Contents

Legal Information ................................................................................................................. 16

Chapter 1: Overview
What’s New ......................................................................................................................... 18
Notational Conventions ........................................................................................................ 18

Chapter 2: Intel(R) Integrated Performance Primitives Concepts
Function Naming ................................................................................................................. 20
  Data-Domain .................................................................................................................. 20
  Name ............................................................................................................................. 20
  Data Types .................................................................................................................. 20
  Descriptors .................................................................................................................. 22
  Parameters .................................................................................................................. 22
  Extensions .................................................................................................................. 23
Function Prototypes in Intel IPP .......................................................................................... 23
Rounding Mode .................................................................................................................. 24
Integer Result Scaling ....................................................................................................... 24
Error Reporting .................................................................................................................. 25
Platform-Aware Functions for Image Processing ............................................................... 27
Threading Layer Functions ............................................................................................... 28
Structures and Enumerators ............................................................................................. 29
  Function Context Structures ....................................................................................... 36
  Structures and Enumerators for Platform-Aware Functions ....................................... 36
Image Data Types and Ranges ........................................................................................... 37
Major Operation Models .................................................................................................... 38
  Neighborhood Operations ......................................................................................... 39
  Regions of Interest in Intel IPP .................................................................................. 39
  Tiled Image Processing .............................................................................................. 42

Chapter 3: Support Functions
Version Information Function ............................................................................................. 43
  GetLibVersion .............................................................................................................. 43
Status Information Function .............................................................................................. 44
  ippGetStatusString ................................................................................................. 44
Memory Allocation Functions ............................................................................................. 45
  Malloc ......................................................................................................................... 45
  Free ............................................................................................................................... 46
Threading Layer Functions ................................................................................................. 47
  SplitUniform2D ......................................................................................................... 47
  ParallelFor ................................................................................................................... 48
  GetTilePointer ............................................................................................................ 48
  GetTileParamsByIndex .............................................................................................. 49
  GetThreadingType ...................................................................................................... 50
  GetThreadIdx ............................................................................................................... 51

Chapter 4: Image Data Exchange and Initialization Functions
Convert ............................................................................................................................... 52
  BinToGray, GrayToBin .............................................................................................. 56
Scale ................................................................................................................................. 58
Chapter 5: Image Arithmetic and Logical Operations

**Arithmetic Operations**
- ScaleC ................................................................. 61
- Set ......................................................................... 62
- Copy ...................................................................... 65
- CopyManaged .......................................................... 69
- CopyConstBorder ..................................................... 70
- CopyMirrorBorder ................................................... 74
- CopyReplicateBorder ................................................. 78
- CopyWrapBorder ....................................................... 81
- CopySubpix .............................................................. 84
- CopySubpixIntersect ................................................ 85
- Dup ........................................................................ 88
- Transpose ................................................................ 89
- SwapChannels ........................................................... 91
- AddRandUniform ...................................................... 94
- AddRandGauss .......................................................... 95
- ImageJaehne ............................................................. 96
- ImageRamp ............................................................... 98
- SampleLine .............................................................. 100
- ZigzagFwd8x8 .......................................................... 101
- ZigzagInv8x8 ............................................................ 103
- Multiply .................................................................. 126
- Add ......................................................................... 140
- AddC ...................................................................... 144
- AddSquare .............................................................. 118
- AddProduct .............................................................. 119
- AddWeighted ............................................................ 120
- Mul ......................................................................... 122
- MulC ........................................................................ 126
- MulC64f ................................................................. 133
- MulScale ................................................................ 135
- MulCscale ............................................................... 137
- Sub ......................................................................... 140
- SubC ...................................................................... 144
- Div ......................................................................... 145
- Div_C ................................................................. 150
- Div_Round .............................................................. 155
- DivC ................................................................. 157
- Abs ......................................................................... 164
- AbsDiff ................................................................. 165
- AbsDiffC ................................................................. 166
- Sqr ......................................................................... 167
- Sqrt ......................................................................... 169
- Ln ........................................................................... 171
- Exp ......................................................................... 173
- DotProd ................................................................. 174
- DotProdCol .............................................................. 177

**Logical Operations**
- Not ........................................................................ 189
- Xor .......................................................................... 187
- Or ............................................................................ 183
- And ......................................................................... 178
- AndC ...................................................................... 179
- Or ............................................................................ 181
- OrC ......................................................................... 185
- Xor ......................................................................... 187
- Not ........................................................................ 189
Chapter 6: Image Color Conversion

Gamma Correction ................................................................. 205
CIE Chromaticity Diagram and Color Gamut ......................... 206
Color Models ....................................................................... 206
Image Downsampling .......................................................... 217
RGB Image Formats ............................................................. 218
Pixel and Planar Image Formats ............................................ 218
Color Model Conversion ....................................................... 222

RGBToYUV ........................................................................... 222
YUVToRGB ......................................................................... 224
RGBToYUV422 ...................................................................... 225
YUV422ToRGB ...................................................................... 226
RGBToYUV420 ...................................................................... 228
YUV420ToRGB ...................................................................... 229
BGRToYUV420 ...................................................................... 231
YUV420ToBGR ...................................................................... 232
YUV422v210ToRGB, YUV422v210ToBGR .............................. 232
YUV422v210ToGray .............................................................. 233
RGBToYCbCr ...................................................................... 234
YCbCrToRGB ....................................................................... 236
YCbCrToBGR ....................................................................... 237
YCbCrToBGR_709CSC ........................................................... 238
RGBToYCbCr422 .................................................................... 239
YCbCr422ToRGB .................................................................... 240
RGBToYCrCb422 .................................................................... 242
YCrCb422ToRGB, YCrCb422ToBGR ................................. 243
BGRToYCbCr422 .................................................................... 244
YCbCr422ToBGR .................................................................... 245
YCbCr422ToGray ................................................................. 246
RGBToCbYCr422, RGBToCbYCr422Gamma ......................... 248
CbYCr422ToRGB ................................................................. 249
BGRToCbYCr422 ................................................................. 249
BGRToCbYCr422_709HDTV .................................................. 250
CbYCr422ToBGR ................................................................. 251
CbYCr422ToBGR_709HDTV .................................................. 252
RGBToYCbCr420 .................................................................... 254
YCbCr420ToRGB, YCbCr420ToBGR ................................. 255
RGBToYCrCb420 .................................................................... 257
YCrCb420ToRGB, YCrCb420ToBGR ................................. 258
BGRToYCbCr420 ................................................................. 260
BGRToYCbCr420_709CSC ..................................................... 261
BGRToYCbCr420_709HDTV .................................................. 262
BGRToYCrCb420_709CSC ..................................................... 263
YCbCr420ToBGR ................................................................. 264
YCbCr420ToBGR_709CSC ..................................................... 265
YCbCr420ToBGR_709HDTV .................................................. 266
BGRToYCrCb420 ................................................................. 268
<table>
<thead>
<tr>
<th>Function</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>BGRToYCbCr411</td>
<td>268</td>
</tr>
<tr>
<td>YCbCr411ToBGRT</td>
<td>269</td>
</tr>
<tr>
<td>RGBToXYZ</td>
<td>270</td>
</tr>
<tr>
<td>XYZToRGB</td>
<td>271</td>
</tr>
<tr>
<td>RGBToLUV, BGRToLUV</td>
<td>272</td>
</tr>
<tr>
<td>LUVToRGB, LUVToBGRT</td>
<td>274</td>
</tr>
<tr>
<td>BGRToLab, RGBToLab</td>
<td>276</td>
</tr>
<tr>
<td>LabToBGRT, LabToRGB</td>
<td>278</td>
</tr>
<tr>
<td>RGBToYCC</td>
<td>280</td>
</tr>
<tr>
<td>YCCToBGRT</td>
<td>281</td>
</tr>
<tr>
<td>RGBToHLS</td>
<td>283</td>
</tr>
<tr>
<td>HLSToRGB</td>
<td>284</td>
</tr>
<tr>
<td>BGRToHLS</td>
<td>285</td>
</tr>
<tr>
<td>HLSToBGR</td>
<td>286</td>
</tr>
<tr>
<td>RGBToHSV</td>
<td>287</td>
</tr>
<tr>
<td>HSVToRGB</td>
<td>288</td>
</tr>
<tr>
<td>RGBToYCoCg</td>
<td>289</td>
</tr>
<tr>
<td>YCoCgToRGB</td>
<td>290</td>
</tr>
<tr>
<td>BGRToYCoCg</td>
<td>291</td>
</tr>
<tr>
<td>SBGRToYCoCg</td>
<td>292</td>
</tr>
<tr>
<td>YCoCgToBGRT</td>
<td>293</td>
</tr>
<tr>
<td>YCoCgToSBGR</td>
<td>294</td>
</tr>
<tr>
<td>BGRToYCoCg_Rev</td>
<td>295</td>
</tr>
<tr>
<td>SBGRToYCoCg_Rev</td>
<td>296</td>
</tr>
<tr>
<td>YCoCgToBGRT_Rev</td>
<td>296</td>
</tr>
<tr>
<td>YCoCgToSBGR_Rev</td>
<td>297</td>
</tr>
<tr>
<td>Color - Gray Scale Conversions</td>
<td>299</td>
</tr>
<tr>
<td>GrayToRGB</td>
<td>299</td>
</tr>
<tr>
<td>RGBToGray</td>
<td>300</td>
</tr>
<tr>
<td>ColorToGray</td>
<td>302</td>
</tr>
<tr>
<td>CFAToBGRA</td>
<td>303</td>
</tr>
<tr>
<td>CFAToRGB</td>
<td>304</td>
</tr>
<tr>
<td>DemosaicAHD</td>
<td>306</td>
</tr>
<tr>
<td>Format Conversion</td>
<td>307</td>
</tr>
<tr>
<td>YCbCr422</td>
<td>307</td>
</tr>
<tr>
<td>YCbCr422ToYCrCb422</td>
<td>308</td>
</tr>
<tr>
<td>YCbCr422ToCbYCr422</td>
<td>309</td>
</tr>
<tr>
<td>YCbCr422ToYCbCr420</td>
<td>310</td>
</tr>
<tr>
<td>YCbCr422To420 _Interlace</td>
<td>312</td>
</tr>
<tr>
<td>YCbCr422ToYCrCb420</td>
<td>313</td>
</tr>
<tr>
<td>YCbCr422ToYCbCr411</td>
<td>314</td>
</tr>
<tr>
<td>YCrCb422ToYCbCr422</td>
<td>315</td>
</tr>
<tr>
<td>YCrCb422ToYCbCr420</td>
<td>316</td>
</tr>
<tr>
<td>YCrCb422ToYCbCr411</td>
<td>317</td>
</tr>
<tr>
<td>CbYCr422ToYCbCr422</td>
<td>317</td>
</tr>
<tr>
<td>CbYCr422ToYCbCr420</td>
<td>318</td>
</tr>
<tr>
<td>CbYCr422ToYCrCb420</td>
<td>319</td>
</tr>
<tr>
<td>CbYCr422ToYCrCb420</td>
<td>320</td>
</tr>
<tr>
<td>CbYCr422ToYCbCr411</td>
<td>321</td>
</tr>
<tr>
<td>YCbCr420</td>
<td>322</td>
</tr>
<tr>
<td>YCbCr420ToYCbCr422</td>
<td>323</td>
</tr>
<tr>
<td>YCbCr420ToYCbCr422 _Filter</td>
<td>324</td>
</tr>
<tr>
<td>YCbCr420To422 _Interlace</td>
<td>326</td>
</tr>
<tr>
<td>YCbCr420ToCbYCr422</td>
<td>327</td>
</tr>
<tr>
<td>YCbCr420ToCbYCr422 _Interlace</td>
<td>328</td>
</tr>
</tbody>
</table>
Chapter 7: Threshold and Compare Operations

Thresholding ................................................................. 369
Threshold ................................................................. 369
Threshold_GT ................................................................. 372
Threshold_LT ................................................................. 373
Threshold_Val ................................................................. 375
Threshold_GTVal ................................................................. 377
Threshold_LTVal ................................................................. 379
Threshold_LTValGTVal ................................................................. 381
ComputeThreshold_Otsu ................................................................. 383
Compare Operations ................................................................. 385
Compare ................................................................. 386
CompareC ................................................................. 387
CompareEqualEps ................................................................. 389
CompareEqualEpsC ................................................................. 390

Chapter 8: Morphological Operations

Dilate3x3 ................................................................. 395
Dilate ................................................................. 396
DilateBorder ................................................................. 398
DilateGetBufferSize ................................................................. 401
DilateGetSpecSize ................................................................. 402
DilateInit ................................................................. 403
Erode3x3 ................................................................. 404
Chapter 9: Filtering Functions

Borders in Neighborhood Operations ........................................................ 451
User-defined Border Types...................................................................... 452
Filters with Borders .............................................................................. 457
  FilterBilateral ................................................................................. 457
  FilterBilateralGetBufferSize .......................................................... 461
  FilterBilateralInit .......................................................................... 462
  FilterBilateralBorderGetBufferSize ............................................... 465
  FilterBilateralBorderInit ............................................................... 466
  FilterBilateralBorder ................................................................. 469
  FilterBoxBorderGetBufferSize ...................................................... 471
  FilterBoxBorder ........................................................................... 472
  FilterBox ...................................................................................... 473
  FilterGaussianBorder ..................................................................... 475
  SumWindow .................................................................................... 476
  SumWindowGetSize .......................................................................... 479
  SumWindowRow .............................................................................. 479
  SumWindowColumn ......................................................................... 481
  FilterMaxBorderGetBufferSize, FilterMinBorderGetBufferSize .......... 482
  FilterMaxBorder, FilterMinBorder .................................................. 483
  DecimateFilterRow, DecimateFilterColumn ...................................... 486
Median Filters ....................................................................................... 488
  FilterMedianBorderGetBufferSize .................................................. 488
  FilterMedianBorder ....................................................................... 489
Chapter 10: Image Linear Transforms

Fourier Transforms ................................................................. 597
  Real - Complex Packed (RCPack2D) Format ................................ 597
  FFTGetSize ............................................................................ 598
  FFTInit .................................................................................. 599
  FFTFwd .................................................................................. 601
  FFTInv .................................................................................. 604
  DFTGetSize ............................................................................ 606
  DFTInit .................................................................................. 607
  DFTFwd .................................................................................. 609
  DFTInv .................................................................................. 611
  MulPack ............................................................................... 612
  MulPackConj ......................................................................... 614
  Magnitude ............................................................................. 616
  MagnitudePackGetSize ........................................................... 617
  MagnitudePack ................................................................. 618
  Phase .................................................................................... 619
  PhasePackGetSize ................................................................. 620
  PhasePack ................................................................. 620
  PolarToCart ........................................................................... 622
  PackToCplxExtend ............................................................... 623
  CplxExtendToPack ............................................................... 624

Windowing Functions .............................................................. 625
  WinBartlettGetSize, WinBartlettSepGetSize .................................. 625
  WinBartlett, WinBartlettSep .................................................. 626
  WinHammingGetSize, WinHammingSepGetSize ............................ 628
  WinHamming, WinHammingSep ............................................ 629

Discrete Cosine Transforms ...................................................... 631
Chapter 11: Image Statistics Functions

Sum ................................................................................................................. 645
Integral............................................................................................................. 647
SqrIntegral ........................................................................................................ 649
TiltedIntegral ..................................................................................................... 651
TiltedSqrIntegral ................................................................................................ 652
Mean .................................................................................................................. 654
Mean_StdDev ..................................................................................................... 656
RectStdDev ......................................................................................................... 658
TiltedRectStdDev ............................................................................................... 659
HistogramGetBufferSize ................................................................................. 661
HistogramGetLevels .......................................................................................... 662
HistogramInit, HistogramUniformInit ............................................................... 663
Histogram .......................................................................................................... 664
CountInRange ..................................................................................................... 667
BlockMinMax ..................................................................................................... 668
Min ...................................................................................................................... 670
MinIndx .............................................................................................................. 672
Max ..................................................................................................................... 673
MaxIndx .............................................................................................................. 674
MinMax ............................................................................................................ 676
MinMaxIndx ....................................................................................................... 677
MaxEvery ........................................................................................................... 679
MinEvery ........................................................................................................... 681
FindPeaks3x3GetBufferSize ............................................................................. 682
FindPeaks3x3 ...................................................................................................... 683
Image Moments .................................................................................................. 685
  MomentGetStateSize ........................................................................................ 686
  MomentInit ........................................................................................................ 686
  Moments ............................................................................................................ 687
  GetSpatialMoment ........................................................................................... 688
  GetCentralMoment .......................................................................................... 689
  GetNormalizedSpatialMoment ......................................................................... 690
  GetNormalizedCentralMoment ...................................................................... 691
  GetHuMoments ............................................................................................... 691
Image Norms ....................................................................................................... 692
  Norm_Inf .......................................................................................................... 692
  Norm_L1 ........................................................................................................... 694
  Norm_L2 .......................................................................................................... 697
  NormDiff_Inf ................................................................................................. 699
  NormDiff_L1 ................................................................................................... 702
  NormDiff_L2 ................................................................................................... 705
  NormRel_Inf .................................................................................................... 707
  NormRel_L1 ..................................................................................................... 709
Chapter 12: Image Geometry Transforms

ROI Processing in Geometric Transforms

Geometric Transform Functions

ResizeYCbCr422GetSize .................................................... 734
ResizeYCbCr422 ................................................................... 735
Resize Functions with Prior Initialization .......................... 736
  Using Intel® IPP Resize Functions with Prior Initialization .... 736
    ResizeGetSize .................................................................. 745
    ResizeGetBufferSize ..................................................... 747
    ResizeGetBorderSize .................................................... 749
    ResizeGetSrcOffset ..................................................... 750
    ResizeGetSrcRoi .......................................................... 751
    ResizeNearestInit ....................................................... 752
    ResizeNearest .................................................................. 753
    ResizeLinearInit .......................................................... 756
    ResizeLinear ..................................................................... 757
    ResizeCubicInit ............................................................ 761
    ResizeCubic ...................................................................... 763
    ResizeLanczosInit ........................................................ 765
    ResizeLanczos ................................................................ 767
    ResizeSuperInit ............................................................ 770
    ResizeSuper ................................................................. 772
    ResizeAntialiasingLinearInit ........................................... 774
    ResizeAntialiasingCubicInit ............................................. 775
    ResizeAntialiasingLanczosInit ........................................ 777
    ResizeAntialiasing ........................................................ 778
    ResizeFilterGetSize ....................................................... 781
    ResizeFilterInit ............................................................ 782
    ResizeFilter ................................................................. 783
    ResizeYUV420GetSize ...................................................... 784
    ResizeYUV420GetSrcRoi ................................................... 785
    ResizeYUV420LanczosInit ................................................. 786
    ResizeYUV420SuperInit ................................................... 788
    ResizeYUV420GetSize ...................................................... 788
    ResizeYUV420GetSrcOffset .............................................. 789
    ResizeYUV420GetBufferSize ............................................. 790
    ResizeYUV420Lanczos .................................................... 791
    ResizeYUV420Super ....................................................... 793
    ResizeYUV422GetSize ...................................................... 795
    ResizeYUV422GetBorderSize ............................................ 796
    ResizeYUV422GetSrcOffset .............................................. 797
    ResizeYUV422GetBufferSize ............................................ 797
    ResizeYUV422GetSrcRoi ................................................... 798
Chapter 13: Miscellaneous Image Transforms

ThresholdAdaptiveBoxGetBufferSize ............................................. 879
ThresholdAdaptiveBox ................................................................. 880
ThresholdAdaptiveGaussGetBufferSize .......................................... 882
ThresholdAdaptiveGaussInit .......................................................... 884
ThresholdAdaptiveGauss ............................................................... 885

Chapter 14: Wavelet Transforms
Chapter 15: Computer Vision

UsingippiAddforBackgroundDifferencing .......................................................... 907
FeatureDetectionFunctions .................................................................................. 907
CornerDetection ................................................................................................. 908
FastNGetSize ....................................................................................................... 908
FastNGetBufferSize .............................................................................................. 910
FastNInit ................................................................................................................ 911
FastN ..................................................................................................................... 912
FastN2DToVec ....................................................................................................... 916
HarrisCornerGetBufferSize .................................................................................. 917
HarrisCorner ......................................................................................................... 918
CannyEdgeDetector ............................................................................................... 922
CannyBorderGetSize ............................................................................................. 923
CannyBorder ......................................................................................................... 924
CannyGetSize ......................................................................................................... 925
Canny ................................................................................................................... 926
EigenValsVecsGetBufferSize .............................................................................. 928
EigenValsVecsBorder ............................................................................................ 929
EigenValsVecs ....................................................................................................... 931
MinEigenValGetBufferSize .................................................................................. 933
MinEigenValBorder ............................................................................................... 934
MinEigenVal .......................................................................................................... 936
HistogramofOrientedGradients(HOG)Descriptor ................................................. 937
HOGGetSize ........................................................................................................... 939
HOGInit ................................................................................................................. 939
HOGGetBufferSize ............................................................................................... 940
HOGGetDescriptorSize ......................................................................................... 941
HOG ....................................................................................................................... 942
HoughTransform .................................................................................................... 944
HoughLineGetSize .................................................................................................. 944
HoughLine ............................................................................................................. 945
HoughLine_Region ............................................................................................... 946
HoughProbLineGetSize ......................................................................................... 947
HoughProbLineInit ............................................................................................... 948
HoughProbLine ...................................................................................................... 949
LineSuppressionGetSize ....................................................................................... 950
LineSuppression ................................................................................................. 951
DistanceTransformFunctions .................................................................................. 954
DistanceTransform ............................................................................................... 954
GetDistanceTransformMask .................................................................................. 957
FastMarchingGetSize ............................................................................................. 958
FastMarching ......................................................................................................... 959
TrueDistanceTransformGetSize .............................................................................. 961
TrueDistanceTransform ....................................................................................... 961
ImageGradients ..................................................................................................... 962
GradientColorToGray ............................................................................................ 963
GradientVectorGetSize ......................................................................................... 964
GradientVectorPrewitt ......................................................................................... 965
GradientVectorScharr ................................................................. 968
GradientVectorSobel ............................................................... 971

Flood Fill Functions ................................................................. 975
FloodFillGetSize .................................................................... 975
FloodFillGetSize_Grad ............................................................ 976
FloodFill .............................................................................. 977
FloodFill_Grad ................................................................. 978
FloodFill_Range ................................................................. 980

Motion Analysis and Object Tracking ...................................... 982
FGMMGetSize ..................................................................... 982
FGMMInit ......................................................................... 983
FGMFForeground .................................................................. 984
FGMMBackground ............................................................... 985

Motion Template Functions ...................................................... 986
Motion Representation ......................................................... 986
Updating MHI Images ......................................................... 987
UpdateMotionHistory .......................................................... 987

Optical Flow ........................................................................ 988
OpticalFlowPyrLKGetSize .................................................... 989
OpticalFlowPyrLKInit ......................................................... 990
OpticalFlowPyrLK ............................................................... 991

Universal Pyramids .................................................................. 993
PyramidGetSize .................................................................... 993
PyramidInit ......................................................................... 994
GetPyramidDownROI .......................................................... 995
GetPyramidUpROI ............................................................. 996
PyramidLayerDownGetSize .................................................. 997
PyramidLayerDownInit ........................................................ 998
PyramidLayerDown ............................................................ 1000
PyramidLayerUpGetSize ........................................................ 1001
PyramidLayerUpInit ............................................................ 1003
PyramidLayerUp ................................................................. 1004

Image Inpainting ................................................................. 1006
InpaintGetSize .................................................................... 1006
InpaintInit ......................................................................... 1007
Inpaint ............................................................................. 1009

Image Segmentation .............................................................. 1010
LabelMarkersGetSize ........................................................... 1011
LabelMarkers ...................................................................... 1011
MarkSpecklesGetSize ............................................................ 1013
MarkSpeckles ...................................................................... 1014
SegmentWatershedGetSize .................................................... 1015
SegmentWatershed .............................................................. 1016
SegmentGradientGetSize ........................................................ 1019
SegmentGradient ................................................................. 1020
BoundSegments ................................................................. 1021

Pattern Recognition .............................................................. 1022
Object Detection Using Haar-like Features .............................. 1023
HaarClassifierGetSize ........................................................... 1024
HaarClassifierInit ............................................................... 1025
GetHaarClassifierSize ........................................................ 1026
TiltedHaarClassifierInit ........................................................ 1027
TiltHaarFeatures ................................................................. 1028
ApplyHaarClassifier ............................................................ 1029
Chapter 16: 3D Data Processing Functions

Appendix A: Handling of Special Cases

Appendix B: Interpolation in Image Geometric Transform Functions

Appendix C: Appendix C: Removed Functions for Image and Video Processing

Appendix D: Bibliography for Image Processing

Appendix E: Glossary
Legal Information

No license (express or implied, by estoppel or otherwise) to any intellectual property rights is granted by this document.

Intel disclaims all express and implied warranties, including without limitation, the implied warranties of merchantability, fitness for a particular purpose, and non-infringement, as well as any warranty arising from course of performance, course of dealing, or usage in trade.

This document contains information on products, services and/or processes in development. All information provided here is subject to change without notice. Contact your Intel representative to obtain the latest forecast, schedule, specifications and roadmaps.

The products and services described may contain defects or errors which may cause deviations from published specifications. Current characterized errata are available on request.


Intel, the Intel logo, Intel Atom, Intel Core, Intel Xeon Phi, VTune and Xeon are trademarks of Intel Corporation in the U.S. and/or other countries.

*Other names and brands may be claimed as the property of others.

Java is a registered trademark of Oracle and/or its affiliates.

Microsoft, Windows, and the Windows logo are trademarks, or registered trademarks of Microsoft Corporation in the United States and/or other countries.

Java is a registered trademark of Oracle and/or its affiliates.

Third Party

Intel® Integrated Performance Primitives (Intel® IPP) includes content from several 3rd party sources that was originally governed by the licenses referenced below:

- zlib library:
  zlib.h -- interface of the 'zlib' general purpose compression library version 1.2.8, April 28th, 2013
  Copyright (C) 1995-2013 Jean-loup Gailly and Mark Adler
  This software is provided 'as-is', without any express or implied warranty. In no event will the authors be held liable for any damages arising from the use of this software.
  Permission is granted to anyone to use this software for any purpose, including commercial applications, and to alter it and redistribute it freely, subject to the following restrictions:
  1. The origin of this software must not be misrepresented; you must not claim that you wrote the original software. If you use this software in a product, an acknowledgment in the product documentation would be appreciated but is not required.
  2. Altered source versions must be plainly marked as such, and must not be misrepresented as being the original software.
  3. This notice may not be removed or altered from any source distribution.
  Jean-loup Gailly Mark Adler
  jloup@gzip.org madler@alumni.caltech.edu

- bzip2:
Copyright © 1996 - 2015 julian@bzip.org

Copyright 2000-2018 Intel Corporation.
This software and the related documents are Intel copyrighted materials, and your use of them is governed by the express license under which they were provided to you (License). Unless the License provides otherwise, you may not use, modify, copy, publish, distribute, disclose or transmit this software or the related documents without Intel's prior written permission.

This software and the related documents are provided as is, with no express or implied warranties, other than those that are expressly stated in the License.
Volume Overview

This manual describes the structure, operation, and functions of the Intel® Integrated Performance Primitives (Intel® IPP) that operate on two-dimensional signals and are used for image and video processing. The manual explains the Intel IPP concepts, as well as specific data type definitions and operation models used in the image and video processing domain, and provides detailed descriptions of the Intel IPP image and video processing functions. The Intel IPP functions are combined in groups by their functionality. Each group of functions is described in a separate chapter (chapters 3 through 16).

For more information about image and video processing concepts and algorithms, refer to the books and materials listed in the Bibliography.

What's New

This Developer Reference documents Intel® Integrated Performance Primitives (Intel® IPP) 2019 release.

The document has been updated to reflect the following changes to the product:

Image Processing:
- Added new Threading Layer (TL) APIs for several functions. See Threading Layer Functions
- Added Threading Layer support functions
- Added the FilterMedian, FilterMedianInit, and FilterMedianGetSize functions. See 3D Data Processing Functions for details.
- Added new flavors to the FilterBorder, FilterBorderInit, and FilterBorderGetSize functions. See 3D Data Processing Functions for details.
- Added new flavors to the FilterBilateral, FilterBilateralInit, and FilterBilateralGetBufferSize functions. See Filters with Borders for details.
- Added new flavors to the LabToRGB and RGBToLab functions. See Color Model Conversion for details.

Additionally, minor updates have been made to fix inaccuracies in the document.

Notational Conventions

The code and syntax used in this manual for function and variable declarations are written in the ANSI C style. However, versions of Intel IPP for different processors or operating systems may, of necessity, vary slightly.

Optimization Notice

Intel's compilers may or may not optimize to the same degree for non-Intel microprocessors for optimizations that are not unique to Intel microprocessors. These optimizations include SSE2, SSE3, and SSSE3 instruction sets and other optimizations. Intel does not guarantee the availability, functionality, or effectiveness of any optimization on microprocessors not manufactured by Intel. Microprocessor-dependent optimizations in this product are intended for use with Intel microprocessors. Certain optimizations not specific to Intel microarchitecture are reserved for Intel microprocessors. Please refer to the applicable product User and Reference Guides for more information regarding the specific instruction sets covered by this notice.

Notice revision #20110804

This manual uses the following notational conventions:
<table>
<thead>
<tr>
<th>Convention</th>
<th>Explanation</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>THIS TYPE STYLE</td>
<td>Used in the text for the Intel IPP constant identifiers.</td>
<td>IPPI_INTER_LINEAR</td>
</tr>
<tr>
<td>This type style</td>
<td>Mixed with the uppercase in structure names; also used in function names, code examples and call statements.</td>
<td>IppiSize, ippiMomentInitAlloc()</td>
</tr>
<tr>
<td>This type style</td>
<td>Parameters in function prototypes and parameters description.</td>
<td>value, srcStep</td>
</tr>
<tr>
<td>$x(n)$ and $x[n]$</td>
<td>Used to represent a discrete 1D signal. The notation $x(n)$ refers to a conceptual signal, while the notation $x[n]$ refers to an actual vector. Both of these are annotated to indicate a specific finite range of values.</td>
<td>$x[n], \ 0 \leq n &lt; len$</td>
</tr>
<tr>
<td>Ipp&lt;domain&gt; and Ipp prefixes</td>
<td>All structures and enumerators, specific for the image and video processing domain have the Ippi prefix, while those common for entire Intel IPP software have the Ipp prefix.</td>
<td>IppiPoint, IppDitherType</td>
</tr>
</tbody>
</table>

See Also
Function Naming
Intel® Integrated Performance Primitives Concepts

This chapter explains the purpose and structure of the Intel® Integrated Performance Primitives (Intel® IPP) for Intel® Architecture software and looks over some of the basic concepts used in the image part of Intel IPP. It also describes the supported data formats and operation modes, and defines function naming conventions in the document.

Function Naming

Naming conventions for the Intel IPP functions are similar for all covered domains. Function names in Intel IPP have the following general format:

```cpp
ipp<data-domain><name>_<datatypedescriptor>[_extension](parameters);
```

The elements of this format are explained in the sections that follow.

Data-Domain

The data-domain is a single character that denotes the subset of functionality to which a given function belongs. The current version of Intel IPP supports the following data-domains:

- **s** for signals (expected data type is a 1D signal)
- **i** for images and video (expected data type is a 2D image)
- **m** for matrices (expected data type is a matrix)
- **r** for realistic rendering functionality and 3D data processing (expected data type depends on supported rendering techniques)
- **g** for signals of fixed length

For example, function names that begin with **ippi** signify that respective functions are used for image or video processing.

Name

The name field identifies what function does and has the following format:

```cpp
<name> = <operation>[_modifier]
```

The operation component is one or more words, acronyms, and abbreviations that describe the core operation.

The modifier component, if present, is a word or abbreviation that denotes a slight modification or variation of the given function.

For example, names without modifiers: Add, RGBToYCbCr, MorphAddGetSize; with modifiers: DCT8x8Inv_2x2, DCT8x8Inv_4x4, RGBToYCbCr_JPEG.

Data Types

The datatype field indicates data types used by the function, in the following format:
<bit depth><bit interpretation> ,
where
bit depth = <1|8|16|32|64>
and
bit interpretation = <u|s|f>[c]
Here u indicates “unsigned integer”, s indicates “signed integer”, f indicates “floating point”, and c indicates “complex”.

Intel IPP supports the following data types for image and video processing functions:

- 8u  
  8 bit, unsigned data

- 8s  
  8 bit, signed data

- 16u  
  16 bit, unsigned data

- 16uc  
  16-bit, complex unsigned short data

- 16s  
  16 bit, signed data

- 16sc  
  16-bit, complex short data

- 32u  
  32 bit, unsigned data

- 32s  
  32 bit, signed data

- 32sc  
  32-bit, complex int data

- 32f  
  32-bit, single-precision real floating point data

- 32fc  
  32-bit, single-precision complex floating point data

- 64s  
  64-bit, quadword signed data

- 64f  
  64-bit, double-precision real floating point data

† - only partial support for intermediate result after transforms (in the so-called “time” domain).

NOTE
For image processing functions that do not support 1u data type, convert bitonal images to 8u gray scale images using theippiConvert_1u8u_C1R function.

The formats for complex data are represented in Intel IPP by structures defined as follows:

```c
typedef struct {     Ipp16s re;     Ipp16s im; } Ipp16sc;
typedef struct {     Ipp32s re;     Ipp32s im; } Ipp32sc;
typedef struct {     Ipp32f re;     Ipp32f im; } Ipp32fc;
```
where re, im denote the real and imaginary parts, respectively.

Complex data formats are used by several arithmetic image processing functions. The 32fc format is also used to store input/output data in some Fourier transform functions.

The 64-bit formats, 64s and 64f, are used for storing data computed by some image statistics functions.

For functions that operate on a single data type, the datatype field contains only one of the values listed above.

If a function operates on source and destination images that have different data types, the respective data type identifiers are listed in the function name in order of source and destination as follows:

```c
<datatype> = <src1Datatype>[src2Datatype][dstDatatype]
```
For example, the function that converts 8-bit unsigned source image data to 32-bit floating point destination image data has the `8u32f` value for the `datatype` field.

**NOTE**
In the lists of function parameters (arguments), the `Ipp` prefix is written in the data type. For example, the 8-bit unsigned data is denoted as `Ipp8u` type. These Intel IPP-specific data types are defined in the respective library header files.

**Descriptors**

The `descriptors` field further describes the operation. Descriptors are individual characters that indicate additional details of the operation.

The following descriptors are used in image and video processing functions:

<table>
<thead>
<tr>
<th>Descriptor</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>Image data contains an alpha channel as the last channel, requires <code>C4</code>, alpha-channel is not processed</td>
</tr>
<tr>
<td>A0</td>
<td>Image data contains an alpha channel as the first channel, requires <code>C4</code>, alpha-channel is not processed</td>
</tr>
<tr>
<td>C1, C2, C3, C4</td>
<td>Image data is in pixel order and made up of 1, 2, 3 or 4 discrete interleaved channels</td>
</tr>
<tr>
<td>C</td>
<td>Channel of interest (COI) is used in the operation</td>
</tr>
<tr>
<td>I</td>
<td>Operation is performed in-place - that is result of operation is written back into the source (default is not-in-place)</td>
</tr>
<tr>
<td>M</td>
<td>Operation uses a mask to determine pixels to be processed</td>
</tr>
<tr>
<td>P2, P3, P4</td>
<td>Image data is in planar order and made up of 2, 3 or 4 discrete planar (non-interleaved) channels, with a separate pointer to each plane</td>
</tr>
<tr>
<td>R</td>
<td>Function operates on a defined region of interest (ROI) for each source image</td>
</tr>
<tr>
<td>Sfs</td>
<td>Saturation and fixed scaling mode is used</td>
</tr>
<tr>
<td>S</td>
<td>Saturation and no scaling (default)</td>
</tr>
<tr>
<td>V</td>
<td>Function operates on a defined volume of interest (VOI) for each source image</td>
</tr>
</tbody>
</table>

If more that one descriptor is used, they are presented in the function name in alphabetical order.

Every function that operates on image data has a channel count descriptor `Cn` (for interleaved image) or `Pn` (for planar). No default channel count is defined.

If input and output channel layouts are different, both source and destination layouts are listed. For example, the function `ippiHLSToBGR_8u_C3P3R` converts three-channel interleaved HLS image to the three-plane BGR image.

**Parameters**

The parameters in functions are in the following order: all source operands, all destination operands, all other operation-specific parameters.
Source parameters are named $Src$ or $SrcN$, if there is more than one input image. Destination parameters are named $Dst$. For in-place operations, the input/output parameter contains the name $SrcDst$. All parameters defined as pointers start with lowercase $p$, for example, $pSrc$, $pMean$, $pSpec$.

**Extensions**

The *extension* field denotes an Intel IPP extension to which the function belongs. The following extensions are supported in Intel IPP Image Processing functions:

<table>
<thead>
<tr>
<th>Extension</th>
<th>Description</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>L</td>
<td>Intel IPP platform-aware functions</td>
<td>ippiAddC_8u_AC4R_L</td>
</tr>
<tr>
<td>LT</td>
<td>Intel IPP threading layer (TL) functions based on the Platform Aware API</td>
<td>ippiSubC_8u_C7RSfs_LT</td>
</tr>
<tr>
<td>T</td>
<td>Intel IPP threading layer (TL) functions based on the Classic API</td>
<td>ipprFilterMedian_64f_C1V_T</td>
</tr>
</tbody>
</table>

**See Also**

Platform-Aware Functions for Image Processing
Threading Layer Functions

**Function Prototypes in Intel IPP**

Function names in Intel IPP contain *datatype* and *descriptor* fields after the *name* field (see *Function Naming* section in this chapter). Most of the Intel IPP functions for image processing have a number of flavors that differ in data types associated with the operation, and in some additional parameters.

Each function flavor has its unique prototype used in function definition and for calling the function from the application program. For many flavors of a given function, these prototypes look quite similar.

To avoid listing all the similar prototypes in function description sections of some chapters in this document, only different templates for such prototypes followed by the table of applicable data types and descriptors for each function may be given. For simplicity, in such cases the data type and descriptor fields in the function name are denoted as $mod$:

$<mod> = <datatype>_<datatype>$

For example, the template for the prototype of the image dilation function that performs not-in-place operation, looks like this:

```c
IppStatus ippiDilate_<mod>(const Ipp<datatype>* pSrc, int srcStep, Ipp<datatype>* pDst, int dstStep, IppiSize roiSize);
```

where the supported values for $mod$ are:

- $8u_C1R$
- $8u_C3R$
- $8u_AC4R$

This notation means that the *ippiDilate* function has three flavors for a not-in-place operation, which process 8-bit unsigned data (of Ipp8u type) and differ in the number of channels in processed images. These flavors have the following prototypes:

```c
IppStatus ippiDilate_8u_C1R(const Ipp8u* pSrc, int srcStep, IppiSize roiSize);  // Ipp8u* pDst, int dstStep,
IppStatus ippiDilate_8u_C3R(const Ipp8u* pSrc, int srcStep, IppiSize roiSize);  // Ipp8u* pDst, int dstStep,
IppStatus ippiDilate_8u_AC4R(const Ipp8u* pSrc, int srcStep, IppiSize roiSize);  // Ipp8u* pDst, int dstStep,
```
Thus, to obtain the full name and parameters list for the specific function flavor, not listed directly, do the following:

1. Choose the function operation mode (denoted in this document as **Case 1**, **2** and so on) and look in the table for the supported data types and descriptors.
2. Set the **mod** field in the function name as the concatenation of the chosen data type and descriptor, delimited by the underscore.
3. Use the respective template, substituting all the **datatype** fields in the parameters list with the chosen data type. Note that **Ipp** prefix is written before the **datatype** in the parameters list (see **Data Types** in this chapter for details).

**Example**

To get the prototype for the **ippiSet** function flavor that sets each channel of a 3-channel destination image to 16-bit signed values, choose **Case 2: Setting each color channel to a specified value** and use **datatype** = **16s**, **descriptors** = **C3R**.

After substituting the **mod** field with **16s_C3R**, obtain the required prototype as

```
ippiSet_16s_C3R(const Ipp16s value[3], Ipp16* pDst, int dstStep, IppiSize roiSize);
```

---

**Rounding Mode**

As many Intel IPP functions have to meet the bit-exact requirement, image processing functions use rounding. The default rounding mode for all functions can be described as "nearest even", that is the fixed point number \( x = N + \alpha, \, 0 \leq \alpha < 0 \), where \( N \) is an integer number, is rounded as given by:

\[
\lceil x \rceil = \begin{cases} 
N & 0 < \alpha < 0.5 \\
N + 1 & 0.5 < \alpha < 1 \\
N + 1 & \alpha = 0.5, N - \text{even} \\
N + 1 & \alpha = 0.5, N - \text{odd}
\end{cases}
\]

For example, 1.5 is rounded to 2, and 2.5 to 2.

Some image processing functions have additional rounding modes, which are set by the parameter **roundMode**.

---

**Integer Result Scaling**

The default for image processing functions is to saturate the results without scaling them.

Some image processing functions operating on integer data use scaling of the internally computed output results by the integer **scaleFactor**, which is specified as one of the function parameters. These functions have the **Sfs** descriptor in their names.

The scale factor can be negative, positive, or zero. Scaling is applied because internal computations are generally performed with a higher precision than the data types used for input and output images.

**NOTE**

The result of integer operations is always saturated to the destination data type range even when scaling is used.

The scaling of an integer result is done by multiplying the output pixel values by \( 2^{-\text{scaleFactor}} \) before the function returns. This helps retain either the output data range or its precision. Usually the scaling with a positive factor is performed by the shift operation. The result is rounded off to the nearest even integer number (see **Rounding Mode**).
For example, the integer \texttt{Ipp16s} result of the square operation \texttt{ippiSqr} for the input value 200 is equal to 32767 instead of 40000, that is, the result is saturated and the exact value cannot be restored. The scaling of the output value with the factor \texttt{scaleFactor} = 1 yields the result 20000, which is not saturated, and the exact value can be restored as 20000*2. Thus, the output data range is retained.

The following example shows how the precision can be partially retained by means of scaling. The integer square root operation \texttt{ippiSqr} (without scaling) for the input value 2 gives the result equal to 1 instead of 1.414. Scaling of the internally computed output value with the factor \texttt{scaleFactor} = -3 gives the result 11, and permits the more precise value to be restored as $11*2^{-3} = 1.375$.

## Error Reporting

The Intel IPP functions return status codes of the performed operation to report errors and warnings to the calling program. Thus, it is up to the application to perform error-related actions and/or recover from the error. The last value of the error status is not stored, and the user is to decide whether to check it or not as the function returns. The status codes are of \texttt{IppStatus} type and are global constant integers.

The status codes and corresponding messages reported by Intel IPP for image and video processing are listed in Table.

<table>
<thead>
<tr>
<th>Status Code</th>
<th>Message</th>
</tr>
</thead>
<tbody>
<tr>
<td>ippStsNotSupportedModeErr</td>
<td>The requested mode is currently not supported.</td>
</tr>
<tr>
<td>ippStsDecimateFractionErr</td>
<td>Unsupported fraction in decimate.</td>
</tr>
<tr>
<td>ippStsWeightErr</td>
<td>Incorrect value for weight.</td>
</tr>
<tr>
<td>ippStsQualityIndexErr</td>
<td>Quality Index cannot be calculated for an image filled with a constant.</td>
</tr>
<tr>
<td>ippStsResizeNoOperationErr</td>
<td>One of the output image dimensions is less than 1 pixel.</td>
</tr>
<tr>
<td>ippStsBlockStepErr</td>
<td>Step for Block is less than 8.</td>
</tr>
<tr>
<td>ippStsMBStepErr</td>
<td>Step for MB is less than 16.</td>
</tr>
<tr>
<td>ippStsNoiseRangeErr</td>
<td>Noise value for Wiener Filter is out of range.</td>
</tr>
<tr>
<td>ippStsLPCalcErr</td>
<td>Cannot evaluate linear prediction.</td>
</tr>
<tr>
<td>ippStsJPEG2KBadPassNumber</td>
<td>Pass number exceeds the limits of $[0,nOfPasses-1]$.</td>
</tr>
<tr>
<td>ippStsJPEG2KDamagedCodeBlock</td>
<td>Codeblock for decoding is damaged.</td>
</tr>
<tr>
<td>ippStsH263CBPYCodeErr</td>
<td>Illegal Huffman code is detected during CBPY stream processing.</td>
</tr>
<tr>
<td>ippStsH263MCBPCInterCodeErr</td>
<td>Illegal Huffman code is detected during MCBPC Inter stream processing.</td>
</tr>
<tr>
<td>ippStsH263MCBFCintraCodeErr</td>
<td>Illegal Huffman code is detected during MCBPC Intra stream processing.</td>
</tr>
<tr>
<td>ippStsNotEvenStepErr</td>
<td>Step value is not pixel multiple.</td>
</tr>
<tr>
<td>ippStsHistoNofLevelsErr</td>
<td>Number of levels for histogram is less than 2.</td>
</tr>
<tr>
<td>ippStsLUTNofLevelsErr</td>
<td>Number of levels for LUT is less than 2.</td>
</tr>
<tr>
<td>ippStsMP4BitOffsetErr</td>
<td>Incorrect value for bit offset.</td>
</tr>
<tr>
<td>ippStsMP4QPErr</td>
<td>Incorrect value for quantization parameter.</td>
</tr>
<tr>
<td>ippStsMP4BlockIdxErr</td>
<td>Incorrect value for block index.</td>
</tr>
<tr>
<td>ippStsMP4BlockTypeErr</td>
<td>Incorrect block type.</td>
</tr>
<tr>
<td>ippStsMP4MVCodeErr</td>
<td>Illegal Huffman code is detected during MV stream processing.</td>
</tr>
<tr>
<td>ippStsMP4VLCCodeErr</td>
<td>Illegal Huffman code is detected during VLC stream processing.</td>
</tr>
<tr>
<td>ippStsMP4DCCodeErr</td>
<td>Illegal code is detected during DC stream processing.</td>
</tr>
<tr>
<td>ippStsMP4FcodeErr</td>
<td>Incorrect value for fcode.</td>
</tr>
<tr>
<td>ippStsMP4AlignErr</td>
<td>Incorrect buffer alignment.</td>
</tr>
<tr>
<td>ippStsMP4TempDiffErr</td>
<td>Incorrect temporal difference.</td>
</tr>
<tr>
<td>ippStsMP4BlockSizeErr</td>
<td>Incorrect size of block or macroblock.</td>
</tr>
<tr>
<td>ippStsMP4ZeroBABErr</td>
<td>All BAB values are equal to zero.</td>
</tr>
<tr>
<td>ippStsMP4PredDirErr</td>
<td>Incorrect prediction direction.</td>
</tr>
<tr>
<td>Status Code</td>
<td>Message</td>
</tr>
<tr>
<td>---------------------------------</td>
<td>-------------------------------------------------------------------------</td>
</tr>
<tr>
<td>ippStsMP4BitsPerPixelErr</td>
<td>Incorrect number of bits per pixel.</td>
</tr>
<tr>
<td>ippStsMP4VideoCompModeErr</td>
<td>Incorrect video component mode.</td>
</tr>
<tr>
<td>ippStsMP4LinearModeErr</td>
<td>Incorrect DC linear mode.</td>
</tr>
<tr>
<td>ippStsH263PredModeErr</td>
<td>Incorrect value for Prediction Mode.</td>
</tr>
<tr>
<td>ippStsH263BlockStepErr</td>
<td>The step value is less than 8.</td>
</tr>
<tr>
<td>ippStsH263MBStepErr</td>
<td>The step value is less than 16.</td>
</tr>
<tr>
<td>ippStsH263FrameWidthErr</td>
<td>The frame width is less than 8.</td>
</tr>
<tr>
<td>ippStsH263FrameHeightErr</td>
<td>The frame height is less than, or equal to zero.</td>
</tr>
<tr>
<td>ippStsH263ExpandPelsErr</td>
<td>The expand pixels number is less than 8.</td>
</tr>
<tr>
<td>ippStsH263PlaneStepErr</td>
<td>Step value is less than the plane width.</td>
</tr>
<tr>
<td>ippStsH263QuantErr</td>
<td>Quantizer value is less than, or equal to zero, or more than 31.</td>
</tr>
<tr>
<td>ippStsH263MVCodeErr</td>
<td>Illegal Huffman code is detected during MV stream processing.</td>
</tr>
<tr>
<td>ippStsH263VLCCodeErr</td>
<td>Illegal Huffman code is detected during VLC stream processing.</td>
</tr>
<tr>
<td>ippStsH263DCCodeErr</td>
<td>Illegal code is detected during DC stream processing.</td>
</tr>
<tr>
<td>ippStsH263ZigzagLenErr</td>
<td>Zigzag compact length is more than 64.</td>
</tr>
<tr>
<td>ippStsJPEGHuffTableErr</td>
<td>JPEG Huffman table is destroyed.</td>
</tr>
<tr>
<td>ippStsJPEGDCTRangeErr</td>
<td>JPEG DCT coefficient is out of range.</td>
</tr>
<tr>
<td>ippStsJPEGOutOfBufErr</td>
<td>An attempt to access out of the buffer.</td>
</tr>
<tr>
<td>ippStsChannelOrderErr</td>
<td>Incorrect order of the destination channels.</td>
</tr>
<tr>
<td>ippStsZeroMaskValueErr</td>
<td>All values of the mask are equal to zero.</td>
</tr>
<tr>
<td>ippStsRangeErr</td>
<td>Incorrect values for bounds: the lower bound is greater than the upper</td>
</tr>
<tr>
<td></td>
<td>bound.</td>
</tr>
<tr>
<td>ippStsQPErr</td>
<td>Incorrect value for a quantizer parameter.</td>
</tr>
<tr>
<td>ippStsQuadErr</td>
<td>The quadrangle is nonconvex or degenerates into triangle, line, or</td>
</tr>
<tr>
<td></td>
<td>point.</td>
</tr>
<tr>
<td>ippStsRectErr</td>
<td>Size of the rectangular region is less than, or equal to 1.</td>
</tr>
<tr>
<td>ippStsCoeffErr</td>
<td>Incorrect values for the transformation coefficients.</td>
</tr>
<tr>
<td>ippStsNoiseValErr</td>
<td>Incorrect value for the noise amplitude for dithering.</td>
</tr>
<tr>
<td>ippStsDitherLevelsErr</td>
<td>Number of dithering levels is out of range.</td>
</tr>
<tr>
<td>ippStsNumChannelsErr</td>
<td>Incorrect or unsupported number of channels.</td>
</tr>
<tr>
<td>ippStsDataTypeErr</td>
<td>Incorrect or unsupported data type.</td>
</tr>
<tr>
<td>ippStsCOIErr</td>
<td>COI is out of range.</td>
</tr>
<tr>
<td>ippStsOutOfRangeErr</td>
<td>Argument is out of range or point is outside the image.</td>
</tr>
<tr>
<td>ippStsDivisorErr</td>
<td>Divisor is equal to zero, function is aborted.</td>
</tr>
<tr>
<td>ippStsAlphaTypeErr</td>
<td>Illegal type of image composition operation.</td>
</tr>
<tr>
<td>ippStsGammaRangeErr</td>
<td>Gamma range bound is less than, or equal to zero.</td>
</tr>
<tr>
<td>ippStsGrayCoefSumErr</td>
<td>Sum of the conversion coefficients must be less than, or equal to 1.</td>
</tr>
<tr>
<td>ippStsChannelErr</td>
<td>Illegal channel number.</td>
</tr>
<tr>
<td>ippStsJaehneErr</td>
<td>Magnitude value is negative.</td>
</tr>
<tr>
<td>ippStsStepErr</td>
<td>Step value is less than, or equal to zero.</td>
</tr>
<tr>
<td>ippStsStrideErr</td>
<td>Stride value is less than the row length.</td>
</tr>
<tr>
<td>ippStsEpsValErr</td>
<td>Negative epsilon value.</td>
</tr>
<tr>
<td>ippStsScaleRangeErr</td>
<td>Scale bounds are out of range.</td>
</tr>
<tr>
<td>ippStsThresholdErr</td>
<td>Invalid threshold bounds.</td>
</tr>
<tr>
<td>ippStsWtOffsetErr</td>
<td>Invalid offset value for the wavelet filter.</td>
</tr>
<tr>
<td>ippStsAnchorErr</td>
<td>Anchor point is outside the mask.</td>
</tr>
<tr>
<td>ippStsMaskSizeErr</td>
<td>Invalid mask size.</td>
</tr>
<tr>
<td>ippStsShiftErr</td>
<td>Shift value is less than zero.</td>
</tr>
<tr>
<td>ippStsSampleFactorErr</td>
<td>Sampling factor is less than, or equal to zero.</td>
</tr>
<tr>
<td>ippStsResizeFactorErr</td>
<td>Resize factor(s) is less than, or equal to zero.</td>
</tr>
<tr>
<td>Status Code</td>
<td>Message</td>
</tr>
<tr>
<td>--------------</td>
<td>-------------------------------------------------------------------------</td>
</tr>
<tr>
<td>ippStsDivByZeroErr</td>
<td>An attempt to divide by zero.</td>
</tr>
<tr>
<td>ippStsInterpolationErr</td>
<td>Invalid interpolation mode.</td>
</tr>
<tr>
<td>ippStsMirrorFlipErr</td>
<td>Invalid flip mode.</td>
</tr>
<tr>
<td>ippStsMoment00ZeroErr</td>
<td>Moment value M(0,0) is too small to continue calculations.</td>
</tr>
<tr>
<td>ippStsThresholdLevelErr</td>
<td>Negative value of the level in the threshold operation.</td>
</tr>
<tr>
<td>ippStsContextMatchErr</td>
<td>Context parameter does not match the operation.</td>
</tr>
<tr>
<td>ippStsFFTFlagErr</td>
<td>Invalid value of the FFT flag parameter.</td>
</tr>
<tr>
<td>ippStsFFTOrderErr</td>
<td>Invalid value of the FFT order parameter.</td>
</tr>
<tr>
<td>ippStsMemAllocErr</td>
<td>Not enough memory for the operation.</td>
</tr>
<tr>
<td>ippStsNullPtrErr</td>
<td>Pointer is NULL.</td>
</tr>
<tr>
<td>ippStsSizeErr</td>
<td>Wrong value for the data size.</td>
</tr>
<tr>
<td>ippStsBadArgErr</td>
<td>Invalid or bad argument.</td>
</tr>
<tr>
<td>ippStsNoErr</td>
<td>No errors.</td>
</tr>
<tr>
<td>ippStsNoOperation</td>
<td>No operation has been executed.</td>
</tr>
<tr>
<td>ippStsMisalignedBuf</td>
<td>Misaligned pointer in operation in which it must be aligned.</td>
</tr>
<tr>
<td>ippStsSqrtNegArg</td>
<td>Negative value(s) of the argument in the function Sqrt.</td>
</tr>
<tr>
<td>ippStsInvZero</td>
<td>INF result. Zero value was met by InvThresh with zero level.</td>
</tr>
<tr>
<td>ippStsEvenMedianMaskSize</td>
<td>Even size of the Median Filter mask was replaced by the odd number.</td>
</tr>
<tr>
<td>ippStsDivByZero</td>
<td>Zero value(s) of the divisor in the function Div.</td>
</tr>
<tr>
<td>ippStsLnZeroArg</td>
<td>Zero value(s) of the argument in the function Ln.</td>
</tr>
<tr>
<td>ippStsLnNegArg</td>
<td>Negative value(s) of the argument in the function Ln.</td>
</tr>
<tr>
<td>ippStsNanArg</td>
<td>Argument value is not a number.</td>
</tr>
<tr>
<td>ippStsDoubleSize</td>
<td>Sizes of image are not multiples of 2.</td>
</tr>
<tr>
<td>ippStsJPEGMarker</td>
<td>JPEG marker in the bitstream.</td>
</tr>
<tr>
<td>ippStsResFloor</td>
<td>All result values are floored.</td>
</tr>
<tr>
<td>ippStsWrongQuadChanged</td>
<td>The fourth vertex of destination quad is not equal to the customer's</td>
</tr>
<tr>
<td>ippStsWrongIntersectROI</td>
<td>Incorrect ROI that has no intersection with the source or destination</td>
</tr>
<tr>
<td>ippStsWrongIntersectQuad</td>
<td>Incorrect quadrangle that has no intersection with the source or</td>
</tr>
<tr>
<td>ippStsSymKernelExpected</td>
<td>The kernel is not symmetric.</td>
</tr>
<tr>
<td>ippStsEvenMedianWeight</td>
<td>Even weight of the Weighted Median Filter was replaced by the odd</td>
</tr>
<tr>
<td>ippStsNoAntialiasing</td>
<td>The mode does not support antialiasing.</td>
</tr>
<tr>
<td>ippStsAlgTypeErr</td>
<td>The algorithm type is not supported.</td>
</tr>
<tr>
<td>ippStsAccurateModeNotSupported</td>
<td>Accurate mode is not supported.</td>
</tr>
</tbody>
</table>

The status codes ending with **Err** (except for the ippStsNoErr status) indicate an error; the integer values of these codes are negative. When an error occurs, the function execution is interrupted.

The status code ippStsNoErr indicates no error. All other status codes indicate warnings. When a specific case is encountered, the function execution is completed and the corresponding warning status is returned. For example, if the function ippiDiv meets an attempt to divide a positive value by zero, the function execution is not aborted. The result of the operation is set to the maximum value that can be represented by the source data type, and the user is warned by the output status ippStsDivByZero. See appendix A Handling of Special Cases for more information.

Platform-Aware Functions for Image Processing

Intel® Integrated Performance Primitives (Intel® IPP) library provides so-called platform-aware functions. These functions use the special data type ippSizeL for object sizes. The ippSizeL data type represents memory-related quantities: it can be 32- or 64-bit wide depending on the target architecture.
While the rest of Intel IPP functions support only objects of 32-bit integer size, platform-aware functions can work with 64-bit object sizes if it is supported by the platform. The API of platform-aware functions is similar to the API of other Intel IPP functions and has only slight differences. You can distinguish platform-aware functions by the L suffix in the function name, for example, ippiAdd_8u_C1RSfs_L.

Currently, the following image processing functions have platform-aware APIs:

<table>
<thead>
<tr>
<th>Function Group</th>
<th>Header</th>
<th>Function Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arithmetic Functions</td>
<td>ippi.h/ippi_l.h</td>
<td>Add, AddC, Sub, SubC, Mul, MulC, Div</td>
</tr>
<tr>
<td>Linear Filters</td>
<td>ippi.h/ippi_l.h</td>
<td>FilterBilateralBorderGetBufferSize, FilterBilateralBorder</td>
</tr>
<tr>
<td>Resize Transform Functions</td>
<td>ippi.h/ippi_l.h</td>
<td>ResizeGetSize, ResizeGetBufferSize, ResizeGetBorderSize, Resize{Nearest</td>
</tr>
<tr>
<td>Support Functions</td>
<td>ippi.h/ippi_l.h</td>
<td>Malloc</td>
</tr>
</tbody>
</table>

Intel IPP platform-aware functions are documented as additional flavors to the existing functions declared in standard Intel IPP headers (without the l suffix). The ippi_l.h header is included into ippi.h.

### Threading Layer Functions

Intel® Integrated Performance Primitives (Intel® IPP) library provides threading layer (TL) functions for image processing. It is a set of functions that are implemented as wrappers over Intel IPP platform-aware functions by using tiling and multithreading with OpenMP*. For implementation details, please see the corresponding source code files.

The API of TL functions is similar to the API of other Intel IPP functions and has only slight differences. You can distinguish Intel IPP TL functions by the _LT or _T suffix in the function name, for example, ippiAdd_8u_C1RSfs_LT. See Extensions for more information about function naming.

For more information about the TL functions usage, refer to the Intel IPP Developer Guide.

### Optimization Notice

Intel's compilers may or may not optimize to the same degree for non-Intel microprocessors for optimizations that are not unique to Intel microprocessors. These optimizations include SSE2, SSE3, and SSSE3 instruction sets and other optimizations. Intel does not guarantee the availability, functionality, or effectiveness of any optimization on microprocessors not manufactured by Intel. Microprocessor-dependent optimizations in this product are intended for use with Intel microprocessors. Certain optimizations not specific to Intel microarchitecture are reserved for Intel microprocessors. Please refer to the applicable product User and Reference Guides for more information regarding the specific instruction sets covered by this notice.

Notice revision #20110804

Currently, the following image processing functions have TL APIs:
<table>
<thead>
<tr>
<th>Function Group</th>
<th>Header</th>
<th>Function Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arithmetic Functions</td>
<td>ippi_tl.h</td>
<td>Add</td>
</tr>
<tr>
<td></td>
<td></td>
<td>AddC</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Sub</td>
</tr>
<tr>
<td></td>
<td></td>
<td>SubC</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Mul</td>
</tr>
<tr>
<td></td>
<td></td>
<td>MulC</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Div</td>
</tr>
<tr>
<td></td>
<td></td>
<td>DivC</td>
</tr>
<tr>
<td>Filters</td>
<td>ippi_tl.h</td>
<td>FilterBilateralBorderGetBufferSize</td>
</tr>
<tr>
<td></td>
<td></td>
<td>FilterBilateralBorderInit</td>
</tr>
<tr>
<td></td>
<td></td>
<td>FilterBilateral</td>
</tr>
<tr>
<td></td>
<td></td>
<td>FilterBilateralGetBufferSize</td>
</tr>
<tr>
<td></td>
<td></td>
<td>FilterBilateralInit</td>
</tr>
<tr>
<td></td>
<td></td>
<td>FilterBilateral</td>
</tr>
<tr>
<td></td>
<td></td>
<td>FilterBorderGetSize</td>
</tr>
<tr>
<td></td>
<td></td>
<td>FilterBorderInit</td>
</tr>
<tr>
<td></td>
<td></td>
<td>FilterBorder</td>
</tr>
<tr>
<td></td>
<td></td>
<td>FilterMedianGetSize</td>
</tr>
<tr>
<td></td>
<td></td>
<td>FilterMedianInit</td>
</tr>
<tr>
<td></td>
<td></td>
<td>FilterMedian</td>
</tr>
<tr>
<td>Resize Functions</td>
<td>ippi_tl.h</td>
<td>ResizeGetSize</td>
</tr>
<tr>
<td></td>
<td></td>
<td>ResizeGetBufferSize</td>
</tr>
<tr>
<td></td>
<td></td>
<td>ResizeGetBorderSize</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Resize{Nearest</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Resize{Nearest</td>
</tr>
<tr>
<td></td>
<td></td>
<td>ResizeAntialiasing{Linear</td>
</tr>
<tr>
<td></td>
<td></td>
<td>ResizeAntialiasing</td>
</tr>
<tr>
<td>Color Conversion Functions</td>
<td>ippcc_tl.h</td>
<td>RGBToLab</td>
</tr>
<tr>
<td></td>
<td></td>
<td>LabToRGB</td>
</tr>
</tbody>
</table>

**Structures and Enumerators**

This section describes the structures and enumerators used by the Intel Integrated Performance Primitives for image processing.

The `IppStatus` constant enumerates the status code values returned by Intel IPP functions, indicating whether the operation was error-free or not.

See section `Error Reporting` in this chapter for more information on the set of valid status codes and corresponding error messages for image and video processing functions.

The structure `IppiPoint` for storing the geometric position of a point is defined as:

```c
typedef struct {
    int x;
    int y;
} IppiPoint;
```

where \( x, y \) denote the coordinates of the point.
The structure `IppPointPolar` for storing the geometric position of a point in polar coordinates is defined as

```c
typedef struct {
    Ipp32f rho;
    Ipp32f theta;
} IppPointPolar;
```

where `rho` - a radial coordinate (radial distance from the origin), `theta` - an angular coordinate (counterclockwise angle from the x-axis).

The structure `IppiSize` for storing the size of a rectangle is defined as

```c
typedef struct {
    int width;
    int height;
} IppiSize;
```

where `width` and `height` denote the dimensions of the rectangle in the x- and y-directions, respectively.

The structure `IppiRect` for storing the geometric position and size of a rectangle is defined as

```c
typedef struct {
    int x;
    int y;
    int width;
    int height;
} IppiRect;
```

where `x`, `y` denote the coordinates of the top left corner of the rectangle that has dimensions `width` in the x-direction by `height` in the y-direction.

The `ippiConnectedComp` structure used in Computer Vision functions defines the connected component as follows:

```c
typedef struct _IppiConnectedComp {
    Ipp64f area;
    Ipp64f value[3];
    IppiRect rect;
} IppiConnectedComp;
```

where `area` - area of the segmented component; `value[3]` - gray scale value of the segmented component; `rect` - bounding rectangle of the segmented component.

The `IppiMaskSize` enumeration defines the neighborhood area for some morphological and filtering functions:

```c
typedef enum {
    ippMskSize1x3 = 13,
    ippMskSize1x5 = 15,
    ippMskSize3x1 = 31,
    ippMskSize3x3 = 33,
    ippMskSize5x1 = 51,
    ippMskSize5x5 = 55
} IppiMaskSize;
```
The **IppCmpOp** enumeration defines the type of compare operation to be used in image comparison functions:

```c
typedef enum {
    ippCmpLess,
    ippCmpLessEq,
    ippCmpEq,
    ippCmpGreaterEq,
    ippCmpGreater
} IppCmpOp;
```

The **IppRoundMode** enumeration defines the rounding mode to be used in some conversion, filtering and arithmetic functions:

```c
typedef enum {
    ippRndZero,
    ippRndNear,
    ippRndFinancial,
    ippRndHintAccurate=0x10
} IppRoundMode;
```

The **IppHintAlgorithm** enumeration defines the type of code to be used in some image transform and statistics functions, that is, faster but less accurate, or vice-versa, more accurate but slower. For more information on using this enumerator, see Table **Hint Arguments for Image Moment Functions**.

```c
typedef enum {
    ippAlgHintNone,
    ippAlgHintFast,
    ippAlgHintAccurate
} IppHintAlgorithm;
```

The types of interpolation used by geometric transform functions are defined as follows:

```c
enum {
    IPPI_INTER_NN        = 1,
    IPPI_INTER_LINEAR    = 2,
    IPPI_INTER_CUBIC     = 4,
    IPPI_INTER_CUBIC2P_BSPLINE,
    IPPI_INTER_CUBIC2P_CATMULLROM,
    IPPI_INTER_CUBIC2P_B05C03,
    IPPI_INTER_SUPER     = 8,
    IPPI_INTER_LANCZOS   = 16,
    IPPI_ANTIALIASING    = (1 << 29),
    IPPI_SUBPIXEL_EDGE   = (1 << 30),
    IPPI_SMOOTH_EDGE     = IPP_MIN_32S
};
```

The **IppAlphaType** enumeration defines the type of the compositing operation to be used in the alpha composition functions:

```c
typedef enum {
    ippAlphaOver,
    ippAlphaIn,
    ippAlphaOut,
    ippAlphaATop,
    ippAlphaXor,
    ippAlphaPlus,
};
```
The **IppiDitherType** enumeration defines the type of dithering to be used by the **ippiReduceBits** function:

```c
typedef enum {
    ippDitherNone,
    ippDitherFS,
    ippDitherJJN,
    ippDitherStucki,
    ippDitherBayer
} IppiDitherType;
```

The layout of the image slices used in some **image format conversion functions** is defined as follows:

```c
enum {
    IPP_UPPER        = 1,
    IPP_LEFT         = 2,
    IPP_CENTER       = 4,
    IPP_RIGHT        = 8,
    IPP_LOWER        = 16,
    IPP_UPPER_LEFT   = 32,
    IPP_UPPER_RIGHT  = 64,
    IPP_LOWER_LEFT   = 128,
    IPP_LOWER_RIGHT  = 256
};
```

The **IppiAxis** enumeration defines the flip axes for the **ippiMirror** functions or direction of the image intensity ramp for the **ippiImageRamp** functions:

```c
typedef enum {
    ippAxsHorizontal,
    ippAxsVertical,
    ippAxsBoth,
    ippAxs45,
    ippAxs135
} IppiAxis;
```

The **IppiBorderType** enumeration defines the border type that is used by some **Separable Filters** and **Fixed Filters** functions:

```c
typedef enum _IppiBorderType {
    ippBorderRepl      =  1,
    ippBorderWrap      =  2,
    ippBorderMirror    =  3,
    ippBorderMirrorR   =  4,
    ippBorderDefault   =  5,
    ippBorderConst     =  6,
    ippBorderTransp    =  7,
};
```
The `IppiFraction` enumeration defines shapes of the structuring element used in some decimate filter functions:

```c
typedef enum {
    ippPolyphase_1_2,
    ippPolyphase_3_5,
    ippPolyphase_2_3,
    ippPolyphase_7_10,
    ippPolyphase_3_4,
} IppiFraction;
```

The `IppiNormOp` enumeration defines the type of normalization that should be applied to the output data:

```c
typedef enum {
    ippiNormNone           = 0x00000000, // default
    ippiNorm          = 0x00000100, // normalized form
    ippiNormCoefficient = 0x00000200, // correlation coefficient in the range [-1.0,..,1.0]
    ippiNormMask          = 0x0000FF00,
} IppiNormOp;
```

The `IppiROIShape` enumeration defines the window shape for the two-dimensional convolution-specific functions:

```c
typedef enum {
    ippiROIFull   = 0x00000000,
    ippiROIValid  = 0x00010000,
    ippiROISame   = 0x00020000,
    ippiROIMask   = 0x00FF0000
} IppiROIShape;
```

The `IppNormType` enumeration defines the norm type that should be applied when computing the magnitude of the gradient:

```c
typedef enum {
    ippNormInf  =   0x00000001, // Infinity norm
    ippNormL1   =   0x00000002, // L1 normalization
    ippNormL2   =   0x00000004  // L2 normalization
} IppNormType;
```

The `IppiHOGConfig` structure defines the configuration parameters for the HOG descriptor:

```c
typedef struct {  
    int   cvCompatible;  /* openCV compatible output format */
    int   cellSize;      /* square cell size (pixels) */
    int   blockSize;     /* square block size (pixels) */
    int   blockStride;   /* block displacement (the same for x- and y- directions) */
    int   nbins;         /* required number of bins */
    Ipp32f  sigma;      /* gaussian factor of HOG block weights */
} IppiHOGConfig;
```
Ipp32f l2thresh; /* normalization factor */
IppiSize winSize; /* detection window size (pixels) */
} IppiHOGConfig;

The code flags used by the FastN functions are defined as follows:

enum {
    IPP_FASTN_ORIENTATION = 0x0001,
    IPP_FASTN_NMS         = 0x0002,
    IPP_FASTN_CIRCLE      = 0x0004,
    IPP_FASTN_SCORE_MODE0 = 0x0020
};

The IppiFastNSpec specification structure is used by the FastN function:

struct FastNSpec;
typedef struct FastNSpec IppiFastNSpec;

The IppiCornerFastN structure used by the FastN2DToVec function stores the destination vector of structures:

typedef struct _IppiCornerFastN {
    int     x;
    int     y;
    int     cornerType;
    int     orientation;
    float   angle;
    float   score;
} IppiCornerFastN;

The IppFGMModel structure contains parameters for the Gaussian mixture-based segmentation algorithm:

typedef struct {
    unsigned int   numFrames; /* length of history */
    unsigned int   numGauss;  /* maximal number of gaussian components per pixel */
        /* (numGauss<=maxNumGauss) */
    Ipp32f varInit;     /* initial value of variance for new gaussian component */
    Ipp32f varMin;      /* minimal bound of variance */
    Ipp32f varMax;      /* maximal bound of variance */
    Ipp32f varWBRatio;  /* background threshold */
    Ipp32f bckgThr;     /* background total weights sum threshold */
    Ipp32f varNGRatio;  /* threshold for adding new gaussian component to list */
    Ipp32f reduction;   /* speed of reduction non-active gaussian components */
    Ipp8u  shadowValue; /* returned shadow value */
    char   shadowFlag;  /* search shadows flag */
    Ipp32f shadowRatio; /* shadow threshold */
} IppFGMModel;

The IppiMorphMode enumerator defines modes for mask processing at the second stage of advanced morphology operations:

typedef enum {
    IPP_MORPH_DEFAULT      = 0x0000,
    IPP_MORPH_MASK_NO_FLIP = 0x0001,
} IppiMorphMode;

The IppChannels enumerator defines the number of channels in the image:

typedef enum {
    ippC0   = 0,
    ippC1   = 1,
    ippC2   = 2,
}
The `IppiFilterBilateralType` enumerator defines the type of the bilateral filter that is used by some Filtering Functions:

```c
typedef enum {
    ippFilterBilateralGauss     = 100,
    ippFilterBilateralGaussFast = 101
} IppiFilterBilateralType
```

The `IppiWarpTransformType` enumerator defines the type of the warp transform for some Warp Functions with Prior Initialization:

```c
typedef enum {
    ippWarpAffine,
    ippWarpPerspective,
    ippWarpBilinear
} IppiWarpTransformType
```

The `IppiDistanceMethodType` structure stores the method of defining the difference in intensity between pixels. It is defined as:

```c
typedef enum {
    ippDistNormL1 = 0x00000002;
    ippDistNormL2 = 0x00000004;
} IppiDistanceMethodType;
```

**Structures for 3D Data Processing Functions**

The `ipprBorderType` enumeration defines the border type that is used by some 3D Data Processing Functions:

```c
typedef enum _IpprBorderType {
    ipprBorderRepl        = ippBorderRepl,
    ipprBorderConst       = ippBorderConst,

    /* Flags to use source image memory pixels from outside of the border in particular directions */
    ipprBorderInMemTop     = 0x0001,
    ipprBorderInMemBottom  = 0x0002,
    ipprBorderInMemLeft    = 0x0004,
    ipprBorderInMemRight   = 0x0008,
    ipprBorderInMemFront   = 0x1000,
    ipprBorderInMemBack    = 0x2000,

    ipprBorderInMem       = ipprBorderInMemLeft|ipprBorderInMemTop|ipprBorderInMemRight|
                         ipprBorderInMemBottom|ipprBorderInMemFront|ipprBorderInMemBack,
} IpprBorderType;
```
The \texttt{IpprVolume} structure stores the volume of a three-dimensional space. It is defined as:

\begin{verbatim}
typedef struct {
    int width;
    int height;
    int depth;
} IpprVolume;
\end{verbatim}

The \texttt{IpprCuboid} structure stores the volume of interest of a three-dimensional space. It is defined as:

\begin{verbatim}
typedef struct {
    int x;
    int y;
    int z;
    int width;
    int height;
    int depth;
} IpprCuboid;
\end{verbatim}

\section*{Function Context Structures}

Some Intel IPP functions use special structures to store function-specific (context) information. For example, the \texttt{IppiFFTSpec} structure stores twiddle factors and bit reverse indexes needed in computing the fast Fourier transform.

Two different kinds of structures are used: structures that are not modified during function operation - they have the suffix \texttt{Spec} in their names, and structures that are modified during operation - they have the suffix \texttt{State} in their names.

These context-related structures are not defined in the public headers, and their fields are not accessible. It was done because the function context interpretation is processor dependent. Thus, you may only use context-related functions and may not create a function context as an automatic variable.

\section*{Structures and Enumerators for Platform-Aware Functions}

This topic describes the structures and enumerators used by the Intel IPP platform-aware functions for image processing.

The \texttt{IppiPointL} structure stores the geometric position of a point. It is defined as:

\begin{verbatim}
typedef struct {
    IppSizeL x;
    IppSizeL y;
} IppiPointL;
\end{verbatim}

where \texttt{x}, \texttt{y} denote the coordinates of the point.

The \texttt{IppiSizeL} structure stores the size of a rectangle. It is defined as:

\begin{verbatim}
typedef struct {
    IppSizeL width;
    IppSizeL height;
} IppiSizeL;
\end{verbatim}

where \texttt{width} and \texttt{height} denote the dimensions of the rectangle in the \texttt{x}– and \texttt{y}–directions, respectively.
The `IppiRectL` structure stores the geometric position and size of a rectangle. It is defined as:

```c
typedef struct {
    IppSizeL x;
    IppSizeL y;
    IppSizeL width;
    IppSizeL height;
} IppiRectL;
```

where \( x, y \) denote the coordinates of the top left corner of the rectangle that has dimensions \( \text{width} \) in the \( x \)-direction by \( \text{height} \) in the \( y \)-direction.

The `IpprVolumeL` structure stores the volume of a three-dimensional space. It is defined as:

```c
typedef struct {
    IppSizeL width;
    IppSizeL height;
    IppSizeL depth;
} IpprVolumeL;
```

where \( \text{width}, \text{height}, \text{depth} \) denote the dimensions of the three-dimensional space.

The `IpprBorderType` enumerator defines the border type that is used by some 3D Data Processing Functions:

```c
typedef enum _IpprBorderType {
    ipprBorderRepl        = ippBorderRepl,
    ipprBorderConst       = ippBorderConst,
    ipprBorderInMemTop     =  0x0010,
    ipprBorderInMemBottom  =  0x0020,
    ipprBorderInMemLeft    =  0x0040,
    ipprBorderInMemRight   =  0x0080,
    ipprBorderInMemFront   =  0x1000,
    ipprBorderInMemBack    =  0x2000
} IpprBorderType;
```

Image Data Types and Ranges

The Intel IPP image processing functions support only absolute color images in which each pixel is represented by its channel intensities. The data storage for an image can be either pixel-oriented or plane-oriented (planar). For images in pixel order, all channel values for each pixel are clustered and stored consecutively, for example, RRGBRGBRGB in case of an RGB image. The number of channels in a pixel-order image can be 1, 2, 3, or 4.

For images in planar order, all image data for each channel is stored contiguously followed by the next channel, for example, RRR...GGG...BBB.

Functions that operate on planar images are identified by the presence of \( P_n \) descriptor in their names. In this case, \( n \) pointers (one for each plane) may be specified.

The image data type is determined by the pixel depth in bits per channel, or bit depth. Bit depth for each channel can be 8, 16 or 32 and is included in the function name as one of these numbers (see Function Naming in this chapter for details). The data may be signed (\( s \)), unsigned (\( u \)), or floating-point real (\( f \)). For some arithmetic and FFT/DFT functions, data in complex format (\( sc \) or \( fc \)) can be used, where each channel value is represented by two numbers: real and imaginary part. All channels in an image must have the same data type.
For example, in an absolute color 24-bit RGB image, three consecutive bytes (24 bits) per pixel represent the three channel intensities in the pixel mode. This data type is identified in function names as the 8u_C3 descriptor, where 8u represents 8-bit unsigned data for each channel and C3 represents three channels.

Some functions operate with images in 16-bit packed RGB format (see RGB Image Formats in Chapter 6 for more details). In this case data of all 3 channels are represented as 16u data type.

For example, in an absolute color 16-bit packed RGB image, two consecutive bytes (16 bits) per pixel represent the three channel intensities in the pixel mode. This data type is identified in function names as the 16u_C3 descriptor, where 16u represents 16-bit unsigned data (not a bit depth) for all packed channels together and C3 stands for three channels.

If an alpha (opacity) channel is present in image data, the image must have four channels, with alpha channel being the last or the first one. This data type is indicated by the AC4 or A0C4 descriptors respectively. The presence of the alpha channel can modify the function’s behavior. For such functions, Intel IPP provides versions with and without alpha. If an alpha channel is specified, the operation usually is not performed on that channel.

The range of values that can be represented by each data type lies between the lower and upper bounds. The following table lists data ranges and constant identifiers used in Intel IPP to denote the respective range bounds:

<table>
<thead>
<tr>
<th>Data Type</th>
<th>Lower Bound</th>
<th>Upper Bound</th>
</tr>
</thead>
<tbody>
<tr>
<td>8s</td>
<td>IPP_MIN_8S</td>
<td>IPP_MAX_8S</td>
</tr>
<tr>
<td>8u</td>
<td>0</td>
<td>IPP_MAX_8U</td>
</tr>
<tr>
<td>16s</td>
<td>IPP_MIN_16S</td>
<td>IPP_MAX_16S</td>
</tr>
<tr>
<td>16u</td>
<td>0</td>
<td>IPP_MAX_16U</td>
</tr>
<tr>
<td>32s</td>
<td>IPP_MIN_32S</td>
<td>IPP_MAX_32S</td>
</tr>
<tr>
<td>32u</td>
<td>0</td>
<td>IPP_MAX_32U</td>
</tr>
</tbody>
</table>
| 32f†      | IPP_MINABS_32F | IPP_MAXABS_32F |}

† The range for absolute values

Major Operation Models

Most Intel IPP image processing functions perform identical and independent operations on all channels of the processed image except for alpha channel. It means that the same operation is applied to each channel, and the computed results do not depend on values of other channels. Some exceptions include theippiFilterMedianColor function and color conversion functions, which process three channels together.

Intel IPP image processing functions can be divided by two major models of operation:

- **Point operations**: functions operate on one pixel to compute the result, for example,ippiAdd
- **Neighborhood operations**: functions operate on a group of pixels, for example,ippiFilterBox

See Also

FilterMedianColor Filters an image using a color median filter.
Add Adds pixel values of two images.
FilterBox Blurs an image using a simple box filter.
**Neighborhood Operations**

The result of a neighborhood operation is based on values of a certain group of pixels located near a given input pixel. The set of neighboring pixels is typically defined by a rectangular mask (or kernel) and anchor cell, specifying the mask alignment with respect to the position of the input pixel as shown in the following figure.

The anchor cell is a fixed cell within the kernel, which is used for positioning the kernel with respect to the currently processed pixel of the source image. The kernel is placed on the image in such a way that the anchor cell coincides with the input pixel. The anchor cell is specified by its coordinates $anchor.x$ and $anchor.y$ in the coordinate system associated with the top left corner of the kernel.

If position of the anchor cell is not specified explicitly in the function description, coordinates of the anchor are computed by the default formula:

$$anchor.x = (kernel.width - 1)/2$$
$$anchor.y = (kernel.height - 1)/2$$

where $kernel.width$ and $kernel.height$ is the width and height of the filter kernel, respectively.

**Regions of Interest in Intel IPP**

Most Intel IPP image processing functions operate not only on entire images but also on image areas. Image region of interest (ROI) is a rectangular area that may be either some part of the image or the whole image.

The Intel IPP functions with ROI support have an $R$ descriptor in their names. ROI of an image is defined by the size and offset from the image origin as shown in figure. The origin of an image is in the top left corner, with $x$ values increasing from left to right and $y$ values increasing downwards.
Both the source and destination images can have a ROI. In such cases the sizes of ROIs are assumed to be the same while offsets may differ. Image processing is performed on data of the source ROI, and the results are written to the destination ROI. In function call sequences, ROI is specified by:

- `roiSize` parameter of the `IppiSize` type
- `pSrc` and `pDst` pointers to the starts of source and destination ROI buffers
- `srcStep` and `dstStep` parameters that are equal to distances in bytes between the starting points of consecutive lines in source and destination images, respectively.

Thus, the `srcStep` and `dstStep` parameters set steps in bytes through image buffers to start processing a new line in the ROI of an image.

The following code example illustrates the use of the `dstStep` parameter in function calls:

```c
Example

IppStatus roi( void ) {
    Ipp8u x[8*3] = {0};
    IppiSize roiSize = {3,2};
    IppiPoint roiPoint = {2,1};
    /// place the pointer to the ROI start position
    return ippiSet_8u_C1R( 7, x+8*roiPoint.y+roiPoint.x, 8, roiSize );
}
```

The resulting image `x` contains the following data:

```
00 00 00 00 00 00 00 00
00 00 07 07 07 00 00 00
00 00 07 07 07 00 00 00
```

If ROI is present:

- source and destination images can have different sizes;
- lines may have padding at the end for aligning the line sizes;
- application must correctly define the `pSrc`, `pDst`, and `roiSize` parameters.

The `pSrc` and `pDst` parameters are the shifted pointers to the image data. For example, in case of ROI operations on 3-bytes-per-pixel image data (`8u_C3R`), `pSrc` points to the start of the source ROI buffer and can be interpreted as follows:

```
pSrc = pSrcImg + 3*(srcImgSize.width * srcRoiOffset.y + srcRoiOffset.x),
```

where
Another example for operations on four-channel image of 32-bit floating point data type `32f_AC4`:

```
pSrc = (Ipp32f*)((Ipp8u*)pSrcImg + srcImgStep * srcRoiOffset.y + 4*SizeOf(Ipp32f)*srcRoiOffset.x.
```

In this example the multiplier 4 is used because the AC4 pixel consists of 4 values - R, G, B, A. Pointer type conversion is required as in Intel IPP all image steps are always in bytes, and it is not recommended to use step/SizeOf(Ipp32f) as in a general case step value may be not a multiple of 4.

For functions using ROI with a neighborhood, you should correctly use values of the `pSrc` and `roiSize` parameters. These functions assume that the points in the neighborhood exist and that therefore `pSrc` is almost never equal to `pSrcImg`. Figure Using ROI with Neighborhood illustrates the case when neighborhood pixels can fall outside the source image.

To ensure valid operation when image pixels are processed, the application should correctly define additional border pixels (see Borders in Neighborhood Operations).

**Warning**

If the required border pixels are not defined prior to calling neighborhood functions that attempt to process such pixels, you may get memory violation errors.

The following code example shows how to process an image with ROI:

**Example**

```c
IppStatus alignedLine( void )
{
    Ipp8u x[8*3] = {0};
```
The image is of size 5x3. Width 8 has been chosen by the user to align every line of the image. The resulting image $x$ contains the following data:

```
07 07 07 07 07 00 00 00
07 07 07 07 07 00 00 00
07 07 07 07 07 00 00 00
```

See Also
Borders in Neighborhood Operations

**Tiled Image Processing**

Intel IPP can process images composed from tiles, or tiled images.
Support Functions

This chapter describes the Intel® IPP support functions that are used to:

- retrieve information about the current Intel IPP software version
- get a brief explanation of the returned status codes
- allocate and free memory that is needed for the operation of other Intel IPP image and video processing functions
- execute Intel IPP Threading Layer service routines.

Version Information Function

This function returns the version number and other information about the active Intel IPP image processing software.

GetLibVersion

*Returns information about the used version of Intel IPP software for image processing.*

**Syntax**

```
const IppLibraryVersion* ippiGetLibVersion(void);
```

**Include Files**

ippi.h

**Domain Dependencies**

**Headers:** ippcore.h, ippvm.h, ipps.h

**Libraries:** ippcore.lib, ippvm.lib, ipps.lib

**Description**

This function returns a pointer to a static data structure `IppLibraryVersion` that contains information about the current version of Intel IPP for image processing. You need not release memory referenced by the returned pointer, as it points to a static variable. The following fields of the `IppLibraryVersion` structure are available:

- **major**
  - the major number of the current library version
- **minor**
  - the minor number of the current library version
- **majorBuild**
  - the number of builds of the major version
- **build**
  - current build number
- **targetCpu[4]**
  - Intel® processor.
- **Name**
  - the name of the current library version
- **Version**
  - the library version string
BuildDate: the library version actual build date

For example, if the library version is 9.0, build revision number is 49671, library name is ippIP AVX2, target CPU is processor with Intel® Advanced Vector Extensions 2 (Intel® AVX2) and build date is “Dec 7 2015”, then the fields in this structure are set as:

\[
\text{major} = 9, \quad \text{minor} = 0, \quad \text{Name} = \text{"ippIP AVX2"}, \quad \text{Version} = \text{"9.0.1 (r49671)"}, \quad \text{targetCpu[4]} = \text{"h9"}, \quad \text{BuildDate} = \text{"Dec 7 2015"}
\]

NOTE
Each sub-library that is used in the image processing domain has its own similar function to retrieve information about the active library version. These functions are: ippiGetLibVersion, ippcvGetLibVersion, and ippccGetLibVersion. They are declared in the following header files: ippcore.h, ippcv.h, ippcc.h, respectively, and have the same interface as the above described function.

**Status Information Function**

Use this function to get a brief description of the status code returned by the current Intel IPP software.

**ippiGetStatusString**

*Translates a status code into a message.*

**Syntax**

```c
const char* ippGetStatusString(IppStatus StsCode);
```

**Include Files**

ippcore.h

**Parameters**

*StsCode*  
Code that indicates the status type (see Table 2-1)

**Description**

This function returns a pointer to the text string associated with a status code *StsCode*. Use this function to produce error and warning messages. The returned pointer is a pointer to an internal static buffer and needs not be released.

The status information function translates this code into the corresponding message *Null Pointer Error*:
Example

A code example below shows how to use the ippGetStatusString function. If you call an Intel IPP function, in this example ippiSet_8u_C1R, with a NULL pointer, it returns an error code -8.

```c
void statusInfo( void ) {
    IppSize roi = {0};
    IppStatus st = ippiSet_8u_C1R(3, 0, 0, roi);
    printf( " %d : %s\n", st, ippGetStatusString( st ));
}
```

Output:

-8, Null Pointer Error

Memory Allocation Functions

This section describes the Intel IPP functions that allocate aligned memory blocks for data of required type, or free previously allocated memory.

**NOTE**
The only function to free the memory allocated by any of these functions is ippiFree().

Malloc

*Allocates memory aligned to 64-byte boundary.*

**Syntax**

**Case 1:** Memory allocation for blocks of 32-bit sizes

```c
Ipp<datatype>* ippiMalloc_<mod>(int widthPixels, int heightPixels, int* pStepBytes);
```

Supported values for `mod`:

- 8u_C1, 16u_C1, 16s_C1, 32s_C1, 32f_C1, 32sc_C1, 32fc_C1
- 8u_C2, 16u_C2, 16s_C2, 32s_C2, 32f_C2, 32sc_C2, 32fc_C2
- 8u_C3, 16u_C3, 16s_C3, 32s_C3, 32f_C3, 32sc_C3, 32fc_C3
- 8u_C4, 16u_C4, 16s_C4, 32s_C4, 32f_C4, 32sc_C4, 32fc_C4
- 8u_AC4, 16u_AC4, 16s_AC4, 32s_AC4, 32f_AC4, 32sc_AC4, 32fc_AC4

**Case 2:** Memory allocation for platform-aware functions

```c
Ipp<datatype>* ippiMalloc_<mod>(IppSizeL widthPixels, IppSizeL heightPixels, IppSizeL* pStepBytes);
```

Supported values for `mod`:

- 8u_C1_L, 16u_C1_L, 16s_C1_L, 32s_C1_L, 32f_C1_L, 32sc_C1_L, 32fc_C1_L
- 8u_C2_L, 16u_C2_L, 16s_C2_L, 32s_C2_L, 32f_C2_L, 32sc_C2_L, 32fc_C2_L
- 8u_C3_L, 16u_C3_L, 16s_C3_L, 32s_C3_L, 32f_C3_L, 32sc_C3_L, 32fc_C3_L
- 8u_C4_L, 16u_C4_L, 16s_C4_L, 32s_C4_L, 32f_C4_L, 32sc_C4_L, 32fc_C4_L
Include Files
ippi.h

Flavors with the _L suffix: ippi_l.h

Domain Dependencies
Headers: ippcore.h, ippvm.h, ipps.h
Libraries: ippcore.lib, ippvm.lib, ipps.lib

Parameters

- **widthPixels**: Width of an image, in pixels.
- **heightPixels**: Height of an image, in pixels.
- **pStepBytes**: Pointer to the distance, in bytes, between the starting points of consecutive lines in the image.

Description

This function allocates a memory block aligned to 64-byte boundary for elements of different data types. Every line of the image is aligned in accordance with the **pStepBytes** parameter, which is calculated by the Malloc function and returned for further use.

The function Malloc allocates one continuous memory block. Functions that operate on planar images require an array of separate pointers (**IppType* plane[3]**) to each plane as an input. In this case, you should call the Malloc function three times.

Example

The code example below demonstrates how to construct an array and set correct values to the pointers to use the allocated memory block with the Intel IPP functions operating on planar images. You need to specify **pStepBytes** for each plane. The example is given for the **8u** data type.

```c
int stepBytes[3];
Ipp8u* plane[3];
plane[0] =ippiMalloc_8u_C1(widthPixels, heightPixels, & (stepBytes [0]));
plane[1] =ippiMalloc_8u_C1(widthPixels/2, heightPixels/2, & (stepBytes [1]));
plane[2] =ippiMalloc_8u_C1(widthPixels/2, heightPixels/2, & (stepBytes [2]));
```

Return Values

The return value of Malloc function is a pointer to an aligned memory block.

If no memory is available in the system, the NULL value is returned.

To free the allocated memory block, use the Free function.

Free

*Frees memory allocated by the function* ippiMalloc.
Syntax

```c
void ippiFree(void* ptr);
```

Include Files

```c
ippi.h
```

Domain Dependencies

Headers: ippcore.h, ippvm.h, ipps.h

Libraries: ippcore.lib, ippvm.lib, ipps.lib

Parameters

`ptr`  
Pointer to a memory block to be freed. This block must have been previously allocated by the function `ippiMalloc`.

Description

This function frees the aligned memory block allocated by the function `ippiMalloc`.

**NOTE**
The function `ippiFree` cannot be used to free memory allocated by standard functions like `malloc` or `calloc`, nor can the memory allocated by `ippiMalloc` be freed by `free`.

Threading Layer Functions

This section described the Intel IPP Threading Layer (TL) functions for image processing. For more information about the TL functions usage, refer to the Intel IPP Developer Guide.

SplitUniform2D

Splits an image into tiles.

Syntax

**Case 1: Operation with TL functions based on the Platform Aware API**

```c
IppStatus ippiSplitUniform2D_LT(IppiSizeL roiSize, IppSizeL minItemNumber, IppiPointL* pSplit, IppiSizeL* pTileSize, IppiSizeL* pTailSize);
```

**Case 2: Operation with TL functions based on the Classic API**

```c
IppStatus ippiSplitUniform2D_T(IppiSize roiSize, int minItemNumber, IppiPoint* pSplit, IppiSize* pTileSize, IppiSize* pTailSize);
```

Include Files

```c
ippi_tl.h
```

Parameters

`roiSize`  
Image ROI size.

`minItemNumber`  
Minimal size of a tile, in pixels.
pSplit
Number of split parts along x and y axes.

pTileSize
Size of a tile.

pTailSize
Size of the last corner tile.

Description
This function operates with ROI (see Regions of Interest in Intel IPP).
This function splits an image into tiles.

Return Values
ippStsNoErr
Indicates no error.

ippStsNullPtrErr
Indicates an error condition when the pSplit, pTileSize, or pTailSize pointer is NULL.

ippStsSizeErr
Indicates an error condition when minItemNumber is less than zero.

ParallelFor
*Performs parallel iterations for a processing function.*

Syntax
Case 1: Operation with TL functions based on the Platform Aware API
IppStatus ippParallelFor_LT(ippSizeL numTiles, void* arg, functype_l func);

Case 2: Operation with TL functions based on the Classic API
IppStatus ippParallelFor_T(int numTiles, void* arg, functype func);

Include Files
ippcore_tl.h

Parameters
numTiles
Number of tiles.

arg
Pointer to the structure that contains arguments for the processing function.

func
Pointer to the processing function used in the "parallel for" loop.

Description
This function performs parallel iterations of a processing function, which is passed as an argument for each tile.

Return Values
ippStsNoErr
Indicates no error.

Any IPP error that the processing function can return.

GetTilePointer
*Returns a pointer to the specified image tile.*
Syntax

Case 1: Operation with TL functions based on the Platform Aware API

```c
IppStatus ippiGetTilePointer_32f_LT(const Ipp32f* pSrc, Ipp32f** pDst, IppSizeL srcStep, IppSizeL x, IppSizeL y, Ipp32s numChannels);
IppStatus ippiGetTilePointer_64f_LT(const Ipp64f* pSrc, Ipp64f** pDst, IppSizeL srcStep, IppSizeL x, IppSizeL y, Ipp32s numChannels);
```

Case 2: Operation with TL functions based on the Classic API

```c
IppStatus ippiGetTilePointer_32f_T(const Ipp32f* pSrc, Ipp32f** pDst, int srcStep, int x, int y, Ipp32s numChannels);
IppStatus ippiGetTilePointer_64f_T(const Ipp64f* pSrc, Ipp64f** pDst, int srcStep, int x, int y, Ipp32s numChannels);
```

Include Files

`ippi_tl.h`

Parameters

- `pSrc` : Pointer to the source image.
- `pDst` : Pointer to the memory location of the pointer to the destination image.
- `srcStep` : Distance in bytes between consecutive lines in the source image.
- `x` : x coordinate of a tile, in pixels.
- `y` : y coordinate of a tile, in pixels.
- `numChannels` : Number of channels in the image. Possible values are 1, 3, and 4.

Description

This function returns a pointer to the specified image tile.

Return Values

- `ippStsNoErr` : Indicates no error.
- `ippStsNullPtrErr` : Indicates an error condition when the `pSrc` or `pDst` pointer is NULL.
- `ippStsSizeErr` : Indicates an error condition when `x` or `y` is less than zero.
- `ippStsNumChannelsErr` : Indicates an error condition when `numChannels` has an illegal value.

GetTileParamsByIndex

*Returns the offset and size of a tile by a given index.*

Syntax

Case 1: Operation with TL functions based on the Platform Aware API

```c
IppStatus ippiGetTileParamsByIndex_LT(IppSizeL index, IppiPointL splitImage, IppiSizeL tileSize, IppiSizeL tailSize, IppiPointL* pTileOffset, IppiSizeL* pTileSize);
```
Case 2: Operation with TL functions based on the Classic API

IppStatus ippiGetSizeByIndex_T(int index, IppiPoint splitImage, IppiSize tileSize, IppiSize tailSize, IppiPoint* pTileOffset, IppiSize* pTileSize);

Include Files
ippi_tl.h

Parameters

- **index**: Ordinal index of a tile.
- **splitImage**: Split of the image by x and y axis correspondingly.
- **tileSize**: Size of a tile.
- **tailSize**: Size of the last bottom right tile.
- **pTileOffset**: Offset of the tile corresponding to the top left image corner.
- **pTileSize**: Size of a tile.

Description
This function returns the offset and size of a tile by a given index.

Return Values

- **ippStsNoErr**: Indicates no error.
- **ippStsNullPtrErr**: Indicates an error condition when the pTileOffset or pTileSize pointer is NULL.

GetThreadingType
Returns type of the threading layer.

Syntax

Case 1: Operation with TL functions based on the Platform Aware API
IppStatus ippGetThreadingType_LT(ippThreadingType* thrType);

Case 2: Operation with TL functions based on the Classic API
IppStatus ippGetThreadingType_T(ippThreadingType* thrType);

Include Files
ippicore_tl.h

Parameters

- **thrType**: Pointer to the threading type.

Description
This function returns OMP if the OpenMP* Threading Layer and TBB if the TBB* Threading Layer is used.

Return Values

- **ippStsNoErr**: Indicates no error.
Indicates an error condition when the \textit{thrType} pointer is NULL.

\textbf{GetThreadIdx}

\textit{Returns a unique thread identification number.}

\textbf{Syntax}

\textbf{Case 1: Operation with TL functions based on the Platform Aware API}

\begin{verbatim}
IppStatus ippGetThreadIdx_LT(int* pThrIdx);
\end{verbatim}

\textbf{Case 2: Operation with TL functions based on the Classic API}

\begin{verbatim}
IppStatus ippGetThreadIdx_T(int* pThrIdx);
\end{verbatim}

\textbf{Include Files}

ippcore_tl.h

\textbf{Parameters}

\begin{itemize}
  \item \textit{pThrIdx} \hspace{1cm} Pointer to the index of a thread.
\end{itemize}

\textbf{Description}

This function returns a unique thread identification number.

\textbf{Return Values}

\begin{itemize}
  \item \texttt{ippStsNoErr} \hspace{1cm} Indicates no error.
  \item \texttt{ippStsNullPtrErr} \hspace{1cm} Indicates an error condition when the \textit{pThrIdx} pointer is NULL.
\end{itemize}
Image Data Exchange and Initialization Functions

This chapter describes the Intel® IPP image processing functions that perform image data manipulation, exchange and initialization operations.

Convert

Converts image pixel values from one data type to another.

Syntax

Case 1: Conversion to increase bit depth and change signed to unsigned type

IppStatus ippiConvert_<mod>(const Ipp<srcDatatype>* pSrc, int srcStep, Ipp<dstDatatype>* pDst, int dstStep, IppiSize roiSize);

Supported values for mod:

- 8u16u_C1R
- 8u16u_C3R
- 8u16u_C4R
- 8s32f_C1R
- 8s32f_C3R
- 8s32f_C4R
- 8s8u_C1R
- 8s8u_C3R
- 8s8u_C4R
- 8s16u_C1R
- 8s16u_C3R
- 8s16u_C4R

IppStatus ippiConvert_<mod>(const Ipp<srcDatatype>* pSrc, int srcStep, Ipp<dstDatatype>* pDst, int dstStep, IppiSize roiSize);

Supported values for mod:

- 8u16u_AC4R
- 8u16s_AC4R
- 8u32s_AC4R
- 8u32f_AC4R
- 8s16u_AC4R
- 16u32s_AC4R
- 16u32f_AC4R
- 16s32s_AC4R
- 16s32f_AC4R

Case 2: Conversion to reduce bit depth and change unsigned to signed type: integer to integer type

IppStatus ippiConvert_<mod>(const Ipp<srcDatatype>* pSrc, int srcStep, Ipp<dstDatatype>* pDst, int dstStep, IppiSize roiSize);

Supported values for mod:

- 16u8u_C1R
- 16u8u_C3R
- 16u8u_C4R
- 32s8u_C1R
- 32s8u_C3R
- 32s8u_C4R
- 32s8s_C1R
- 32s8s_C3R
- 32s8s_C4R

IppStatus ippiConvert_<mod>(const Ipp<srcDatatype>* pSrc, int srcStep, Ipp<dstDatatype>* pDst, int dstStep, IppiSize roiSize);

Supported values for mod:

- 16u8u_AC4R
- 16u8s_AC4R
- 16u32s_AC4R
- 16u32f_AC4R
- 16s32s_AC4R
- 16s32f_AC4R
- 32s8u_AC4R
- 32s8s_AC4R
- 32s8s_AC4R

Intel® Integrated Performance Primitives Developer Reference, Volume 2: Image Processing
IppStatusippiConvert_<mod>(const Ipp<srcDatatype>*pSrc, int srcStep, Ipp<dstDatatype>*pDst, int dstStep, IppiSize roiSize, IppRoundMode roundMode, int scaleFactor);

Supported values for mod:

```
8u8s_C1RSfs  16u8s_C1RSfs  32u8s_C1RSfs  32s16u_C1RSfs
16u8s_C1RSfs  32u8s_C1RSfs  32s16u_C1RSfs
16u16s_C1RSfs  32u16u_C1RSfs
16u16s_C1RSfs  32u16u_C1RSfs
32u16s_C1RSfs  32u32s_C1RSfs
```

IppStatusippiConvert_<mod>(const Ipp<srcDatatype>*pSrc, int srcStep, Ipp<dstDatatype>*pDst, int dstStep, IppiSize roiSize);

Supported values for mod:

```
16u8u_AC4R  16s8u_AC4R  32s8u_AC4R  32s8s_AC4R
```

**Floating point to integer type:**

IppStatusippiConvert_<mod>(const Ipp32f*pSrc, int srcStep, Ipp<dstDatatype>*pDst, int dstStep, IppiSize roiSize, IppRoundMode roundMode);

Supported values for mod:

```
32f8u_C1R  32f8s_C1R  32f16u_C1R  32f16s_C1R
32f8u_C3R  32f8s_C3R  32f16u_C3R  32f16s_C3R
32f8u_C4R  32f8s_C4R  32f16u_C4R  32f16s_C4R
```

IppStatusippiConvert_<mod>(const Ipp<srcDatatype>*pSrc, int srcStep, Ipp<dstDatatype>*pDst, int dstStep, IppiSize roiSize, IppRoundMode roundMode, int scaleFactor);

Supported values for mod:

```
32f8u_C1RSfs  32f8s_C1RSfs  32f16u_C1RSfs  32f16s_C1RSfs  32f32u_C1RSfs
64f8u_C1RSfs  64f8s_C1RSfs  64f16u_C1RSfs  64f16s_C1RSfs
```
IppStatus ippiConvert_32f32u_C1IRSfs(Ipp32u* pSrcDst, int srcDstStep, IippiSize roiSize, IppRoundMode roundMode, int scaleFactor);
IppStatus ippiConvert_<mod>(const Ipp32f* pSrc, int srcStep, Ipp<dstDatatype>* pDst, int dstStep, IippiSize roiSize, IppRoundMode roundMode);

Supported values for `mod`:

- 32f8u_AC4R
- 32f8s_AC4R
- 32f16u_AC4R
- 32f16s_AC4R

**Include Files**

ippi.h

**Domain Dependencies**

Headers: ippcore.h, ippvm.h, ipps.h

Libraries: ippcore.lib, ippvm.lib, ipps.lib

**Parameters**

- **pSrc**
  Pointer to the source image ROI.
- **srcStep**
  Distance, in bytes, between the starting points of consecutive lines in the source image.
- **pDst**
  Pointer to the destination image ROI.
- **pSrcDst**
  Pointer to the source and destination image ROI for in-place operation.
- **dstStep**
  Distance, in bytes, between the starting points of consecutive lines in the destination image.
- **srcDstStep**
  Distance, in bytes, between the starting points of consecutive lines in the source and destination image for in-place operation.
- **roiSize**
  Size of the source and destination ROI in pixels.
- **scaleFactor**
  Scale factor (see Integer Result Scaling).
- **roundMode**
  Rounding mode, the following values are possible:
  - **ippRndZero**
    specifies that floating-point values are truncated to zero,
  - **ippRndNear**
    specifies that floating-point values are rounded to the nearest even integer when the fractional part equals 0.5; otherwise they are rounded to the nearest integer,
  - **ippRndFinancial**
    specifies that floating-point values are rounded down to the nearest integer when the fractional part is less than 0.5, or rounded up to the nearest integer if the fractional part is equal or greater than 0.5.

**Description**

This function operates with ROI.

This function converts pixel values in the source image ROI `pSrc` to a different data type and writes them to the destination image ROI `pDst`. 

The result of integer operations is always saturated to the destination data type range. It means that if the value of the source pixel is out of the data range of the destination image, the value of the corresponding destination pixel is set to the value of the lower or upper bound (minimum or maximum) of the destination data range:

\[ x = p_{Src}[i,j] \]

- If \( x > \text{MAX\_VAL} \) then \( x = \text{MAX\_VAL} \)
- If \( x < \text{MIN\_VAL} \) then \( x = \text{MIN\_VAL} \)

\[ p_{Dst}[i,j] = \text{(CASTING)} x \]

If you want to shift data from the signed range to the unsigned range and vice-versa, see "Application Notes" below.

The function flavors with the \( Sfs \) descriptor in their names perform scaling of the internally computed results in accordance with the \( \text{scaleFactor} \) parameter.

When converting from floating-point to integer type, rounding defined by \( \text{roundMode} \) is performed, and the result is saturated to the destination data type range.

NOTE
The bit order of each byte in the source image is inverse to the pixel order. It means that the first pixel in a row represents the last (seventh) bit of the first byte in a row.

Application Notes
When data is converted from the signed integer to the corresponding unsigned integer and vice versa (8s -> 8u, 16u --> 16s), the pixel information may be lost because all negative values will be set to zero (signed-unsigned conversion), or unsigned values from the high half of the range will be set to the maximum value of the signed range (unsigned - signed conversion).

If you need just to shift the data from the signed range to the unsigned range and vice versa, use the function \( \text{ippiXorC} \) with the parameter \( \text{value} \) specified in such a way that the most significant bit is set to 1, and all other bits are set to 0. For example, if you want to convert pixel values from \( \text{Ipps16s} \) type to \( \text{Ipp16u} \) type with the range shift call the function:

\[ \text{ippiXorC}_\text{16u}_\text{C1R}( \text{Ipp16u *} p_{Src}, \text{srcStep}, 0x8000, p_{Dst}, \text{dstStep}, \text{roiSize}); \]

In this case the pixels values are converted as follows:

- \(-32768 \rightarrow 0\)
- \(-32767 \rightarrow 1\)
- \(\ldots\)
- \(-1 \rightarrow 32767\)
- \(0 \rightarrow 32768\)
- \(1 \rightarrow 32769\)
- \(\ldots\)
- \(32766 \rightarrow 65534\)
- \(32767 \rightarrow 65535\)

Return Values
\( \text{ippStsNoErr} \) Indicates no error. Any other value indicates an error or a warning.
ippStsNullPtrErr  
Indicates an error when any of the specified pointers is NULL, with the exception of second mode in Case 4.

ippStsSizeErr  
Indicates an error when roiSize has a field with zero or negative value, or srcBitOffset/dstBitOffset is less than zero.

ippStsStepErr  
Indicates an error when srcStep or dstStep has a zero or negative value.

ippStsMemAllocErr  
Indicates an error when memory allocation fails.

Example

The code example below shows data conversion without scaling.

```c
IppStatus convert( void ) {
    IppiSize roi={5,4};
    Ipp32f x[5*4];
    Ipp8u y[5*4];
   ippiSet_32f_C1R( -1.0f, x, 5*sizeof(Ipp32f), roi );
    x[1] = 300; x[2] = 150;
    return ippiConvert_32f8u_C1R( x, 5*sizeof(Ipp32f), y, 5, roi, ippRndNear );
}
```

The destination image \( y \) contains:

```
00 FF 96 00 00
00 00 00 00 00
00 00 00 00 00
00 00 00 00 00
```

See Also

Integer Result Scaling
Regions of Interest in Intel IPP

BinToGray, GrayToBin

Converts a bitonal image to a grayscale image and vice versa.

Syntax

Case 1: Conversion of a bitonal image to a grayscale image

```c
IppStatus ippiBinToGray_<dstDataType>_C1R(const Ipp8u* pSrc, int srcStep, int srcBitOffset, Ipp<dstDataType>* pDst, int dstStep, IppiSize roiSize, Ipp<dstDataType> loVal, Ipp<dstDataType> hiVal);
```

Supported values for \( \text{dstDataType} \):

- 8u
- 16u
- 16s
- 32f
Case 2: Conversion of a grayscale image to a bitonal image

IppStatus ippiGrayToBin_<srcDataType>1u_C1R(const Ipp<srcDataType>* pSrc, int srcStep, Ipp8u* pDst, int dstStep, int dstBitOffset, IppiSize roiSize, Ipp<srcDataType>* threshold);

Supported values for srcDataType:
8u 16u 16s 32f

Include Files
ippi.h

Domain Dependencies
Headers: ippcore.h, ippvm.h, ipps.h
Libraries: ippcore.lib, ippvm.lib, ipps.lib

Parameters
pSrc
Pointer to the source image ROI.

srcStep
Distance, in bytes, between the starting points of consecutive lines in the source image.

srcBitOffset
Offset, in bits, from the first byte of the source image row.

pDst
Pointer to the destination image ROI.

dstStep
Distance, in bytes, between the starting points of consecutive lines in the destination image.

dstBitOffset
Offset, in bits, from the first byte of the destination image row.

roiSize
Size of the ROI in pixels.

loVal
Destination value that corresponds to the "0" value of the corresponding source element.

hiVal
Destination value that corresponds to the "1" value of the corresponding source element.

threshold
Threshold level.

Description
These functions operate with ROI.

The ippiBinToGray function converts a bitonal image to grayscale, and the ippiGrayToBin function converts a grayscale image to bitonal. The data type of the bitonal image is 8u. It means that each byte consists of eight consecutive pixels of the image (1 bit per pixel). You need to specify the start position of the ROI buffer in the srcBitOffset and dstBitOffset parameters.

The ippiBinToGray function transforms each bit of the source image into the pixel of the destination image in the following way:
• If the input pixel is equal to 0, the corresponding output pixel is set to loVal.
• If the input pixel is equal to 1, the corresponding output pixel is set to hiVal.
The `ippiGrayToBin` function transforms each pixel of the source image into the bit of the destination image in the following way:

- If the input pixel is more than the `threshold` value, the corresponding output bit is set to 1.
- If the input pixel is less than, or equal to the `threshold` value, the corresponding output bit is set to 0.

### Return Values

- **ippStsNoErr**
  - Indicates no error.

- **ippStsNullPtrErr**
  - Indicates an error when any of the specified pointers is `NULL`.

- **ippStsStepErr**
  - Indicates an error when the `srcStep` or `dstStep` value is less than, or equal to zero.

- **ippStsSizeErr**
  - Indicates an error when:
    - `roiSize` has a zero or negative value
    - the `srcBitOffset` or `dstBitOffset` value is less than zero

### See Also

Regions of Interest in Intel IPP

### Scale

**Scales pixel values of an image and converts them to another bit depth.**

#### Syntax

**Case 1: Scaling with conversion to integer data of increased bit depth**

```c
IppStatus ippiScale_<mod>(const Ipp<srcDatatype>* pSrc, int srcStep, Ipp<dstDatatype>* pDst, int dstStep, IppiSize roiSize);
```

Supported values for `mod`:

- `8u16u_C1R`
- `8u16u_C3R`
- `8u16u_C4R`
- `8u16s_C1R`
- `8u16s_C3R`
- `8u16s_C4R`
- `8u32s_C1R`
- `8u32s_C3R`
- `8u32s_C4R`

**Case 2: Scaling with conversion to floating-point data**

```c
IppStatus ippiScale_<mod>(const Ipp8u* pSrc, int srcStep, Ipp32f* pDst, int dstStep, IppiSize roiSize, Ipp32f vMin, Ipp32f vMax);
```

Supported values for `mod`:

- `8u32f_C1R`
- `8u32f_C3R`
- `8u32f_C4R`
IppStatusippiScale_8u32f_AC4R(constIpp8u*pSrc,intsrcStep,Ipp32f*pDst,int
dstStep, IppiSizeroiSize, Ipp32f vMin, Ipp32f vMax);

**Case 3: Scaling of integer data with conversion to reduced bit depth**

IppStatusippiScale_<mod>(constIpp<srcDatatype>*pSrc,intsrcStep, Ipp<dstDatatype>*
pDst, int dstStep, IppiSizeroiSize, IppHintAlgorithm hint);

Supported values for mod:

- 16u8u_C1R
- 16u8u_C3R
- 16u8u_C4R
- 16s8u_C1R
- 16s8u_C3R
- 16s8u_C4R
- 32s8u_C1R
- 32s8u_C3R
- 32s8u_C4R

IppStatusippiScale_<mod>(constIpp<srcDatatype>*pSrc,intsrcStep, Ipp<dstDatatype>*
pDst, int dstStep, IppiSizeroiSize, IppHintAlgorithm hint);

Supported values for mod:

- 16u8u_AC4R
- 16s8u_AC4R
- 32s8u_AC4R

**Case 4: Scaling of floating-point data with conversion to integer data type**

IppStatusippiScale_<mod>(constIpp32f*pSrc,intsrcStep,Ipp8u*pDst,int dstStep,
IppiSize roiSize, Ipp32f vMin, Ipp32f vMax);

Supported values for mod:

- 32f8u_C1R
- 32f8u_C3R
- 32f8u_C4R

IppStatusippiScale_32f8u_AC4R(constIpp32f*pSrc,intsrcStep, Ipp8u*pDst, int dstStep,
IppiSize roiSize, Ipp32f vMin, Ipp32f vMax);

**Include Files**

ippi.h

**Domain Dependencies**

Headers: ippcore.h, ippvm.h, ipps.h

Libraries: ippcore.lib, ippvm.lib, ipps.lib

**Parameters**

- **pSrc**
  Pointer to the source image ROI.

- **srcStep**
  Distance, in bytes, between the starting points of consecutive lines in the
  source image.

- **pDst**
  Pointer to the destination image ROI.

- **dstStep**
  Distance, in bytes, between the starting bytes of consecutive lines in the
  destination image.

- **roiSize**
  Size of the source and destination ROI in pixels.
Minimum and maximum values of the input data.

Option to select the algorithmic implementation of the function (see Hint Arguments for Image Moment Functions).

**Description**

This function operates with ROI.

This function converts pixel values of a source image ROI $p_{Src}$ to the destination data type, using a linear mapping. The computation algorithm is specified by the `hint` argument. For conversion between integer data types, the whole range $[src_{Min}..src_{Max}]$ of the input data type is mapped onto the range $[dst_{Min}..dst_{Max}]$ of the output data type.

The source pixel $p$ is mapped to the destination pixel $p'$ by the following formula:

$$p' = dst_{Min} + k*(p - src_{Min})$$

where

$$k = (dst_{Max} - dst_{Min})/(src_{Max} - src_{Min})$$

For conversions to and from floating-point data type, the user-defined floating-point data range $[v_{Min}..v_{Max}]$ is mapped onto the source or destination data type range.

If the conversion is from `Ipp32f` type and some of the input floating-point values are outside the specified input data range $[v_{Min}..v_{Max}]$, the corresponding output values saturate. To determine the actual floating-point data range in your image, use the `ippiMinMax` function.

**Return Values**

- `ippStsNoErr` Indicates no error. Any other value indicates an error or a warning.
- `ippStsNullPtrErr` Indicates an error when any of the specified pointers is `NULL`.
- `ippStsSizeErr` Indicates an error condition if `roiSize` has a field with a zero or negative value.
- `ippStsStepErr` Indicates an error condition if `srcStep` or `dstStep` has a zero or negative value.
- `ippStsScaleRangeErr` Indicates an error condition if the input data bounds are incorrect, that is, $v_{Max}$ is less than or equal to $v_{Min}$.

**Example**

The code example below shows how to use scaling to preserve the data range.

```c
IppStatus scale( void ) {
    IppiSize roi = {5,4};
    Ipp32f x[5*4];
    Ipp8u y[5*4];
    ippiSet_32f_C1R( -1.0f, x, 5*sizeof(Ipp32f), roi );
    x[1] = 300; x[2] = 150;
    return ippiScale_32f8u_C1R( x, 5*sizeof(Ipp32f), y, 5, roi, -1, 300 );
}
```
The destination image \( y \) contains:

\[
00 \text{ FF } 80 \text{ 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 }
\]

See Also
Regions of Interest in Intel IPP
Image Moments
Intel® Integrated Performance Primitives Concepts
MinMax Computes the minimum and maximum of image pixel values.

**ScaleC**

*Scales pixel values of an image and converts them to another bit depth.*

**Syntax**

**Case 1: Not-in-place operation**

IppStatus ippiScaleC_<mod>_C1R(const Ipp<srcDatatype>* pSrc, int srcStep, Ipp64f mVal, Ipp64f aVal, Ipp<dstDatatype>* pDst, int dstStep, IppiSize roiSize, IppHintAlgorithm hint);

Supported values for mod:

<table>
<thead>
<tr>
<th>srcDatatype</th>
<th>dstDatatype</th>
</tr>
</thead>
<tbody>
<tr>
<td>8u</td>
<td>8u</td>
</tr>
<tr>
<td>8s8u</td>
<td>8s</td>
</tr>
<tr>
<td>16u8u</td>
<td>16u</td>
</tr>
<tr>
<td>16s8s</td>
<td>16s</td>
</tr>
<tr>
<td>32s8u</td>
<td>32s</td>
</tr>
<tr>
<td>64f8u</td>
<td>64f</td>
</tr>
</tbody>
</table>

where the first value is srcDatatype and the second value is dstDatatype.

**Case 2: In-place operation**

IppStatus ippiScaleC_<mod>_C1IR(const Ipp<datatype>* pSrcDst, int srcDstStep, Ipp64f mVal, Ipp64f aVal, IppiSize roiSize, IppHintAlgorithm hint);

Supported values for mod:

<table>
<thead>
<tr>
<th>datatype</th>
</tr>
</thead>
<tbody>
<tr>
<td>8u</td>
</tr>
<tr>
<td>8s</td>
</tr>
<tr>
<td>16u</td>
</tr>
<tr>
<td>16s</td>
</tr>
<tr>
<td>32s</td>
</tr>
<tr>
<td>64f</td>
</tr>
</tbody>
</table>

**Include Files**

ippi.h

**Domain Dependencies**

Headers: ippcore.h, ippvm.h, ipps.h

Libraries: ippcore.lib, ippvm.lib, ipps.lib
Parameters

- **pSrc**
  Pointer to the source image ROI.

- **pSrcDst**
  Pointer to the source and destination buffer or an array of pointers to separate source and destination color planes for in-place operation.

- **srcDstStep**
  Distance, in bytes, between the starting points of consecutive lines in the source and destination image for in-place operation.

- **srcStep**
  Distance, in bytes, between the starting points of consecutive lines in the source image.

- **mVal**
  Value of the multiplier used for scaling.

- **aVal**
  Offset value for scaling.

- **pDst**
  Pointer to the destination image ROI.

- **dstStep**
  Distance, in bytes, between the starting bytes of consecutive lines in the destination image.

- **roiSize**
  Size of the source and destination ROI, in pixels.

- **hint**
  Option to select the algorithmic implementation of the function. Supported values are ippAlgHintFast (default) and ippAlgHintAccurate.

Description

This function operates with ROI (see Regions of Interest in Intel IPP).

This function scales pixel values of the source image ROI and converts them to the destination data type according to the following formula:

\[
\text{dst} = \text{saturate}\_\text{to}\_\text{dstType}(\text{src} \times \text{mVal} + \text{aVal})
\]

Return Values

- **ippStsNoErr**
  Indicates no error. Any other value indicates an error or a warning.

- **ippStsNullPtrErr**
  Indicates an error when any of the specified pointers is NULL.

- **ippStsSizeErr**
  Indicates an error when **roiSize** has a field with a zero or negative value.

- **ippStsStepErr**
  Indicates an error when the step value is less than, or equal to zero.

Example

To better understand usage of this function, refer to the following example in the examples archive available for download from https://software.intel.com/en-us/ipp-manual-examples:

**ScaleC.c**

See Also

Regions of Interest in Intel IPP

Set

Sets pixels of an array to a constant value.
**Syntax**

**Case 1: Setting one-channel data to a value**

IppStatus ippiSet_<mod>(Ipp<datatype> value, Ipp<datatype>* pDst, int dstStep, IppiSize roiSize);

Supported values for mod:

| 8u_C1R | 16u_C1R | 16s_C1R | 32s_C1R | 32f_C1R |

**Case 2: Setting each color channel to a specified value**

IppStatus ippiSet_<mod>(const Ipp<datatype> value[3], Ipp<datatype>* pDst, int dstStep, IppiSize roiSize);

Supported values for mod:

| 8u_C3R | 16u_C3R | 16s_C3R | 32s_C3R | 32f_C3R |
| 8u_AC4R | 16u_AC4R | 16s_AC4R | 32s_AC4R | 32f_AC4R |

**Case 3: Setting color channels and alpha channel to specified values**

IppStatus ippiSet_<mod>(const Ipp<datatype> value[4], Ipp<datatype>* pDst, int dstStep, IppiSize roiSize);

Supported values for mod:

| 8u_C4R | 16u_C4R | 16s_C4R | 32s_C4R | 32f_C4R |

**Case 4: Setting masked one-channel data to a value**

IppStatus ippiSet_<mod>(Ipp<datatype> value, Ipp<datatype>* pDst, int dstStep, IppiSize roiSize, const Ipp8u* pMask, int maskStep);

Supported values for mod:

| 8u_C1MR | 16u_C1MR | 16s_C1MR | 32s_C1MR | 32f_C1MR |

**Case 5: Setting color channels of masked multi-channel data to specified values**

IppStatus ippiSet_<mod>(const Ipp<datatype> value[3], Ipp<datatype>* pDst, int dstStep, IppiSize roiSize, const Ipp8u* pMask, int maskStep);

Supported values for mod:

| 8u_C3MR | 16u_C3MR | 16s_C3MR | 32s_C3MR | 32f_C3MR |
| 8u_AC4MR | 16u_AC4MR | 16s_AC4MR | 32s_AC4MR | 32f_AC4MR |

**Case 6: Setting all channels of masked multi-channel data to specified values**

IppStatus ippiSet_<mod>(const Ipp<datatype> value[4], Ipp<datatype>* pDst, int dstStep, IppiSize roiSize, const Ipp8u* pMask, int maskStep);

Supported values for mod:

| 8u_C4MR | 16u_C4MR | 16s_C4MR | 32s_C4MR | 32f_C4MR |
Case 7: Setting selected channel of multi-channel data to a value

IppStatus ippiSet_<mod>(Ipp<datatype> value, Ipp<datatype>* pDst, int dstStep, IppSize roiSize);

Supported values for mod:

8u_C3CR  16u_C3CR  16s_C3CR  32s_C3CR  32f_C3CR
8u_C4CR  16u_C4CR  16s_C4CR  32s_C4CR  32f_C4CR

Include Files

ippi.h

Domain Dependencies

Headers: ippcore.h, ippvm.h, ipps.h
Libraries: ippcore.lib, ippvm.lib, ipps.lib

Parameters

value

Constant value to assign to each pixel in the destination image ROI.

pDst

Pointer to the destination image ROI.

dstStep

Distance, in bytes, between the starting points of consecutive lines in the destination image.

roiSize

Size of the image ROI in pixels.

pMask

Pointer to the mask image buffer.

maskStep

Distance, in bytes, between the starting points of consecutive lines in the mask image buffer.

Description

This function operates with ROI.

This function sets pixels in the destination image ROI pDst to the value constant. Either all pixels in a rectangular ROI, or only those selected by the specified mask pMask, can be set to a value. In case of masked operation, the function sets pixel values in the destination buffer only if the spatially corresponding mask array value is non-zero. When a channel of interest is selected, that is only one channel of a multi-channel image must be set (see Case 7), the pDst pointer points to the start of ROI buffer in the required channel. If alpha channel is present in the source image data, the alpha components may be either skipped, or set to a value, depending on the chosen ippiSet function flavor.

This function supports negative step value.

Return Values

ippStsNoErr

Indicates no error. Any other value indicates an error or a warning.

ippStsNullPtrErr

Indicates an error when any of the specified pointers is NULL.

ippStsSizeErr

Indicates an error condition if roiSize has a field with zero or negative value.

ippStsStepErr

Indicates an error condition if dstStep or maskStep has a zero value.
Example

The code example below shows how to use the function ippiSet_8u_C1R.

```c
void func_set()
{
    IppiSize roi = {5,4};
    Ipp8u x[8*4] = {0};
    ippiSet_8u_C1R(1, x, 8, roi);
}
```

Result:

```
01 01 01 01 00 00 00
01 01 01 01 00 00 00
01 01 01 01 00 00 00
01 01 01 01 00 00 00
```

See Also
Regions of Interest in Intel IPP

Copy

Copies pixel values between two buffers.

Syntax

Case 1: Copying all pixels of all color channels

```c
IppStatus ippiCopy_<mod>(const Ipp<datatype>* pSrc, int srcStep, Ipp<datatype>* pDst, int dstStep, IppiSize roiSize);
```

Supported values for `mod`:

```
8u_C1R   16u_C1R   16s_C1R   32s_C1R   32f_C1R
8u_C3R   16u_C3R   16s_C3R   32s_C3R   32f_C3R
8u_C4R   16u_C4R   16s_C4R   32s_C4R   32f_C4R
8u_AC4R  16u_AC4R  16s_AC4R  32s_AC4R  32f_AC4R
8u_C3AC4R 16u_C3AC4R 16s_C3AC4R 32s_C3AC4R 32f_C3AC4R
8u_AC4C3R 16u_AC4C3R 16s_AC4C3R 32s_AC4C3R 32f_AC4C3R
```

Case 2: Copying masked pixels only

```c
IppStatus ippiCopy_<mod>(const Ipp<datatype>* pSrc, int srcStep, Ipp<datatype>* pDst, int dstStep, IppiSize roiSize, const Ipp8u* pMask, int maskStep);
```

Supported values for `mod`:

```
8u_C1MR  16u_C1MR  16s_C1MR  32s_C1MR  32f_C1MR
8u_C3MR  16u_C3MR  16s_C3MR  32s_C3MR  32f_C3MR
8u_C4MR  16u_C4MR  16s_C4MR  32s_C4MR  32f_C4MR
8u_AC4MR 16u_AC4MR 16s_AC4MR 32s_AC4MR 32f_AC4MR
```
Case 3: Copying a selected channel in a multi-channel image

IppStatusippiCopy_<mod>(const Ipp<datatype>* pSrc, int srcStep, Ipp<datatype>* pDst, int dstStep, IppiSize roiSize);

Supported values for mod:
- 8u_C3CR
- 16u_C3CR
- 16s_C3CR
- 32s_C3CR
- 32f_C3CR
- 8u_C4CR
- 16u_C4CR
- 16s_C4CR
- 32s_C4CR
- 32f_C4CR

Case 4: Copying a selected channel to a one-channel image

IppStatusippiCopy_<mod>(const Ipp<datatype>* pSrc, int srcStep, Ipp<datatype>* pDst, int dstStep, IppiSize roiSize);

Supported values for mod:
- 8u_C3C1R
- 16u_C3C1R
- 16s_C3C1R
- 32s_C3C1R
- 32f_C3C1R
- 8u_C4C1R
- 16u_C4C1R
- 16s_C4C1R
- 32s_C4C1R
- 32f_C4C1R

Case 5: Copying a one-channel image to a multi-channel image

IppStatusippiCopy_<mod>(const Ipp<datatype>* pSrc, int srcStep, Ipp<datatype>* pDst, int dstStep, IppiSize roiSize);

Supported values for mod:
- 8u_C1C3R
- 16u_C1C3R
- 16s_C1C3R
- 32s_C1C3R
- 32f_C1C3R
- 8u_C1C4R
- 16u_C1C4R
- 16s_C1C4R
- 32s_C1C4R
- 32f_C1C4R

Case 6: Splitting color image into separate planes

IppStatusippiCopy_<mod>(const Ipp<datatype>* pSrc, int srcStep, Ipp<datatype>* const pDst[3], int dstStep, IppiSize roiSize);

Supported values for mod:
- 8u_C3P3R
- 16u_C3P3R
- 16s_C3P3R
- 32s_C3P3R
- 32f_C3P3R

IppStatusippiCopy_<mod>(const Ipp<datatype>* pSrc, int srcStep, Ipp<datatype>* const pDst[4], int dstStep, IppiSize roiSize);

Supported values for mod:
- 8u_C4P4R
- 16u_C4P4R
- 16s_C4P4R
- 32s_C4P4R
- 32f_C4P4R

Case 7: Composing color image from separate planes

IppStatusippiCopy_<mod>(const Ipp<datatype>* const pSrc[3], int srcStep, Ipp<datatype>* pDst, int dstStep, IppiSize roiSize);

Supported values for mod:
- 8u_P3C3R
- 16u_P3C3R
- 16s_P3C3R
- 32s_P3C3R
- 32f_P3C3R
IppStatus ippiCopy_<mod>(const Ipp<datatype>* const pSrc[4], int srcStep,
Ipp<datatype>* pDst, int dstStep, IppiSize roiSize);

Supported values for mod:
8u_P4C4R  16u_P4C4R  16s_P4C4R  32s_P4C4R  32f_P4C4R

Case 8: Copying all pixels of all color channels with platform-aware functions
IppStatus ippiCopy_<mod>(const Ipp<datatype>* pSrc, IppSizeL srcStep, Ipp<datatype>*
pDst, IppSizeL dstStep, IppiSizeL roiSize);

Supported values for mod:
8u_C1R_L  16s_C1R_L  16u_C1R_L  32s_C1R_L  32f_C1R_L
8u_C3R_L  16s_C3R_L  16u_C3R_L  32s_C3R_L  32f_C3R_L
8u_C4R_L  16s_C4R_L  16u_C4R_L  32s_C4R_L  32f_C4R_L
8u_AC4R_L  16s_AC4R_L  16u_AC4R_L  32s_AC4R_L  32f_AC4R_L

Include Files
ippi.h

Flavors with the _L suffix: ippi_l.h

Domain Dependencies

Headers: ippcore.h, ippvm.h, ipps.h
Libraries: ippcore.lib, ippvm.lib, ipps.lib

Parameters

pSrc
Pointer to the source image ROI. The array storing pointers to the color planes of the source planar image.

srcStep
Distance, in bytes, between the starting points of consecutive lines in the source image.

pDst
Pointer to the destination image ROI. The array storing pointers to the color planes of the destination planar image.

dstStep
Distance, in bytes, between the starting points of consecutive lines in the destination image.

roiSize
Size of the source and destination ROI in pixels.

pMask
Pointer to the mask image buffer.

maskStep
Distance, in bytes, between the starting points of consecutive lines in the mask image buffer.

Description

This function operates with ROI.

This function copies data from the source image pSrc to the destination image pDst. Copying pixels selected by a mask pMask is supported as well.
For masked operation (Case 2), the function writes pixel values in the destination buffer only if the spatially corresponding mask array value is non-zero (as illustrated in the code example below).

Function flavors operating with the channel of interest (descriptor C) copy only one specified channel of a source multi-channel image to the channel of another multi-channel image (see Case 3). For these functions, the pSrc and pDst pointers point to the starts of ROI buffers in the specified channels of source and destination images, respectively.

Some function flavors add alpha channel to the 3-channel source image (flavors with the _C3AC4R descriptor), or discard alpha channel from the source image (flavors with the _AC4C3R descriptor) - see Case 1.

Special function flavors copy data from only one specified channel pSrc of a multi-channel image to a one-channel image pDst (see Case 4), as well as to copy data from a one-channel image pSrc to only one specified channel of a multi-channel image pDst (see Case 5).

You can also use the ippiCopy function to convert the interleaved color image into separate planes and vice versa (see Case 6 and Case 7).

**Return Values**

- ippStsNoErr Indicates no error. Any other value indicates an error or a warning.
- ippStsNullPtrErr Indicates an error when any of the specified pointers is NULL, with the exception of second mode in Case 4.
- ippStsSizeErr Indicates an error condition if roiSize has a field with a zero or negative value.
- ippStsStepErr Indicates an error condition if srcStep or dstStep is less than roiSize.width * <pixelSize> for Cases 4 and 5.

**Example**

The code example below shows how to copy masked data.

```c
IppStatus copyWithMask( void ) {
    Ipp8u mask[3*3], x[5*4], y[5*4]={0};
    IppiSize imgroi={5,4}, mskroi={3,3};
    ippiSet_8u_C1R( 3, x, 5, imgroi );
    /// set mask with a hole in upper left corner
    ippiSet_8u_C1R( 1, mask, 3, mskroi );
    mask[0] = 0;
    /// copy roi with mask
    return ippiCopy_8u_C1MR( x, 5, y, 5, mskroi, mask, 3 );
}
```

The destination image y contains:

```
00 03 03 00 00  
03 03 03 00 00  
03 03 03 00 00  
00 00 00 00 00  
```

**See Also**

Regions of Interest in Intel IPP
CopyManaged

Copies pixel values between two images in accordance with the specified type of copying.

Syntax

IppStatus ippiCopyManaged_8u_C1R(const Ipp8u* pSrc, int srcStep, Ipp8u* pDst, int dstStep, IppiSize roiSize, int flags);

Include Files

ippi.h

Domain Dependencies

Headers: ippcore.h, ippvm.h, ipps.h
Libraries: ippcore.lib, ippvm.lib, ipps.lib

Parameters

pSrc       Pointer to the source image ROI.
srcStep    Distance, in bytes, between the starting points of consecutive lines in the source image.
pDst       Pointer to the destination image ROI.
dstStep    Distance, in bytes, between the starting points of consecutive lines in the destination image.
roiSize    Size of the image ROI in pixels.
flags      Specifies the type of copying. Possible values are:
            • IPP_TEMPORAL_COPY - standard copying
            • IPP_NONTEMPORAL_STORE - copying without caching the destination image
            • IPP_NONTEMPORAL_LOAD - processor uses non-temporal load instructions

Description

This function operates with ROI.

This function copies data from a source image ROI pSrc to the destination image ROI pDst. The flags parameter specifies the type of copying that the function performs:

- When flags is set to IPP_TEMPORAL_COPY, the function is identical to the ippiCopy_8u_C1R function.
- When flags is set to IPP_NONTEMPORAL_STORE, the processor uses non-temporal store instructions. Copying is performed without caching the data of the destination image.
- When flags is set to IPP_NONTEMPORAL_LOAD, the processor uses non-temporal load instructions.

To achieve better performance, align data to 64-byte boundary.
Optimization Notice

Intel's compilers may or may not optimize to the same degree for non-Intel microprocessors for optimizations that are not unique to Intel microprocessors. These optimizations include SSE2, SSE3, and SSSE3 instruction sets and other optimizations. Intel does not guarantee the availability, functionality, or effectiveness of any optimization on microprocessors not manufactured by Intel. Microprocessor-dependent optimizations in this product are intended for use with Intel microprocessors. Certain optimizations not specific to Intel microarchitecture are reserved for Intel microprocessors. Please refer to the applicable product User and Reference Guides for more information regarding the specific instruction sets covered by this notice.

Notice revision #20110804

Return Values

ippStsNoErr
Indicates no error. Any other value indicates an error or a warning.

ippStsNullPtrErr
Indicates an error when one of the specified pointers is NULL.

ippStsSizeErr
Indicates an error when roiSize has a field with a zero or negative value.

See Also
Regions of Interest in Intel IPP

CopyConstBorder

Copies pixels values between two images and adds the border pixels with a constant value.

Syntax

Case 1: Operation on one-channel data

IppStatus ippiCopyConstBorder_<mod>(const Ipp<datatype>* pSrc, int srcStep, IppSize srcRoiSize, Ipp<datatype>* pDst, int dstStep, IppSize dstRoiSize, int topBorderHeight, int leftBorderWidth, Ipp<datatype> value);

Supported values for mod:

- 8u_C1R
- 16u_C1R
- 16s_C1R
- 32s_C1R
- 32f_C1R

Case 2: Operation on multi-channel data

IppStatus ippiCopyConstBorder_<mod>(const Ipp<datatype>* pSrc, int srcStep, IppSize srcRoiSize, Ipp<datatype>* pDst, int dstStep, IppSize dstRoiSize, int topBorderHeight, int leftBorderWidth, const Ipp<datatype> value[3]);

Supported values for mod:

- 8u_C3R
- 16u_C3R
- 16s_C3R
- 32s_C3R
- 32f_C3R
- 8u_AC4R
- 16u_AC4R
- 16s_AC4R
- 32s_AC4R
- 32f_AC4R
IppStatus ippiCopyConstBorder_<mod>(const Ipp<datatype>* pSrc, int srcStep, IppiSize srcRoiSize, Ipp<datatype>* pDst, int dstStep, IppiSize dstRoiSize, int topBorderHeight, int leftBorderWidth, const Ipp<datatype> value[4]);

Supported values for <mod>:

<p>| | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>8u_C4R</td>
<td>16u_C4R</td>
<td>16s_C4R</td>
<td>32s_C4R</td>
<td>32f_C4R</td>
</tr>
</tbody>
</table>

**Case 3: Operation on one-channel data with platform-aware functions**


Supported values for <mod>:

<p>| | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>8u_C1R_L</td>
<td>16u_C1R_L</td>
<td>16s_C1R_L</td>
<td>32s_C1R_L</td>
<td>32f_C1R_L</td>
</tr>
</tbody>
</table>

IppStatus ippiCopyConstBorder_<mod>(Ipp<datatype>* pSrcDst, IppSizeL srcDstStep, IppSizeL srcRoiSize, Ipp<datatype> value);

Supported values for <mod>:

<p>| | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>8u_C1IR_L</td>
<td>16u_C1IR_L</td>
<td>16s_C1IR_L</td>
<td>32s_C1IR_L</td>
<td>32f_C1IR_L</td>
</tr>
</tbody>
</table>

**Case 4: Operation on multi-channel data with platform-aware functions**


Supported values for <mod>:

<p>| | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>8u_C3R_L</td>
<td>16u_C3R_L</td>
<td>16s_C3R_L</td>
<td>32s_C3R_L</td>
<td>32f_C3R_L</td>
</tr>
<tr>
<td>8u_AC4R_L</td>
<td>16u_AC4R_L</td>
<td>16s_AC4R_L</td>
<td>32s_AC4R_L</td>
<td>32f_AC4R_L</td>
</tr>
</tbody>
</table>


Supported values for <mod>:

<p>| | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>8u_C4R_L</td>
<td>16u_C4R_L</td>
<td>16s_C4R_L</td>
<td>32s_C4R_L</td>
<td>32f_C4R_L</td>
</tr>
</tbody>
</table>


Supported values for <mod>:

<p>| | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>8u_C3IR_L</td>
<td>16u_C3IR_L</td>
<td>16s_C3IR_L</td>
<td>32s_C3IR_L</td>
<td>32f_C3IR_L</td>
</tr>
<tr>
<td>8u_AC4IR_L</td>
<td>16u_AC4IR_L</td>
<td>16s_AC4IR_L</td>
<td>32s_AC4IR_L</td>
<td>32f_AC4IR_L</td>
</tr>
</tbody>
</table>

Supported values for <mod>:
8u_C4IR_L 16u_C4IR_L 16s_C4IR_L 32s_C4IR_L 32f_C4IR_L

Include Files
ippi.h

Flavors with the _L suffix: ippi_l.h

Domain Dependencies
Headers: ippcore.h, ippvm.h, ipps.h
Libraries: ippcore.lib, ippvm.lib, ipps.lib

Parameters
pSrc  Pointer to the source image ROI.
srcStep  Distance in bytes between starts of consecutive lines in the source image.
pNext  Pointer to the destination image.
pSrcDst  Pointer to the source/destination image (for in-place flavors).
dstStep  Distance, in bytes, between the starting points of consecutive lines in the destination image.
srcDstStep  Distance, in bytes, between the starting points of consecutive lines in the source/destination image (for in-place flavors).
srcRoiSize  Size of the source ROI, in pixels.
dstRoiSize  Size of the destination ROI, in pixels.
topBorderHeight  Height of the top border, in pixels.
leftBorderWidth  Width of the left border, in pixels.
value  The constant value to assign to the border pixels (constant vector in case of multi-channel images).

Description
This function operates with ROI (see Regions of Interest in Intel IPP).
This function copies the source image \( p_{Src} \) to the destination image \( p_{Dst} \) and creates border outside the copied area; pixel values of the border are set to the specified constant value that is passed by the \texttt{value} argument. The parameters \texttt{topBorderHeight} and \texttt{leftBorderWidth} specify the position of the first pixel of the source ROI in the destination image ROI (see Figure Creating a Border of Pixels with Constant Value.) Squares marked in red correspond to pixels copied from the source image, that is, the source image ROI.

Creating a Border of Pixels with Constant Value

\[
\begin{array}{cccccccc}
\text{v} & \text{v} & \text{v} & \text{v} & \text{v} & \text{v} & \text{v} & \text{v} \\
\text{v} & \text{v} & \text{v} & \text{v} & \text{v} & \text{v} & \text{v} & \text{v} \\
\text{v} & \text{v} & \text{1} & \text{2} & \text{3} & \text{4} & \text{5} & \text{v} \\
\text{v} & \text{v} & \text{6} & \text{7} & \text{8} & \text{9} & \text{10} & \text{v} \\
\text{v} & \text{v} & \text{11} & \text{12} & \text{13} & \text{14} & \text{15} & \text{v} \\
\text{v} & \text{v} & \text{16} & \text{17} & \text{18} & \text{19} & \text{20} & \text{v} \\
\text{v} & \text{v} & \text{v} & \text{v} & \text{v} & \text{v} & \text{v} & \text{v} \\
\end{array}
\]

\( \text{topBorderHeight}=2 \quad \text{leftBorderWidth}=2 \)

The height (width) of the destination ROI cannot be less than the sum of the height (width) of source ROI and the \texttt{topBorderHeight (leftBorderWidth)} parameter.

Return Values

- \texttt{ippStsNoErr}: Indicates no error. Any other value indicates an error or a warning.
- \texttt{ippStsNullPtrErr}: Indicates an error when any of the specified pointers is \texttt{NULL}.
- \texttt{ippStsSizeErr}: Indicates an error condition if \texttt{srcRoiSize} or \texttt{dstRoiSize} has a field with a zero or negative value, or \texttt{topBorderHeight} or \texttt{leftBorderWidth} is less than zero, or \texttt{dstRoiSize.width} < \texttt{srcRoiSize.width + leftBorderWidth}, or \texttt{dstRoiSize.height} < \texttt{srcRoiSize.height + topBorderHeight}.
- \texttt{ippStsStepErr}: Indicates an error condition if \texttt{srcStep} or \texttt{dstStep} has a zero or negative value.

Example

The code example below shows how to use the function \texttt{ippiCopyConstBorder\_8u\_C1R}.

```c
Ipp8u src[8*4] = {3, 3, 3, 3, 8, 8, 8,
3, 2, 1, 2, 3, 8, 8, 8,
3, 2, 1, 2, 3, 8, 8, 8,
3, 3, 3, 3, 8, 8, 8};
Ipp8u dst[8*6];
IppiSize srcRoi = {5, 4};
```
IppiSize dstRoi = { 7, 6 };  
int borderWidth = 1; 
int borderHeight = 1;  
int borderVal = 0; 
ippiCopyConstBorder_8u_C1R(src, 8, srcRoi, dst, 8, dstRoi, borderHeight, borderWidth, borderVal);

Results  
source image:  
3 3 3 3 3 8 8 8  
3 2 1 2 3 8 8 8  
3 2 1 2 3 8 8 8  src  
3 3 3 3 3 8 8 8  

destination image:  
0 0 0 0 0 0 0 0  
0 3 3 3 3 3 3 0  
0 3 2 1 2 3 0  dst  
0 3 3 3 3 3 0  
0 0 0 0 0 0 0  

CopyMirrorBorder

Copies pixels values between two images and adds the mirrored border pixels.

Syntax

Case 1: Not-in-place operation

IppStatus ippiCopyMirrorBorder_<mod>(const Ipp<datatype>* pSrc, int srcStep, IppiSize srcRoiSize, Ipp<datatype>* pDst, int dstStep, IppiSize dstRoiSize, int topBorderHeight, int leftBorderWidth);

Supported values for mod:

8u_C1R  16u_C1R  16s_C1R  32s_C1R  32f_C1R  
8u_C3R  16u_C3R  16s_C3R  32s_C3R  32f_C3R  
8u_C4R  16u_C4R  16s_C4R  32s_C4R  32f_C4R  

Case 2: In-place operation

IppStatus ippiCopyMirrorBorder_<mod>(const Ipp<datatype>* pSrc, int srcDstStep, IppiSize srcRoiSize, IppiSize dstRoiSize, int topBorderHeight, int leftBorderWidth);

Supported values for mod:

8u_C1IR  16u_C1IR  16s_C1IR  32s_C1IR  32f_C1IR  
8u_C3IR  16u_C3IR  16s_C3IR  32s_C3IR  32f_C3IR  
8u_C4IR  16u_C4IR  16s_C4IR  32s_C4IR  32f_C4IR
Case 3: Not-in-place operation with platform-aware functions

IppStatus ippiCopyMirrorBorder_<mod>(const Ipp<datatype>* pSrc, IppSizeL srcStep, IppSizeL srcRoiSize, Ipp<datatype>* pDst, IppSizeL dstStep, IppSizeL dstRoiSize, IppSizeL topBorderHeight, IppSizeL leftBorderWidth);

Supported values for mod:

- 8u_C1R_L
- 16u_C1R_L
- 16s_C1R_L
- 32s_C1R_L
- 32f_C1R_L
- 8u_C3R_L
- 16u_C3R_L
- 16s_C3R_L
- 32s_C3R_L
- 32f_C3R_L
- 8u_C4R_L
- 16u_C4R_L
- 16s_C4R_L
- 32s_C4R_L
- 32f_C4R_L

Case 4: In-place operation with platform-aware functions

IppStatus ippiCopyMirrorBorder_<mod>(const Ipp<datatype>* pSrc, IppSizeL srcDstStep, IppiSizeL srcRoiSize, IppiSizeL dstRoiSize, IppSizeL topBorderHeight, IppSizeL leftBorderWidth);

Supported values for mod:

- 8u_C1IR_L
- 16u_C1IR_L
- 16s_C1IR_L
- 32s_C1IR_L
- 32f_C1IR_L
- 8u_C3IR_L
- 16u_C3IR_L
- 16s_C3IR_L
- 32s_C3IR_L
- 32f_C3IR_L
- 8u_C4IR_L
- 16u_C4IR_L
- 16s_C4IR_L
- 32s_C4IR_L
- 32f_C4IR_L

Include Files

ippi.h

Flavors with the _L suffix: ippi_l.h

Domain Dependencies

Headers: ippcore.h, ippvm.h, ipps.h
Libraries: ippcore.lib, ippvm.lib, ipps.lib

Parameters

- **pSrc**: Pointer to the source image ROI.
- **srcStep**: Distance, in bytes, between the starting points of consecutive lines in the source image.
- **pDst**: Pointer to the destination image.
- **dstStep**: Distance, in bytes, between the starting points of consecutive lines in the destination image.
- **srcDstStep**: Distance, in bytes, between the starting points of consecutive lines in the source and destination image.
- **srcRoiSize**: Size of the source ROI, in pixels.
- **dstRoiSize**: Size of the destination ROI, in pixels.
- **topBorderHeight**: Height of the top border, in pixels.
- **leftBorderWidth**: Width of the left border, in pixels.
Description

This function operates with ROI (see Regions of Interest in Intel IPP).

This function copies the source image \( pSrc \) to the destination image \( pDst \) and fills pixels outside the copied area (border pixels) in the destination image with the values of the source image pixels according to the scheme illustrated in the figure below. Squares marked in red correspond to pixels copied from the source image, that is, the source image ROI.

Creating a Mirrored Border

<table>
<thead>
<tr>
<th>13</th>
<th>12</th>
<th>11</th>
<th>12</th>
<th>13</th>
<th>14</th>
<th>14</th>
<th>13</th>
</tr>
</thead>
<tbody>
<tr>
<td>8</td>
<td>7</td>
<td>6</td>
<td>7</td>
<td>8</td>
<td>9</td>
<td>10</td>
<td>9</td>
</tr>
<tr>
<td>3</td>
<td>2</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>4</td>
</tr>
<tr>
<td>8</td>
<td>7</td>
<td>6</td>
<td>7</td>
<td>8</td>
<td>9</td>
<td>10</td>
<td>9</td>
</tr>
<tr>
<td>13</td>
<td>12</td>
<td>11</td>
<td>12</td>
<td>13</td>
<td>14</td>
<td>15</td>
<td>14</td>
</tr>
<tr>
<td>18</td>
<td>17</td>
<td>16</td>
<td>17</td>
<td>18</td>
<td>19</td>
<td>20</td>
<td>19</td>
</tr>
<tr>
<td>13</td>
<td>12</td>
<td>11</td>
<td>12</td>
<td>13</td>
<td>14</td>
<td>15</td>
<td>14</td>
</tr>
</tbody>
</table>

**NOTE**

In-place flavors actually add border pixels to the source image ROI, thus a destination image ROI is larger than the initial image.

The parameters \( \text{topBorderHeight} \) and \( \text{leftBorderWidth} \) specify the position of the first pixel of the source ROI in the destination image ROI.

The height (width) of the destination ROI cannot be less than the sum of the height (width) of source ROI and the \( \text{topBorderHeight} \) (\( \text{leftBorderWidth} \)) parameter.

**NOTE**

If border width is greater than the image size in the corresponding dimension, multiple reflections are obtained for this border.
NOTE
If you use this function for a tiled image, note that to perform correct mirroring, the size of a tile must be more than the size of the used border. For example, if the image referenced above is divided into two tiles of size 3x3 and 2x3, the second tile (cells are highlighted in yellow) is extended with top, right, and bottom borders (highlighted in gray). The width of the right border is not less than the second tile width, so the pixels (blue) required for constructing the border of the tiled image are out of the processed tile. Therefore the last column (red) of the border extended image is computed incorrectly:

<p>| | | | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>13</td>
<td>12</td>
<td>11</td>
<td>12</td>
<td>13</td>
<td>14</td>
<td>15</td>
<td>14</td>
</tr>
<tr>
<td>8</td>
<td>7</td>
<td>6</td>
<td>7</td>
<td>8</td>
<td>9</td>
<td>10</td>
<td>9</td>
</tr>
<tr>
<td>3</td>
<td>2</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>4</td>
</tr>
<tr>
<td>8</td>
<td>7</td>
<td>6</td>
<td>7</td>
<td>8</td>
<td>9</td>
<td>10</td>
<td>9</td>
</tr>
<tr>
<td>13</td>
<td>12</td>
<td>11</td>
<td>12</td>
<td>13</td>
<td>14</td>
<td>15</td>
<td>14</td>
</tr>
<tr>
<td>18</td>
<td>17</td>
<td>16</td>
<td>17</td>
<td>18</td>
<td>19</td>
<td>20</td>
<td>19</td>
</tr>
<tr>
<td>13</td>
<td>12</td>
<td>11</td>
<td>12</td>
<td>13</td>
<td>14</td>
<td>15</td>
<td>14</td>
</tr>
</tbody>
</table>

Return Values

- ippStsNoErr: Indicates no error. Any other value indicates an error or a warning.
- ippStsNullPtrErr: Indicates an error when any of the specified pointers is NULL.
- ippStsSizeErr: Indicates an error in any of the following cases:
  - srcRoiSize or dstRoiSize has a field with a zero or negative value
  - topBorderHeight or leftBorderWidth is less than zero
  - dstRoiSize.width < srcRoiSize.width + leftBorderWidth
  - dstRoiSize.height < srcRoiSize.height + topBorderHeight
- ippStsStepErr: Indicates an error when srcStep or dstStep has a zero or negative value.

Example

The code example below shows how to use the ippiCopyMirrorBorder_8u_C1R function.

```c
Ipp8u src[8*4] = { 1, 2, 3, 8, 8, 8, 8, 8,
                  10, 9, 8, 8, 8, 8, 8, 8,
                  11, 12, 13, 8, 8, 8, 8, 8,
                  19, 18, 17, 8, 8, 8, 8, 8};
Ipp8u dst[10*8];
IppiSize srcRoi = { 3, 4 }; IppiSize dstRoi = { 10, 8 };
int topBorderHeight = 2;
```
int leftBorderWidth = 2;
ippiCopyMirrorBorder_8u_C1R(src, 8, srcRoi, dst, 10, dstRoi, topBorderHeight, leftBorderWidth);

Result:

source image:
1  2  3  8  8  8  8  8
10  9  8  8  8  8  8  8
11 12 13  8  8  8  8  8
19 18 17  8  8  8  8  8

destination image:
13 12 11  12 11 12 13 12 11 12 13 12 13 12 11 12 13 12
8  9 10  9  8  9 10  9  8  9  8  9  8  9  8  9
3  2  1  2  3  2  1  2  3  2
8  9 10  9  8  9 10  9  8  9  8  9  8  9  8  9
13 12 11  12 11 12 13 12 11 12 13 12 17 18 19 18 17 18
13 12 11  12 11 12 13 12 11 12 13 12
8  9 10  9  8  9 10  9  8  9  8  9

See Also
Regions of Interest in Intel IPP

CopyReplicateBorder

Copies pixels values between two images and adds the replicated border pixels.

Syntax

Case 1: Not-in-place operation

IppStatus ippiCopyReplicateBorder_<mod>(const Ipp<datatype>* pSrc, int srcStep, IppiSize srcRoiSize, Ipp<datatype>* pDst, int dstStep, IppiSize dstRoiSize, int topBorderHeight, int leftBorderWidth);

Supported values for mod:

<table>
<thead>
<tr>
<th>8u_C1R</th>
<th>16u_C1R</th>
<th>16s_C1R</th>
<th>32s_C1R</th>
<th>32f_C1R</th>
</tr>
</thead>
<tbody>
<tr>
<td>8u_C3R</td>
<td>16u_C3R</td>
<td>16s_C3R</td>
<td>32s_C3R</td>
<td>32f_C3R</td>
</tr>
<tr>
<td>8u_C4R</td>
<td>16u_C4R</td>
<td>16s_C4R</td>
<td>32s_C4R</td>
<td>32f_C4R</td>
</tr>
<tr>
<td>8u_AC4R</td>
<td>16u_AC4R</td>
<td>16s_AC4R</td>
<td>32s_AC4R</td>
<td>32f_AC4R</td>
</tr>
</tbody>
</table>

Case 2: In-place operation

IppStatus ippiCopyReplicateBorder_<mod>(const Ipp<datatype>* pSrc, int srcDstStep, IppiSize srcRoiSize, IppiSize dstRoiSize, int topBorderHeight, int leftBorderWidth);

Supported values for mod:

<table>
<thead>
<tr>
<th>8u_C1IR</th>
<th>16u_C1IR</th>
<th>16s_C1IR</th>
<th>32s_C1IR</th>
<th>32f_C1IR</th>
</tr>
</thead>
<tbody>
<tr>
<td>8u_C3IR</td>
<td>16u_C3IR</td>
<td>16s_C3IR</td>
<td>32s_C3IR</td>
<td>32f_C3IR</td>
</tr>
<tr>
<td>8u_C4IR</td>
<td>16u_C4IR</td>
<td>16s_C4IR</td>
<td>32s_C4IR</td>
<td>32f_C4IR</td>
</tr>
</tbody>
</table>
Case 3: Not-in-place operation for platform-aware functions

IppStatus ippiCopyReplicateBorder_<mod>(const Ipp<datatype>* pSrc, IppSizeL srcStep, IppSizeL srcRoiSize, Ipp<datatype>* pDst, IppSizeL dstStep, IppSizeL dstRoiSize, IppSizeL topBorderHeight, IppSizeL leftBorderWidth);

Supported values for mod:

- 8u_C1R_L
- 16u_C1R_L
- 16s_C1R_L
- 32s_C1R_L
- 32f_C1R_L
- 8u_C3R_L
- 16u_C3R_L
- 16s_C3R_L
- 32s_C3R_L
- 32f_C3R_L
- 8u_C4R_L
- 16u_C4R_L
- 16s_C4R_L
- 32s_C4R_L
- 32f_C4R_L
- 8u_AC4R_L
- 16u_AC4R_L
- 16s_AC4R_L
- 32s_AC4R_L
- 32f_AC4R_L

Case 4: In-place operation for platform-aware functions

IppStatus ippiCopyReplicateBorder_<mod>(const Ipp<datatype>* pSrc, IppSizeL srcDstStep, IppSizeL srcRoiSize, IppSizeL dstRoiSize, IppSizeL topBorderHeight, IppSizeL leftBorderWidth);

Supported values for mod:

- 8u_C1IR_L
- 16u_C1IR_L
- 16s_C1IR_L
- 32s_C1IR_L
- 32f_C1IR_L
- 8u_C3IR_L
- 16u_C3IR_L
- 16s_C3IR_L
- 32s_C3IR_L
- 32f_C3IR_L
- 8u_C4IR_L
- 16u_C4IR_L
- 16s_C4IR_L
- 32s_C4IR_L
- 32f_C4IR_L
- 8u_AC4IR_L
- 16u_AC4IR_L
- 16s_AC4IR_L
- 32s_AC4IR_L
- 32f_AC4IR_L

Include Files

ippi.h

Flavors with the _L suffix:ippi_l.h

Domain Dependencies

Headers: ippcore.h, ippvm.h, ipps.h

Libraries: ippcore.lib, ippvm.lib, ipps.lib

Parameters

- pSrc: Pointer to the source image ROI.
- srcStep: Distance in bytes between starts of consecutive lines in the source image.
- pDst: Pointer to the destination image.
- dstStep: Distance in bytes between starts of consecutive lines in destination image.
- srcDstStep: Distance in bytes between starts of consecutive lines in the source and the destination image.
- srcRoiSize: Size of the source ROI in pixels.
- dstRoiSize: Size of the destination ROI in pixels.
- topBorderHeight: Height of the top border in pixels.
- leftBorderWidth: Width of the left border in pixels.
Description
This function operates with ROI (see Regions of Interest in Intel IPP).

This function copies the source image \( p_{Src} \) to the destination image \( p_{Dst} \) and fills pixels ("border") outside the copied area in the destination image with the values of the source image pixels according to the scheme illustrated in Figure Creating a Replicated Border. Squares marked in red correspond to pixels copied from the source image, that is, the source image ROI.

Note that the in-place function flavor actually adds border pixels to the source image ROI, thus a destination image ROI is larger than the initial image.

The parameters \( topBorderHeight \) and \( leftBorderWidth \) specify the position of the first pixel of the source ROI in the destination image ROI.

Creating a Replicated Border

<table>
<thead>
<tr>
<th>1</th>
<th>1</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>5</th>
<th>5</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>5</td>
<td>5</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>5</td>
<td>5</td>
</tr>
<tr>
<td>6</td>
<td>6</td>
<td>6</td>
<td>7</td>
<td>8</td>
<td>9</td>
<td>10</td>
<td>10</td>
<td></td>
</tr>
<tr>
<td>11</td>
<td>11</td>
<td>11</td>
<td>12</td>
<td>13</td>
<td>14</td>
<td>15</td>
<td>15</td>
<td></td>
</tr>
<tr>
<td>16</td>
<td>16</td>
<td>16</td>
<td>17</td>
<td>18</td>
<td>19</td>
<td>20</td>
<td>20</td>
<td></td>
</tr>
<tr>
<td>16</td>
<td>16</td>
<td>16</td>
<td>17</td>
<td>18</td>
<td>19</td>
<td>20</td>
<td>20</td>
<td></td>
</tr>
</tbody>
</table>

The height (width) of the destination ROI cannot be less than the sum of the height (width) of source ROI and the \( topBorderHeight \) (\( leftBorderWidth \)) parameter.

Return Values
ippStsNoErr Indicates no error. Any other value indicates an error or a warning.

ippStsNullPtrErr Indicates an error when any of the specified pointers is NULL.

ippStsSizeErr Indicates an error condition if \( srcRoiSize \) or \( dstRoiSize \) has a field with a zero or negative value, or \( topBorderHeight \) or \( leftBorderWidth \) is less than zero, or \( dstRoiSize.width < srcRoiSize.width + leftBorderWidth \), or \( dstRoiSize.height < srcRoiSize.height + topBorderHeight \).

ippStsStepErr Indicates an error condition if \( srcStep \) or \( dstStep \) has a zero or negative value.
**Example**

The code example below shows how to use the `ippiCopyReplicateBorder_8u_C1R` function.

```c
Ipp8u src[8*4] = {5, 4, 3, 4, 5, 8, 8, 8,
                   3, 2, 1, 2, 3, 8, 8, 8,
                   3, 2, 1, 2, 3, 8, 8, 8,
                   5, 4, 3, 4, 5, 8, 8, 8};
Ipp8u dst[9*8];
IppiSize srcRoi = { 5, 4 };      // Source ROI
IppiSize dstRoi = { 9, 8 };      // Destination ROI
int topborderHeight = 2;
int leftborderWidth  = 2;
ippiCopyReplicateBorder_8u_C1R(src, 8, srcRoi, dst, 9, dstRoi, topBorderHeight, leftBorderWidth);
```

**Results**

**source image:**

```
5 4 3 4 5 8 8 8
3 2 1 2 3 8 8 8
3 2 1 2 3 8 8 8
5 4 3 4 5 8 8 8
```

**destination image:**

```
5 5 5 4 3 4 5 5 5
5 5 5 4 3 4 5 5 5
5 5 5 4 3 4 5 5 5
3 3 3 2 1 2 3 3 3
3 3 3 2 1 2 3 3 3
5 5 5 4 3 4 5 5 5
5 5 5 4 3 4 5 5 5
5 5 5 4 3 4 5 5 5
```

---

**CopyWrapBorder**

Copies pixels values between two images and adds the border pixels.

**Syntax**

**Case 1: Not-in-place operation**

```c
IppStatus ippiCopyWrapBorder_32s_C1R(const Ipp32s* pSrc, int srcStep, IppiSize srcRoiSize, Ipp32s* pDst, int dstStep, IppiSize dstRoiSize, int topBorderHeight, int leftBorderWidth);
IppStatus ippiCopyWrapBorder_32f_C1R(const Ipp32f* pSrc, int srcStep, IppiSize srcRoiSize, Ipp32f* pDst, int dstStep, IppiSize dstRoiSize, int topBorderHeight, int leftBorderWidth);
```

**Case 2: In-place operation**

```c
IppStatus ippiCopyWrapBorder_32s_C1IR(const Ipp32s* pSrc, int srcDstStep, IppiSize srcRoiSize, IppiSize dstRoiSize, int topBorderHeight, int leftBorderWidth);
IppStatus ippiCopyWrapBorder_32f_C1IR(const Ipp32f* pSrc, int srcDstStep, IppiSize srcRoiSize, IppiSize dstRoiSize, int topBorderHeight, int leftBorderWidth);
```
Case 3: Not-in-place operation with platform-aware functions

IppStatus ippiCopyWrapBorder_32s_C1R_L(const Ipp32s* pSrc, IppSizeL srcStep, IppiSizeL srcRoiSize, Ipp32s* pDst, IppSizeL dstStep, IppiSizeL dstRoiSize, IppSizeL topBorderHeight, IppSizeL leftBorderWidth);
IppStatus ippiCopyWrapBorder_32f_C1R_L(const Ipp32f* pSrc, IppSizeL srcStep, IppiSizeL srcRoiSize, Ipp32f* pDst, IppSizeL dstStep, IppiSizeL dstRoiSize, IppSizeL topBorderHeight, IppSizeL leftBorderWidth);

Case 4: In-place operation with platform-aware functions

IppStatus ippiCopyWrapBorder_32s_C1IR_L(const Ipp32s* pSrc, IppSizeL srcDstStep, IppiSizeL srcRoiSize, IppSizeL dstRoiSize, IppSizeL topBorderHeight, IppSizeL leftBorderWidth);
IppStatus ippiCopyWrapBorder_32f_C1IR_L(const Ipp32f* pSrc, IppSizeL srcDstStep, IppiSizeL srcRoiSize, IppSizeL dstRoiSize, IppSizeL topBorderHeight, IppSizeL leftBorderWidth);

Include Files
ippi.h

Flavors with the _L suffix: ippi_l.h

Domain Dependencies
Headers: ippcore.h, ippvm.h, ipps.h
Libraries: ippcore.lib, ippvm.lib, ipps.lib

Parameters

pSrc Pointer to the source image ROI.
srcStep Distance in bytes between starts of consecutive lines in the source image.
pDst Pointer to the destination image ROI.
dstStep Distance in bytes between starts of consecutive lines in the destination image.
srcDstStep Distance in bytes between starts of consecutive lines in the source and destination image for in-place flavor.
srcRoiSize Size of the source ROI in pixels.
dstRoiSize Size of the destination ROI in pixels.
topBorderHeight Height of the top border in pixels.
leftBorderWidth Width of the left border in pixels.

Description

This function operates with ROI (see Regions of Interest in Intel IPP).

This function copies the source image pSrc to the destination image pDst and fills pixels ("border") outside the copied area in the destination image with the values of the source image pixels according to the scheme illustrated in Figure Creating a Border of Pixels by ippiCopyWrapBorder Function. Squares marked in red correspond to pixels copied from the source image.
Note that the in-place function flavor actually adds border pixels to the source image ROI, thus a destination image ROI is larger than the initial image.

The parameters \textit{topBorderHeight} and \textit{leftBorderWidth} specify the position of the first pixel of the source ROI in the destination image ROI.

**Creating a Border of Pixels by ippiCopyWrapBorder Function**

<table>
<thead>
<tr>
<th>14</th>
<th>15</th>
<th>11</th>
<th>12</th>
<th>13</th>
<th>14</th>
<th>15</th>
<th>11</th>
<th>12</th>
<th>13</th>
<th>14</th>
<th>15</th>
<th>11</th>
</tr>
</thead>
<tbody>
<tr>
<td>19</td>
<td>20</td>
<td>16</td>
<td>17</td>
<td>18</td>
<td>19</td>
<td>20</td>
<td>16</td>
<td>17</td>
<td>18</td>
<td>19</td>
<td>20</td>
<td>16</td>
</tr>
<tr>
<td>4</td>
<td>5</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>1</td>
</tr>
<tr>
<td>9</td>
<td>10</td>
<td>6</td>
<td>7</td>
<td>8</td>
<td>9</td>
<td>10</td>
<td>6</td>
<td>7</td>
<td>8</td>
<td>9</td>
<td>10</td>
<td>6</td>
</tr>
<tr>
<td>14</td>
<td>15</td>
<td>11</td>
<td>12</td>
<td>13</td>
<td>14</td>
<td>15</td>
<td>11</td>
<td>12</td>
<td>13</td>
<td>14</td>
<td>15</td>
<td>11</td>
</tr>
<tr>
<td>19</td>
<td>20</td>
<td>16</td>
<td>17</td>
<td>18</td>
<td>19</td>
<td>20</td>
<td>16</td>
<td>17</td>
<td>18</td>
<td>19</td>
<td>20</td>
<td>16</td>
</tr>
<tr>
<td>4</td>
<td>5</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>1</td>
</tr>
<tr>
<td>9</td>
<td>10</td>
<td>6</td>
<td>7</td>
<td>8</td>
<td>9</td>
<td>10</td>
<td>6</td>
<td>7</td>
<td>8</td>
<td>9</td>
<td>10</td>
<td>6</td>
</tr>
<tr>
<td>14</td>
<td>15</td>
<td>11</td>
<td>12</td>
<td>13</td>
<td>14</td>
<td>15</td>
<td>11</td>
<td>12</td>
<td>13</td>
<td>14</td>
<td>15</td>
<td>11</td>
</tr>
<tr>
<td>19</td>
<td>20</td>
<td>16</td>
<td>17</td>
<td>18</td>
<td>19</td>
<td>20</td>
<td>16</td>
<td>17</td>
<td>18</td>
<td>19</td>
<td>20</td>
<td>16</td>
</tr>
<tr>
<td>4</td>
<td>5</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>1</td>
</tr>
<tr>
<td>9</td>
<td>10</td>
<td>6</td>
<td>7</td>
<td>8</td>
<td>9</td>
<td>10</td>
<td>6</td>
<td>7</td>
<td>8</td>
<td>9</td>
<td>10</td>
<td>6</td>
</tr>
</tbody>
</table>

The height (width) of the destination ROI cannot be less than the sum of the height (width) of source ROI and the \textit{topBorderHeight} (\textit{leftBorderWidth}) parameter.

**Return Values**

- \textbf{ippStsNoErr} Indicates no error. Any other value indicates an error or a warning.
- \textbf{ippStsNullPtrErr} Indicates an error when any of the specified pointers is NULL.
- \textbf{ippStsSizeErr} Indicates an error condition if \textit{srcRoiSize} or \textit{dstRoiSize} has a field with a zero or negative value, or \textit{topBorderHeight} or \textit{leftBorderWidth} is less than zero, or \textit{dstRoiSize.width} < \textit{srcRoiSize.width} + \textit{leftBorderWidth}, or \textit{dstRoiSize.height} < \textit{srcRoiSize.height} + \textit{topBorderHeight}.
- \textbf{ippStsStepErr} Indicates an error condition if \textit{srcStep} or \textit{dstStep} has a zero or negative value.
- \textbf{ippStsNotEvenStepErr} Indicates an error condition if one of the step values is not divisible by 4 for floating point images, or by 2 for short integer images.

**Example**

The code example below shows how to use the \textit{ippiCopyWrapBorder_32s_C1R} function.

```c
Ipp32s src[8*4] = {
    5, 4, 3, 4, 5, 8, 8, 8,
    3, 2, 1, 2, 3, 8, 8, 8,
    3, 2, 1, 2, 3, 8, 8, 8,
    5, 4, 3, 4, 5, 8, 8, 8
};
```
Ipp32s dst[9*8];
IppSize srcRoi = { 5, 4 };
IppSize dstRoi = { 9, 8 };
int topborderHeight = 2;
int leftborderWidth = 2;
ippiCopyWrapBorder_32s_C1R (src, 8*sizeof(Ipp32s), srcRoi, dst, 9*sizeof(Ipp32s), dstRoi, topBorderHeight, leftBorderWidth);

Results
source image:
5 4 3 4 5 8 8 8
3 2 1 2 3 8 8 8
3 2 1 2 3 8 8 8
5 4 3 4 5 8 8 8
destination image:
2 3 3 2 1 2 3 3 2
4 5 5 4 3 4 5 5 4
4 5 5 4 3 4 5 5 4
2 3 3 2 1 2 3 3 2
2 3 3 2 1 2 3 3 2
4 5 5 4 3 4 5 5 4
4 5 5 4 3 4 5 5 4
2 3 3 2 1 2 3 3 2

CopySubpix
Copies pixel values between two images with subpixel precision.

Syntax
Case 1: Copying without conversion or with conversion to floating point data
IppStatus ippiCopySubpix_<mod>(const Ipp<srcDatatype>* pSrc, int srcStep, Ipp<dstDatatype>* pDst, int dstStep, IppiSize roiSize, Ipp32f dx, Ipp32f dy);

Supported values for mod:

<table>
<thead>
<tr>
<th>8u_C1R</th>
<th>16u_C1R</th>
<th>32f_C1R</th>
</tr>
</thead>
<tbody>
<tr>
<td>8u32f_C1R</td>
<td>16u32f_C1R</td>
<td></td>
</tr>
</tbody>
</table>

Case 2: Copying with conversion to integer data
IppStatus ippiCopySubpix_8u16u_C1R_Sfs(const Ipp8u* pSrc, int srcStep, Ipp16u* pDst, int dstStep, IppiSize roiSize, Ipp32f dx, Ipp32f dy, int scaleFactor);

Include Files
ippcv.h

Domain Dependencies
Headers: ippcore.h, ippvm.h, ipps.h,ippi.h
Libraries: ippcore.lib, ippvm.lib, ipps.lib,ippi.lib
Parameters

- **pSrc**: Pointer to the source image ROI.
- **srcStep**: Distance in bytes between starts of consecutive lines in the source image.
- **pDst**: Pointer to the destination image ROI.
- **dstStep**: Distance in bytes between starts of consecutive lines in the destination image.
- **roiSize**: Size of the source and destination ROI in pixels.
- **dx**: Fractional part of the x-coordinate in the source image.
- **dy**: Fractional part of the y-coordinate in the source image.
- **scaleFactor**: Scale factor (see Integer Result Scaling).

Description

This function operates with ROI (see Regions of Interest in Intel IPP).

This function computes the pixel value of the destination image using linear interpolation (see Linear Interpolation in Appendix B) in accordance with the following formula:

\[
p_{DST_{j,i}} = p_{SRC_{j*dx,i*dy}}
\]

where \( i = 0, \ldots, \text{roiSize.height} - 1, j = 0, \ldots, \text{roiSize.width} - 1 \).

Return Values

- **ippStsNoErr**: Indicates no error. Any other value indicates an error or a warning.
- **ippStsNullPtrErr**: Indicates an error when any of the specified pointers is NULL.
- **ippStsSizeErr**: Indicates an error condition if **roiSize** has a field with a zero or negative value.
- **ippStsStepErr**: Indicates an error condition if **srcStep** or **dstStep** is less than \( \text{roiSize.width} * \text{<pixelSize>} \).
- **ippStsNotEvenStepErr**: Indicates an error condition if one of **srcStep** or **dstStep** is not divisible by 4 for floating point images, or by 2 for short integer images.

**CopySubpixIntersect**

Copies pixel values of the intersection with specified window with subpixel precision.
Syntax

Case 1: Copying without conversion or with conversion to floating point data

IppStatus ippiCopySubpixIntersect_<mod>(const Ipp<srcDatatype>* pSrc, int srcStep,
IppSize srcRoiSize, Ipp<srcDatatype>* pDst, int dstStep, IppSize dstRoiSize,
IppiPoint_32f point, IppiPoint* pMin, IppiPoint* pMax);

Supported values for mod:
  8u_C1R  16u_C1R  32f_C1R
  8u32f_C1R  16u32f_C1R

Case 2: Copying with conversion to integer data

IppStatus ippiCopySubpixIntersect_8u16u_C1R_Sfs(const Ipp8u* pSrc, int srcStep,
IppSize srcRoiSize, Ipp16u* pDst, int dstStep, IppSize dstRoiSize, IppiPoint_32f
point, IppiPoint* pMin, IppiPoint* pMax, int scaleFactor);

Include Files

ippcv.h

Domain Dependencies

Headers: ippcore.h, ippvm.h, ipps.h,ippi.h
Libraries: ippcore.lib, ippvm.lib, ipps.lib,ippi.lib

Parameters

pSrc  Pointer to the source image ROI.
srcStep  Distance in bytes between starts of consecutive lines in the source image.
srcRoiSize  Size of the source image ROI in pixels.
pDst  Pointer to the destination image ROI.
dstStep  Distance in bytes between starts of consecutive lines in the source and
destination image.
dstRoiSize  Size of the destination ROI in pixels.
point  Center point of the window.
pMin  Pointer to coordinates of the top left pixel of the intersection in the
destination image.
pMax  Pointer to coordinates of the bottom right pixel of the intersection in the
destination image.
scaleFactor  Scale factor (see Integer Result Scaling).

Description

This function operates with ROI (see Regions of Interest in Intel IPP).

This function determines the intersection of the source image and the window of size dstRoiSize centered
in point point. The corresponding pixels of the destination image are calculated using linear interpolation
(see Linear Interpolation in Appendix B) in accordance with the following formula:
where \( X_{\text{subpix}}(j) = \min(\max(point.x + j - 0.5*(dstRoiSize.width - 1), 0), srcRoiSize.width - 1) \),
\[ Y_{\text{subpix}}(i) = \min(\max(point.y + i - 0.5*(dstRoiSize.height - 1), 0), srcRoiSize.height - 1) \],
\( i = 0, \ldots dstRoiSize.height - 1 \), \( j = 0, \ldots dstRoiSize.width - 1 \).

Minimal values of \( j \) and \( i \) (coordinates of the top left calculated destination pixel) are assigned to \( pMin.x \) and \( pMin.y \), maximal values (coordinates of the top left calculated destination pixel) - to \( pMin.x \) and \( pMin.y \).

(See Figure Image Copying with Subpixel Precision Using ippiCopySubpixIntersect Function.)

**Image Copying with Subpixel Precision Using ippiCopySubpixIntersect Function**

The height (width) of the destination ROI cannot be less than the sum of the height (width) of source ROI and the topBorderHeight (leftBorderWidth) parameter.

**Return Values**

- **ippiStsNoErr**
  Indicates no error. Any other value indicates an error or a warning.

- **ippiStsNullPtrErr**
  Indicates an error when any of the specified pointers is NULL.

- **ippiStsSizeErr**
  Indicates an error condition if srcRoiSize or dstRoiSize has a field with a zero or negative value.

- **ippiStsStepErr**
  Indicates an error condition if srcStep is less than srcRoiSize.width * <pixelSize>, or dstStep is less than dstRoiSize.width * <pixelSize>.

**Example**

The code example below shows how to use the function ippiCopySubpixIntersect_8u_C1R.

```c
Ipp8u src[8*6] = {
    7, 7, 6, 6, 6, 6, 7, 7,
    6, 5, 5, 5, 5, 5, 6,
}```
};
Ipp8u dst[7*4];
IppiSize srcRoi = { 8, 6 };
IppiSize dstRoi = { 7, 4 };
IppiPoint_32f point = { 4, 1 };
IppiPoint min;
IppiPoint max;

ippiCopySubpixIntersect_8u_C1R (src, 8, srcRoi, dst, 7, dstRoi, point, &min, &max );

Results
source image:
7 7 6 6 6 6 7 7
6 5 5 5 5 5 6
6 5 4 3 3 4 6
6 5 4 3 3 4 6
6 5 5 5 5 5 6
6 6 6 6 6 6 6
destination image:
7 6 6 6 6 7 7
6 6 6 6 6 7
5 5 4 4 5 6
5 4 3 3 5 6
min = { 0, 1 }
max = { 5, 3 }

Dup
Copies a gray scale image to all channels of the color image.

Syntax
IppStatus ippiDup_8u_C1C3R(const Ipp8u* pSrc, int srcStep, Ipp8u* pDst, int dstStep, IppiSize roiSize);
IppStatus ippiDup_8u_C1C4R(const Ipp8u* pSrc, int srcStep, Ipp8u* pDst, int dstStep, IppiSize roiSize);

Include Files
ippi.h

Domain Dependencies
Headers: ippcore.h, ippvm.h, ipps.h
Libraries: ippcore.lib, ippvm.lib, ipps.lib
**Parameters**

- **pSrc**: Pointer to the source image ROI.
- **srcStep**: Distance in bytes between starts of consecutive lines in the source image.
- **pDst**: Pointer to the destination image ROI.
- **dstStep**: Distance in bytes between starts of consecutive lines in the destination image.
- **roiSize**: Size of the source and destination ROI in pixels.

**Description**

This function operates with ROI (see Regions of Interest in Intel IPP).

This function copies a one-channel (gray scale) image `pSrc` to each channel of the multi-channel image `pDst`.

**Return Values**

- **ippStsNoErr**: Indicates no error. Any other value indicates an error or a warning.
- **ippStsNullPtrErr**: Indicates an error when any of the specified pointers is NULL.
- **ippStsSizeErr**: Indicates an error condition if `RoiSize` has a field with a zero or negative value.

**Transpose**

Transposes a source image.

**Syntax**

**Case 1: Not-in-place operation**

```c
IppStatus ippiTranspose_<mod>(const Ipp<datatype>* pSrc, int srcStep, Ipp<datatype>* pDst, int dstStep, IppiSize roiSize);
```

Supported values for `mod`:

- 8u_C1R
- 16u_C1R
- 16s_C1R
- 32s_C1R
- 32f_C1R
- 8u_C3R
- 16u_C3R
- 16s_C3R
- 32s_C3R
- 32f_C3R
- 8u_C4R
- 16u_C4R
- 16s_C4R
- 32s_C4R
- 32f_C4R

**Case 2: In-place operation**

```c
IppStatus ippiTranspose_<mod>(Ipp<datatype>* pSrcDst, int srcDstStep, IppiSize roiSize);
```

Supported values for `mod`:

- 8u_C1IR
- 16u_C1IR
- 16s_C1IR
- 32s_C1IR
- 32f_C1IR
- 8u_C3IR
- 16u_C3IR
- 16s_C3IR
- 32s_C3IR
- 32f_C3IR
- 8u_C4IR
- 16u_C4IR
- 16s_C4IR
- 32s_C4IR
- 32f_C4IR
Include Files
ippi.h

Domain Dependencies
Headers: ippcore.h, ippvm.h, ipps.h
Libraries: ippcore.lib, ippvm.lib, ipps.lib

Parameters

pSrc  
Pointer to the source image ROI.

srcStep  
Distance, in bytes, between the starting points of consecutive lines in the source image.

pDst  
Pointer to the destination image ROI.

dstStep  
Distance, in bytes, between the starting points of consecutive lines in the destination image.

pSrcDst  
Pointer to the source and destination ROI for in-place operation.

srcDstStep  
Distance, in bytes, between the starting points of consecutive lines in the source and destination image buffer for the in-place operation.

roiSize  
Size of the source ROI in pixels.

Description

This function operates with ROI.

This function transposes the source image pSrc (pSrcDst for in-place flavors) and stores the result in pDst (pSrcDst). The destination image is obtained from the source image by transforming the columns to the rows: pDst(x,y) = pSrc(y,x)

The parameter roiSize is specified for the source image. The value of the roiSize.width parameter for the destination image is equal to roiSize.height for the source image, and roiSize.height for the destination image is equal to roiSize.width for the source image.

NOTE
For in-place operations, roiSize.width must be equal to roiSize.height.

Return Values

ippStsNoErr  
Indicates no error. Any other value indicates an error or a warning.

ippStsNullPtrErr  
Indicates an error when any of the specified pointers is NULL, with the exception of second mode in Case 4.

ippStsSizeErr  
Indicates an error when:

• roiSize has a field with a zero or negative value
• roiSize.width is not equal to roiSize.height for in-place flavors
Example

The code example below shows how to use the ippiTranspose_8u_C1R function.

```c
Ipp8u src[8*4] = {1, 2, 3, 4, 8, 8, 8, 8,
                 1, 2, 3, 4, 8, 8, 8, 8,
                 1, 2, 3, 4, 8, 8, 8, 8};
Ipp8u dst[4*4];
IppSize srcRoi = { 4, 4 };
ippiTranspose_8u_C1R ( src, 8, dst, 4, srcRoi );
```

Result:

```
1 2 3 4 8 8 8 8
1 2 3 4 8 8 8 8    src
1 2 3 4 8 8 8 8
1 1 1 1
2 2 2 2
3 3 3 3      dst
4 4 4 4
```

See Also

Regions of Interest in Intel IPP

SwapChannels

Copies channels of the source image to the destination image.

Syntax

Case 1: Not-in-place operation

```c
IppStatus ippiSwapChannels_<mod>(const Ipp<datatype>* pSrc, int srcStep, Ipp<datatype>* pDst, int dstStep, IppSize roiSize, const int dstOrder[3]);
```

Supported values for `mod`:

<table>
<thead>
<tr>
<th></th>
<th>8u_C3R</th>
<th>16u_C3R</th>
<th>16s_C3R</th>
<th>32s_C3R</th>
<th>32f_C3R</th>
</tr>
</thead>
<tbody>
<tr>
<td>8u_AC4R</td>
<td></td>
<td>16u_AC4R</td>
<td>16s_AC4R</td>
<td>32s_AC4R</td>
<td>32f_AC4R</td>
</tr>
</tbody>
</table>

```c
IppStatus ippiSwapChannels_<mod>(const Ipp<datatype>* pSrc, int srcStep, Ipp<datatype>* pDst, int dstStep, IppSize roiSize, const int dstOrder[4]);
```

Supported values for `mod`:

<table>
<thead>
<tr>
<th></th>
<th>8u_C4R</th>
<th>16u_C4R</th>
<th>16s_C4R</th>
<th>32s_C4R</th>
<th>32f_C4R</th>
</tr>
</thead>
<tbody>
<tr>
<td>8u_AC4R</td>
<td></td>
<td>16u_AC4R</td>
<td>16s_AC4R</td>
<td>32s_AC4R</td>
<td>32f_AC4R</td>
</tr>
</tbody>
</table>

Case 2: In-place operation

```c
IppStatus ippiSwapChannels_8u_C3IR(Ipp8u* pSrcDst, int srcDstStep, IppSize roiSize, const int dstOrder[3]);
```
Case 3: Operation with converting 3-channel image to the 4-channel image

IppStatus ippiSwapChannels_8u_C4IR(Ipp8u* pSrcDst, int srcDstStep, IppSize roiSize, const int dstOrder[4]);

Supported values for mod:
8u_C3C4R   16u_C3C4R   16s_C3C4R   32s_C3C4R   32f_C3C4R

Case 4: Operation with converting 4-channel image to the 3-channel image

IppStatus ippiSwapChannels_<mod>(const Ipp<datatype>* pSrc, int srcStep, Ipp<datatype>* pDst, int dstStep, IppSize roiSize, const int dstOrder[3]);

Supported values for mod:
8u_C4C3R   16u_C4C3R   16s_C4C3R   32s_C4C3R   32f_C4C3R

Include Files
ippi.h

Domain Dependencies
Headers: ippcore.h, ippvm.h, ipps.h
Libraries: ippcore.lib, ippvm.lib, ipps.lib

Parameters
pSrc  Pointer to the source image ROI.
srcStep  Distance in bytes between starts of consecutive lines in the source image.
pDst  Pointer to the destination image ROI.
dstStep  Distance in bytes between starts of consecutive lines in the destination image.
pSrcDst  Pointer to the source and destination ROI for in-place operation.
srcDstStep  Distance in bytes between starts of consecutive lines in the source and destination image for the in-place operation.
roiSize  Size of the source and destination ROI in pixels.
dstOrder  Order of channels in the destination image.
val  Constant value.

Description
This function operates with ROI (see Regions of Interest in Intel IPP).

This function copies the data from specified channels of the source image ROI pSrc to the specified channels of the destination image ROI pDst.

The first channel in the destination image is determined by the first component of dstOrder. Its value lies in the range [0..2] for a 3-channel image, and [0..3] for a 4-channel image, and indicates the corresponding channel number of the source image. Other channels are specified in the similar way. For example, if the
sequence of channels in the source 3-channel image is $A, B, C$, and $dstOrder[0]=2, dstOrder[1]=0, dstOrder[2]=1$, then the order of channels in the 3-channel destination image is $C, A, B$. Some or all components of $dstOrder$ may have the same values. It means that data from a certain channel of the source image may be copied to several channels of the destination image.

Some functions flavors convert a 3-channel source image to the 4-channel destination image (see Case 3). In this case an additional channel contains data from any specified source channel, or its pixel values are set to the specified constant value $val$ (corresponding component $dstOrder[n]$ should be set to 3), or its pixel values are not set at all (corresponding component $dstOrder[n]$ should be set to an arbitrary value greater than 3). For example, the sequence of channels in the source 3-channel image is $A, B, C$, if $dstOrder[0]=1, dstOrder[1]=0, dstOrder[2]=1, dstOrder[3]=2$, then the order of channels in the 4-channel destination image will be $B, A, B, C$; if $dstOrder[0]=4, dstOrder[1]=0, dstOrder[2]=1, dstOrder[3]=2$, then the order of channels in the 4-channel destination image will be $D, A, B, C$, where $D$ is a channel whose pixel values are not set.

The function flavors that support image with the alpha channel do not perform operation on it.

This function supports negative step values.

**Return Values**

- **ippStsNoErr**
  Indicates no error. Any other value indicates an error or a warning.

- **ippStsNullPtrErr**
  Indicates an error when any of the specified pointers is NULL.

- **ippStsSizeErr**
  Indicates an error condition if $roiSize$ has a field with zero or negative value.

- **ippStsStepErr**
  Indicates an error condition if $srcStep$ or $dstStep$ has a zero value.

**Example**

The code example below shows how to use the function `ippiSwapChannels_8u_C3R`.

```c
Ipp8u src[12*3] = { 255, 0, 0, 255, 0, 0, 255, 0, 0, 255, 0, 0,
                      0, 255, 0, 0, 255, 0, 0, 255, 0, 0, 255, 0,
                      0, 0, 255, 0, 0, 255, 0, 0, 255, 0, 0, 255};
Ipp8u dst[12*3];
IppiSize roiSize = { 4, 3 };  
int order[3] = { 2, 1, 0 };
ippiSwapChannels_8u_C3R ( src, 12, dst, 12, roiSize, order );
```

Result:

<table>
<thead>
<tr>
<th>src</th>
<th>[rgb]</th>
</tr>
</thead>
<tbody>
<tr>
<td>255 0 0 255 0 0 255 0 0 255 0 0</td>
<td></td>
</tr>
<tr>
<td>0 255 0 0 255 0 0 255 0 0 255 0</td>
<td></td>
</tr>
<tr>
<td>0 0 255 0 0 255 0 0 255 0 0 255</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>dst</th>
<th>[bgr]</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 0 255 0 0 255 0 0 255 0 0 255</td>
<td></td>
</tr>
<tr>
<td>0 255 0 0 255 0 0 255 0 0 255 0</td>
<td></td>
</tr>
<tr>
<td>255 0 0 255 0 0 255 0 0 255 0 0</td>
<td></td>
</tr>
</tbody>
</table>
AddRandUniform

Generates random samples with uniform distribution and adds them to an image data.

Syntax

IppStatus ippiAddRandUniform_<mod>(Ipp<datatype>* pSrcDst, int srcDstStep, IppSize roiSize, Ipp<datatype> low, Ipp<datatype> high, unsigned int* pSeed);

Supported values for mod:

- 8u_C1IR
- 16u_C1IR
- 16s_C1IR
- 32f_C1IR
- 8u_C3IR
- 16u_C3IR
- 16s_C3IR
- 32f_C3IR
- 8u_C4IR
- 16u_C4IR
- 16s_C4IR
- 32f_C4IR
- 8u_AC4IR
- 16u_AC4IR
- 16s_AC4IR
- 32f_AC4IR

Include Files

ippi.h

Domain Dependencies

Headers: ippcore.h, ippvm.h, ipps.h
Libraries: ippcore.lib, ippvm.lib, ipps.lib

Parameters

- pSrcDst: Pointer to the source and destination image ROI.
- srcDstStep: Distance in bytes between starts of consecutive lines in the source and destination image.
- roiSize: Size of the image ROI in pixels.
- low: The lower bound for the range of uniformly distributed values.
- high: The upper bound for the range of uniformly distributed values.
- pSeed: The initial seed value for the pseudo-random number generator.

Description

This function operates with ROI (see Regions of Interest in Intel IPP).

The function generates samples with uniform distribution over the range \([low, high]\) and adds them to a source image pointed to by pSrcDst.

The resulting pixel values that exceed the image data range are saturated to the respective data-range limits. To obtain an image that contains pure noise with uniform distribution, call ippiAddRandUniform using a source image with zero data as input.

Return Values

- ippStsNoErr: Indicates no error. Any other value indicates an error or a warning.
- ippStsNullPtrErr: Indicates an error when any of the specified pointers is NULL.
Indicates an error condition if `roiSize` has a field with zero or negative value.

Indicates an error condition if `srcDstStep` has a zero or negative value.

Example

The code example below shows data conversion without scaling.

```c
IppStatus randUniform( void ) {
    unsigned int seed = 123456;
    Ipp8u img[2048], mn, mx;
    IppiSize roi=(2048,1);
    Ipp64f mean;
    IppStatus st;
    ippiSet_8u_C1R( 0, img, 2048, roi);
    st = ippiAddRandUniform_8u_C1IR(img, 2048, roi, 0, 255, &seed);
    ippiMean_8u_C1R( img, 2048, roi, &mean );
    ippiMinMax_8u_C1R( img, 2048, roi, &mn, &mx );
    printf("[\d..\d], mean=%.3f\n", mn, mx, mean );
    return st;
}
```

AddRandGauss

Generates random samples with Gaussian distribution and adds them to an image data.

Syntax

```c
IppStatus ippiAddRandGauss_<mod>(Ipp<datatype>* pSrcDst, int srcDstStep, IppiSize roiSize, Ipp<datatype> mean, Ipp<datatype> stDev, unsigned int* pSeed);
```

Supported values for `mod`:

- `8u_C1IR`
- `16u_C1IR`
- `16s_C1IR`
- `32f_C1IR`
- `8u_C3IR`
- `16u_C3IR`
- `16s_C3IR`
- `32f_C3IR`
- `8u_C4IR`
- `16u_C4IR`
- `16s_C4IR`
- `32f_C4IR`
- `8u_AC4IR`
- `16u_AC4IR`
- `16s_AC4IR`
- `32f_AC4IR`

Include Files

`ippi.h`

Domain Dependencies

Headers: `ippcore.h`, `ippvm.h`, `ipps.h`

Libraries: `ippcore.lib`, `ippvm.lib`, `ipps.lib`

Parameters

- `pSrcDst`: Pointer to the source and destination image ROI.
- `srcDstStep`: Distance in bytes between starts of consecutive lines in the source and destination image.
roiSize
Size of the image ROI in pixels.

mean
The mean of the Gaussian distribution.

stDev
The standard deviation of the Gaussian distribution.

pSeed
The initial seed value for the pseudo-random number generator.

Description
This function operates with ROI (see Regions of Interest in Intel IPP).
The function generates samples with Gaussian distribution that have the mean value mean and standard deviation stdev and adds them to a source image ROI pointed to by pSrcDst.
The resulting pixel values that exceed the image data range are saturated to the respective data-range limits. To obtain an image which contains pure noise with Gaussian distribution, callippiAddRandGauss using a source image with zero data as input.

Return Values
ippStsNoErr Indicates no error. Any other value indicates an error or a warning.
ippStsNullPtrErr Indicates an error condition if pSrcDst or pSeed pointer is NULL.
ippStsSizeErr Indicates an error condition if roiSize has a field with zero or negative value.
ippStsStepErr Indicates an error condition if srcDstStep has a zero or negative value.

ImageJaehne
Creates Jaehne test image.

Syntax
IppStatusippiImageJaehne_<mod>(Ipp<datatype>* pDst, int dstStep, IppiSize roiSize);

Supported values for mod:
8u_C1R 16u_C1R 16s_C1R 32f_C1R
8u_C3R 16u_C3R 16s_C3R 32f_C3R
8u_C4R 16u_C4R 16s_C4R 32f_C4R
8u_AC4R 16u_AC4R 16s_AC4R 32f_AC4R

Include Files
ippi.h

Domain Dependencies
Headers: ippcore.h, ippvm.h, ipps.h
Libraries: ippcore.lib, ippvm.lib, ipps.lib
**Parameters**

- **pDst**: Pointer to the destination image ROI.
- **dstStep**: Distance in bytes between starts of consecutive lines in the destination image.
- **roiSize**: Size of the destination image ROI in pixels.

**Description**

This function operates with ROI (see Regions of Interest in Intel IPP).

This function creates a specific one- or three-channel test image that has been first introduced to digital image processing by B.Jaehne (see [Jae95]).

The destination image pixel values are computed according to the following formula:

\[
\text{Dst}(x,y) = A \times \sin\left(0.5 \times \text{IPP}_\pi \times \frac{x^2 + y^2}{\text{roiSize}.height}\right),
\]

where \(x, y\) are pixel coordinates varying in the range

\[0 \leq x \leq \text{roiSize}.width - 1, 0 \leq y \leq \text{roiSize}.height - 1;\]

\text{IPP}_\pi \text{ is the library constant that stands for } \pi \text{ value.}

\[x^2 = \frac{(x - \text{roiSize}.width + 1)}{2.0},\]
\[y^2 = \frac{(y - \text{roiSize}.height + 1)}{2.0},\]

\(A\) is the constant value that depends upon the image type being created.

For the 32f floating point data, the pixel values in the created image can vary in the range between 0 (inclusive) and 1 (exclusive).
Figure Example of a Generated Jaehne's Test Image illustrates an example of a test image generated by the \texttt{ippiImageJaehne} function.

**Example of a Generated Jaehne's Test Image**

These test images can be effectively used when you need to visualize and interpret the results of applying filtering functions, similarly to what is proposed in [Jae95].

**Return Values**

- \texttt{ippStsNoErr} Indicates no error. Any other value indicates an error or a warning.
- \texttt{ippStsNullPtrErr} Indicates an error condition if the \texttt{pDst} pointer is NULL.
- \texttt{ippStsSizeErr} Indicates an error condition if \texttt{roiSize} has a field with a zero or negative value, or if \texttt{dstStep} is less than or equal to zero.

**ImageRamp**

*Creates a test image that has an intensity ramp.*

**Syntax**

\begin{verbatim}
IppStatus ippiImageRamp_<mod>(Ipp<datatype>* pDst, int dstStep, IppiSize roiSize, float offset, float slope, IppiAxis axis);
\end{verbatim}

Supported values for \texttt{mod}:

- 8u_C1R
- 16u_C1R
- 16s_C1R
- 32f_C1R
- 8u_C3R
- 16u_C3R
- 16s_C3R
- 32f_C3R
- 8u_C4R
- 16u_C4R
- 16s_C4R
- 32f_C4R
Include Files

ippi.h

Domain Dependencies

Headers: ippcore.h, ippvm.h, ipps.h
Libraries: ippcore.lib, ippvm.lib, ipps.lib

Parameters

pDst  
Pointer to the destination image ROI.

dstStep  
Distance in bytes between starts of consecutive lines in the destination image.

roiSize  
Size of the destination image ROI in pixels.

offset  
Offset value.

slope  
Slope coefficient.

axis  
Specifies the direction of the image intensity ramp; can be one of the following:

ippAxsHorizontal  
for the ramp in X-direction,

ippAxsVertical  
for the ramp in Y-direction,

ippAxsBoth  
for the ramp in both X and Y-directions.

Description

This function operates with ROI (see Regions of Interest in Intel IPP).
The function creates a one- or three-channel image that can be used as a test image to examine the effect of applying different image processing functions.
The destination image pixel values are computed according to one of the following formulas:
dst\( (x,y) = offset + slope \times x \), if \( axis = ippAxsHorizontal \),
dst\( (x,y) = offset + slope \times y \), if \( axis = ippAxsVertical \),
dst\( (x,y) = offset + slope \times x \times y \), if \( axis = ippAxsBoth \),

where \( x, y \) are pixel coordinates varying in the range
\( 0 \leq x \leq roiSize.width - 1, 0 \leq y \leq roiSize.height - 1 \);

Note that linear transform coefficients \( offset \) and \( slope \) have floating-point values for all function flavors.
The computed pixel values that exceed the image data range are saturated to the respective data-range limits.

Return Values

ippStsNoErr  
Indicates no error. Any other value indicates an error or a warning.

ippStsNullPtrErr  
Indicates an error condition if the \( pDst \) pointer is NULL.
Indicates an error condition if roiSize has a field with a zero or negative value, or if dstStep is less than or equal to zero.

Example

The code example below illustrates how to use the ippiImageRamp function.

```c
IppStatus ramp( void ){
    Ipp8u dst[8*4];
    IppiSize roiSize = { 8, 4 };
    return ippiImageRamp_8u_C1R( dst, 8, roiSize, 0.0f, 256.0f/7, ippAxsHorizontal);
}
```

The destination image contains the following data:

| 00 25 49 6E 92 B7 DB FF |
| 00 25 49 6E 92 B7 DB FF |
| 00 25 49 6E 92 B7 DB FF |
| 00 25 49 6E 92 B7 DB FF |

SampleLine

**Puts a raster line into buffer.**

**Syntax**

```c
IppStatus ippiSampleLine_<mod>( const Ipp<datatype>* pSrc, int srcStep, IppiSize roiSize, Ipp<datatype>* pDst, IppiPoint pt1, IppiPoint pt2);
```

**Supported values for mod:**

- 8u_C1R
- 16u_C1R
- 32f_C1R
- 8u_C3R
- 16u_C3R
- 32f_C3R

**Include Files**

ippcv.h

**Domain Dependencies**

Headers: ippcore.h, ippvm.h, ipps.h,ippi.h

Libraries: ippcore.lib, ippvm.lib,ipps.lib,ippi.lib

**Parameters**

- **pSrc**
  - Pointer to the ROI in the source raster image.
- **srcStep**
  - Distance in bytes between starts of consecutive lines in the raster image.
- **roiSize**
  - Size of the image ROI in pixels.
- **pDst**
  - Pointer to the destination buffer. The buffer is to store at least \( \max(|pt2.x - pt1.x| + 1, |pt2.y - pt1.y| + 1) \) points.
- **pt1**
  - Starting point of the line.
Description
This function operates with ROI (see Regions of Interest in Intel IPP).
This function iterates through the points that belong to the raster line using the 8-point connected Bresenham algorithm, and puts the resulting pixels into the destination buffer.

Return Values

<table>
<thead>
<tr>
<th>Return Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ippStsNoErr</td>
<td>Indicates no error. Any other value indicates an error or a warning.</td>
</tr>
<tr>
<td>ippStsNullPtrErr</td>
<td>Indicates an error condition if any of the specified pointers is NULL.</td>
</tr>
<tr>
<td>ippStsSizeErr</td>
<td>Indicates an error condition if roiSize.width or roiSize.height is less than or equal to zero.</td>
</tr>
<tr>
<td>ippStsStepErr</td>
<td>Indicates an error condition if srcStep is less than roiSize.width * &lt;pixelSize&gt;.</td>
</tr>
<tr>
<td>ippStsNotEvenStepErr</td>
<td>Indicates an error when the step for the floating-point image cannot be divided by 4.</td>
</tr>
<tr>
<td>ippStsOutOfRangeErr</td>
<td>Indicates an error when any of the line ending points is outside the image.</td>
</tr>
</tbody>
</table>

Example
The code example below shows how to use the function ippiSampleLine_8u_C1R.

```c
void func_sampleline()
{
    Ipp8u pSrc[5*4] = { 0, 1, 2, 3, 4,
                        5, 6, 7, 8, 9,
                        0, 9, 8, 7, 6,
                        5, 4, 3, 2, 1 };
    IppSize roiSize = {5, 4};
    IppiPoint pt1 = {1, 1};
    IppiPoint pt2 = {2, 3};
    Ipp8u pDst[3];
    int srcStep = 5;
    ippiSampleLine_8u_C1R( pSrc, srcStep, roiSize, pDst, pt1, pt2 );
    printf("Result: %d, %d, %d\n", pDst[0], pDst[1], pDst[2] );  // << this wrong line
    printf("Result: %d, %d, %d\n", pDst[0], pDst[1], pDst[2] );  // this is correct line
}
```

Result: 6, 9, 3

ZigzagFwd8x8

Converts a conventional order to the zigzag order.

Syntax

```c
IppStatus ippiZigzagFwd8x8_16s_C1(const Ipp16s* pSrc, Ipp16s* pDst);```

pt2

Ending point of the line.
Include Files
ippi.h

Domain Dependencies
Headers: ippcore.h, ippvm.h, ipps.h
Libraries: ippcore.lib, ippvm.lib, ipps.lib

Parameters

pSrc  
Pointer to the source data.

pDst  
Pointer to the destination data.

Description
This function rearranges data in an 8x8 block from a conventional order (left-to-right, top-to-bottom) to the zigzag sequence.

Figure Zigzag Sequence specifies the resulting zigzag sequence.

Zigzag Sequence

```
0  1  5  6  14 15 27 28
 2  4  7  13 16 26 29 42
 3  8 12 17 25 30 41 43
 9 11 18 24 31 40 44 53
10 19 23 32 39 45 52 54
20 22 33 38 46 51 55 60
21 34 37 47 50 56 59 61
35 36 48 49 57 58 62 63
```

Return Values

ippStsNoErr  
Indicates no error.

ippStsNullPtrErr  
Indicates an error when any of the specified pointers is NULL.

Example
The code example below shows how to use the ippiZigzagFwd8x8_16s_C1 function.

```c
Ipp16s src[8*8] = {
    0, 1, 5, 7, 9, 2, 4, 1,
    5, 4, 8, 6, 3, 8, 0, 3,
    6, 2, 6, 8, 1, 4, 2, 8,
    4, 3, 2, 9, 3, 0, 6, 6,
    7, 7, 3, 0, 4, 1, 0, 9,
    5, 1, 9, 2, 5, 7, 1, 7,
    0, 3, 5, 0, 7, 5, 9, 8,
    2, 9, 1, 4, 6, 8, 2, 3
};
Ipp16s dst[8*8];
```
ippiZigzagFwd8x8_16s_C1 (src, dst);

Result:
0 1 5 7 9 2 4 1
5 4 8 6 3 8 0 3
6 2 6 8 1 4 2 8
4 3 2 9 3 0 6 6 src //conventional order
7 7 3 0 4 1 0 9
5 1 9 2 5 7 1 7
0 3 5 0 7 5 9 8
2 9 1 4 6 8 2 3

0 1 5 6 4 5 7 8
dst //zigzag order
2 4 7 3 6 6 9 2
3 8 2 7 5 0 1 3
9 1 8 4 1 0 4 3
0 9 3 2 9 5 2 4
0 2 3 8 6 1 5 0
1 4 7 7 0 6 9 1
5 6 8 9 7 8 2 3

ZigzagInv8x8

Converts a zigzag order to the conventional order.

Syntax

IppStatusippiZigzagInv8x8_16s_C1(const Ipp16s* pSrc, Ipp16s* pDst);

Include Files

ippi.h

Domain Dependencies

Headers: ippcore.h, ippvm.h, ipps.h
Libraries: ippcore.lib, ippvm.lib, ipps.lib

Parameters

pSrc Pointer to the source data.
pDst Pointer to the destination data.

Description

This function rearranges data in an 8x8 block from a zigzag sequence to the conventional order (left-to-right, top-to-bottom).

Figure Zigzag Sequence specifies the resulting zigzag sequence.

Return Values

ippStsNoErr Indicates no error.
ippStsNullPtrErr Indicates an error when any of the specified pointers is NULL.
Image Arithmetic and Logical Operations

This chapter describes functions that modify pixel values of an image buffer using arithmetic or logical operations. It also includes functions that perform image compositing based on opacity (alpha-blending).

An additional suffix \texttt{C}, if present in the function name, indicates operation with a constant. Arithmetic functions that operate on integer data perform fixed scaling of the internally computed results. In case of overflow the result value is saturated to the destination data type range.

\textbf{NOTE}
Most arithmetic and logical functions support data with 1-, 3-, or 4-channel pixel values. In the alpha channel case (AC4), the alpha channels are not processed.

\section*{Arithmetic Operations}

Functions described in this section perform arithmetic operations on pixel values. Arithmetic operations include addition, multiplication, subtraction, and division of pixel values of two images as well as similar operations on a single image and a constant. Computation of an absolute value, square, square root, exponential, and natural logarithm of pixels in an image buffer is also supported. Functions of this group perform operations on each pixel in the source buffer(s), and write the results into the destination buffer. Some functions also support processing of images with complex data.

\subsection*{Add}
Adds pixel values of two images.

\textbf{Syntax}

\texttt{IppStatus ippiAdd}\_\texttt{_<mod>}(const Ipp<datatype>* \texttt{pSrc1}, int \texttt{src1Step}, const Ipp<datatype>* \texttt{pSrc2}, int \texttt{src2Step}, Ipp<datatype>* \texttt{pDst}, int \texttt{dstStep}, IppSize \texttt{roiSize}, int \texttt{scaleFactor});

\textbf{Supported values for mod:}

\begin{itemize}
  \item \texttt{8u\_C1RSfs}
  \item \texttt{16u\_C1RSfs}
  \item \texttt{16s\_C1RSfs}
  \item \texttt{8u\_C3RSfs}
  \item \texttt{16u\_C3RSfs}
  \item \texttt{16s\_C3RSfs}
  \item \texttt{8u\_C4RSfs}
  \item \texttt{16u\_C4RSfs}
  \item \texttt{16s\_C4RSfs}
  \item \texttt{8u\_AC4RSfs}
  \item \texttt{16u\_AC4RSfs}
  \item \texttt{16s\_AC4RSfs}
\end{itemize}
Case 2: Not-in-place operation on floating point or complex data

IppStatus ippiAdd_<mod>(const Ipp<datatype>* pSrc1, int src1Step, const Ipp<datatype>* pSrc2, int src2Step, Ipp<datatype>* pDst, int dstStep, IppSize roiSize);

Supported values for mod:

32f_C1R
32f_C3R
32f_C4R

IppStatus ippiAdd_32f_AC4R(const Ipp32f* pSrc1, int src1Step, const Ipp32f* pSrc2, int src2Step, Ipp32f* pDst, int dstStep, IppSize roiSize);

Case 3: In-place operation on integer or complex data

IppStatus ippiAdd_<mod>(const Ipp<datatype>* pSrc, int srcStep, Ipp<datatype>* pSrcDst, int srcDstStep, IppSize roiSize, int scaleFactor);

Supported values for mod:

8u_C1IRSfs 16u_C1IRSfs 16s_C1IRSfs
8u_C3IRSfs 16u_C3IRSfs 16s_C3IRSfs
8u_AC4IRSfs 16u_AC4IRSfs 16s_AC4IRSfs
8u_C4IRSfs 16u_C4IRSfs 16s_C4IRSfs

Case 4: In-place operation on floating point or complex data

IppStatus ippiAdd_<mod>(const Ipp<datatype>* pSrc, int srcStep, Ipp<datatype>* pSrcDst, int srcDstStep, IppSize roiSize);

Supported values for mod:

32f_C1IR
32f_C3IR
32f_AC4IR
32f_C4IR

Case 5: In-place operation using a floating point accumulator image

IppStatus ippiAdd_<mod>(const Ipp<srcDatatype>* pSrc, int srcStep, Ipp32f* pSrcDst, int srcDstStep, IppSize roiSize);

Supported values for mod:

8u32f_C1IR 16u32f_C1IR
Case 6: Masked in-place operation using a floating point accumulator image

IppStatus ippiAdd_<mod>(const Ipp<srcDatatype> * pSrc, int srcStep, const Ipp8u* pMask, int maskStep, Ipp32f* pSrcDst, int srcDstStep, IppSize roiSize);

Supported values for mod:

8u32f_C1IMR 16u32f_C1IMR 32f_C1IMR

Case 7: Not-in-place operation on integer data with platform-aware functions

IppStatus ippiAdd_<mod>(const Ipp<datatype> * pSrc1, IppSizeL src1Step, const Ipp<datatype>* pSrc2, IppSizeL src2Step, Ipp<datatype>* pDst, IppSizeL dstStep, IppSizeL roiSize, int scaleFactor);

Supported values for mod:

8u_C1RSfs_L 16u_C1RSfs_L 16s_C1RSfs_L
8u_C3RSfs_L 16u_C3RSfs_L 16s_C3RSfs_L
8u_C4RSfs_L 16u_C4RSfs_L 16s_C4RSfs_L
8u_AC4RSfs_L 16u_AC4RSfs_L 16s_AC4RSfs_L

Case 8: Not-in-place operation on floating point data with platform-aware functions

IppStatus ippiAdd_<mod>(const Ipp<datatype>* pSrc1, IppSizeL src1Step, const Ipp<datatype>* pSrc2, IppSizeL src2Step, Ipp<datatype>* pDst, IppSizeL dstStep, IppSizeL roiSize);

Supported values for mod:

32f_C1R_L
32f_C3R_L
32f_C4R_L
32f_AC4R_L

Case 9: In-place operation on integer data with platform-aware functions

IppStatus ippiAdd_<mod>(const Ipp<datatype>* pSrc, IppSizeL srcStep, Ipp<datatype>* pSrcDst, IppSizeL srcDstStep, IppSizeL roiSize, int scaleFactor);

Supported values for mod:

8u_C1IRSfs_L 16u_C1IRSfs_L 16s_C1IRSfs_L
8u_C3IRSfs_L 16u_C3IRSfs_L 16s_C3IRSfs_L
8u_C4IRSfs_L 16u_C4IRSfs_L 16s_C4IRSfs_L
8u_AC4IRSfs_L 16u_AC4IRSfs_L 16s_AC4IRSfs_L
Case 10: In-place operation on floating point data with platform-aware functions

IppStatus ippiAdd_<mod>(const Ipp<datatype>* pSrc1, IppSizeL src1Step, Ipp<datatype>* pDst, IppSizeL dstStep, IppiSizeL roiSize);

Supported values for mod:

32f_C1IR_L
32f_C3IR_L
32f_C4IR_L
32f_AC4IR_L

Case 11: Not-in-place operation on integer data with threading layer (TL) functions

IppStatus ippiAdd_<mod>(const Ipp<datatype>* pSrc1, IppSizeL src1Step, const Ipp<datatype>* pSrc2, IppSizeL src2Step, Ipp<datatype>* pDst, IppSizeL dstStep, IppiSizeL roiSize, int scaleFactor);

Supported values for mod:

8u_C1RSfs_LT  16u_C1RSfs_LT  16s_C1RSfs_LT
8u_C3RSfs_LT  16u_C3RSfs_LT  16s_C3RSfs_LT
8u_C4RSfs_LT  16u_C4RSfs_LT  16s_C4RSfs_LT
8u_AC4RSfs_LT  16u_AC4RSfs_LT  16s_AC4RSfs_LT

Case 12: Not-in-place operation on floating point data with TL functions

IppStatus ippiAdd_<mod>(const Ipp<datatype>* pSrc1, IppSizeL src1Step, const Ipp<datatype>* pSrc2, IppSizeL src2Step, Ipp<datatype>* pDst, IppSizeL dstStep, IppiSizeL roiSize);

Supported values for mod:

32f_C1R_LT
32f_C3R_LT
32f_C4R_LT
32f_AC4R_LT

Case 13: In-place operation on integer data with TL functions

IppStatus ippiAdd_<mod>(const Ipp<datatype>* pSrc, IppSizeL srcStep, Ipp<datatype>* pSrcDst, IppSizeL srcDstStep, IppiSizeL roiSize, int scaleFactor);

Supported values for mod:

8u_C1IRSfs_LT  16u_C1IRSfs_LT  16s_C1IRSfs_LT
8u_C3IRSfs_LT  16u_C3IRSfs_LT  16s_C3IRSfs_LT
8u_C4IRSfs_LT  16u_C4IRSfs_LT  16s_C4IRSfs_LT
Case 14: In-place operation on floating point data with TL functions

IppStatus ippiAdd_<mod>(const Ipp<datatype>* pSrc1, IppSizeL src1Step, Ipp<datatype>* pSrc2, IppSizeL src2Step, Ipp<datatype>* pDst, IppSizeL dstStep, IppiSizeL roiSize);

Supported values for <mod>:

32f_C1IR_LT
32f_C3IR_LT
32f_C4IR_LT
32f_AC4IR_LT

Include Files
ippcv.h
ippi.h
ippi_l.h
ippi_tl.h

Domain Dependencies
Flavors declared in ippi.h:
Headers: ippcore.h, ippvm.h, ipps.h
Libraries: ippcore.lib, ippvm.lib, ipps.lib
Flavors declared in ippcv.h:
Headers: ippcore.h, ippvm.h, ipps.h,ippi.h
Libraries: ippcore.lib, ippvm.lib, ipps.lib,ippi.lib
Flavors declared in ipp64x.h:

Parameters

pSrc, pSrc1, pSrc2
srcStep, src1Step, src2Step
pDst
dstStep
pSrc
srcStep
pSrcDst

Pointer to the ROI in the source images.
Distance, in bytes, between the starting points of consecutive lines in the source images.
Pointer to the destination image ROI.
Distance, in bytes, between the starting points of consecutive lines in the destination image.
Pointer to the first source image ROI for the in-place operation.
Distance, in bytes, between the starting points of consecutive lines in the first source image for the in-place operation.
Pointer to the second source and destination image ROI for the in-place operation.
Description

This function operates with ROI.

This function adds corresponding pixel values of two source image buffers and places the results in a
destination buffer. In case of operations on integer data, the resulting values are scaled by scaleFactor. For
complex data, the function processes both real and imaginary parts of pixel values. Some function flavors
add 8u, 8s, 16u, or 32f source image pixel values to a floating point accumulator image in-place. Addition of
pixel values in case of a masked operation is performed only if the respective mask value is non-zero;
otherwise, the accumulator pixel value remains unchanged.

NOTE
For the functions that operate on complex data, step values must be positive. For the
functions that use an accumulator image, step values must be no less than
 roiSize.width*<pixelSize>.

Functions with AC4 descriptor do not process alpha channels.

Function flavors described in Case 5 and Case 6 are declared in the ippcv.h.

Return Values

  ippStsNoErr  Indicates no error. Any other value indicates an error or a
               warning.
  ippStsNullPtrErr  Indicates an error condition when any of the specified pointers is
                    NULL.
  ippStsSizeErr  Indicates an error condition when roiSize has a field with zero
                 or negative value.
  ippStsStepErr  Indicates an error condition in the following cases:
                 • For functions that operate on complex data, if any of the specified
                   step values is zero or negative.
                 • For functions using an accumulator image, if any of the specified
                   step values is less than roiSize.width * <pixelSize>.
  ippStsNotEvenStepErr  Indicates an error condition when one of step values for
                         floating-point images cannot be divided by 4.

Example

The code example below shows how to use the function ippiAdd_8u_C1RSfs.

Ipp8u src[8*4] = {8, 4, 2, 1, 0, 0, 0, 0,
                  8, 4, 2, 1, 0, 0, 0, 0,
                  8, 4, 2, 1, 0, 0, 0, 0,
                  8, 4, 2, 1, 0, 0, 0, 0};
Ipp8u src2[8*4] = {4, 3, 2, 1, 0, 0, 0, 0,
                  4, 3, 2, 1, 0, 0, 0, 0,
                  4, 3, 2, 1, 0, 0, 0, 0,
                  4, 3, 2, 1, 0, 0, 0, 0};
Ipp8u dst[8*4];
IppiSize srcRoi = { 4, 4 }; Int scaleFactor = 1;    // later examples for 2 and -2 values
ippiAdd_8u_C1RSfs (src1, 8, src2, 8, dst, 4, srcRoi, scaleFactor );

Result:

<table>
<thead>
<tr>
<th>src1</th>
</tr>
</thead>
<tbody>
<tr>
<td>8 4 2 1 0 0 0 0</td>
</tr>
<tr>
<td>8 4 2 1 0 0 0 0</td>
</tr>
<tr>
<td>8 4 2 1 0 0 0 0</td>
</tr>
<tr>
<td>8 4 2 1 0 0 0 0</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>src2</th>
</tr>
</thead>
<tbody>
<tr>
<td>4 3 2 1 0 0 0 0</td>
</tr>
<tr>
<td>4 3 2 1 0 0 0 0</td>
</tr>
<tr>
<td>4 3 2 1 0 0 0 0</td>
</tr>
<tr>
<td>4 3 2 1 0 0 0 0</td>
</tr>
</tbody>
</table>

dst >> scaleFactor = 1        scaleFactor = 2    ScaleFactor = -2

<table>
<thead>
<tr>
<th>6 4 2 1</th>
</tr>
</thead>
<tbody>
<tr>
<td>6 4 2 1</td>
</tr>
<tr>
<td>6 4 2 1</td>
</tr>
<tr>
<td>6 4 2 1</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>3 2 1 0</th>
</tr>
</thead>
<tbody>
<tr>
<td>3 2 1 0</td>
</tr>
<tr>
<td>3 2 1 0</td>
</tr>
<tr>
<td>3 2 1 0</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>48 28 16 8</th>
</tr>
</thead>
<tbody>
<tr>
<td>48 28 16 8</td>
</tr>
<tr>
<td>48 28 16 8</td>
</tr>
<tr>
<td>48 28 16 8</td>
</tr>
</tbody>
</table>

See Also
Regions of Interest in Intel IPP

AddC
Add a constant to pixel values of an image.

Syntax

Case 1: Not-in-place operation on one-channel integer or complex data

IppStatus ippiAddC_<mod>(const Ipp<datatype>* pSrc, int srcStep, Ipp<datatype> value,
                         Ipp<datatype>* pDst, int dstStep, IppiSize roiSize, int scaleFactor);

Supported values for <mod>:

<table>
<thead>
<tr>
<th>8u_C1RSfs</th>
<th>16u_C1RSfs</th>
<th>16s_C1RSfs</th>
</tr>
</thead>
</table>

Case 2: Not-in-place operation on multi-channel integer or complex data

IppStatus ippiAddC_<mod>(const Ipp<datatype>* pSrc, int srcStep, const Ipp<datatype> value[3],
                         Ipp<datatype>* pDst, int dstStep, IppiSize roiSize, int scaleFactor);

Supported values for <mod>:

<table>
<thead>
<tr>
<th>8u_C3RSfs</th>
<th>16u_C3RSfs</th>
<th>16s_C3RSfs</th>
</tr>
</thead>
</table>
IppStatus ippiAddC_<mod>(const Ipp<datatype>* pSrc, int srcStep, const Ipp<datatype>* value[4], Ipp<datatype>* pDst, int dstStep, IppiSize roiSize, int scaleFactor);

Supported values for mod:

8u_C4RSfs  16u_C4RSfs  16s_C4RSfs

IppStatus ippiAddC_<mod>(const Ipp<datatype>* pSrc, int srcStep, const Ipp<datatype>* value[3], Ipp<datatype>* pDst, int dstStep, IppiSize roiSize, int scaleFactor);

Supported values for mod:

8u_AC4RSfs  16u_AC4RSfs  16s_AC4RSfs

Case 3: Not-in-place operation on one-channel floating-point or complex data

IppStatus ippiAddC_<mod>(const Ipp<datatype>* pSrc, int srcStep, Ipp<datatype>* value, Ipp<datatype>* pDst, int dstStep, IppiSize roiSize);

Supported values for mod:

32f_C1R

Case 4: Not-in-place operation on multi-channel floating-point or complex data

IppStatus ippiAddC_<mod>(const Ipp<datatype>* pSrc, int srcStep, const Ipp<datatype>* value[3], Ipp<datatype>* pDst, int dstStep, IppiSize roiSize);

Supported values for mod:

32f_C3R

IppStatus ippiAddC_32f_C4R(const Ipp32f* pSrc, int srcStep, const Ipp32f value[4], Ipp32f* pDst, int dstStep, IppiSize roiSize);

IppStatus ippiAddC_<mod>(const Ipp<datatype>* pSrc, int srcStep, const Ipp<datatype>* value[3], Ipp<datatype>* pDst, int dstStep, IppiSize roiSize);

Supported values for mod:

32f_AC4R

Case 5: In-place operation on one-channel integer or complex data

IppStatus ippiAddC_<mod>(Ipp<datatype> value, Ipp<datatype>* pSrcDst, int srcDstStep, IppiSize roiSize, int scaleFactor);

Supported values for mod:

8u_C1IRSfs  16u_C1IRSfs  16s_C1IRSfs
Case 6: In-place operation on multi-channel integer or complex data

IppStatus ippiAddC_<mod>(const Ipp<datatype> value[3], Ipp<datatype>* pSrcDst, int srcDstStep, IppiSize roiSize, int scaleFactor);

Supported values for mod:

<table>
<thead>
<tr>
<th>8u_C3IRSfs</th>
<th>16u_C3IRSfs</th>
<th>16s_C3IRSfs</th>
</tr>
</thead>
<tbody>
<tr>
<td>8u_AC4IRSfs</td>
<td>16u_AC4IRSfs</td>
<td>16s_AC4IRSfs</td>
</tr>
</tbody>
</table>

IppStatus ippiAddC_<mod>(const Ipp<datatype> value[4], Ipp<datatype>* pSrcDst, int srcDstStep, IppiSize roiSize, int scaleFactor);

Supported values for mod:

| 8u_C4IRSfs | 16u_C4IRSfs | 16s_C4IRSfs |

Case 7: In-place operation on one-channel floating-point or complex data

IppStatus ippiAddC_32f_C1IR(Ipp32f value, Ipp32f* pSrcDst, int srcDstStep, IppiSize roiSize);

Case 8: In-place operation on multi-channel floating-point or complex data

IppStatus ippiAddC_<mod>(const Ipp<datatype> value[3], Ipp<datatype>* pSrcDst, int srcDstStep, IppiSize roiSize);

Supported values for mod:

<table>
<thead>
<tr>
<th>32f_C3IR</th>
</tr>
</thead>
<tbody>
<tr>
<td>32f_AC4IR</td>
</tr>
</tbody>
</table>

IppStatus ippiAddC_32f_C4IR(const Ipp32f value[4], Ipp32f* pSrcDst, int srcDstStep, IppiSize roiSize);

Case 9: Not-in-place operation on one-channel integer data with platform-aware functions

IppStatus ippiAddC_<mod>(const Ipp<datatype>* pSrc, IppSizeL srcStep, Ipp<datatype>* pDst, IppSizeL dstStep, IppiSizeL roiSize, int scaleFactor);

Supported values for mod:

| 8u_C1RSfs_L | 16u_C1RSfs_L | 16s_C1RSfs_L |

Case 10: Not-in-place operation on multi-channel integer data with platform-aware functions

IppStatus ippiAddC_<mod>(const Ipp<datatype>* pSrc, IppSizeL srcStep, const Ipp<datatype> value[3], Ipp<datatype>* pDst, IppSizeL dstStep, IppiSizeL roiSize, int scaleFactor);

Supported values for mod:

<table>
<thead>
<tr>
<th>8u_C3RSfs_L</th>
<th>16u_C3RSfs_L</th>
<th>16s_C3RSfs_L</th>
</tr>
</thead>
<tbody>
<tr>
<td>8u_AC4RSfs_L</td>
<td>16u_AC4RSfs_L</td>
<td>16s_AC4RSfs_L</td>
</tr>
</tbody>
</table>
IppStatus ippiAddC_<mod>(const Ipp<datatype>* pSrc, IppSizeL srcStep, const Ipp<datatype>* value[4], Ipp<datatype>* pDst, IppSizeL dstStep, IppiSizeL roiSize, int scaleFactor);

Supported values for mod:

8u_C4RSfs_L   16u_C4RSfs_L   16s_C4RSfs_L

Case 11: Not-in-place operation on one-channel floating point data with platform-aware functions
IppStatus ippiAddC_32f_C1R_L(const Ipp32f* pSrc, IppSizeL srcStep, Ipp32f value, Ipp32f* pDst, IppSizeL dstStep, IpipiSizeL roiSize);

Case 12: Not-in-place operation on multi-channel floating point data with platform-aware functions
IppStatus ippiAddC_<mod>(const Ipp32f* pSrc, IppSizeL srcStep, const Ipp32f value[3], Ipp32f* pDst, IppSizeL dstStep, IpipiSizeL roiSize);

Supported values for mod:

32f_C3R_L
32f_AC4R_L

IppStatus ippiAddC_32f_C4R_L(const Ipp32f* pSrc, IppSizeL srcStep, const Ipp32f value[4], Ipp32f* pDst, IppSizeL dstStep, IpipiSizeL roiSize);

Case 13: In-place operation on one-channel integer data with platform-aware functions
IppStatus ippiAddC_<mod>(Ipp<datatype> value, Ipp<datatype>* pSrcDst, IppSizeL srcDstStep, IpipiSizeL roiSize, int scaleFactor);

Supported for mod:

8u_C1IRSfs_L   16u_C1IRSfs_L   16s_C1IRSfs_L

Case 14: In-place operation on multi-channel integer data with platform-aware functions
IppStatus ippiAddC_<mod>(const Ipp<datatype> value[3], Ipp<datatype>* pSrcDst, IppSizeL srcDstStep, IpipiSizeL roiSize, int scaleFactor);

Supported values for mod:

8u_C3IRSfs_L   16u_C3IRSfs_L   16s_C3IRSfs_L
8u_AC4IRSfs_L   16u_AC4IRSfs_L   16s_AC4IRSfs_L

IppStatus ippiAddC_<mod>(const Ipp<datatype> value[4], Ipp<datatype>* pSrcDst, IppSizeL srcDstStep, IpipiSizeL roiSize, int scaleFactor);

Supported values for mod:

8u_C4IRSfs_L   16u_C4IRSfs_L   16s_C4IRSfs_L

Case 15: In-place operation on one-channel floating point data with platform-aware functions
IppStatus ippiAddC_32f_C1IR_L(Ipp32f value, Ipp32f* pSrcDst, IppSizeL srcDstStep, IpipiSizeL roiSize);
Case 16: In-place operation on multi-channel integer data with platform-aware functions

IppStatus ippiAddC_<mod>(const Ipp32f value[3], Ipp32f* pSrcDst, IppSizeL srcDstStep, IppiSizeL roiSize);

Supported values for mod:

32f_C3IR_L
32f_AC4IR_L

IppStatus ippiAddC_32f_C4IR_L(const Ipp32f value[4], Ipp32f* pSrcDst, IppSizeL srcDstStep, IppiSizeL roiSize);

Case 17: Not-in-place operation on one-channel integer data with threading layer (TL) functions

IppStatus ippiAddC_<mod>(const Ipp<datatype>* pSrc, IppSizeL srcStep, Ipp<datatype> value, Ipp<datatype>* pDst, IppSizeL dstStep, IppiSizeL roiSize, int scaleFactor);

Supported values for mod:

8u_C1RSfs_LT  16u_C1RSfs_LT  16s_C1RSfs_LT

Case 18: Not-in-place operation on multi-channel integer data with TL functions

IppStatus ippiAddC_<mod>(const Ipp<datatype>* pSrc, IppSizeL srcStep, const Ipp<datatype> value[3], Ipp<datatype>* pDst, IppSizeL dstStep, IppiSizeL roiSize, int scaleFactor);

Supported values for mod:

8u_C3RSfs_LT  16u_C3RSfs_LT  16s_C3RSfs_LT
8u_AC4RSfs_LT  16u_AC4RSfs_LT  16s_AC4RSfs_LT

IppStatus ippiAddC_<mod>(const Ipp<datatype>* pSrc, IppSizeL srcStep, const Ipp<datatype> value[4], Ipp<datatype>* pDst, IppSizeL dstStep, IppiSizeL roiSize, int scaleFactor);

Supported values for mod:

8u_C4RSfs_LT  16u_C4RSfs_LT  16s_C4RSfs_LT

Case 19: Not-in-place operation on one-channel floating point data with TL functions

IppStatus ippiAddC_32f_C1R_LT(const Ipp32f* pSrc, IppSizeL srcStep, Ipp32f value, Ipp32f* pDst, IppSizeL dstStep, IppiSizeL roiSize);

Case 20: Not-in-place operation on multi-channel floating point data with TL functions

IppStatus ippiAddC_<mod>(const Ipp32f* pSrc, IppSizeL srcStep, const Ipp32f value[3], Ipp32f* pDst, IppSizeL dstStep, IppiSizeL roiSize);

Supported values for mod:

32f_C3R_LT
32f_AC4R_LT
IppStatus ippiAddC_32f_C4R_LT(const Ipp32f* pSrc, IppSizeL srcStep, const Ipp32f value[4], Ipp32f* pDst, IppSizeL dstStep, IppSizeL roiSize);

**Case 21: In-place operation on one-channel integer data with TL functions**

IppStatus ippiAddC_<mod>(Ipp<datatype> value, Ipp<datatype>* pSrcDst, IppSizeL srcDstStep, IppSizeL roiSize, int scaleFactor);

Supported values for `mod`:

8u_C1IRSfs_LT  16u_C1IRSfs_LT  16s_C1IRSfs_LT

**Case 22: In-place operation on multi-channel integer data with TL functions**

IppStatus ippiAddC_<mod>(const Ipp<datatype> value[3], Ipp<datatype>* pSrcDst, IppSizeL srcDstStep, IppSizeL roiSize, int scaleFactor);

Supported values for `mod`:

8u_C3IRSfs_LT  16u_C3IRSfs_LT  16s_C3IRSfs_LT
8u_AC4IRSfs_LT  16u_AC4IRSfs_LT  16s_AC4IRSfs_LT

IppStatus ippiAddC_<mod>(const Ipp<datatype> value[4], Ipp<datatype>* pSrcDst, IppSizeL srcDstStep, IppSizeL roiSize, int scaleFactor);

Supported values for `mod`:

8u_C4IRSfs_LT  16u_C4IRSfs_LT  16s_C4IRSfs_LT

**Case 23: In-place operation on one-channel floating point data with TL functions**

IppStatus ippiAddC_32f_C1IR_LT(Ipp32f value, Ipp32f* pSrcDst, IppSizeL srcDstStep, IppSizeL roiSize);

**Case 24: In-place operation on multi-channel integer data with TL functions**

IppStatus ippiAddC_<mod>(const Ipp32f value[3], Ipp32f* pSrcDst, IppSizeL srcDstStep, IppSizeL roiSize);

Supported values for `mod`:

32f_C3IR_LT
32f_AC4IR_LT

IppStatus ippiAddC_32f_C4IR_LT(const Ipp32f value[4], Ipp32f* pSrcDst, IppSizeL srcDstStep, IppSizeL roiSize);

**Include Files**

ippi.h

Flavors with the `_LT` suffix: ippi_tl.h

Flavors with the `_L` suffix: ippi_l.h

**Domain Dependencies**

Flavors declared in ippi.h:

Headers: ippcore.h, ippvm.h, ipps.h
Libraries: ippcore.lib, ippvm.lib, ipps.lib

Flavors declared inippi_tl.h:

Parameters

- **pSrc**: Pointer to the source image ROI.
- **srcStep**: Distance in bytes between starts of consecutive lines in the source image.
- **value**: The constant value to add to image pixel values (constant vector in case of multi-channel images).
- **pDst**: Pointer to the destination image ROI.
- **dstStep**: Distance in bytes between starts of consecutive lines in the destination image.
- **pSrcDst**: Pointer to the source and destination image ROI for the in-place operation.
- **srcDstStep**: Distance in bytes between starts of consecutive lines in the source and destination image buffer for the in-place operation.
- **roiSize**: Size of the source and destination ROI in pixels.
- **scaleFactor**: Scale factor (see Integer Result Scaling).

Description

This function operates with ROI (see Regions of Interest in Intel IPP).

This function changes the image intensity by adding `value` to image pixel values. For one-channel images, a positive `value` brightens the image (increases the intensity); a negative value darkens the image (decreases the intensity). For multi-channel images, the components of a constant vector `value` are added to pixel channel values. For complex data, the function processes both real and imaginary parts of pixel values.

**NOTE**
Step values must be positive for functions that operate on complex data.

In case of operations on integer data, the resulting values are scaled by `scaleFactor`.

Note that the functions with AC4 descriptor do not process alpha channels.

Return Values

- ippStsNoErr: Indicates no error. Any other value indicates an error or a warning.
- ippStsNullPtrErr: Indicates an error condition if any of the specified pointers is NULL.
- ippStsSizeErr: Indicates an error condition if `roiSize` has a field with zero or negative value.
- ippStsStepErr: Indicates an error condition if any of the specified buffer step values is zero or negative for functions that operate on complex data.
AddSquare

Adds squared pixel values of a source image to floating-point pixel values of an accumulator image.

Syntax

Case 1: In-place operation

IppStatus ippiAddSquare_<mod>(const Ipp<srcDatatype>* pSrc, int srcStep, Ipp32f* pSrcDst, int srcDstStep, IppiSize roiSize);

Supported values for mod:

- 8u32f_C1IR
- 16u32f_C1IR
- 32f_C1IR

Case 2: Masked in-place operation

IppStatus ippiAddSquare_<mod>(const Ipp<srcDatatype>* pSrc, int srcStep, const Ipp8u* pMask, int maskStep, Ipp32f* pSrcDst, int srcDstStep, IppiSize roiSize);

Supported values for mod:

- 8u32f_C1IMR
- 16u32f_C1IMR
- 32f_C1IMR

Include Files

ippcv.h

Domain Dependencies

Headers: ippcore.h, ippvm.h, ipps.h,ippi.h
Libraries: ippcore.lib, ippvm.lib, ipps.lib,ippi.lib

Parameters

- pSrc: Pointer to the source image ROI.
- srcStep: Distance in bytes between starts of consecutive lines in the source image.
- pMask: Pointer to the mask image.
- maskStep: Distance in bytes between starts of consecutive lines in the mask image.
- pSrcDst: Pointer to the destination (accumulator) image ROI.
- srcDstStep: Distance in bytes between starts of consecutive lines in the accumulator image.
- roiSize: Size of the image ROI in pixels.

Description

This function operates with ROI (see Regions of Interest in Intel IPP).

This function adds squared pixel values of the source image pSrc to floating-point pixel values of the accumulator image pSrcDst as follows:

\[ pSrcDst(x,y) = pSrcDst(x,y) + pSrc(x,y)^2 \]
Addition of the squared pixel values in case of a masked operation is performed only if the respective mask value is nonzero; otherwise, the accumulator pixel value remains unchanged.

**Return Values**

- **ippStsNoErr**: Indicates no error. Any other value indicates an error or a warning.
- **ippStsNullPtrErr**: Indicates an error condition if any of the specified pointers is NULL.
- **ippStsSizeErr**: Indicates an error when `roiSize.width` or `roiSize.height` is negative.
- **ippStsStepErr**: Indicates an error when `srcStep`, `maskStep`, or `srcDstStep` is less than `roiSize.width * <pixelSize>`.
- **ippStsNotEvenStepErr**: Indicates an error condition if one of step values for floating-point images cannot be divided by 4.

**AddProduct**

*Adds product of pixel values of two source images to floating-point pixel values of an accumulator image.*

**Syntax**

**Case 1: In-place operation**

```c
IppStatus ippiAddProduct_<mod>(const Ipp<srcDatatype>* pSrc1, int src1Step, const Ipp<srcDatatype>* pSrc2, int src2Step, Ipp32f* pSrcDst, int srcDstStep, IppiSize roiSize);
```

Supported values for `mod`:

- `8u32f_C1IR`
- `16u32f_C1IR`
- `32f_C1IR`

**Case 2: Masked in-place operation**

```c
IppStatus ippiAddProduct_<mod>(const Ipp<srcDatatype>* pSrc1, int src1Step, const Ipp<srcDatatype>* pSrc2, int src2Step, const Ipp8u* pMask, int maskStep, Ipp32f* pSrcDst, int srcDstStep, IppiSize roiSize);
```

Supported values for `mod`:

- `8u32f_C1IMR`
- `16u32f_C1IMR`
- `32f_C1IMR`

**Include Files**

- `ippcv.h`

**Domain Dependencies**

- **Headers**: `ippcore.h`, `ippvm.h`, `ipps.h`, `ippi.h`
- **Libraries**: `ippcore.lib`, `ippvm.lib`, `ipps.lib`, `ippi.lib`

**Parameters**

- `pSrc1, pSrc2`: Pointers to the source images ROI.
**src1Step, src2Step**
Distances in bytes between starts of consecutive lines in the source images.

**pMask**
Pointer to the mask image.

**maskStep**
Distance in bytes between starts of consecutive lines in the mask image.

**pSrcDst**
Pointer to the destination (accumulator) image ROI.

**srcDstStep**
Distance in bytes between starts of consecutive lines in the accumulator image.

**roiSize**
Size of the image ROI in pixels.

**Description**
This function operates with ROI (see Regions of Interest in Intel IPP).

This function adds the product of pixel values of two source images pSrc1 and pSrc2 to floating-point pixel values of the accumulator image pSrcDst as given by:

\[ p_{SrcDst}(x,y) = p_{SrcDst}(x,y) + p_{Src1}(x,y) \cdot p_{Src2}(x,y) \]

The products of pixel values in case of a masked operation are added only if the respective mask value is nonzero; otherwise, the accumulator pixel value remains unchanged.

**Return Values**
- **ippStsNoErr**: Indicates no error.
- **ippStsNullPtrErr**: Indicates an error when any of the specified pointers is null.
- **ippStsSizeErr**: Indicates an error when roiSize.width or roiSize.height is negative.
- **ippStsStepErr**: Indicates an error if src1Step, src2Step, maskStep, or srcDstStep is less than roiSize.width * <pixelSize>.
- **ippStsNotEvenStepErr**: Indicates an error condition if one of step values for floating-point images cannot be divided by 4.

**AddWeighted**
*Adds weighted pixel values of a source image to floating-point pixel values of an accumulator image.*

**Syntax**

**Case 1: In-place operation**

```c
IppStatus ippiAddWeighted_<mod>(const Ipp<srcDatatype>* pSrc, int srcStep, Ipp32f* pSrcDst, int srcDstStep, IppiSize roiSize, Ipp32f alpha);
```

Supported values for mod:

- 8u32f_C1IR
- 16u32f_C1IR
- 32f_C1IR
Case 2: Masked in-place operation

IppStatus ippiAddWeighted_<mod>(const Ipp<srcDatatype>* pSrc, int srcStep, const Ipp8u* pMask, int maskStep, Ipp32f* pSrcDst, int srcDstStep, IppiSize roiSize, Ipp32f alpha);

Supported values for mod:

8u32f_C1IMR 16u32f_C1IMR 32f_C1IMR

Case 3: Not-in-place operation

IppStatus ippiAddWeighted_32f_C1R(const Ipp32f* pSrc1, int src1Step, const Ipp32f* pSrc2, int src2Step, Ipp32f* pDst, int dstStep, IppiSize roiSize, Ipp32f alpha);

Include Files

ippcv.h

Domain Dependencies

Headers: ippcore.h, ippvm.h, ipps.h,ippi.h
Libraries: ippcore.lib, ippvm.lib, ipps.lib,ippi.lib

Parameters

pSrc  
Pointer to the source image ROI for the in-place operation.

pSrc1, pSrc2  
Pointers to the ROI in the source images.

srcStep  
Distance in bytes between starts of consecutive lines in the source image for the in-place operation.

dstStep, src2Step  
Distance in bytes between starts of consecutive lines in the source images.

pMask  
Pointer to the mask image.

maskStep  
Distance in bytes between starts of consecutive lines in the mask image.

pSrcDst  
Pointer to the destination (accumulator) image ROI for the in-place operation.

srcDstStep  
Distance in bytes between starts of consecutive lines in the accumulator image.

pDst  
Pointer to the destination image ROI.

dstStep  
Distance in bytes between starts of consecutive lines in the destination image.

roiSize  
Size of the image ROI in pixels.

alpha  
Weight a of the source image.

Description

This function operates with ROI (see Regions of Interest in Intel IPP).

This function adds pixel values of the source image pSrc1 multiplied by a weight factor alpha to pixel values of the image pSrc2 multiplied by (1-alpha) and stores result in the pDst as follows:

\[
pDst(x,y) = pSrc1(x,y) \times \alpha + pSrc2(x,y) \times (1 - \alpha).
\]
The in-place flavors of the function adds pixel values of the source image pSrc multiplied by a weight factor alpha to floating-point pixel values of the accumulator image pSrcDst multiplied by (1-alpha) as follows:

\[ p\text{SrcDst}(x,y) = p\text{SrcDst}(x,y) \times (1-\alpha) + p\text{Src}(x,y) \times \alpha \]

Addition of the weighted pixel values in case of a masked operation is performed only if the respective mask value is nonzero; otherwise, the accumulator pixel value remains unchanged.

**Return Values**

- ippStsNoErr: Indicates no error.
- ippStsNullPtrErr: Indicates an error when any of the specified pointers is NULL.
- ippStsSizeErr: Indicates an error when roiSize.width or roiSize.height is equal to 0 or negative.
- ippStsStepErr: Indicates an error when one of the step values is equal to zero, or is less than roiSize.width * <pixelSize>.
- ippStsNotEvenStepErr: Indicates an error when one of step values for floating-point images cannot be divided by 4.

**Mul**

Multiplies pixel values of two images.

**Case 1: Not-in-place operation on integer or complex data**

IppStatusippiMul_\text{<mod>}(\text{const Ipp<datatype>* pSrc1}, \text{int src1Step}, \text{const Ipp<datatype>* pSrc2}, \text{int src2Step}, \text{Ipp<datatype>* pDst}, \text{int dstStep}, \text{IppiSize roiSize}, \text{int scaleFactor});

Supported values for mod:

- 8u_\text{C1RSfs}
- 8u_\text{C3RSfs}
- 8u_\text{AC4RSfs}
- 8u_\text{C4RSfs}
- 16u_\text{C1RSfs}
- 16u_\text{C3RSfs}
- 16u_\text{AC4RSfs}
- 16u_\text{C4RSfs}
- 16s_\text{C1RSfs}
- 16s_\text{C3RSfs}
- 16s_\text{C4RSfs}

**Case 2: Not-in-place operation on floating-point or complex data**

IppStatusippiMul_\text{<mod>}(\text{const Ipp<datatype>* pSrc1}, \text{int src1Step}, \text{const Ipp<datatype>* pSrc2}, \text{int src2Step}, \text{Ipp<datatype>* pDst}, \text{int dstStep}, \text{IppiSize roiSize});

Supported values for mod:

- 32f_\text{C1R}
- 32f_\text{AC4R}
- 32f_\text{C3R}
- 32f_\text{C4R}
**Case 3: In-place operation on integer or complex data**

IppStatus ippiMul_<mod>(const Ipp<datatype>* pSrc, int srcStep, Ipp<datatype>* pSrcDst, int srcDstStep, IppiSize roiSize, int scaleFactor);

Supported values for mod:

- 8u_C1IRSfs
- 8u_C3IRSfs
- 8u_AC4IRSfs
- 8u_C4IRSfs
- 16u_C1IRSfs
- 16u_C3IRSfs
- 16u_AC4IRSfs
- 16u_C4IRSfs
- 16s_C1IRSfs
- 16s_C3IRSfs
- 16s_AC4IRSfs
- 16s_C4IRSfs

**Case 4: In-place operation on floating-point or complex data**

IppStatus ippiMul_<mod>(const Ipp<datatype>* pSrc, int srcStep, Ipp<datatype>* pSrcDst, int srcDstStep, IppiSize roiSize);

Supported values for mod:

- 32f_C1IR
- 32f_C3IR
- 32f_AC4IR
- 32f_C4IR

**Case 5: Not-in-place operation on integer data with platform-aware functions**

IppStatus ippiMul_<mod>(const Ipp<datatype>* pSrc1, IppSizeL src1Step, const Ipp<datatype>* pSrc2, IppSizeL src2Step, Ipp<datatype>* pDst, IppSizeL dstStep, IppiSizeL roiSize, int scaleFactor);

Supported values for mod:

- 8u_C1RSfs_L
- 8u_C3RSfs_L
- 8u_AC4RSfs_L
- 8u_C4RSfs_L
- 16u_C1RSfs_L
- 16u_C3RSfs_L
- 16u_AC4RSfs_L
- 16u_C4RSfs_L
- 16s_C1RSfs_L
- 16s_C3RSfs_L
- 16s_AC4RSfs_L
- 16s_C4RSfs_L

**Case 6: Not-in-place operation on floating-point data with platform-aware functions**

IppStatus ippiMul_<mod>(const Ipp<datatype>* pSrc1, IppSizeL src1Step, const Ipp<datatype>* pSrc2, IppSizeL src2Step, Ipp<datatype>* pDst, IppSizeL dstStep, IppiSizeL roiSize);

Supported values for mod:

- 32f_C1R_L
- 32f_C3R_L
- 32f_C4R_L
Case 7: In-place operation on integer data with platform-aware functions

IppStatus ippiMul_<mod>(const Ipp<datatype>* pSrc, IppSizeL srcStep, Ipp<datatype>* pSrcDst, IppSizeL srcDstStep, IppiSizeL roiSize, int scaleFactor);

Supported values for mod:

8u_C1IRSfs_L  16u_C1IRSfs_L  16s_C1IRSfs_L
8u_C3IRSfs_L  16u_C3IRSfs_L  16s_C3IRSfs_L
8u_AC4IRSfs_L  16u_AC4IRSfs_L  16s_AC4IRSfs_L
8u_C4IRSfs_L  16u_C4IRSfs_L  16s_C4IRSfs_L

Case 8: In-place operation on floating-point data with platform-aware functions

IppStatus ippiMul_<mod>(const Ipp<datatype>* pSrc, IppSizeL srcStep, Ipp<datatype>* pSrcDst, IppSizeL srcDstStep, IppiSizeL roiSize);

Supported values for mod:

32f_C1IR_L
32f_C3IR_L
32f_AC4IR_L
32f_C4IR_L

Case 9: Not-in-place operation on integer data with threading layer (TL) functions

IppStatus ippiMul_<mod>(const Ipp<datatype>* pSrc1, IppSizeL src1Step, const Ipp<datatype>* pSrc2, IppSizeL src2Step, Ipp<datatype>* pDst, IppSizeL dstStep, IppiSizeL roiSize, int scaleFactor);

Supported values for mod:

8u_C1RSfs_LT  16u_C1RSfs_LT  16s_C1RSfs_LT
8u_C3RSfs_LT  16u_C3RSfs_LT  16s_C3RSfs_LT
8u_AC4RSfs_LT  16u_AC4RSfs_LT  16s_AC4RSfs_LT
8u_C4RSfs_LT  16u_C4RSfs_LT  16s_C4RSfs_LT

Case 10: Not-in-place operation on floating-point data with TL functions

IppStatus ippiMul_<mod>(const Ipp<datatype>* pSrc1, IppSizeL src1Step, const Ipp<datatype>* pSrc2, IppSizeL src2Step, Ipp<datatype>* pDst, IppSizeL dstStep, IppiSizeL roiSize);

Supported values for mod:

32f_C1R_LT
32f_C3R_LT
Case 11: In-place operation on integer data with TL functions

IppStatus ippiMul_<mod>(const Ipp<datatype>* pSrc, IppSizeL srcStep, Ipp<datatype>* pSrcDst, IppSizeL srcDstStep, IppSizeL roiSize, int scaleFactor);

Supported values for mod:

- 8u_C1IRSfs_LT
- 16u_C1IRSfs_LT
- 16s_C1IRSfs_LT
- 8u_C3IRSfs_LT
- 16u_C3IRSfs_LT
- 16s_C3IRSfs_LT
- 8u_AC4IRSfs_LT
- 16u_AC4IRSfs_LT
- 16s_AC4IRSfs_LT
- 8u_C4IRSfs_LT
- 16u_C4IRSfs_LT
- 16s_C4IRSfs_LT

Case 12: In-place operation on floating-point data with TL functions

IppStatus ippiMul_<mod>(const Ipp<datatype>* pSrc, IppSizeL srcStep, Ipp<datatype>* pSrcDst, IppSizeL srcDstStep, IppSizeL roiSize);

Supported values for mod:

- 32f_C1IR_LT
- 32f_C3IR_LT
- 32f_AC4IR_LT
- 32f_C4IR_LT

Include Files

ippi.h

Flavors with the _LT suffix: ippi_tl.h
Flavors with the _L suffix: ippi_l.h

Domain Dependencies

Flavors declared in ippi.h:

- Headers: ippcore.h, ippvm.h, ipps.h
- Libraries: ippcore.lib, ippvm.lib, ipps.lib

Flavors declared in ippi_tl.h:


Parameters

- pSrc, pSrc1, pSrc2
- srcStep, src1Step, src2Step
- pDst

Pointers to the source images ROI.
Distances in bytes between starts of consecutive lines in the source images.
Pointer to the destination image ROI.
**dstStep**
Distance in bytes between starts of consecutive lines in the destination image.

**pSrcDst**
Pointer to the source and destination image ROI for the in-place operation.

**srcDstStep**
Distance in bytes between starts of consecutive lines in the source and destination image for the in-place operation.

**roiSize**
Size of the source and destination ROI in pixels.

**scaleFactor**
Scale factor (see Integer Result Scaling).

**Description**
This function operates with ROI (see Regions of Interest in Intel IPP). This function multiplies corresponding pixel values of two source image buffers and places the results in a destination buffer. In case of operations on integer data, the resulting values are scaled by `scaleFactor`. For complex data, the function processes both real and imaginary parts of pixel values.

**NOTE**
Step values must be positive for functions that operate on complex data.

Note that the functions with `AC4` descriptor do not process alpha channels.

**Return Values**
- `ippStsNoErr` Indicates no error. Any other value indicates an error or a warning.
- `ippStsNullPtrErr` Indicates an error condition if any of the pointers is `NULL`.
- `ippStsSizeErr` Indicates an error condition if `roiSize` has a field with zero or negative value.
- `ippStsStepErr` Indicates an error condition if any of the specified buffer step values is zero or negative for functions that operate on complex data.

**MulC**
*Multiplies pixel values of an image by a constant.*

**Syntax**
**Case 1: Not-in-place operation on one-channel integer or complex data**

```c
IppStatus ippiMulC_<mod>(const Ipp<datatype>* pSrc, int srcStep, Ipp<datatype> value, Ipp<datatype>* pDst, int dstStep, IppiSize roiSize, int scaleFactor);
```

Supported values for `mod`:
- `8u_C1RSfs`
- `16u_C1RSfs`
- `16s_C1RSfs`
Case 2: Not-in-place operation on multi-channel integer or complex data

IppStatus ippiMulC_<mod>(const Ipp<datatype>* pSrc, int srcStep, const Ipp<datatype>* pDst, int dstStep, IppiSize roiSize, int scaleFactor);

**Supported values for mod:**

8u_C3RSfs    16u_C3RSfs    16s_C3RSfs

IppStatus ippiMulC_<mod>(const Ipp<datatype>* pSrc, int srcStep, const Ipp<datatype>* pDst, int dstStep, IppiSize roiSize, int scaleFactor);

**Supported values for mod:**

8u_C4RSfs    16u_C4RSfs    16s_C4RSfs

IppStatus ippiMulC_<mod>(const Ipp<datatype>* pSrc, int srcStep, const Ipp<datatype>* pDst, int dstStep, IppiSize roiSize, int scaleFactor);

**Supported values for mod:**

8u_AC4RSfs    16u_AC4RSfs    16s_AC4RSfs

Case 3: Not-in-place operation on one-channel floating-point or complex data

IppStatus ippiMulC_32f_C1R(const Ipp<datatype>* pSrc, int srcStep, Ipp<datatype>* pDst, int dstStep, IppiSize roiSize);

Case 4: Not-in-place operation on multi-channel floating-point or complex data

IppStatus ippiMulC_32f_C3R(const Ipp<datatype>* pSrc, int srcStep, const Ipp<datatype>* pDst, int dstStep, IppiSize roiSize);

IppStatus ippiMulC_32f_C4R(const Ipp32f* pSrc, int srcStep, const Ipp32f* pDst, int dstStep, IppiSize roiSize);

IppStatus ippiMulC_<mod>(const Ipp<datatype>* pSrc, int srcStep, const Ipp<datatype>* pDst, int dstStep, IppiSize roiSize);

**Supported values for mod:**

32f_AC4R

IppStatus ippiMulC_32f_C4R(const Ipp32f* pSrc, int srcStep, const Ipp32f value[4], Ipp32f* pDst, int dstStep, IppiSize roiSize);

Case 5: In-place operation on one-channel integer or complex data

IppStatus ippiMulC_<mod>(Ipp<datatype>* value, Ipp<datatype>* pSrcDst, int srcDstStep, IppiSize roiSize, int scaleFactor);

**Supported values for mod:**

8u_C1IRSfs    16u_C1IRSfs    16s_C1IRSfs
Case 6: In-place operation on multi-channel integer or complex data

IppStatus ippiMulC_<mod>(const Ipp<datatype> value[3], Ipp<datatype>* pSrcDst, int srcDstStep, IppiSize roiSize, int scaleFactor);

Supported values for mod:

- 8u_C3IRSfs
- 16u_C3IRSfs
- 16s_C3IRSfs
- 8u_AC4IRSfs
- 16u_AC4IRSfs
- 16s_AC4IRSfs

IppStatus ippiMulC_<mod>(const Ipp<datatype> value[4], Ipp<datatype>* pSrcDst, int srcDstStep, IppiSize roiSize, int scaleFactor);

Supported values for mod:

- 8u_C4IRSfs
- 16u_C4IRSfs
- 16s_C4IRSfs

Case 7: In-place operation on one-channel floating-point or complex data

IppStatus ippiMulC_32f_C1IR(Ipp32f value, Ipp32f* pSrcDst, int srcDstStep, IppiSize roiSize);

Case 8: In-place operation on multi-channel floating-point or complex data

IppStatus ippiMulC_<mod>(const Ipp<datatype> value[3], Ipp<datatype>* pSrcDst, int srcDstStep, IppiSize roiSize);

Supported values for mod:

- 32f_C3IR
- 32f_AC4IR

IppStatus ippiMulC_32f_C4IR(const Ipp32f value[4], Ipp32f* pSrcDst, int srcDstStep, IppiSize roiSize);

Case 9: Not-in-place operation on one-channel integer data with platform-aware functions

IppStatus ippiMulC_<mod>(const Ipp<datatype>* pSrc, IppSizeL srcStep, Ipp<datatype>* pDst, IppSizeL dstStep, IppiSizeL roiSize, int scaleFactor);

Supported values for mod:

- 8u_C1RSfs_L
- 16u_C1RSfs_L
- 16s_C1RSfs_L

Case 10: Not-in-place operation on multi-channel integer data with platform-aware functions

IppStatus ippiMulC_<mod>(const Ipp<datatype>* pSrc, IppSizeL srcStep, const Ipp<datatype> value[3], Ipp<datatype>* pDst, IppSizeL dstStep, IppiSizeL roiSize, int scaleFactor);

Supported values for mod:

- 8u_C3RSfs_L
- 16u_C3RSfs_L
- 16s_C3RSfs_L
- 8u_AC4RSfs_L
- 16u_AC4RSfs_L
- 16s_AC4RSfs_L
IppStatus ippiMulC_<mod>(const Ipp<datatype>* pSrc, IppSizeL srcStep, const Ipp<datatype>* pDst, IppSizeL dstStep, IppiSizeL roiSize, int scaleFactor);

Supported values for mod:

8u_C4RSfs_L   16u_C4RSfs_L   16s_C4RSfs_L

Case 11: Not-in-place operation on one-channel floating point data with platform-aware functions
IppStatus ippiMulC_32f_C1R_L(const Ipp32f* pSrc, IppSizeL srcStep, Ipp32f value, Ipp32f* pDst, IppSizeL dstStep, IppiSizeL roiSize);

Case 12: Not-in-place operation on multi-channel floating point data with platform-aware functions
IppStatus ippiMulC_<mod>(const Ipp32f* pSrc, IppSizeL srcStep, const Ipp32f value[3], Ipp32f* pDst, IppSizeL dstStep, IppiSizeL roiSize);

Supported values for mod:

32f_C3R_L
32f_AC4R_L

IppStatus ippiMulC_32f_C4R_L(const Ipp32f* pSrc, IppSizeL srcStep, const Ipp32f value[4], Ipp32f* pDst, IppSizeL dstStep, IppiSizeL roiSize);

Case 13: In-place operation on one-channel integer data with platform-aware functions
IppStatus ippiMulC_<mod>(Ipp<datatype> value, Ipp<datatype>* pSrcDst, IppSizeL srcDstStep, IppiSizeL roiSize, int scaleFactor);

Supported values for mod:

8u_C1IRSfs_L   16u_C1IRSfs_L   16s_C1IRSfs_L

Case 14: In-place operation on multi-channel integer data with platform-aware functions
IppStatus ippiMulC_<mod>(const Ipp<datatype> value[3], Ipp<datatype>* pSrcDst, IppSizeL srcDstStep, IppiSizeL roiSize, int scaleFactor);

Supported values for mod:

8u_C3IRSfs_L   16u_C3IRSfs_L   16s_C3IRSfs_L
8u_AC4IRSfs_L   16u_AC4IRSfs_L   16s_AC4IRSfs_L

IppStatus ippiMulC_<mod>(const Ipp<datatype> value[4], Ipp<datatype>* pSrcDst, IppSizeL srcDstStep, IppiSizeL roiSize, int scaleFactor);

Supported values for mod:

8u_C4IRSfs_L   16u_C4IRSfs_L   16s_C4IRSfs_L

Case 15: In-place operation on one-channel floating point data with platform-aware functions
IppStatus ippiMulC_32f_C1IR_L(Ipp32f value, Ipp32f* pSrcDst, IppSizeL srcDstStep, IppiSizeL roiSize);
Case 16: In-place operation on multi-channel integer data with platform-aware functions

IppStatus ippiMulC_<mod>(const Ipp32f value[3], Ipp32f* pSrcDst, IppSizeL srcDstStep, IppiSizeL roiSize);

Supported values for mod:

32f_C3IR_L
32f_AC4IR_L

IppStatus ippiMulC_32f_C4IR_L(const Ipp32f value[4], Ipp32f* pSrcDst, IppSizeL srcDstStep, IppiSizeL roiSize);

Case 17: Not-in-place operation on one-channel integer data with threading layer (TL) functions

IppStatus ippiMulC_<mod>(const Ipp<datatype>* pSrc, IppSizeL srcStep, Ipp<datatype> value, Ipp<datatype>* pDst, IppSizeL dstStep, IppiSizeL roiSize, int scaleFactor);

Supported values for mod:

8u_C1RSfs_LT  16u_C1RSfs_LT  16s_C1RSfs_LT

Case 18: Not-in-place operation on multi-channel integer data with TL functions

IppStatus ippiMulC_<mod>(const Ipp<datatype>* pSrc, IppSizeL srcStep, const Ipp<datatype> value[3], Ipp<datatype>* pDst, IppSizeL dstStep, IppiSizeL roiSize, int scaleFactor);

Supported values for mod:

8u_C3RSfs_LT  16u_C3RSfs_LT  16s_C3RSfs_LT
8u_AC4RSfs_LT  16u_AC4RSfs_LT  16s_AC4RSfs_LT

IppStatus ippiMulC_<mod>(const Ipp<datatype>* pSrc, IppSizeL srcStep, const Ipp<datatype> value[4], Ipp<datatype>* pDst, IppSizeL dstStep, IppiSizeL roiSize, int scaleFactor);

Supported values for mod:

8u_C4RSfs_LT  16u_C4RSfs_LT  16s_C4RSfs_LT

Case 19: Not-in-place operation on one-channel floating point data with TL functions

IppStatus ippiMulC_32f_C1R_LT(const Ipp32f* pSrc, IppSizeL srcStep, Ipp32f value, Ipp32f* pDst, IppSizeL dstStep, IppiSizeL roiSize, int scaleFactor);

Case 20: Not-in-place operation on multi-channel floating point data with TL functions

IppStatus ippiMulC_<mod>(const Ipp32f* pSrc, IppSizeL srcStep, const Ipp32f value[3], Ipp32f* pDst, IppSizeL dstStep, IppiSizeL roiSize);

Supported values for mod:

32f_C3R_LT
32f_AC4R_LT
IppStatus ippiMulC_32f_C4R_LT(const Ipp32f* pSrc, IppSizeL srcStep, const Ipp32f value[4], Ipp32f* pDst, IppSizeL dstStep, IppiSizeL roiSize);

Case 21: In-place operation on one-channel integer data with TL functions
IppStatus ippiMulC_<mod>(const Ipp<datatype> value, Ipp<datatype>* pSrcDst, IppSizeL srcDstStep, IppiSizeL roiSize, int scaleFactor);

Supported values for mod:

8u_C1IRSfs_LT    16u_C1IRSfs_LT    16s_C1IRSfs_LT

Case 22: In-place operation on multi-channel integer data with TL functions
IppStatus ippiMulC_<mod>(const Ipp<datatype> value[3], Ipp<datatype>* pSrcDst, IppSizeL srcDstStep, IppiSizeL roiSize, int scaleFactor);

Supported values for mod:

8u_C3IRSfs_LT    16u_C3IRSfs_LT    16s_C3IRSfs_LT
8u_AC4IRSfs_LT    16u_AC4IRSfs_LT    16s_AC4IRSfs_LT

IppStatus ippiMulC_<mod>(const Ipp<datatype> value[4], Ipp<datatype>* pSrcDst, IppSizeL srcDstStep, IppiSizeL roiSize, int scaleFactor);

Supported values for mod:

8u_C4IRSfs_LT    16u_C4IRSfs_LT    16s_C4IRSfs_LT

Case 23: In-place operation on one-channel floating point data with TL functions
IppStatus ippiMulC_32f_C1IR_LT(Ipp32f value, Ipp32f* pSrcDst, IppSizeL srcDstStep, IppiSizeL roiSize);

Case 24: In-place operation on multi-channel integer data with TL functions
IppStatus ippiMulC_<mod>(const Ipp32f value[3], Ipp32f* pSrcDst, IppSizeL srcDstStep, IppiSizeL roiSize);

Supported values for mod:

32f_C3IR_LT
32f_AC4IR_LT

IppStatus ippiMulC_32f_C4IR_LT(const Ipp32f value[4], Ipp32f* pSrcDst, IppSizeL srcDstStep, IppiSizeL roiSize);

Include Files
ippi.h

Flavors with the _LT suffix: ippi_tl.h
Flavors with the _L suffix: ippi_l.h

Domain Dependencies
Flavors declared in ippi.h:
Headers: ippcore.h, ippvm.h, ipps.h
Libraries: ippcore.lib, ippvm.lib, ipps.lib

Flavors declared inippi_tl.h:

Parameters

**pSrc**
- Pointer to the source image ROI.

**srcStep**
- Distance, in bytes, between the starting points of consecutive lines in the source image.

**value**
- The constant value to add to image pixel values (constant vector in case of multi-channel images).

**pDst**
- Pointer to the destination image ROI.

**dstStep**
- Distance, in bytes, between the starting points of consecutive lines in the destination image.

**pSrcDst**
- Pointer to the source and destination image ROI for the in-place operation.

**srcDstStep**
- Distance, in bytes, between the starting points of consecutive lines in the source and destination image for the in-place operation.

**roiSize**
- Size of the source and destination ROI in pixels.

**scaleFactor**
- Scale factor (see **Integer Result Scaling**).

Description

This function operates with ROI (see Regions of Interest in Intel IPP).

This function multiplies pixel values of an image by a constant **value**. For multi-channel images, pixel channel values are multiplied by the components of a constant vector **value**. For complex data, the function processes both real and imaginary parts of pixel values.

**NOTE**
Step values must be positive for functions that operate on complex data.

In case of operations on integer data, the resulting values are scaled by **scaleFactor**.

Note that the functions with **AC4** descriptor do not process alpha channels.

Return Values

**ippStsNoErr**
- Indicates no error. Any other value indicates an error or a warning.

**ippStsNullPtrErr**
- Indicates an error condition if any of the specified pointers is NULL.

**ippStsSizeErr**
- Indicates an error condition if **roiSize** has a field with zero or negative value.

**ippStsStepErr**
- Indicates an error condition if any of the specified buffer step values is zero or negative for functions that operate on complex data.
MulC64f

Multiplies pixel values of an image by a constant array.

**Syntax**

**Not-in-place operations**

IppStatus ippiMulC64f_8u_C1R(const Ipp8u* pSrc, int srcStep, const Ipp64f value[1], Ipp8u* pDst, int dstStep, IppiSize roiSize, IppHintAlgorithm hint, IppRoundMode rndMode);

IppStatus ippiMulC64f_8u_C3R(const Ipp8u* pSrc, int srcStep, const Ipp64f value[3], Ipp8u* pDst, int dstStep, IppiSize roiSize, IppHintAlgorithm hint, IppRoundMode rndMode);

IppStatus ippiMulC64f_8u_C4R(const Ipp8u* pSrc, int srcStep, const Ipp64f value[4], Ipp8u* pDst, int dstStep, IppiSize roiSize, IppHintAlgorithm hint, IppRoundMode rndMode);

IppStatus ippiMulC64f_16u_C1R(const Ipp16u* pSrc, int srcStep, const Ipp64f value[1], Ipp16u* pDst, int dstStep, IppiSize roiSize, IppHintAlgorithm hint, IppRoundMode rndMode);

IppStatus ippiMulC64f_16u_C3R(const Ipp16u* pSrc, int srcStep, const Ipp64f value[3], Ipp16u* pDst, int dstStep, IppiSize roiSize, IppHintAlgorithm hint, IppRoundMode rndMode);

IppStatus ippiMulC64f_16u_C4R(const Ipp16u* pSrc, int srcStep, const Ipp64f value[4], Ipp16u* pDst, int dstStep, IppiSize roiSize, IppHintAlgorithm hint, IppRoundMode rndMode);

IppStatus ippiMulC64f_16s_C1R(const Ipp16s* pSrc, int srcStep, const Ipp64f value[1], Ipp16s* pDst, int dstStep, IppiSize roiSize, IppHintAlgorithm hint, IppRoundMode rndMode);

IppStatus ippiMulC64f_16s_C3R(const Ipp16s* pSrc, int srcStep, const Ipp64f value[3], Ipp16s* pDst, int dstStep, IppiSize roiSize, IppHintAlgorithm hint, IppRoundMode rndMode);

IppStatus ippiMulC64f_16s_C4R(const Ipp16s* pSrc, int srcStep, const Ipp64f value[4], Ipp16s* pDst, int dstStep, IppiSize roiSize, IppHintAlgorithm hint, IppRoundMode rndMode);

IppStatus ippiMulC64f_32f_C1R(const Ipp32f* pSrc, int srcStep, const Ipp64f value[1], Ipp32f* pDst, int dstStep, IppiSize roiSize, IppHintAlgorithm hint, IppRoundMode rndMode);

IppStatus ippiMulC64f_32f_C3R(const Ipp32f* pSrc, int srcStep, const Ipp64f value[3], Ipp32f* pDst, int dstStep, IppiSize roiSize, IppHintAlgorithm hint, IppRoundMode rndMode);
IppStatus ippiMulC64f_32f_C4R(const Ipp32f* pSrc, int srcStep, const Ipp64f value[4], Ipp32f* pDst, int dstStep, IppiSize roiSize, IppHintAlgorithm hint, IppRoundMode rndMode);

**In-place operations**

IppStatus ippiMulC64f_8u_C1IR(Ipp8u* pSrcDst, int srcDstStep, const Ipp64f value[1], IppiSize roiSize, IppHintAlgorithm hint, IppRoundMode rndMode);

IppStatus ippiMulC64f_8u_C3IR(Ipp8u* pSrcDst, int srcDstStep, const Ipp64f value[3], IppiSize roiSize, IppHintAlgorithm hint, IppRoundMode rndMode);

IppStatus ippiMulC64f_8u_C4IR(Ipp8u* pSrcDst, int srcDstStep, const Ipp64f value[4], IppiSize roiSize, IppHintAlgorithm hint, IppRoundMode rndMode);

IppStatus ippiMulC64f_16u_C1IR(Ipp16u* pSrcDst, int srcDstStep, const Ipp64f value[1], IppiSize roiSize, IppHintAlgorithm hint, IppRoundMode rndMode);

IppStatus ippiMulC64f_16u_C3IR(Ipp16u* pSrcDst, int srcDstStep, const Ipp64f value[3], IppiSize roiSize, IppHintAlgorithm hint, IppRoundMode rndMode);

IppStatus ippiMulC64f_16u_C4IR(Ipp16u* pSrcDst, int srcDstStep, const Ipp64f value[4], IppiSize roiSize, IppHintAlgorithm hint, IppRoundMode rndMode);

IppStatus ippiMulC64f_16s_C1IR(Ipp16s* pSrcDst, int srcDstStep, const Ipp64f value[1], IppiSize roiSize, IppHintAlgorithm hint, IppRoundMode rndMode);

IppStatus ippiMulC64f_16s_C3IR(Ipp16s* pSrcDst, int srcDstStep, const Ipp64f value[3], IppiSize roiSize, IppHintAlgorithm hint, IppRoundMode rndMode);

IppStatus ippiMulC64f_16s_C4IR(Ipp16s* pSrcDst, int srcDstStep, const Ipp64f value[4], IppiSize roiSize, IppHintAlgorithm hint, IppRoundMode rndMode);

IppStatus ippiMulC64f_32f_C1IR(Ipp32f* pSrcDst, int srcDstStep, const Ipp64f value[1], IppiSize roiSize, IppHintAlgorithm hint, IppRoundMode rndMode);

IppStatus ippiMulC64f_32f_C3IR(Ipp32f* pSrcDst, int srcDstStep, const Ipp64f value[3], IppiSize roiSize, IppHintAlgorithm hint, IppRoundMode rndMode);

IppStatus ippiMulC64f_32f_C4IR(Ipp32f* pSrcDst, int srcDstStep, const Ipp64f value[4], IppiSize roiSize, IppHintAlgorithm hint, IppRoundMode rndMode);

**Include Files**

ippi.h

**Domain Dependencies**

**Headers:** ippcore.h, ippvm.h, ipps.h

**Libraries:** ippcore.lib, ippvm.lib, ipps.lib

**Parameters**

- **pSrc**  
  Pointer to the source image.

- **pSrcDst**  
  Pointer to the source and destination image ROI for in-place operations.

- **srcStep**  
  Distance, in bytes, between the starting points of consecutive lines in the source image.

- **srcDstStep**  
  Distance, in bytes, between the starting points of consecutive lines in the source and destination image for in-place operations.
value

Constant vector to add to image pixel values.

pDst

Pointer to the destination image.

dstStep

Distance, in bytes, between the starting points of consecutive lines in the destination image.

roiSize

Size of the image ROI, in pixels.

hint

Option to select the algorithmic implementation of the function, the following values are supported:

ippAlgHintAccurate

All output pixels are exact; accuracy takes precedence over performance.

ippAlgHintFast, ippAlgHintNone

Function performance takes precedence over accuracy and some output pixels can differ by ±1 from the exact result.

rndMode

Rounding mode, the following values are supported:

ippRndZero

Floating-point values are truncated to zero.

ippRndNear

Floating-point values are rounded to the nearest even integer when the fractional part equals 0.5; otherwise they are rounded to the nearest integer.

ippRndFinancial

Floating-point values are rounded down to the nearest integer when the fractional part is less than 0.5, or rounded up to the nearest integer if the fractional part is equal to or greater than 0.5.

Description

This function multiplies pixel values of the source image by the specified constant array and places the scaled results to the same image.

Return Values

ippStsNoErr

Indicates no error.

ippStsNullPtrErr

Indicates an error when at least one of the specified pointers is NULL.

ippStsSizeErr

Indicates an error when width or height of the image is less than, or equal to zero.

Example

To better understand usage of this function, refer to the following example in the examples archive available for download from https://software.intel.com/en-us/ipp-manual-examples:

MulC64f.c

MulScale

Multiplies pixel values of two images and scales the products.
Syntax

Case 1: Not-in-place operation

IppStatus ippiMulScale_<mod>(const Ipp<datatype>* pSrc1, int src1Step, const Ipp<datatype>* pSrc2, int src2Step, Ipp<datatype>* pDst, int dstStep, IppiSize roiSize);

Supported values for mod:

8u_C1R   16u_C1R
8u_C3R   16u_C3R
8u_C4R   16u_C4R
8u_AC4R  16u_AC4R

Case 2: In-place operation

IppStatus ippiMulScale_<mod>(const Ipp<datatype>* pSrc, int srcStep, Ipp<datatype>* pSrcDst, int srcDstStep, IppiSize roiSize);

Supported values for mod:

8u_C1IR  16u_C1IR
8u_C3IR  16u_C3IR
8u_C4IR  16u_C4IR
8u_AC4IR 16u_AC4IR

Include Files

ippi.h

Domain Dependencies

Headers: ippcore.h, ippvm.h, ipps.h
Libraries: ippcore.lib, ippvm.lib, ipps.lib

Parameters

pSrc, pSrc1, pSrc2
srcStep, src1Step, src2Step

Pointers to the source images ROI.

pDst
dstStep

Distance in bytes between starts of consecutive lines in the source images.

Pointer to the destination image ROI.

pSrcDst
srcDstStep

Distance in bytes between starts of consecutive lines in the destination image.

Pointer to the source and destination image ROI for the in-place operation.

roiSize

Distance in bytes between starts of consecutive lines in the source and destination image for the in-place operation.

Size of the source and destination ROI in pixels.
Description
This function operates with ROI (see Regions of Regions of Interest in Intel IPP).

This function multiplies corresponding pixel values of two input buffers and scales the products using the following formula:

\[ \text{dst}_\text{pixel} = \frac{\text{src}_1\text{pixel} \times \text{src}_2\text{pixel}}{\text{max}_\text{val}}, \]

where \text{src}_1\text{pixel} and \text{src}_2\text{pixel} are pixel values of the source buffers, \text{dst}_\text{pixel} is the resultant pixel value, and \text{max}_\text{val} is the maximum value of the pixel data range (see Table “Image Data Types and Ranges” for details). The function is implemented for 8-bit and 16-bit unsigned data types only.

Note that the functions with \text{AC4} descriptor do not process alpha channelss.

Return Values
ippStsNoErr Indicates no error. Any other value indicates an error or a warning.
ippStsNullPtrErr Indicates an error condition if any of the specified pointers is NULL.
ippStsSizeErr Indicates an error condition if roiSize has a field with zero or negative value.
ippStsStepErr Indicates an error condition if any of the specified buffer step values is zero or negative.

MulCScale
Multiplies pixel values of an image by a constant and scales the products.

Syntax
Case 1: Not-in-place operation on one-channel data
IppStatus ippiMulCScale_<mod>(const Ipp<datatype>* pSrc, int srcStep, Ipp<datatype> value, Ipp<datatype>* pDst, int dstStep, IppiSize roiSize);

Supported values for mod:
8u_C1R  16u_C1R

Case 2: Not-in-place operation on multi-channel data
IppStatus ippiMulCScale_<mod>(const Ipp<datatype>* pSrc, int srcStep, const Ipp<datatype> value[3], Ipp<datatype>* pDst, int dstStep, IppiSize roiSize);

Supported values for mod:
8u_C3R  16u_C3R
8u_AC4R  16u_AC4R
IppStatusippiMulCScale_<mod>(const Ipp<datatype>* pSrc, int srcStep, const Ipp<datatype> value[4], Ipp<datatype>* pDst, int dstStep, IppiSize roiSize);

Supported values for mod:

8u_C4R  16u_C4R

**Case 3: In-place operation on one-channel data**

IppStatusippiMulCScale_<mod>(Ipp<datatype> value, Ipp<datatype>* pSrcDst, int srcDstStep, IppiSize roiSize);

Supported values for mod:

8u_C1IR  16u_C1IR

**Case 4: In-place operation on multi-channel data**

IppStatusippiMulCScale_<mod>(const Ipp<datatype> value[3], const Ipp<datatype>* pSrcDst, int srcDstStep, IppiSize roiSize);

Supported values for mod:

8u_C3IR  16u_C3IR
8u_AC4IR  16u_AC4IR

IppStatusippiMulCScale_<mod>(const Ipp<datatype> value[4], Ipp<datatype>* pSrcDst, int srcDstStep, IppiSize roiSize);

Supported values for mod:

8u_C4IR  16u_C4IR

**Include Files**

ippi.h

**Domain Dependencies**

Headers: ippcore.h, ippvm.h, ipps.h
Libraries: ippcore.lib, ippvm.lib, ipps.lib

**Parameters**

- **pSrc**: Pointer to the source image ROI.
- **srcStep**: Distance in bytes between starts of consecutive lines in the source image.
- **value**: The constant value to multiply each pixel value in a source image (constant vector in case of 3- or four-channel images).
- **pDst**: Pointer to the destination image ROI.
- **dstStep**: Distance in bytes between starts of consecutive lines in the destination image.
- **pSrcDst**: Pointer to the source and destination image ROI for the in-place operation.
**srcDstStep**

Distance in bytes between starts of consecutive lines in the source and destination image for the in-place operation.

**roiSize**

Size of the source and destination ROI in pixels.

**Description**

This function operates with ROI (see Regions of Interest in Intel IPP).

This function multiplies pixel values in the input buffer by a constant `value` and scales the products using the following formula:

\[
\text{dst\_pixel} = \frac{\text{src\_pixel} \times \text{value}}{\text{max\_val}},
\]

where `src\_pixel` is a pixel values of the source buffer, `dst\_pixel` is the resultant pixel value, and `max\_val` is the maximum value of the pixel data range (see Table “Image Data Types and Ranges” for details).

The function is implemented for 8-bit and 16-bit unsigned data types only. It can be used to multiply pixel values by a number between 0 and 1.

Note that the functions with AC4 descriptor do not process alpha channels.

**Return Values**

- `ippStsNoErr` indicates no error. Any other value indicates an error or a warning.
- `ippStsNullPtrErr` Indicates an error condition if one of the specified pointers is NULL.
- `ippStsSizeErr` Indicates an error condition if the `roiSize` has a field with zero or negative value.
- `ippStsStepErr` Indicates an error condition if any of the specified buffer step values is zero or negative.

**Example**

The code example below shows how to use the function `ippiMulCScale_8u_C1R`.

```c
void func_mulcscale()
{
    IppiSize ROI = {8,4};
    IppiSize ROI2 = {5,4};
    Ipp8u src[8*4];
    Ipp8u dst[8*4];
    Ipp8u v = 100;

    ippSet_8u_C1R(100,src,8,ROI);
    ippSet_8u_C1R(0,dst,8,ROI);
    ippMulCScale_8u_C1R(src,8,v,dst,8,ROI2);
}
```

**Result:**

<table>
<thead>
<tr>
<th>src</th>
<th>dst</th>
</tr>
</thead>
<tbody>
<tr>
<td>100 100 100 100 100 100 100 100</td>
<td>39 39 39 39 39 0 0 0</td>
</tr>
<tr>
<td>100 100 100 100 100 100 100 100</td>
<td>39 39 39 39 39 0 0 0</td>
</tr>
<tr>
<td>100 100 100 100 100 100 100 100</td>
<td>39 39 39 39 39 0 0 0</td>
</tr>
<tr>
<td>100 100 100 100 100 100 100 100</td>
<td>39 39 39 39 39 0 0 0</td>
</tr>
</tbody>
</table>
**Sub**

Subtracts pixel values of two images.

**Syntax**

**Case 1: Not-in-place operation on integer or complex data**

IppStatus ippiSub_<mod>(const Ipp<datatype>* pSrc1, int src1Step, const Ipp<datatype>* pSrc2, int src2Step, Ipp<datatype>* pDst, int dstStep, IppiSize roiSize, int scaleFactor);

Supported values for **mod**:

- 8u_C1RSfs
- 8u_C3RSfs
- 8u_AC4RSfs
- 8u_C4RSfs
- 16u_C1RSfs
- 16u_C3RSfs
- 16u_AC4RSfs
- 16u_C4RSfs
- 16s_C1RSfs
- 16s_C3RSfs
- 16s_AC4RSfs
- 16s_C4RSfs

**Case 2: Not-in-place operation on floating-point or complex data**

IppStatus ippiSub_<mod>(const Ipp<datatype>* pSrc1, int src1Step, const Ipp<datatype>* pSrc2, int src2Step, Ipp<datatype>* pDst, int dstStep, IppiSize roiSize);

Supported values for **mod**:

- 32f_C1R
- 32f_C3R
- 32f_AC4R
- 32f_C4R

**Case 3: In-place operation on integer or complex data**

IppStatus ippiSub_<mod>(const Ipp<datatype>* pSrc, int srcStep, Ipp<datatype>* pSrcDst, int srcDstStep, IppiSize roiSize, int scaleFactor);

Supported values for **mod**:

- 8u_C1IRSfs
- 8u_C3IRSfs
- 8u_AC4IRSfs
- 8u_C4IRSfs
- 16u_C1IRSfs
- 16u_C3IRSfs
- 16u_AC4IRSfs
- 16u_C4IRSfs
- 16s_C1IRSfs
- 16s_C3IRSfs
- 16s_AC4IRSfs
- 16s_C4IRSfs
Case 4: In-place operation on floating-point or complex data
IppStatus ippiSub_<mod>(const Ipp<datatype>* pSrc, int srcStep, Ipp<datatype>* pSrcDst, int srcDstStep, IppiSize roiSize);

Supported values for mod:

32f_C1IR
32f_C3IR
32f_AC4IR
32f_C4IR

Case 5: Not-in-place operation on integer data with platform-aware functions
IppStatus ippiSub_<mod>(const Ipp<datatype>* pSrc1, IppSizeL src1Step, const Ipp<datatype>* pSrc2, IppSizeL src2Step, Ipp<datatype>* pDst, IppSizeL dstStep, IppiSizeL roiSize, int scaleFactor);

Supported values for mod:

8u_C1RSfs_L 16u_C1RSfs_L 16s_C1RSfs_L
8u_C3RSfs_L 16u_C3RSfs_L 16s_C3RSfs_L
8u_AC4RSfs_L 16u_AC4RSfs_L 16s_AC4RSfs_L
8u_C4RSfs_L 16u_C4RSfs_L 16s_C4RSfs_L

Case 6: Not-in-place operation on floating-point data with platform-aware functions
IppStatus ippiSub_<mod>(const Ipp<datatype>* pSrc1, IppSizeL src1Step, const Ipp<datatype>* pSrc2, IppSizeL src2Step, Ipp<datatype>* pDst, IppSizeL dstStep, IppiSizeL roiSize);

Supported values for mod:

32f_C1R_L
32f_C3R_L
32f_C4R_L
32f_AC4R_L

Case 7: In-place operation on integer data with platform-aware functions
IppStatus ippiSub_<mod>(const Ipp<datatype>* pSrc, IppSizeL srcStep, Ipp<datatype>* pSrcDst, IppSizeL srcDstStep, IppiSizeL roiSize, int scaleFactor);

Supported values for mod:

8u_C1IRSfs_L 16u_C1IRSfs_L 16s_C1IRSfs_L
8u_C3IRSfs_L 16u_C3IRSfs_L 16s_C3IRSfs_L
8u_AC4IRSfs_L 16u_AC4IRSfs_L 16s_AC4IRSfs_L
Case 8: In-place operation on floating-point data with platform-aware functions

IppStatus ippiSub_<mod>(const Ipp<datatype>* pSrc1, IppSizeL src1Step, Ipp<datatype>* pDst, IppSizeL dstStep, IppiSizeL roiSize);

Supported values for mod:

32f_C1IR_L
32f_C3IR_L
32f_AC4IR_L
32f_C4IR_L

Case 9: Not-in-place operation on integer data with threading layer (TL) functions

IppStatus ippiSub_<mod>(const Ipp<datatype>* pSrc1, IppSizeL src1Step, const Ipp<datatype>* pSrc2, IppSizeL src2Step, Ipp<datatype>* pDst, IppSizeL dstStep, IppiSizeL roiSize, int scaleFactor);

Supported values for mod:

8u_C1RSfs_LT
16u_C1RSfs_LT
16s_C1RSfs_LT
8u_C3RSfs_LT
16u_C3RSfs_LT
16s_C3RSfs_LT
8u_AC4RSfs_LT
16u_AC4RSfs_LT
16s_AC4RSfs_LT
8u_C4RSfs_LT
16u_C4RSfs_LT
16s_C4RSfs_LT

Case 10: Not-in-place operation on floating-point data with TL functions

IppStatus ippiSub_<mod>(const Ipp<datatype>* pSrc1, IppSizeL src1Step, const Ipp<datatype>* pSrc2, IppSizeL src2Step, Ipp<datatype>* pDst, IppSizeL dstStep, IppiSizeL roiSize);

Supported values for mod:

32f_C1R_LT
32f_C3R_LT
32f_C4R_LT
32f_AC4R_LT

Case 11: In-place operation on integer data with TL functions

IppStatus ippiSub_<mod>(const Ipp<datatype>* pSrc, IppSizeL srcStep, Ipp<datatype>* pSrcDst, IppSizeL srcDstStep, IppiSizeL roiSize, int scaleFactor);

Supported values for mod:

8u_C1IRSfs_LT
16u_C1IRSfs_LT
16s_C1IRSfs_LT
8u_C3IRSfs_LT
16u_C3IRSfs_LT
16s_C3IRSfs_LT
Case 12: In-place operation on floating-point data with TL functions

IppStatus ippiSub_<mod>(const Ipp<datatype>* pSrc1, IppSizeL src1Step, Ipp<datatype>* pDst, IppSizeL dstStep, IppSizeL roiSize);

Supported values for mod:

32f_C1IR_LT
32f_C3IR_LT
32f_AC4IR_LT
32f_C4IR_LT

Include Files

ippi.h

Flavors with the _LT suffix: ippi_tl.h
Flavors with the _L suffix: ippi_l.h

Domain Dependencies

Flavors declared in ippi.h:

Headers: ippcore.h, ippvm.h, ipps.h
Libraries: ippcore.lib, ippvm.lib, ipps.lib

Flavors declared in ippi_tl.h:


Parameters

pSrc, pSrc1, pSrc2
srcStep, src1Step, src2Step

pDst
dstStep

pSrcDst
srcDstStep

roiSize
scaleFactor

Points to the source images ROI.
Distances in bytes between starts of consecutive lines in the source images.
Pointer to the destination image ROI.
Distance in bytes between starts of consecutive lines in the destination image.
Pointer to the source and destination image ROI for the in-place operation.
Distance in bytes between starts of consecutive lines in the source and destination image for the in-place operation.
Size of the source and destination ROI in pixels.
Scale factor (see Integer Result Scaling).

Description

This function operates with ROI (see Regions of Interest in Intel IPP).
This function subtracts pixel values of the source buffer \( p\text{Src1} \) from the corresponding pixel values of the buffer \( p\text{Src2} \) and places the result in the destination buffer \( p\text{Dst} \). For in-place operations, the values in \( p\text{Src} \) are subtracted from the values in \( p\text{SrcDst} \) and the results are placed into \( p\text{SrcDst} \). For complex data, the function processes both real and imaginary parts of pixel values.

**NOTE**
Step values must be positive for functions that operate on complex data.

In case of operations on integer data, the resulting values are scaled by \( \text{scaleFactor} \).

Note that the functions with AC4 descriptor do not process alpha channels.

**Return Values**

- \( \text{ippStsNoErr} \): Indicates no error. Any other value indicates an error or a warning.
- \( \text{ippStsNullPtrErr} \): Indicates an error condition if one of the specified pointers is NULL.
- \( \text{ippStsSizeErr} \): Indicates an error condition if \( \text{roiSize} \) has a field with zero or negative value.
- \( \text{ippStsStepErr} \): Indicates an error condition if any of the specified buffer step values is zero or negative for functions that operate on complex data.

**SubC**

*Subtracts a constant from pixel values of an image.*

**Syntax**

**Case 1: Not-in-place operation on one-channel integer or complex data**

\[
\text{IppStatus ippiSubC}_{<\text{mod}}(\text{const Ipp}<\text{datatype}>* \ p\text{Src}, \text{int } \text{srcStep, Ipp}<\text{datatype}> \ \text{value}, \text{Ipp}<\text{datatype}>* \ p\text{Dst}, \text{int } \text{dstStep}, \text{IppiSize } \text{roiSize}, \text{int } \text{scaleFactor});
\]

Supported values for \( \text{mod} \):

- \( 8\text{u}_\text{C1RSfs} \)
- \( 16\text{u}_\text{C1RSfs} \)
- \( 16\text{s}_\text{C1RSfs} \)

**Case 2: Not-in-place operation on multi-channel integer or complex data**

\[
\text{IppStatus ippiSubC}_{<\text{mod}}(\text{const Ipp}<\text{datatype}>* \ p\text{Src}, \text{int } \text{srcStep, const Ipp}<\text{datatype}> \ \text{value}[3], \text{Ipp}<\text{datatype}>* \ p\text{Dst}, \text{int } \text{dstStep, IppiSize roiSize, int } \text{scaleFactor});
\]

Supported values for \( \text{mod} \):

- \( 8\text{u}_\text{C3RSfs} \)
- \( 16\text{u}_\text{C3RSfs} \)
- \( 16\text{s}_\text{C3RSfs} \)
IppStatusippiSubC_<mod>(const Ipp<datatype>* pSrc, int srcStep, const Ipp<datatype> value[4], Ipp<datatype>* pDst, int dstStep, IppiSize roiSize, int scaleFactor);

Supported values for mod:

8u_C4RSfs  16u_C4RSfs  16s_C4RSfs

IppStatusippiSubC_<mod>(const Ipp<datatype>* pSrc, int srcStep, const Ipp<datatype> value[3], Ipp<datatype>* pDst, int dstStep, IppiSize roiSize, int scaleFactor);

Supported values for mod:

8u_AC4RSfs  16u_AC4RSfs  16s_AC4RSfs

Case 3: Not-in-place operation on one-channel floating-point or complex data

IppStatusippiSubC_32f_C1R(const Ipp<datatype>* pSrc, int srcStep, Ipp<datatype> value, Ipp<datatype>* pDst, int dstStep, IppiSize roiSize);

Case 4: Not-in-place operation on multi-channel floating-point or complex data

IppStatusippiSubC_32f_C3R(const Ipp<datatype>* pSrc, int srcStep, const Ipp<datatype> value[3], Ipp<datatype>* pDst, int dstStep, IppiSize roiSize);

Case 5: In-place operation on one-channel integer or complex data

IppStatusippiSubC_<mod>(Ipp<datatype> value, Ipp<datatype>* pSrcDst, int srcDstStep, IppiSize roiSize, int scaleFactor);

Supported values for mod:

8u_C1IRSfs  16u_C1IRSfs  16s_C1IRSfs

Case 6: In-place operation on multi-channel integer or complex data

IppStatusippiSubC_<mod>(const Ipp<datatype> value[3], Ipp<datatype>* pSrcDst, int srcDstStep, IppiSize roiSize, int scaleFactor);

Supported values for mod:

8u_C3IRSfs  16u_C3IRSfs  16s_C3IRSfs

8u_AC4IRSfs  16u_AC4IRSfs  16s_AC4IRSfs
IppStatus ippiSubC_<mod>(const Ipp<datatype> value[4], Ipp<datatype>* pSrcDst, int srcDstStep, IppSize roiSize, int scaleFactor);

Supported values for mod:

8u_C4IRSfs  16u_C4IRSfs  16s_C4IRSfs

**Case 7: In-place operation on one-channel floating-point or complex data**

IppStatus ippiSubC_32f_C1IR(Ipp<datatype> value, Ipp<datatype>* pSrcDst, int srcDstStep, IppSize roiSize);

**Case 8: In-place operation on multi-channel floating-point or complex data:**

IppStatus ippiSubC_<mod>(const Ipp<datatype> value[3], Ipp<datatype>* pSrcDst, int srcDstStep, IppSize roiSize);

Supported values for mod:

32f_C3IR
32f_AC4IR

IppStatus ippiSubC_32f_C4IR(const Ipp32f value[4], Ipp32f* pSrcDst, int srcDstStep, IppSize roiSize);

**Case 9: Not-in-place operation on one-channel integer data with platform-aware functions**

IppStatus ippiSubC_<mod>(const Ipp<datatype>* pSrc, IppSizeL srcStep, Ipp<datatype> value, Ipp<datatype>* pDst, IppSizeL dstStep, IppSizeL roiSize, int scaleFactor);

Supported values for mod:

8u_C1RSfs_L  16u_C1RSfs_L  16s_C1RSfs_L

**Case 10: Not-in-place operation on multi-channel integer data with platform-aware functions**

IppStatus ippiSubC_<mod>(const Ipp<datatype>* pSrc, IppSizeL srcStep, const Ipp<datatype> value[3], Ipp<datatype>* pDst, IppSizeL dstStep, IppSizeL roiSize, int scaleFactor);

Supported values for mod:

8u_C3RSfs_L  16u_C3RSfs_L  16s_C3RSfs_L
8u_AC4RSfs_L  16u_AC4RSfs_L  16s_AC4RSfs_L

IppStatus ippiSubC_<mod>(const Ipp<datatype>* pSrc, IppSizeL srcStep, const Ipp<datatype> value[4], Ipp<datatype>* pDst, IppSizeL dstStep, IppSizeL roiSize, int scaleFactor);

Supported values for mod:

8u_C4RSfs_L  16u_C4RSfs_L  16s_C4RSfs_L

**Case 11: Not-in-place operation on one-channel floating point data with platform-aware functions**

IppStatus ippiSubC_32f_C1R_L(const Ipp32f* pSrc, IppSizeL srcStep, Ipp32f value, Ipp32f* pDst, IppSizeL dstStep, IppSizeL roiSize);
Case 12: Not-in-place operation on multi-channel floating point data with platform-aware functions

IppStatus ippiSubC_<mod>(const Ipp32f* pSrc, IppSizeL srcStep, const Ipp32f value[3], Ipp32f* pDst, IppSizeL dstStep, IppSizeL roiSize);

Supported values for \texttt{mod}:

32f_C3R_L
32f_AC4R_L

IppStatus ippiSubC_32f_C4R_L(const Ipp32f* pSrc, IppSizeL srcStep, const Ipp32f value[4], Ipp32f* pDst, IppSizeL dstStep, IppSizeL roiSize);

Case 13: In-place operation on one-channel integer data with platform-aware functions

IppStatus ippiSubC_<mod>(Ipp<datatype> value, Ipp<datatype>* pSrcDst, IppSizeL srcDstStep, IppSizeL roiSize, int scaleFactor);

Supported values for \texttt{mod}:

8u_C1IRSfs_L 16u_C1IRSfs_L 16s_C1IRSfs_L

Case 14: In-place operation on multi-channel integer data with platform-aware functions

IppStatus ippiSubC_<mod>(const Ipp<datatype> value[3], Ipp<datatype>* pSrcDst, IppSizeL srcDstStep, IppSizeL roiSize, int scaleFactor);

Supported values for \texttt{mod}:

8u_C3IRSfs_L 16u_C3IRSfs_L 16s_C3IRSfs_L
8u_AC4IRSfs_L 16u_AC4IRSfs_L 16s_AC4IRSfs_L

IppStatus ippiSubC_<mod>(const Ipp<datatype> value[4], Ipp<datatype>* pSrcDst, IppSizeL srcDstStep, IppSizeL roiSize, int scaleFactor);

Supported values for \texttt{mod}:

8u_C4IRSfs_L 16u_C4IRSfs_L 16s_C4IRSfs_L

Case 15: In-place operation on one-channel floating point data with platform-aware functions

IppStatus ippiSubC_32f_C1IR_L(Ipp32f value, Ipp32f* pSrcDst, IppSizeL srcDstStep, IppSizeL roiSize);

Case 16: In-place operation on multi-channel integer data with platform-aware functions

IppStatus ippiSubC_<mod>(const Ipp32f value[3], Ipp32f* pSrcDst, IppSizeL srcDstStep, IppSizeL roiSize);

Supported values for \texttt{mod}:

32f_C3IR_L
32f_AC4IR_L

IppStatus ippiSubC_32f_C4IR_L(const Ipp32f value[4], Ipp32f* pSrcDst, IppSizeL srcDstStep, IppSizeL roiSize);
Case 17: Not-in-place operation on one-channel integer data with threading layer (TL) functions

IppStatus ippiSubC_<mod>(const Ipp<datatype>* pSrc, IppSizeL srcStep, Ipp<datatype>* pDst, IppSizeL dstStep, IppSizeL roiSize, int scaleFactor);

Supported values for mod:

8u_C1RSfs_LT  16u_C1RSfs_LT  16s_C1RSfs_LT

Case 18: Not-in-place operation on multi-channel integer data with TL functions

IppStatus ippiSubC_<mod>(const Ipp<datatype>* pSrc, IppSizeL srcStep, const Ipp<datatype>* value[3], Ipp<datatype>* pDst, IppSizeL dstStep, IppSizeL roiSize, int scaleFactor);

Supported values for mod:

8u_C3RSfs_LT  16u_C3RSfs_LT  16s_C3RSfs_LT
8u_AC4RSfs_LT  16u_AC4RSfs_LT  16s_AC4RSfs_LT

Case 19: Not-in-place operation on one-channel floating point data with TL functions

IppStatus ippiSubC_32f_C1R_LT(const Ipp32f* pSrc, IppSizeL srcStep, Ipp32f value, Ipp32f* pDst, IppSizeL dstStep, IppSizeL roiSize);

Case 20: Not-in-place operation on multi-channel floating point data with TL functions

IppStatus ippiSubC_<mod>(const Ipp32f* pSrc, IppSizeL srcStep, const Ipp32f value[3], Ipp32f* pDst, IppSizeL dstStep, IppSizeL roiSize);

Supported values for mod:

32f_C3R_LT
32f_AC4R_LT

Case 21: In-place operation on one-channel integer data with TL functions

IppStatus ippiSubC_<mod>(Ipp<datatype> value, Ipp<datatype>* pSrcDst, IppSizeL srcDstStep, IppSizeL roiSize, int scaleFactor);

Supported values for mod:

8u_C1IRSfs_LT  16u_C1IRSfs_LT  16s_C1IRSfs_LT
Case 22: In-place operation on multi-channel integer data with TL functions

IppStatus ippiSubC_<mod>(const Ippdatatype value[3], Ippdatatype* pSrcDst, IppSizeL srcDstStep, IppiSizeL roiSize, int scaleFactor);

Supported values for mod:

8u_C3IRSfs_LT  16u_C3IRSfs_LT  16s_C3IRSfs_LT
8u_AC4IRSfs_LT  16u_AC4IRSfs_LT  16s_AC4IRSfs_LT

IppStatus ippiSubC_<mod>(const Ippdatatype value[4], Ippdatatype* pSrcDst, IppSizeL srcDstStep, IppiSizeL roiSize, int scaleFactor);

Supported values for mod:

8u_C4IRSfs_LT  16u_C4IRSfs_LT  16s_C4IRSfs_LT

Case 23: In-place operation on one-channel floating point data with TL functions

IppStatus ippiSubC_32f_C1IR_LT(Ipp32f value, Ipp32f* pSrcDst, IppSizeL srcDstStep, IppiSizeL roiSize);

Case 24: In-place operation on multi-channel integer data with TL functions

IppStatus ippiSubC_<mod>(const Ipp32f value[3], Ipp32f* pSrcDst, IppSizeL srcDstStep, IppiSizeL roiSize);

Supported values for mod:

32f_C3IR_LT
32f_AC4IR_LT

IppStatus ippiSubC_32f_C4IR_LT(const Ipp32f value[4], Ipp32f* pSrcDst, IppSizeL srcDstStep, IppiSizeL roiSize);

Include Files

ippi.h

Flavors with the _LT suffix: ippi_tl.h

Flavors with the _L suffix: ippi_l.h

Domain Dependencies

Flavors declared in ippi.h:

Headers: ippcore.h, ippvm.h, ipps.h
Libraries: ippcore.lib, ippvm.lib, ipps.lib

Flavors declared in ippi_tl.h:


Parameters

pSrc  Pointer to the source image ROI.
srcStep  Distance in bytes between starts of consecutive lines in the source image.
**value**

The constant value to subtract from each pixel value in a source image (constant vector in case of multi-channel images).

**pDst**

Pointer to the destination image ROI.

**dstStep**

Distance in bytes between starts of consecutive lines in the destination image.

**pSrcDst**

Pointer to the source and destination image ROI for the in-place operation.

**srcDstStep**

Distance in bytes between starts of consecutive lines in the source and destination image for the in-place operation.

**roiSize**

Size of the source and destination ROI in pixels.

**scaleFactor**

Scale factor (see Integer Result Scaling).

**Description**

This function operates with ROI (see Regions of Interest in Intel IPP).

This function changes image intensity by subtracting the constant `value` from pixel values of an image buffer. For multi-channel images, the components of a constant vector `value` are subtracted from pixel channel values. For complex data, the function processes both real and imaginary parts of pixel values.

**NOTE**

Step values must be positive for functions that operate on complex data.

In case of operations on integer data, the resulting values are scaled by `scaleFactor`.

Note that the functions with the AC4 descriptor do not process alpha channels.

**Return Values**

- **ippStsNoErr**
  
  Indicates no error. Any other value indicates an error or a warning.

- **ippStsNullPtrErr**
  
  Indicates an error condition if one of the specified pointers is NULL.

- **ippStsSizeErr**
  
  Indicates an error condition if `roiSize` has a field with zero or negative value.

- **ippStsStepErr**
  
  Indicates an error condition if any of the specified buffer step values is zero or negative for functions that operate on complex data.

**Div**

Divides pixel values of an image by pixel values of another image.
Syntax

Case 1: Not-in-place operation on integer or complex data

IppStatus ippiDiv_<mod>(const Ipp<datatype>* pSrc1, int src1Step, const Ipp<datatype>* pSrc2, int src2Step, Ipp<datatype>* pDst, int dstStep, IppiSize roiSize, int scaleFactor);

Supported values for mod:

<table>
<thead>
<tr>
<th>8u_C1RSfs</th>
<th>16u_C1RSfs</th>
<th>16s_C1RSfs</th>
</tr>
</thead>
<tbody>
<tr>
<td>8u_C3RSfs</td>
<td>16u_C3RSfs</td>
<td>16s_C3RSfs</td>
</tr>
<tr>
<td>8u_AC4RSfs</td>
<td>16u_AC4RSfs</td>
<td>16s_AC4RSfs</td>
</tr>
<tr>
<td>8u_C4RSfs</td>
<td>16u_C4RSfs</td>
<td>16s_C4RSfs</td>
</tr>
</tbody>
</table>

Case 2: Not-in-place operation on floating-point or complex data

IppStatus ippiDiv_<mod>(const Ipp<datatype>* pSrc1, int src1Step, const Ipp<datatype>* pSrc2, int src2Step, Ipp<datatype>* pDst, int dstStep, IppiSize roiSize);

Supported values for mod:

<table>
<thead>
<tr>
<th>32f_C1R</th>
</tr>
</thead>
<tbody>
<tr>
<td>32f_C3R</td>
</tr>
<tr>
<td>32f_AC4R</td>
</tr>
<tr>
<td>32f_C4R</td>
</tr>
</tbody>
</table>

Case 3: In-place operation on integer or complex data

IppStatus ippiDiv_<mod>(const Ipp<datatype>* pSrc, int srcStep, Ipp<datatype>* pSrcDst, int dstStep, IppiSize roiSize, int scaleFactor);

Supported values for mod:

<table>
<thead>
<tr>
<th>8u_C1IRSfs</th>
<th>16u_C1IRSfs</th>
<th>16s_C1IRSfs</th>
</tr>
</thead>
<tbody>
<tr>
<td>8u_C3IRSfs</td>
<td>16u_C3IRSfs</td>
<td>16s_C3IRSfs</td>
</tr>
<tr>
<td>8u_AC4IRSfs</td>
<td>16u_AC4IRSfs</td>
<td>16s_AC4IRSfs</td>
</tr>
<tr>
<td>8u_C4IRSfs</td>
<td>16u_C4IRSfs</td>
<td>16s_C4IRSfs</td>
</tr>
</tbody>
</table>

Case 4: In-place operation on floating-point or complex data

IppStatus ippiDiv_<mod>(const Ipp<datatype>* pSrc, int srcStep, Ipp<datatype>* pSrcDst, int dstStep, IppiSize roiSize);

Supported values for mod:

<table>
<thead>
<tr>
<th>32f_C1IR</th>
</tr>
</thead>
<tbody>
<tr>
<td>32f_C3IR</td>
</tr>
<tr>
<td>32f_AC4IR</td>
</tr>
</tbody>
</table>
32f_C4IR

Case 5: Not-in-place operation on integer data with platform-aware functions

\[
\text{IppStatus ippiDiv}_{\text{mod}}(\text{const Ipp<datatype>\star pSrc1, IppSizeL src1Step, const Ipp<datatype>\star pSrc2, IppSizeL src2Step, Ipp<datatype>\star pDst, IppSizeL dstStep, IppSizeL roiSize, int scaleFactor);}
\]

Supported values for mod:

\[
8u_{C1RSfs_L} \\ 16u_{C1RSfs_L} \\ 16s_{C1RSfs_L} \\ 8u_{C3RSfs_L} \\ 16u_{C3RSfs_L} \\ 16s_{C3RSfs_L} \\ 8u_{AC4RSfs_L} \\ 16u_{AC4RSfs_L} \\ 16s_{AC4RSfs_L} \\ 8u_{C4RSfs_L} \\ 16u_{C4RSfs_L} \\ 16s_{C4RSfs_L}
\]

Case 6: Not-in-place operation on floating-point data with platform-aware functions

\[
\text{IppStatus ippiDiv}_{\text{mod}}(\text{const Ipp<datatype>\star pSrc1, IppSizeL src1Step, const Ipp<datatype>\star pSrc2, IppSizeL src2Step, Ipp<datatype>\star pDst, IppSizeL dstStep, IppSizeL roiSize);}
\]

Supported values for mod:

\[
32f_{C1R_L} \\ 32f_{C3R_L} \\ 32f_{AC4R_L} \\ 32f_{C4R_L}
\]

Case 7: In-place operation on integer data with platform-aware functions

\[
\text{IppStatus ippiDiv}_{\text{mod}}(\text{const Ipp<datatype>\star pSrc, IppSizeL srcStep, Ipp<datatype>\star pSrcDst, IppSizeL srcDstStep, IppSizeL roiSize, int scaleFactor);}
\]

Supported values for mod:

\[
8u_{C1IRSfs_L} \\ 16u_{C1IRSfs_L} \\ 16s_{C1IRSfs_L} \\ 8u_{C3IRSfs_L} \\ 16u_{C3IRSfs_L} \\ 16s_{C3IRSfs_L} \\ 8u_{AC4IRSfs_L} \\ 16u_{AC4IRSfs_L} \\ 16s_{AC4IRSfs_L} \\ 8u_{C4IRSfs_L} \\ 16u_{C4IRSfs_L} \\ 16s_{C4IRSfs_L}
\]

Case 8: In-place operation on floating-point data with platform-aware functions

\[
\text{IppStatus ippiDiv}_{\text{mod}}(\text{const Ipp<datatype>\star pSrc, IppSizeL srcStep, Ipp<datatype>\star pSrcDst, IppSizeL srcDstStep, IppSizeL roiSize, int scaleFactor);}
\]

Supported values for mod:

\[
32f_{C1R_L} \\ 32f_{C3R_L}
\]
32f_AC4IR_L
32f_C4IR_L

Case 9: Not-in-place operation on integer data with threading layer (TL) functions

IppStatus ippiDiv_<mod>(const Ipp<datatype>* pSrc1, IppSizeL src1Step, const Ipp<datatype>* pSrc2, IppSizeL src2Step, Ipp<datatype>* pDst, IppSizeL dstStep, IppSizeL roiSize, int scaleFactor);

Supported values for mod:

- 8u_C1RSfs_LT
- 16u_C1RSfs_LT
- 16s_C1RSfs_LT
- 8u_C3RSfs_LT
- 16u_C3RSfs_LT
- 16s_C3RSfs_LT
- 8u_AC4RSfs_LT
- 16u_AC4RSfs_LT
- 16s_AC4RSfs_LT
- 8u_C4RSfs_LT
- 16u_C4RSfs_LT
- 16s_C4RSfs_LT

Case 10: Not-in-place operation on floating-point data with TL functions

IppStatus ippiDiv_<mod>(const Ipp<datatype>* pSrc1, IppSizeL src1Step, const Ipp<datatype>* pSrc2, IppSizeL src2Step, Ipp<datatype>* pDst, IppSizeL dstStep, IppSizeL roiSize);

Supported values for mod:

- 32f_C1R_LT
- 32f_C3R_LT
- 32f_C4R_LT
- 32f_AC4R_LT

Case 11: In-place operation on integer data with TL functions

IppStatus ippiDiv_<mod>(const Ipp<datatype>* pSrc, IppSizeL srcStep, Ipp<datatype>* pSrcDst, IppSizeL srcDstStep, IppSizeL roiSize, int scaleFactor);

Supported values for mod:

- 8u_C1IRSfs_LT
- 16u_C1IRSfs_LT
- 16s_C1IRSfs_LT
- 8u_C3IRSfs_LT
- 16u_C3IRSfs_LT
- 16s_C3IRSfs_LT
- 8u_AC4IRSfs_LT
- 16u_AC4IRSfs_LT
- 16s_AC4IRSfs_LT
- 8u_C4IRSfs_LT
- 16u_C4IRSfs_LT
- 16s_C4IRSfs_LT

Case 12: In-place operation on floating-point data with TL functions

IppStatus ippiDiv_<mod>(const Ipp<datatype>* pSrc, IppSizeL srcStep, Ipp<datatype>* pSrcDst, IppSizeL srcDstStep, IppSizeL roiSize);

Supported values for mod:

- 32f_C1IR_LT
32f_C3IR_LT
32f_AC4IR_LT
32f_C4IR_LT

Include Files

ippi.h

Flavors with the _LT suffix: ippi_tl.h
Flavors with the _L suffix: ippi_l.h

Domain Dependencies

Flavors declared in ippi.h:
Headers: ippcore.h, ippvm.h, ipps.h
Libraries: ippcore.lib, ippvm.lib, ipps.lib

Flavors declared in ippi_tl.h:

Parameters

pSrc, pSrc1, pSrc2
srcStep, src1Step, src2Step
pDst
dstStep
pSrcDst
srcDstStep
roiSize
scaleFactor

Pointers to the source images ROI.
Distances in bytes between starts of consecutive lines in the source images.
Pointer to the destination image ROI.
Distance in bytes between starts of consecutive lines in the destination image.
Pointer to the source and destination image ROI for the in-place operation.
Distance in bytes between starts of consecutive lines in the source and destination image for the in-place operation.
Size of the source and destination ROI in pixels.
Scale factor (see Integer Result Scaling).

Description

This function operates with ROI (see Regions of Interest in Intel IPP).

This function divides pixel values of the source buffer pSrc2 by the corresponding pixel values of the buffer pSrc1 and places the result in a destination buffer pDst. For in-place operations, the values in pSrcDst are divided by the values in pSrc and placed into pSrcDst. For complex data, the function processes both real and imaginary parts of pixel values. In case of operations on integer data, the resulting values are scaled by scaleFactor and rounded (not truncated). When the function encounters a zero divisor value, the execution is not interrupted. The function returns the warning message and corresponding result value (see appendix A “Handling of Special Cases” for more information).

Note that the functions with the AC4 descriptor do not process alpha channels.
## Return Values

- **ippStsNoErr**: Indicates no error. Any other value indicates an error or a warning.
- **ippStsNullPtrErr**: Indicates an error condition if one of the specified pointers is NULL.
- **ippStsSizeErr**: Indicates an error condition if `roiSize` has a field with zero or negative value.
- **ippStsStepErr**: Indicates an error condition if any of the specified buffer step values is zero or negative.
- **ippStsDivByZero**: Indicates a warning that a divisor value is zero. The function execution is continued.

## Example

The code example below illustrates how the function `ippiDiv` can be used.

```c
IppStatus div32f( void ) {
    Ipp32f a[4*3], b[4*3];
    IppSize roi = {2,2};
    int i;
    for( i=0; i<4*3; ++i ) a[i] = b[i] = (float)i;
    return ippiDiv_32f_C1IR( a, 4*sizeof(Ipp32f), b,
        4*sizeof(Ipp32f), roi );
}
```

The destination image `b` contains

<table>
<thead>
<tr>
<th></th>
<th>-1.#IND</th>
<th>+1.000</th>
<th>+2.000</th>
<th>+3.000</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>+1.000</td>
<td>+1.000</td>
<td>+6.000</td>
<td>+7.000</td>
</tr>
<tr>
<td></td>
<td>+8.000</td>
<td>+9.000</td>
<td>+10.00</td>
<td>+11.00</td>
</tr>
</tbody>
</table>

Console output:

```
-- warning in div32f:
(6) Zero value(s) of divisor in the function Div
```

## Div_Round

Divides pixel values of an image by pixel values of another image with different rounding modes.

### Syntax

**Case 1: Not-in-place operation on integer data**

```
IppStatus ippiDiv_Round_<mod>(
    const Ipp<datatype>* pSrc1, int src1Step,
    const Ipp<datatype>* pSrc2, int src2Step,
    Ipp<datatype>* pDst, int dstStep, IppSize roiSize,
    IppRoundMode rndMode, int scaleFactor);
```

Supported values for `mod`:

- 8u_C1RSfs  16u_C1RSfs  16s_C1RSfs
- 8u_C3RSfs  16u_C3RSfs  16s_C3RSfs
Case 2: In-place operation on integer data

IppStatus ippiDiv_Round_<mod>(const Ipp<datatype>* pSrc, int srcStep, Ipp<datatype>* pSrcDst, int srcDstStep, IppiSize roiSize, IppRoundMode rndMode, int scaleFactor);

Supported values for mod:

8u_C1IRSfs  16u_C1IRSfs  16s_C1IRSfs
8u_C3IRSfs  16u_C3IRSfs  16s_C3IRSfs
8u_AC4IRSfs  16u_AC4IRSfs  16s_AC4IRSfs
8u_C4IRSfs  16u_C4IRSfs  16s_C4IRSfs

Include Files

ippi.h

Domain Dependencies

Headers: ippcore.h, ippvm.h, ipps.h
Libraries: ippcore.lib, ippvm.lib, ipps.lib

Parameters

pSrc, pSrc1, pSrc2  Pointers to the source images ROI.
srcStep, src1Step, src2Step  Distances in bytes between starts of consecutive lines in the source images.
pDst  Pointer to the destination image ROI.
dstStep  Distance in bytes between starts of consecutive lines in the destination image.
pSrcDst  Pointer to the source and destination image ROI for the in-place operation.
srcDstStep  Distance in bytes between starts of consecutive lines in the source and destination image for the in-place operation.
roiSize  Size of the source and destination ROI in pixels.
roundMode  Rounding mode, the following values are possible:

ippiRndZero  specifies that floating-point values are truncated toward zero,
ippiRndNear  specifies that floating-point values are rounded to the nearest even integer when the fractional part equals 0.5; otherwise they are rounded to the nearest integer,
ippRndFinancial specifies that floating-point values are rounded down to the nearest integer when the fractional part is less than 0.5, or rounded up to the nearest integer if the fractional part is equal or greater than 0.5.

scaleFactor Scale factor (see Integer Result Scaling).

Description
This function operates with ROI (see Regions of Interest in Intel IPP).

This function divides pixel values of the source buffer pSrc2 by the corresponding pixel values of the buffer pSrc1 and places the result in a destination buffer pDst. For in-place operations, the values in pSrcDst are divided by the values in pSrc and placed into pSrcDSt. The resulting values are scaled by scaleFactor and rounded using the rounding method specified by the parameter roundMode. When the function encounters a zero divisor value, the execution is not interrupted. The function returns the warning message and corresponding result value (see appendix A "Handling of Special Cases" for more information).

Note that the functions with the AC4 descriptor do not process alpha channels.

Return Values

ippStsNoErr Indicates no error. Any other value indicates an error or a warning.
ippStsNullPtrErr Indicates an error condition if one of the specified pointers is NULL.
ippStsSizeErr Indicates an error condition if roiSize has a field with zero or negative value.
ippStsStepErr Indicates an error condition if any of the specified buffer step values is zero or negative.
ippStsDivByZero Indicates a warning that a divisor value is zero. The function execution is continued.
ippStsRndModeNotSupported Indicates an error condition if the roundMode has an illegal value.
Eff

DivC
Divides pixel values of an image by a constant.

Syntax

Case 1: Not-in-place operation on one-channel integer or complex data

IppStatus ippiDivC_<mod>(const Ipp<datatype>* pSrc, int srcStep, Ipp<datatype> value, Ipp<datatype>* pDst, int dstStep, IppSize roiSize, int scaleFactor);

Supported values for mod:

8u_C1RSfs 16u_C1RSfs 16s_C1RSfs
Case 2: Not-in-place operation on multi-channel integer or complex data

IppStatus ippiDivC_<mod>(const Ipp<datatype>* pSrc, int srcStep, const Ipp<datatype> value[3], Ipp<datatype>* pDst, int dstStep, IppiSize roiSize, int scaleFactor);

Supported values for mod:

8u_C3RSfs 16u_C3RSfs 16s_C3RSfs 16s_AC4RSfs

IppStatus ippiDivC_<mod>(const Ipp<datatype>* pSrc, int srcStep, const Ipp<datatype> value[4], Ipp<datatype>* pDst, int dstStep, IppiSize roiSize, int scaleFactor);

Supported values for mod:

8u_C4RSfs 16u_C4RSfs 16s_C4RSfs

IppStatus ippiDivC_<mod>(const Ipp<datatype>* pSrc, int srcStep, const Ipp<datatype> value[3], Ipp<datatype>* pDst, int dstStep, IppiSize roiSize, int scaleFactor);

Supported values for mod:

8u_AC4RSfs 16u_AC4RSfs

Case 3: Not-in-place operation on one-channel floating-point or complex data

IppStatus ippiDivC_32f_C1R(const Ipp<datatype>* pSrc, int srcStep, Ipp<datatype> value, Ipp<datatype>* pDst, int dstStep, IppiSize roiSize);

Case 4: Not-in-place operation on multi-channel floating-point or complex data

IppStatus ippiDivC_32f_C3R(const Ipp<datatype>* pSrc, int srcStep, const Ipp<datatype> value[3], Ipp<datatype>* pDst, int dstStep, IppiSize roiSize);

IppStatus ippiDivC_32f_C4R(const Ipp32f* pSrc, int srcStep, const Ipp32f value[4], Ipp32f* pDst, int dstStep, IppiSize roiSize);

IppStatus ippiDivC_<mod>(const Ipp<datatype>* pSrc, int srcStep, const Ipp<datatype> value[3], Ipp<datatype>* pDst, int dstStep, IppiSize roiSize);

Supported values for mod:

32f_AC4R

IppStatus ippiDivC_32f_C4R(const Ipp32f* pSrc, int srcStep, const Ipp32f value[4], Ipp32f* pDst, int dstStep, IppiSize roiSize);

Case 5: In-place operation on one-channel integer or complex data

IppStatus ippiDivC_<mod>(Ipp<datatype> value, Ipp<datatype>* pSrcDst, int srcDstStep, IppiSize roiSize, int scaleFactor);

Supported values for mod:

8u_C1IRSfs 16u_C1IRSfs 16s_C1IRSfs
Case 6: In-place operation on multi-channel integer or complex data

IppStatus ippiDivC_<mod>(const Ipp<datatype> value[3], Ipp<datatype>* pSrcDst, int srcDstStep, IppiSize roiSize, int scaleFactor);

Supported values for mod:

<table>
<thead>
<tr>
<th>8u_C3IRSfs</th>
<th>16u_C3IRSfs</th>
<th>16s_C3IRSfs</th>
</tr>
</thead>
<tbody>
<tr>
<td>8u_AC4IRSfs</td>
<td>16u_AC4IRSfs</td>
<td>16s_AC4IRSfs</td>
</tr>
</tbody>
</table>

IppStatus ippiDivC_<mod>(const Ipp<datatype> value[4], Ipp<datatype>* pSrcDst, int srcDstStep, IppiSize roiSize, int scaleFactor);

Supported values for mod:

| 8u_C4IRSfs | 16u_C4IRSfs | 16s_C4IRSfs |

Case 7: In-place operation on one-channel floating-point or complex data

IppStatus ippiDivC_<mod>(Ipp<datatype> value, Ipp<datatype>* pSrcDst, int srcDstStep, IppiSize roiSize);

Supported values for mod:

| 32f_C1IR |

Case 8: In-place operation on multi-channel floating-point or complex data

IppStatus ippiDivC_<mod>(const Ipp<datatype> value[3], Ipp<datatype>* pSrcDst, int srcDstStep, IppiSize roiSize);

Supported values for mod:

| 32f_C3IR |
| 32f_AC4IR |

IppStatus ippiDivC_<mod>(const Ipp32f value[4], Ipp32f* pSrcDst, int srcDstStep, IppiSize roiSize);

Case 9: Not-in-place operation on one-channel integer data with platform-aware functions

IppStatus ippiDivC_<mod>(const Ipp<datatype>* pSrc, IppSizeL srcStep, Ipp<datatype>* pDst, IppSizeL dstStep, IppiSizeL roiSize, int scaleFactor);

Supported values for mod:

| 8u_C1RSfs_L | 16u_C1RSfs_L | 16s_C1RSfs_L |

Case 10: Not-in-place operation on multi-channel integer data with platform-aware functions

IppStatus ippiDivC_<mod>(const Ipp<datatype>* pSrc, IppSizeL srcStep, const Ipp<datatype> value[3], Ipp<datatype>* pDst, IppSizeL dstStep, IppiSizeL roiSize, int scaleFactor);

Supported values for mod:

| 8u_C3RSfs_L | 16u_C3RSfs_L | 16s_C3RSfs_L |
8u_AC4RSfs_L  16u_AC4RSfs_L  16s_AC4RSfs_L

IppStatusippiDivC_<mod>(const Ipp<datatype>* pSrc, IppSizeL srcStep, const Ipp<datatype>* pDst, IppSizeL dstStep, IppSizeL roiSize, int scaleFactor);

Supported values for mod:

8u_C4RSfs_L  16u_C4RSfs_L  16s_C4RSfs_L

Case 11: Not-in-place operation on one-channel floating point data with platform-aware functions

IppStatusippiDivC_32f_C1R_L(const Ipp32f* pSrc, IppSizeL srcStep, Ipp32f value, Ipp32f* pDst, IppSizeL dstStep, IppSizeL roiSize);

Case 12: Not-in-place operation on multi-channel floating point data with platform-aware functions

IppStatusippiDivC_<mod>(const Ipp32f* pSrc, IppSizeL srcStep, const Ipp32f value[3], Ipp32f* pDst, IppSizeL dstStep, IppSizeL roiSize);

Supported values for mod:

32f_C3R_L
32f_AC4R_L

IppStatusippiDivC_32f_C4R_L(const Ipp32f* pSrc, IppSizeL srcStep, const Ipp32f value[4], Ipp32f* pDst, IppSizeL dstStep, IppSizeL roiSize);

Case 13: In-place operation on one-channel integer data with platform-aware functions

IppStatusippiDivC_<mod>(Ipp<datatype> value, Ipp<datatype>* pSrcDst, IppSizeL srcDstStep, IppSizeL roiSize, int scaleFactor);

Supported values for mod:

8u_C1IRSfs_L  16u_C1IRSfs_L  16s_C1IRSfs_L

Case 14: In-place operation on multi-channel integer data with platform-aware functions

IppStatusippiDivC_<mod>(const Ipp<datatype> value[3], Ipp<datatype>* pSrcDst, IppSizeL srcDstStep, IppSizeL roiSize, int scaleFactor);

Supported values for mod:

8u_C3IRSfs_L  16u_C3IRSfs_L  16s_C3IRSfs_L
8u_AC4IRSfs_L  16u_AC4IRSfs_L  16s_AC4IRSfs_L

IppStatusippiDivC_<mod>(const Ipp<datatype> value[4], Ipp<datatype>* pSrcDst, IppSizeL srcDstStep, IppSizeL roiSize, int scaleFactor);

Supported values for mod:

8u_C4IRSfs_L  16u_C4IRSfs_L  16s_C4IRSfs_L
Case 15: In-place operation on one-channel floating point data with platform-aware functions

IppStatus ippiDivC_32f_C1IR_L(Ipp32f value, Ipp32f* pSrcDst, IppSizeL srcDstStep, IppSizeL roiSize);

Case 16: In-place operation on multi-channel integer data with platform-aware functions

IppStatus ippiDivC_<mod>(const Ipp32f value[3], Ipp32f* pSrcDst, IppSizeL srcDstStep, IppSizeL roiSize);

Supported values for mod:

32f_C3IR_L
32f_AC4IR_L

IppStatus ippiDivC_32f_C4IR_L(const Ipp32f value[4], Ipp32f* pSrcDst, IppSizeL srcDstStep, IppSizeL roiSize);

Case 17: Not-in-place operation on one-channel integer data with threading layer (TL) functions

IppStatus ippiDivC_<mod>(const Ipp<datatype>* pSrc, IppSizeL srcStep, const Ipp<datatype> value, Ipp<datatype>* pDst, IppSizeL dstStep, IppSizeL roiSize, int scaleFactor);

Supported values for mod:

8u_C1RSfs_LT  16u_C1RSfs_LT  16s_C1RSfs_LT
8u_C2RSfs_LT  16u_C2RSfs_LT  16s_C2RSfs_LT
8u_AC4RSfs_LT  16u_AC4RSfs_LT  16s_AC4RSfs_LT

Case 18: Not-in-place operation on multi-channel integer data with TL functions

IppStatus ippiDivC_<mod>(const Ipp<datatype>* pSrc, IppSizeL srcStep, const Ipp<datatype> value[3], Ipp<datatype>* pDst, IppSizeL dstStep, IppSizeL roiSize, int scaleFactor);

Supported values for mod:

8u_C3RSfs_LT  16u_C3RSfs_LT  16s_C3RSfs_LT
8u_AC4RSfs_LT  16u_AC4RSfs_LT  16s_AC4RSfs_LT

IppStatus ippiDivC_<mod>(const Ipp<datatype>* pSrc, IppSizeL srcStep, const Ipp<datatype> value[4], Ipp<datatype>* pDst, IppSizeL dstStep, IppSizeL roiSize, int scaleFactor);

Supported values for mod:

8u_C4RSfs_LT  16u_C4RSfs_LT  16s_C4RSfs_LT

Case 19: Not-in-place operation on one-channel floating point data with TL functions

IppStatus ippiDivC_32f_C1R_LT(const Ipp32f* pSrc, IppSizeL srcStep, Ipp32f value, Ipp32f* pDst, IppSizeL dstStep, IppSizeL roiSize);

Case 20: Not-in-place operation on multi-channel floating point data with TL functions

IppStatus ippiDivC_<mod>(const Ipp32f* pSrc, IppSizeL srcStep, const Ipp32f value[3], Ipp32f* pDst, IppSizeL dstStep, IppSizeL roiSize);

Supported values for mod:

32f_C3R_LT
Case 21: In-place operation on one-channel integer data with TL functions

IppStatus ippiDivC_<mod>(const Ipp<datatype> value, Ipp<datatype>* pSrcDst, IppSizeL srcDstStep, IppSizeL roiSize, int scaleFactor);

Supported values for mod:

8u_C1IRSfs_LT  16u_C1IRSfs_LT  16s_C1IRSfs_LT

Case 22: In-place operation on multi-channel integer data with TL functions

IppStatus ippiDivC_<mod>(const Ipp<datatype> value[3], Ipp<datatype>* pSrcDst, IppSizeL srcDstStep, IppSizeL roiSize, int scaleFactor);

Supported values for mod:

8u_C3IRSfs_LT  16u_C3IRSfs_LT  16s_C3IRSfs_LT
8u_AC4IRSfs_LT  16u_AC4IRSfs_LT  16s_AC4IRSfs_LT

IppStatus ippiDivC_<mod>(const Ipp<datatype> value[4], Ipp<datatype>* pSrcDst, IppSizeL srcDstStep, IppSizeL roiSize, int scaleFactor);

Supported values for mod:

8u_C4IRSfs_LT  16u_C4IRSfs_LT  16s_C4IRSfs_LT

Case 23: In-place operation on one-channel floating point data with TL functions

IppStatus ippiDivC_32f_C1IR_LT(Ipp32f value, Ipp32f* pSrcDst, IppSizeL srcDstStep, IppSizeL roiSize);

Case 24: In-place operation on multi-channel integer data with TL functions

IppStatus ippiDivC_<mod>(const Ipp32f value[3], Ipp32f* pSrcDst, IppSizeL srcDstStep, IppSizeL roiSize);

Supported values for mod:

32f_C3IR_LT
32f_AC4IR_LT

IppStatus ippiDivC_32f_C4IR_LT(const Ipp32f* pSrc, IppSizeL srcStep, const Ipp32f value[4], Ipp32f* pDst, IppSizeL dstStep, IppSizeL roiSize);

Include Files

ippi.h

Flavors with the _LT suffix: ippi_tl.h
Flavors with the _L suffix: ippi_l.h
Domain Dependencies
Flavors declared in ippi.h:
Headers: ippcore.h, ippvm.h, ipps.h
Libraries: ippcore.lib, ippvm.lib, ipps.lib
Flavors declared in ippi_tl.h:

Parameters

pSrc
Pointer to the source image ROI.

srcStep
Distance in bytes between starts of consecutive lines in the source image.

value
The constant value to divide each pixel value in a source buffer (constant vector in case of 3- or four-channel images).

pDst
Pointer to the destination image ROI.

dstStep
Distance in bytes between starts of consecutive lines in the destination image.

pSrcDst
Pointer to the source and destination image ROI for the in-place operation.

srcDstStep
Distance in bytes between starts of consecutive lines in the source and destination image for the in-place operation.

roiSize
Size of the source and destination ROI in pixels.

scaleFactor
Scale factor (see InInteger Result Scaling).

Description
This function operates with ROI (see Regions of Interest in Intel IPP).
This function changes image intensity by dividing pixel values of an image buffer by the constant value. For multi-channel images, pixel channel values are divided by the components of a constant vector value. For complex data, the function processes both real and imaginary parts of pixel values. In case of operations on integer data, the resulting values are scaled by scaleFactor and rounded (not truncated).
When the divisor value is zero, the function execution is aborted and the ippStsDivByZeroErr error status is set. Note that in the alpha channel case (AC4), the alpha channels are not processed.

Return Values

ippStsNoErr
Indicates no error. Any other value indicates an error or a warning.

ippStsNullPtrErr
Indicates an error condition if one of the specified pointers is NULL.

ippStsSizeErr
Indicates an error condition if roiSize has a field with zero or negative value.

ippStsStepErr
Indicates an error condition if srcStep, dstStep, or srcDstStep has a zero or negative value.

ippStsDivByZeroErr
Indicates an error condition if the divisor value is zero.
**Abs**

Comprises absolute pixel values of a source image and places them into the destination image.

**Case 1: Not-in-place operation**

IppStatus ippiAbs_<mod>(const Ipp<datatype>* pSrc, int srcStep, Ipp<datatype>* pDst, int dstStep, IppiSize roiSize);

Supported values for mod:

- 16s_C1R 32f_C1R
- 16s_C3R 32f_C3R
- 16s_C4R 32f_C4R
- 32f_AC4R

IppStatus ippiAbs_16s_AC4R(const Ipp<datatype>* pSrc, int srcStep, Ipp<datatype>* pDst, int dstStep, IppiSize roiSize);

**Case 2: In-place operation**

IppStatus ippiAbs_<mod>(Ipp<datatype>* pSrcDst, int srcDstStep, IppiSize roiSize);

Supported values for mod:

- 16s_C1IR 32f_C1IR
- 16s_C3IR 32f_C3IR
- 16s_C4IR 32f_C4IR
- 16s_AC4IR 32f_AC4IR

**Include Files**

ippi.h

**Domain Dependencies**

Headers: ippcore.h, ippvm.h, ipps.h

Libraries: ippcore.lib, ippvm.lib, ipps.lib

**Parameters**

- **pSrc**
  - Pointer to the source image ROI.
- **srcStep**
  - Distance in bytes between starts of consecutive lines in the source image.
- **pDst**
  - Pointer to the destination image ROI.
- **dstStep**
  - Distance in bytes between starts of consecutive lines in the destination image.
- **pSrcDst**
  - Pointer to the source and destination image ROI for the in-place operation.
srcDstStep
Distance in bytes between starts of consecutive lines in the source and destination image for the in-place operation.

roiSize
Size of the source and destination ROI in pixels.

Description
This function operates with ROI (see Regions of Interest in Intel IPP).

This function takes the absolute value of each channel in each pixel of the source image ROI and places the result into a destination image ROI. It operates on signed data only. Note that the functions with the AC4 descriptor do not process alpha channels.

Return Values
- ippStsNoErr: Indicates no error. Any other value indicates an error or a warning.
- ippStsNullPtrErr: Indicates an error condition if one of the specified pointers is NULL.
- ippStsSizeErr: Indicates an error condition if roiSize has a field with zero or negative value.

AbsDiff
*Calculates absolute difference between two images.*

Syntax
IppStatus ippiAbsDiff_<mod>(const Ippdatatype*pSrc1, int src1Step, const Ippdatatype*pSrc2, int src2Step, Ippdatatype*pDst, int dstStep, IppiSize roiSize);

Supported values for mod:
- 8u_C1R
- 16u_C1R
- 32f_C1R
- 8u_C3R

Include Files
ippcv.h

Domain Dependencies
Headers: ippcore.h, ippvm.h, ipps.h,ippi.h
Libraries: ippcore.lib, ippvm.lib, ipps.lib,ippi.lib

Parameters
- pSrc1: Pointer to the first source image.
- src1Step: Distance in bytes between starts of consecutive lines in the first source image.
- pSrc2: Pointer to second source image.
- src2Step: Distance in bytes between starts of consecutive lines in the second source image.
pDst
Distance in bytes between starts of consecutive lines in the
destination image.
Size of the image ROI in pixels.

Description
This function operates with ROI (see Regions of Interest in Intel IPP).
This function calculates the absolute pixel-wise difference between two images by the formula:
\[ pDst(x,y) = \text{abs}(pSrc_1(x,y) - pSrc_2(x,y)) \].

Return Values
ippStsNoErr Indicates no error.
ippStsNullPtrErr Indicates an error condition if one of the specified pointers is NULL.
ippStsSizeErr Indicates an error condition if roiSize has a field with zero or negative value.
ippStsStepErr Indicates an error when src1Step, src2Step or dstStep is less than roiSize.width * <pixelSize>.
ippStsNotEvenStepErr Indicates an error condition if one of step values for floating-point images cannot be divided by 4.

AbsDiffC
Calculates absolute difference between image and scalar value.

Syntax
IppStatus ippiAbsDiffC_<mod>(const Ipp16u* pSrc, int srcStep, Ipp16u* pDst, int dstStep, IppiSize roiSize, int value);

Supported values for mod:
8u_C1R 16u_C1R

IppStatus ippiAbsDiffC_32f_C1R(const Ipp32f* pSrc, int srcStep, Ipp32f* pDst, int dstStep, IppiSize roiSize, Ipp32f value);

Include Files
ippcv.h

Domain Dependencies
Headers: ippcore.h, ippvm.h, ipps.h,ippi.h
Libraries: ippcore.lib, ippvm.lib, ipps.lib,ippi.lib

Parameters
pSrc Pointer to the source image.
**srcStep**
Distance in bytes between starts of consecutive lines in the source image.

**pDst**
Pointer to the destination image.

**dstStep**
Distance in bytes between starts of consecutive lines in the destination image.

**roiSize**
Size of the image ROI in pixels.

**value**
Scalar value used to decrement each element of the source image.

**Description**
This function operates with ROI (see Regions of Interest in Intel IPP).
This function calculates the absolute pixel-wise difference between the source image `pSrc` and the scalar value by the formula:

\[
pDst(x,y) = \text{abs}(pSrc(x,y) - value).
\]

The function clips the `value` to the range \([0, 255]\) for the \(8\)u data type, and to the range \([0, 65535]\) for the \(16\)u data type.

**Return Values**
- ippStsNoErr: Indicates no error.
- ippStsNullPtrErr: Indicates an error condition if one of the specified pointers is NULL.
- ippStsSizeErr: Indicates an error if `roiSize` has a field with zero or negative value.
- ippStsStepErr: Indicates an error when `srcStep` or `dstStep` is less than `roiSize.width * <pixelSize>`.
- ippStsNotEvenStepErr: Indicates an error condition if one of step values for floating-point images cannot be divided by 4.

**Sqr**
Squares pixel values of an image and writes them into the destination image.

**Syntax**
**Case 1: Not-in-place operation on integer data**

\[
\text{IppStatus ippiSqr_<mod>}(\text{const Ipp<datatype>* } pSrc, \text{ int srcStep, Ipp<datatype>* } pDst, \text{ int dstStep, IppiSize roiSize, int scaleFactor});
\]

**Supported values for mod:**
- 8u_C1RSfs 16u_C1RSfs 16s_C1RSfs
- 8u_C3RSfs 16u_C3RSfs 16s_C3RSfs
- 8u_C4RSfs 16u_C4RSfs 16s_C4RSfs
- 8u_AC4RSfs 16u_AC4RSfs 16s_AC4RSfs
Case 2: Not-in-place operation on floating-point data

IppStatus ippiSqr_<mod>(const Ipp32f* pSrc, int srcStep, Ipp32f* pDst, int dstStep, IppiSize roiSize);

Supported values for mod:

32f_C1R
32f_C3R
32f_C4R
32f_AC4R

Case 3: In-place operation on integer data

IppStatus ippiSqr_<mod>(Ipp<datatype>* pSrcDst, int srcDstStep, IppiSize roiSize, int scaleFactor);

Supported values for mod:

8u_C1IRSfs 16u_C1IRSfs 16s_C1IRSfs
8u_C3IRSfs 16u_C3IRSfs 16s_C3IRSfs
8u_C4IRSfs 16u_C4IRSfs 16s_C4IRSfs
8u_AC4IRSfs 16u_AC4IRSfs 16s_AC4IRSfs

Case 4: In-place operation on floating-point data

IppStatus ippiSqr_<mod>(Ipp32f* pSrcDst, int srcDstStep, IppiSize roiSize);

Supported values for mod:

32f_C1IR
32f_C3IR
32f_C4IR
32f_AC4IR

Include Files

ippi.h

Domain Dependencies

Headers: ippcore.h, ippvm.h, ipps.h
Libraries: ippcore.lib, ippvm.lib, ipps.lib

Parameters

pSrc

Pointer to the source image ROI.

srcStep

Distance in bytes between starts of consecutive lines in the source image.
**Description**

This function operates with ROI (see Regions of Interest in Intel IPP).

This function squares pixel values of the source image ROI and writes them to the destination image ROI. The function flavors operating on integer data apply fixed scaling defined by `scaleFactor` to the internally computed values, and saturate the results before writing them to the destination image ROI.

Note that the functions with the AC4 descriptor do not process alpha channel.

**Return Values**

- `ippStsNoErr`: Indicates an error. Any other value indicates an error or a warning.
- `ippStsNullPtrErr`: Indicates an error condition if one of the specified pointers is NULL.
- `ippStsSizeErr`: Indicates an error condition if `roiSize` has a field with zero or negative value.

**Sqrt**

Computes square roots of pixel values of a source image and writes them into the destination image.

**Syntax**

**Case 1: Not-in-place operation on integer data**

```c
IppStatus ippiSqrt_<mod>(const Ipp<datatype>* pSrc, int srcStep, Ipp<datatype>* pDst, int dstStep, IppiSize roiSize, int scaleFactor);
```

Supported values for `mod`:

- 8u_C1RSfs 16u_C1RSfs 16s_C1RSfs
- 8u_C3RSfs 16u_C3RSfs 16s_C3RSfs
- 8u_AC4RSfs 16u_AC4RSfs 16s_AC4RSfs
Case 2: Not-in-place operation on floating-point data

IppStatus ippiSqrt_<mod>(const Ipp32f* pSrc, int srcStep, Ipp32f* pDst, int dstStep, IppiSize roiSize);

Supported values for mod:

32f_C1R
32f_C3R
32f_AC4R

Case 3: In-place operation on integer data

IppStatus ippiSqrt_<mod>(Ipp<datatype>* pSrcDst, int srcDstStep, IppiSize roiSize, int scaleFactor);

Supported values for mod:

8u_C1IRSfs 16u_C1IRSfs 16s_C1IRSfs
8u_C3IRSfs 16u_C3IRSfs 16s_C3IRSfs
8u_AC4IRSfs 16u_AC4IRSfs 16s_AC4IRSfs

Case 4: In-place operation on floating-point data

IppStatus ippiSqrt_<mod>(Ipp32f* pSrcDst, int srcDstStep, IppiSize roiSize);

Supported values for mod:

32f_C1IR
32f_C3IR
32f_C4IR
32f_AC4IR

Include Files

ippi.h

Domain Dependencies

Headers: ippcore.h, ippvm.h, ipps.h
Libraries: ippcore.lib, ippvm.lib, ipps.lib

Parameters

pSrc Pointer to the source image ROI.
srcStep Distance in bytes between starts of consecutive lines in the source image.
pDst Pointer to the destination image ROI.
dstStep Distance in bytes between starts of consecutive lines in the destination image.
Description
This function operates with ROI (see Regions of Interest in Intel IPP).

This function computes square roots of pixel values of the source image ROI and writes them into the
destination image ROI. The function flavors operating on integer data apply fixed scaling defined by
scaleFactor to the internally computed values, and saturate the results before writing them to the
destination image ROI. If a source pixel value is negative, the function issues a warning and continues
execution with the corresponding result value (see appendix A "Handling of Special Cases" for more
information).

Note that the functions with the AC4 descriptor do not process alpha channelss.

Return Values
ippStsNoErr Indicates no error. Any other value indicates an error or a
warning.
ippStsNullPtrErr Indicates an error condition if one of the specified pointers is
NULL.
ippStsSizeErr Indicates an error condition if roiSize has a field with zero or
negative value.
ippStsSqrtNegArg Indicates a warning that a source pixel has a negative value.

Ln
Computes the natural logarithm of pixel values in a
source image and writes the results into the
destination image.

Syntax
Case 1: Not-in-place operation on integer data
IppStatus ippiLn_<mod>(const Ipp<datatype>* pSrc, int srcStep, Ipp<datatype>* pDst, int
dstStep, IppiSize roiSize, int scaleFactor);

Supported values for mod:
8u_C1RSfs 16u_C1RSfs 16s_C1RSfs
8u_C3RSfs 16u_C3RSfs 16s_C3RSfs
Case 2: Not-in-place operation on floating-point data

```c
IppStatus ippiLn_<mod>(const Ipp32f* pSrc, int srcStep, Ipp32f* pDst, int dstStep, IppiSize roiSize);
```

Supported values for mod:

- 32f_C1R
- 32f_C3R

Case 3: In-place operation on integer data

```c
IppStatus ippiLn_<mod>(Ipp<datatype>* pSrcDst, int srcDstStep, IppiSize roiSize, int scaleFactor);
```

Supported values for mod:

- 8u_C1IRSfs  16u_C1IRSfs  16s_C1IRSfs
- 8u_C3IRSfs  16u_C3IRSfs  16s_C3IRSfs

Case 4: In-place operation on floating-point data

```c
IppStatus ippiLn_<mod>(Ipp32f* pSrcDst, int srcDstStep, IppiSize roiSize);
```

Supported values for mod:

- 32f_C1IR
- 32f_C3IR

Include Files

ippi.h

Domain Dependencies

Headers: ippcore.h, ippvm.h, ipps.h
Libraries: ippcore.lib, ippvm.lib, ipps.lib

Parameters

- `pSrc`  Pointer to the source image ROI.
- `srcStep`  Distance in bytes between starts of consecutive lines in the source image.
- `pDst`  Pointer to the destination image ROI.
- `dstStep`  Distance in bytes between starts of consecutive lines in the destination image.
- `pSrcDst`  Pointer to the source and destination image ROI for the in-place operation.
- `srcDstStep`  Distance in bytes between starts of consecutive lines in the source and destination image for the in-place operation.
- `roiSize`  Size of the source and destination ROI in pixels.
- `scaleFactor`  Scale factor (see Integer Result Scaling).
Description
This function operates with ROI (see Regions of Interest in Intel IPP).

This function computes natural logarithms of pixel values of the source image ROI and writes the resultant values to the destination image ROI. The function flavors operating on integer data apply fixed scaling defined by scaleFactor to the internally computed values, and saturate the results before writing them to the destination image ROI.

If a source pixel value is zero or negative, the function issues a corresponding warning and continues execution with the corresponding result value (see appendix A "Handling of Special Cases" for more information).

When several inputs have zero or negative value, the status code returned by the function corresponds to the first encountered case as illustrated in the code example below.

Return Values

<table>
<thead>
<tr>
<th>Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ippStsNoErr</td>
<td>Indicates no error. Any other value indicates an error or a warning.</td>
</tr>
<tr>
<td>ippStsNullPtrErr</td>
<td>Indicates an error condition if one of the specified pointers is NULL.</td>
</tr>
<tr>
<td>ippStsSizeErr</td>
<td>Indicates an error condition if roiSize has a field with zero or negative value.</td>
</tr>
<tr>
<td>ippStsStepErr</td>
<td>Indicates an error condition if srcStep, dstStep, or srcDstStep has a zero or negative value.</td>
</tr>
<tr>
<td>ippStsLnZeroArg</td>
<td>Indicates a warning that a source pixel has a zero value.</td>
</tr>
<tr>
<td>ippStsLnNegArg</td>
<td>Indicates a warning that a source pixel has a negative value.</td>
</tr>
</tbody>
</table>

Example
The code example below shows how to use Ln function.

```c
IppStatus ln( void ) {
    Ipp32f img[8*8];
    IppiSize roi = { 8, 8 };
    IppStatus st;
    ippiSet_32f_C1R( (float)IPP_E, img, 8*4, roi );
    img[0] = -0;
    img[1] = -1;
    st = ippiLn_32f_C1IR( img, 8*sizeof(Ipp32f), roi );
    printf("%f %f %f\n", img[0], img[1], img[2] );
    return st;
}
```

Output values:

-1.#INF00 -1.#IND00 1.000000

Status value and message:

(7) Zero value(s) of argument in the Ln function

Exp
Computes the exponential of pixel values in a source image and writes the results into the destination image.
Syntax

Case 1: Not-in-place operation on integer data

IppStatus ippiExp_<mod>(const Ipp<datatype>* pSrc, int srcStep, Ipp<datatype>* pDst, int dstStep, IppiSize roiSize, int scaleFactor);

Supported values for mod:

8u_C1RSfs 16u_C1RSfs 16s_C1RSfs
8u_C3RSfs 16u_C3RSfs 16s_C3RSfs

Case 2: Not-in-place operation on floating-point data

IppStatus ippiExp_<mod>(const Ipp32f* pSrc, int srcStep, Ipp32f* pDst, int dstStep, IppiSize roiSize);

Supported values for mod:

32f_C1R
32f_C3R

Case 3: In-place operation on integer data

IppStatus ippiExp_<mod>(Ipp<datatype>* pSrcDst, int srcDstStep, IppiSize roiSize, int scaleFactor);

Supported values for mod:

8u_C1IRSfs 16u_C1IRSfs 16s_C1IRSfs
8u_C3IRSfs 16u_C3IRSfs 16s_C3IRSfs

Case 4: In-place operation on floating-point data

IppStatus ippiExp_<mod>(Ipp32f* pSrcDst, int srcDstStep, IppiSize roiSize);

Supported values for mod:

32f_C1IR
32f_C3IR

Include Files

ippi.h

Domain Dependencies

Headers: ippcore.h, ippvm.h, ipps.h
Libraries: ippcore.lib, ippvm.lib, ipps.lib

Parameters

- **pSrc**
  - Pointer to the source image ROI.
- **srcStep**
  - Distance in bytes between starts of consecutive lines in the source image.
**Description**

This function operates with ROI (see Regions of Interest in Intel IPP).

This function computes $e$ to the power of pixel values of the source image ROI and writes the resultant values into the destination image ROI. The function flavors operating on integer data apply fixed scaling defined by `scaleFactor` to the internally computed values, and saturate the results before writing them to the destination image ROI.

When the overflow occurs, the resultant value is determined in accordance with the data type (see appendix A “Handling of Special Cases” for more information).

**Return Values**

- `ippStsNoErr` Indicates no error. Any other value indicates an error or a warning.
- `ippStsNullPtrErr` Indicates an error condition if one of the specified pointers is NULL.
- `ippStsSizeErr` Indicates an error condition if `roiSize` has a field with zero or negative value.
- `ippStsStepErr` Indicates an error condition if `srcStep`, `dstStep`, or `srcDstStep` has a zero or negative value.

**DotProd**

*Computes the dot product of pixel values of two source images.*

**Syntax**

**Case 1: Operation on one-channel integer data**

```c
IppStatus ippiDotProd_<mod>(const Ipp<srcDatatype>* pSrc1, int src1Step, const Ipp<srcDatatype>* pSrc2, int src2Step, IppiSize roiSize, Ipp64f* pDp);
```

Supported values for `mod`:

- 8u64f_C1R
- 16u64f_C1R
- 16s64f_C1R
- 32u64f_C1R
- 32s64f_C1R
Case 2: Operation on three-channel integer data

IppStatusippiDotProd_<mod>(const Ipp<srcDatatype>* pSrc1, int src1Step, const Ipp<srcDatatype>* pSrc2, int src2Step, IppiSize roiSize, Ipp64f pDp[3]);

Supported values for mod:

8u64f_C3R    16u64f_C3R    16s64f_C3R    32u64f_C3R    32s64f_C3R

Case 3: Operation on four-channel integer data

IppStatusippiDotProd_<mod>(const Ipp<srcDatatype>* pSrc1, int src1Step, const Ipp<srcDatatype>* pSrc2, int src2Step, IppiSize roiSize, Ipp64f pDp[4]);

Supported values for mod:

8u64f_C4R    16u64f_C4R    16s64f_C4R    32u64f_C4R    32s64f_C4R

IppStatusippiDotProd_<mod>(const Ipp<srcDatatype>* pSrc1, int src1Step, const Ipp<srcDatatype>* pSrc2, int src2Step, IppiSize roiSize, Ipp64f pDp[3]);

Supported values for mod:

8u64f_AC4R   16u64f_AC4R   32u64f_AC4R
16s64f_AC4R   32s64f_AC4R

Case 4: Operation on floating-point data

IppStatusippiDotProd_32f64f_C1R(const Ipp32f* pSrc1, int src1Step, const Ipp32f* pSrc2, int src2Step, IppiSize roiSize, Ipp64f* pDp, IppHintAlgorithm hint);

IppStatusippiDotProd_32f64f_C3R(const Ipp32f* pSrc1, int src1Step, const Ipp32f* pSrc2, int src2Step, IppiSize roiSize, Ipp64f pDp[3], IppHintAlgorithm hint);

IppStatusippiDotProd_32f64f_C4R(const Ipp32f* pSrc1, int src1Step, const Ipp32f* pSrc2, int src2Step, IppiSize roiSize, Ipp64f pDp[4], IppHintAlgorithm hint);

IppStatusippiDotProd_32f64f_AC4R(const Ipp32f* pSrc1, int src1Step, const Ipp32f* pSrc2, int src2Step, IppiSize roiSize, Ipp64f pDp[3], IppHintAlgorithm hint);

Include Files

ippi.h

Domain Dependencies

Headers: ippcore.h, ippvm.h, ipps.h
Libraries: ippcore.lib, ippvm.lib, ipps.lib

Parameters

pSrc1, pSrc2
src1Step, src2Step
roiSize

Pointer to the ROI in the source images.
Distance, in bytes, between the starting points of consecutive lines in the source images.
Size of the source image ROI.
pDp  

Pointer to the dot product or to the array containing the computed dot products for multi-channel images.

hint  

Option to select the algorithmic implementation of the function, see Table “Hint Arguments for Image Moment Functions”.

Description

This function operates with ROI (see Regions of Interest in Intel IPP).

This function computes the dot product of pixel values of two source images \( p\text{Src1} \) and \( p\text{Src2} \) using algorithm indicated by the hint argument (see Table “Hint Arguments for Image Moment Functions”) and stores the result in \( pDp \). In case of multi-channel images, the dot product is computed for each channel separately and stored in the array \( pDp \).

Return Values

ippStsNoErr  
Indicates no error. Any other value indicates an error.

ippStsNullPtrErr  
Indicates an error condition if one of the specified pointers is NULL.

ippStsSizeErr  
Indicates an error condition if roiSize has a field with zero or negative value.

ippStsStepErr  
Indicates an error condition if one of the step values has zero or negative value.

DotProdCol  

*Calculates the dot product of taps vector and columns of the specified set of rows.*

Syntax

IppStatus ippiDotProdCol_32f_L2(const Ipp32f* const ppSrcRow[], const Ipp32f* pTaps, int tapsLen, Ipp32f* pDst, int width);

IppStatus ippiDotProdCol_64f_L2(const Ipp64f* const ppSrcRow[], const Ipp64f* pTaps, int tapsLen, Ipp64f* pDst, int width);

Include Files

ippi.h

Domain Dependencies

Headers: ippicore.h, ippvm.h, ipps.h

Libraries: ippicore.lib, ippvm.lib, ipps.lib

Parameters

\( pp\text{SrcRow} \)  
Pointer to the set of rows.

\( p\text{Taps} \)  
Pointer to the taps vector.

\( \text{tapsLen} \)  
Length of taps vector, is equal to the number of rows.

\( p\text{Dst} \)  
Pointer to the destination row.

\( \text{width} \)  
Width of the source and destination rows.
Description

This function calculates the dot product of taps vector \( pTaps \) and columns of the specified set of the rows \( ppSrcRow \). It is useful for external vertical filtering pipeline implementation.

Return Values

- \texttt{ippStsNoErr} Indicates no error. Any other value indicates an error.
- \texttt{ippStsNullPtrErr} Indicates an error condition if one of the specified pointers is NULL.
- \texttt{ippStsSizeErr} Indicates an error condition if \( width \) is less than or equal to 0.

Logical Operations

Functions described in this section perform bitwise operations on pixel values. The operations include logical AND, NOT, inclusive OR, exclusive OR, and bit shifts.

And

*Performs a bitwise AND operation between corresponding pixels of two images.*

Syntax

Case 1: Not-in-place operation

\[
\text{IppStatus ippiAnd\_<mod>}(\text{const Ipp<datatype>* } pSrc1, \text{int src1Step, const Ipp<datatype>* } pSrc2, \text{int src2Step, Ipp<datatype>* } pDst, \text{int dstStep, IppSize roiSize});
\]

Supported values for \texttt{mod}:

- 8u\_C1R  16u\_C1R  32s\_C1R
- 8u\_C3R  16u\_C3R  32s\_C3R
- 8u\_C4R  16u\_C4R  32s\_C4R
- 8u\_AC4R  16u\_AC4R  32s\_AC4R

Case 2: In-place operation

\[
\text{IppStatus ippiAnd\_<mod>}(\text{const Ipp<datatype>* } pSrc, \text{int srcStep, Ipp<datatype>* } pSrcDst, \text{int srcDstStep, IppSize roiSize});
\]

Supported values for \texttt{mod}:

- 8u\_C1IR  16u\_C1IR  32s\_C1IR
- 8u\_C3IR  16u\_C3IR  32s\_C3IR
- 8u\_C4IR  16u\_C4IR  32s\_C4IR
- 8u\_AC4IR  16u\_AC4IR  32s\_AC4IR
Include Files
ippi.h

Domain Dependencies
Headers: ippcore.h, ippvm.h, ipps.h
Libraries: ippcore.lib, ippvm.lib, ipps.lib

Parameters
pSrc, pSrc1, pSrc2  Pointers to the source images ROI.
srcStep, src1Step, src2Step  Distances in bytes between starts of consecutive lines in the source images.
pDst  Pointer to the destination image ROI.
dstStep  Distance in bytes between starts of consecutive lines in the destination image.
pSrcDst  Pointer to the source and destination image ROI for the in-place operation.
srcDstStep  Distance in bytes between starts of consecutive lines in the source and destination images for the in-place operation.
roiSize  Size of the source and destination ROI in pixels.

Description
This function operates with ROI (see Regions of Interest in Intel IPP).
This function performs a bitwise AND operation between the values of corresponding pixels of two source image ROIs, and writes the result into a destination image ROI.
Note that the functions with the AC4 descriptor do not process alpha channels.

Return Values
ippStsNoErr  Indicates no error. Any other value indicates an error or a warning.
ippStsNullPtrErr  Indicates an error condition if one of the specified pointers is NULL.
ippStsSizeErr  Indicates an error condition if roiSize has a field with zero or negative value.
ippStsStepErr  Indicates an error condition if any of the specified buffer step values is zero or negative.

AndC
Performs a bitwise AND operation of each pixel with a constant.
### Syntax

**Case 1: Not-in-place operation on one-channel data**

```
IppStatus ippiAndC_<mod>(const Ipp<datatype>* pSrc, int srcStep, Ipp<datatype> value, Ipp<datatype>* pDst, int dstStep, IppiSize roiSize);
```

Supported values for `mod`:

- 8u_C1R
- 16u_C1R
- 32s_C1R

**Case 2: Not-in-place operation on multi-channel data**

```
IppStatus ippiAndC_<mod>(const Ipp<datatype>* pSrc, int srcStep, const Ipp<datatype> value[3], Ipp<datatype>* pDst, int dstStep, IppiSize roiSize);
```

Supported values for `mod`:

- 8u_C3R
- 16u_C3R
- 32s_C3R
- 8u_AC4R
- 16u_AC4R
- 32s_AC4R

**Case 3: In-place operation on one-channel data**

```
IppStatus ippiAndC_<mod>(Ipp<datatype> value, Ipp<datatype>* pSrcDst, int srcDstStep, IppiSize roiSize);
```

Supported values for `mod`:

- 8u_C1IR
- 16u_C1IR
- 32s_C1IR

**Case 4: In-place operation on multi-channel data**

```
IppStatus ippiAndC_<mod>(const Ipp<datatype> value[3], Ipp<datatype>* pSrcDst, int srcDstStep, IppiSize roiSize);
```

Supported values for `mod`:

- 8u_C3IR
- 16u_C3IR
- 32s_C3IR
- 8u_AC4IR
- 16u_AC4IR
- 32s_AC4IR

- 8u_C4IR
- 16u_C4IR
- 32s_C4IR

**Include Files**

ippi.h
Domain Dependencies

Headers: ippcore.h, ippvm.h, ipps.h
Libraries: ippcore.lib, ippvm.lib, ipps.lib

Parameters

pSrc  
Pointer to the source image ROI.

srcStep  
Distance in bytes between starts of consecutive lines in the source image.

value  
The constant value to perform the bitwise AND operation on each pixel of the source image ROI (constant vector in case of multi-channel images).

pDst  
Pointer to the destination image ROI.

dstStep  
Distance in bytes between starts of consecutive lines in the destination image.

pSrcDst  
Pointer to the source and destination image ROI for the in-place operation.

srcDstStep  
Distance in bytes between starts of consecutive lines in the source and destination image buffer for the in-place operation.

roiSize  
Size of the source and destination ROI in pixels.

Description

This function operates with ROI (see Regions of Interest in Intel IPP).

This function performs a bitwise AND operation between each pixel value of a source image ROI and constant value.

Note that the functions with the AC4 descriptor do not process alpha channels.

Return Values

ippStsNoErr  
Indicates no error. Any other value indicates an error or a warning.

ippStsNullPtrErr  
Indicates an error condition if one of the specified pointers is NULL.

ippStsSizeErr  
Indicates an error condition if roiSize has a field with a zero or negative value.

ippStsStepErr  
Indicates an error condition if srcStep, dstStep, or srcDstStep has a zero or negative value.

Or

Performs bitwise inclusive OR operation between pixels of two source buffers.
Syntax

Case 1: Not-in-place operation

IppStatus ippiOr_<mod>(const Ipp<datatype>* pSrc1, int src1Step, const Ipp<datatype>* pSrc2, int src2Step, Ipp<datatype>* pDst, int dstStep, IppiSize roiSize);

Supported values for mod:

- 8u_C1R
- 16u_C1R
- 32s_C1R
- 8u_C3R
- 16u_C3R
- 32s_C3R
- 8u_C4R
- 16u_C4R
- 32s_C4R
- 8u_AC4R
- 16u_AC4R
- 32s_AC4R

Case 2: In-place operation

IppStatus ippiOr_<mod>(const Ipp<datatype>* pSrc, int srcStep, Ipp<datatype>* pSrcDst, int srcDstStep, IppiSize roiSize);

Supported values for mod:

- 8u_C1IR
- 16u_C1IR
- 32s_C1IR
- 8u_C3IR
- 16u_C3IR
- 32s_C3IR
- 8u_C4IR
- 16u_C4IR
- 32s_C4IR
- 8u_AC4IR
- 16u_AC4IR
- 32s_AC4IR

Include Files

ippi.h

Domain Dependencies

Headers: ippcore.h, ippvm.h, ipps.h
Libraries: ippcore.lib, ippvm.lib, ipps.lib

Parameters

- pSrc, pSrc1, pSrc2: Pointers to the source images ROI.
- srcStep, src1Step, src2Step: Distances in bytes between starts of consecutive lines in the source images.
- pDst: Pointer to the destination image ROI.
- dstStep: Distance in bytes between starts of consecutive lines in the destination image.
- pSrcDst: Pointer to the source and destination image ROI for the in-place operation.
- srcDstStep: Distance in bytes between starts of consecutive lines in the source and destination image for the in-place operation.
- roiSize: Size of the source and destination ROI in pixels.

Description

This function operates with ROI (see Regions of Interest in Intel IPP).
This function performs a bitwise inclusive OR operation between the values of corresponding pixels of two source image ROIs, and writes the result into a destination image ROI. Note that the functions with the AC4 descriptor do not process alpha channels.

**Return Values**

- **ippStsNoErr**
  - Indicates no error. Any other value indicates an error or a warning.
- **ippStsNullPtrErr**
  - Indicates an error condition if one of the specified pointers is NULL.
- **ippStsSizeErr**
  - Indicates an error condition if roiSize has a field with zero or negative value.
- **ippStsStepErr**
  - Indicates an error condition if any of the specified buffer step values is zero or negative.

**Example**

The code example below show how to use the function `ippiOr_8u_C1R`.

```c
void func_or()
{
    IppiSize Src1ROI = {8,4};
    IppiSize Src2ROI = {8,4};
    IppiSize DstROI = {5,4};
    Ipp8u src1[8*4];
    Ipp8u src2[8*4];
    Ipp8u dst[8*4];

    ippiSet_8u_C1R(0,dest,8,Src1ROI);
    ippiSet_8u_C1R(5,src1,8,Src1ROI);
    ippiSet_8u_C1R(6,src2,8,Src2ROI);
    ippiOr_8u_C1R(src1,8,src2,8,dst,8,DstROI);
}
```

**Result:**

```
<table>
<thead>
<tr>
<th>src1</th>
<th>src2</th>
<th>dst</th>
</tr>
</thead>
<tbody>
<tr>
<td>05</td>
<td>05</td>
<td>05</td>
</tr>
<tr>
<td>05</td>
<td>05</td>
<td>05</td>
</tr>
<tr>
<td>05</td>
<td>05</td>
<td>05</td>
</tr>
<tr>
<td>05</td>
<td>05</td>
<td>05</td>
</tr>
</tbody>
</table>
```

**OrC**

*Performs a bitwise inclusive OR operation between each pixel of a buffer and a constant.*
Syntax

Case 1: Not-in-place operation on one-channel data

IppStatus ippiOrC_<mod>(const Ipp<datatype>* pSrc, int srcStep, Ipp<datatype> value, Ipp<datatype>* pDst, int dstStep, IppiSize roiSize);

Supported values for mod:

- 8u_C1R
- 16u_C1R
- 32s_C1R

Case 2: Not-in-place operation on multi-channel data

IppStatus ippiOrC_<mod>(const Ipp<datatype>* pSrc, int srcStep, const Ipp<datatype> value[3], Ipp<datatype>* pDst, int dstStep, IppiSize roiSize);

Supported values for mod:

- 8u_C3R
- 16u_C3R
- 32s_C3R
- 8u_AC4R
- 16u_AC4R
- 32s_AC4R

IppStatus ippiOrC_<mod>(const Ipp<datatype>* pSrc, int srcStep, const Ipp<datatype> value[4], Ipp<datatype>* pDst, int dstStep, IppiSize roiSize);

Supported values for mod:

- 8u_C4R
- 16u_C4R
- 32s_C4R

Case 3: In-place operation on one-channel data

IppStatus ippiOrC_<mod>(Ipp<datatype> value, Ipp<datatype>* pSrcDst, int srcDstStep, IppiSize roiSize);

Supported values for mod:

- 8u_C1IR
- 16u_C1IR
- 32s_C1IR

Case 4: In-place operation on multi-channel data

IppStatus ippiOrC_<mod>(const Ipp<datatype> value[3], Ipp<datatype>* pSrcDst, int srcDstStep, IppiSize roiSize);

Supported values for mod:

- 8u_C3IR
- 16u_C3IR
- 32s_C3IR
- 8u_AC4IR
- 16u_AC4IR
- 32s_AC4IR

IppStatus ippiOrC_<mod>(const Ipp<datatype> value[4], Ipp<datatype>* pSrcDst, int srcDstStep, IppiSize roiSize);

Supported values for mod:

- 8u_C4IR
- 16u_C4IR
- 32s_C4IR

Include Files

ippi.h
Domain Dependencies

Headers: ippcore.h, ippvm.h, ipps.h
Libraries: ippcore.lib, ippvm.lib, ipps.lib

Parameters

- **pSrc**: Pointer to the source image ROI.
- **srcStep**: Distance in bytes between starts of consecutive lines in the source image.
- **value**: The constant value to perform the bitwise OR operation on each pixel of the source image (constant vector in case of multi-channel images).
- **pDst**: Pointer to the destination image ROI.
- **dstStep**: Distance in bytes between starts of consecutive lines in the destination image.
- **pSrcDst**: Pointer to the source and destination image ROI for the in-place operation.
- **srcDstStep**: Distance in bytes between starts of consecutive lines in the source and destination image for the in-place operation.
- **roiSize**: Size of the source and destination ROI in pixels.

Description

This function operates with ROI (see Regions of Interest in Intel IPP).
This function performs a bitwise inclusive OR operation between each pixel value of a source image ROI and constant value.
Note that the functions with the AC4 descriptor do not process alpha channels.

Return Values

- **ippStsNoErr**: Indicates no error. Any other value indicates an error or a warning.
- **ippStsNullPtrErr**: Indicates an error condition if one of the specified pointers is NULL.
- **ippStsSizeErr**: Indicates an error condition if roiSize has a field with zero or negative value.
- **ippStsStepErr**: Indicates an error condition if srcStep, dstStep, or srcDstStep has a zero or negative value.

Xor

Performs bitwise exclusive OR operation between pixels of two source buffers.
Syntax

Case 1: Not-in-place operation

IppStatus ippiXor_<mod>(
const Ipp<datatype>* pSrc1, int src1Step, const Ipp<datatype>* pSrc2, int src2Step, Ipp<datatype>* pDst, int dstStep, IppiSize roiSize);

Supported values for mod:

- 8u_C1R
- 16u_C1R
- 32s_C1R
- 8u_C3R
- 16u_C3R
- 32s_C3R
- 8u_C4R
- 16u_C4R
- 32s_C4R
- 8u_AC4R
- 16u_AC4R
- 32s_AC4R

Case 2: In-place operation

IppStatus ippiXor_<mod>(
const Ipp<datatype>* pSrc, int srcStep, Ipp<datatype>* pSrcDst, int srcDstStep, IppiSize roiSize);

Supported values for mod:

- 8u_C1IR
- 16u_C1IR
- 32s_C1IR
- 8u_C3IR
- 16u_C3IR
- 32s_C3IR
- 8u_C4IR
- 16u_C4IR
- 32s_C4IR
- 8u_AC4IR
- 16u_AC4IR
- 32s_AC4IR

Include Files

ippi.h

Domain Dependencies

Headers: ippcore.h, ippvm.h, ipps.h
Libraries: ippcore.lib, ippvm.lib, ipps.lib

Parameters

- pSrc, pSrc1, pSrc2: Pointers to the source images ROI.
- srcStep, src1Step, src2Step: Distances in bytes between starts of consecutive lines in the source images.
- pDst: Pointer to the destination image ROI.
- dstStep: Distance in bytes between starts of consecutive lines in the destination image.
- pSrcDst: Pointer to the source and destination image ROI for the in-place operation.
- srcDstStep: Distance in bytes between starts of consecutive lines in the source and destination image for the in-place operation.
- roiSize: Size of the source and destination ROI in pixels.

Description

This function operates with ROI (see Regions of Interest in Intel IPP).
This function performs a bitwise exclusive OR operation between the values of corresponding pixels of two source image ROIs, and writes the result into a destination image ROI.

Note that the functions with the AC4 descriptor do not process alpha channels.

Return Values

- `ippStsNoErr` Indicates no error. Any other value indicates an error or a warning.
- `ippStsNullPtrErr` Indicates an error condition if one of the specified pointers is NULL.
- `ippStsSizeErr` Indicates an error condition if `roiSize` has a field with zero or negative value.
- `ippStsStepErr` Indicates an error condition if any of the specified buffer step values is zero or negative.

**XorC**

*Performs a bitwise exclusive OR operation between each pixel of a buffer and a constant.*

**Syntax**

**Case 1: Not-in-place operation on one-channel data**

```c
IppStatus ippiXorC_<mod>(const Ipp<datatype>* pSrc, int srcStep, Ipp<datatype> value, Ipp<datatype>* pDst, int dstStep, IppiSize roiSize);
```

Supported values for mod:

- 8u_C1R  16u_C1R  32s_C1R

**Case 2: Not-in-place operation on multi-channel data**

```c
IppStatus ippiXorC_<mod>(const Ipp<datatype>* pSrc, int srcStep, const Ipp<datatype> value[3], Ipp<datatype>* pDst, int dstStep, IppiSize roiSize);
```

Supported values for mod:

- 8u_C3R  16u_C3R  32s_C3R
- 8u_AC4R  16u_AC4R  32s_AC4R

```c
IppStatus ippiXorC_<mod>(const Ipp<datatype>* pSrc, int srcStep, const Ipp<datatype> value[4], Ipp<datatype>* pDst, int dstStep, IppiSize roiSize);
```

Supported values for mod:

- 8u_C4R  16u_C4R  32s_C4R
Case 3: In-place operation on one-channel data

IppStatus ippiXorC_<mod>(Ipp<datatype> value, Ipp<datatype>* pSrcDst, int srcDstStep, IppiSize roiSize);

Supported values for mod:

  8u_C1IR  16u_C1IR  32s_C1IR

Case 4: In-place operation on multi-channel data

IppStatus ippiXorC_<mod>(const Ipp<datatype> value[3], Ipp<datatype>* pSrcDst, int srcDstStep, IppiSize roiSize);

Supported values for mod:

  8u_C3IR  16u_C3IR  32s_C3IR
  8u_AC4IR 16u_AC4IR  32s_AC4IR

IppStatus ippiXorC_<mod>(const Ipp<datatype> value[4], Ipp<datatype>* pSrcDst, int srcDstStep, IppiSize roiSize);

Supported values for mod:

  8u_C4IR  16u_C4IR  32s_C4IR

Include Files

ippi.h

Domain Dependencies

Headers: ippcore.h, ippvm.h, ipps.h
Libraries: ippcore.lib, ippvm.lib, ipps.lib

Parameters

pSrc
Pointer to the source image ROI.

srcStep
Distance in bytes between starts of consecutive lines in the source image.

value
The constant value to perform the bitwise XOR operation on each pixel of the source image ROI (constant vector in case of multi-channel images).

pDst
Pointer to the destination image ROI.

dstStep
Distance in bytes between starts of consecutive lines in the destination image.

pSrcDst
Pointer to the source and destination image ROI for the in-place operation.

srcDstStep
Distance in bytes between starts of consecutive lines in the source and destination image for the in-place operation.

roiSize
Size of the source and destination ROI in pixels.

Description

This function operates with ROI (see Regions of Interest in Intel IPP).
This function performs a bitwise exclusive OR operation between each pixel value of a source image ROI and constant value.

Note that the functions with the AC4 descriptor do not process alpha channels.

**Return Values**

- ippStsNoErr: Indicates no error. Any other value indicates an error or a warning.
- ippStsNullPtrErr: Indicates an error condition if one of the specified pointers is NULL.
- ippStsSizeErr: Indicates an error condition if roiSize has a field with zero or negative value.
- ippStsStepErr: Indicates an error condition if srcStep, dstStep, or srcDstStep has a zero or negative value.

**Not**

*Performs a bitwise NOT operation on each pixel of a source buffer.*

**Syntax**

**Case 1: Not-in-place operation**

```c
IppStatusippiNot_<mod>(const Ipp8u* pSrc, int srcStep, Ipp8u* pDst, int dstStep, IppiSize roiSize);
```

Supported values for `mod`:

- 8u_C1R
- 8u_C3R
- 8u_C4R
- 8u_AC4R

**Case 2: In-place operation**

```c
IppStatusippiNot_<mod>(Ipp8u* pSrcDst, int srcDstStep, IppiSize roiSize);
```

Supported values for `mod`:

- 8u_C1IR
- 8u_C3IR
- 8u_C4IR
- 8u_AC4IR

**Include Files**

ippi.h

**Domain Dependencies**

Headers: ippcore.h, ippvm.h, ipps.h
Libraries: ippcore.lib, ippvm.lib, ipps.lib

**Parameters**

*pSrc*  
Pointer to the source image ROI.

*srcStep*  
Distance in bytes between starts of consecutive lines in the source image.

*pDst*  
Pointer to the destination image ROI.

*dstStep*  
Distance in bytes between starts of consecutive lines in the destination image.

*pSrcDst*  
Pointer to the source and destination image ROI for the in-place operation.

*srcDstStep*  
Distance in bytes between starts of consecutive lines in the source and destination image for the in-place operation.

*roiSize*  
Size of the source and destination ROI in pixels.

**Description**

This function operates with ROI (see Regions of Interest in Intel IPP).

This function performs a bitwise NOT operation on each pixel value of a source image ROI.

Note that the functions with the AC4 descriptor do not process alpha channels.

**Return Values**

- **ippStsNoErr**
  
  Indicates no error. Any other value indicates an error or a warning.

- **ippStsNullPtrErr**
  
  Indicates an error condition if one of the specified pointers is NULL.

- **ippStsSizeErr**
  
  Indicates an error condition if *roiSize* has a field with zero or negative value.

- **ippStsStepErr**
  
  Indicates an error condition if *srcStep*, *dstStep*, or *srcDstStep* has a zero or negative value.

**RShiftC**

*Shifts bits in pixel values to the right.*

**Case 1: Not-in-place operation on one-channel data**

```c
IppStatus ippiRShiftC_<mod>(const Ipp<datatype>* pSrc, int srcStep, Ipp32u value,
Ipp<datatype>* pDst, int dstStep, IppiSize roiSize);
```

Supported values for *mod*:

- 8u_C1R
- 16u_C1R
- 16s_C1R
- 32s_C1R
Case 2: Not-in-place operation on multi-channel data

IppStatus ippiRShiftC_<mod>(const Ipp<datatype>* pSrc, int srcStep, const Ipp32u value[3], Ipp<datatype>* pDst, int dstStep, IppiSize roiSize);

Supported values for mod:

8u_C3R     16u_C3R     16s_C3R     32s_C3R

IppStatus ippiRShiftC_<mod>(const Ipp<datatype>* pSrc, int srcStep, const Ipp32u value[4], Ipp<datatype>* pDst, int dstStep, IppiSize roiSize);

Supported values for mod:

8u_C4R     16u_C4R     16s_C4R     32s_C4R

IppStatus ippiRShiftC_<mod>(const Ipp<datatype>* pSrc, int srcStep, const Ipp32u value[3], Ipp<datatype>* pDst, int dstStep, IppiSize roiSize);

Supported values for mod:

8u_AC4R    16u_AC4R    16s_AC4R    32s_AC4R

Case 3: In-place operation on one-channel data

IppStatus ippiRShiftC_<mod>(Ipp32u value, Ipp<datatype>* pSrcDst, int srcDstStep, IppiSize roiSize);

Supported values for mod:

8u_C1IR    16u_C1IR    16s_C1IR    32s_C1IR

Case 4: In-place operation on multi-channel data

IppStatus ippiRShiftC_<mod>(const Ipp32u value[3], Ipp<datatype>* pSrcDst, int srcDstStep, IppiSize roiSize);

Supported values for mod:

8u_C3IR    16u_C3IR    16s_C3IR    32s_C3IR
8u_AC4IR    16u_AC4IR    16s_AC4IR    32s_AC4IR

IppStatus ippiRShiftC_<mod>(const Ipp32u value[4], Ipp<datatype>* pSrcDst, int srcDstStep, IppiSize roiSize);

Supported values for mod:

8u_C4IR    16u_C4IR    16s_C4IR    32s_C4IR

Include Files

ippi.h
Domain Dependencies

Headers: ippcore.h, ippvm.h, ipps.h

Libraries: ippcore.lib, ippvm.lib, ipps.lib

Parameters

pSrc  
Pointer to the source image ROI.

srcStep  
Distance in bytes between starts of consecutive lines in the source image.

value  
The number of bits to shift (constant vector in case of multi-channel images).

pDst  
Pointer to the destination image ROI.

dstStep  
Distance in bytes between starts of consecutive lines in the destination image.

pSrcDst  
Pointer to the source and destination image ROI for the in-place operation.

srcDstStep  
Distance in bytes between starts of consecutive lines in the source and destination image for the in-place operation.

roiSize  
Size of the source and destination ROI in pixels.

Description

This function operates with ROI (see Regions of Interest in Intel IPP).

This function decreases the intensity of pixels in the source image ROI by shifting the bits in each pixel value by value bits to the right. The positions vacated after shifting the bits are filled with the sign bit. In case of multi-channel data, each color channel can have its own shift value. This operation is equivalent to dividing the pixel values by a constant power of 2.

Note that the functions with the AC4 descriptor do not process alpha channels.

Return Values

ippStsNoErr  
Indicates no error. Any other value indicates an error or a warning.

ippStsNullPtrErr  
Indicates an error condition if one of the specified pointers is NULL.

ippStsSizeErr  
Indicates an error condition if roiSize has a field with a zero or negative value.

ippStsStepErr  
Indicates an error condition if srcStep, dstStep, or srcDstStep has a zero or negative value.

LShiftC

Shifts bits in pixel values to the left.
Syntax

Case 1: Not-in-place operation on one-channel data

IppStatus ippiLShiftC_<mod>(const Ipp<datatype>* pSrc, int srcStep, Ipp32u value, Ipp<datatype>* pDst, int dstStep, IppiSize roiSize);

Supported values for mod:
- 8u_C1R
- 16u_C1R
- 32s_C1R

Case 2: Not-in-place operation on multi-channel data

IppStatus ippiLShiftC_<mod>(const Ipp<datatype>* pSrc, int srcStep, const Ipp32u value[3], Ipp<datatype>* pDst, int dstStep, IppiSize roiSize);

Supported values for mod:
- 8u_C3R
- 16u_C3R
- 32s_C3R
- 32s_AC4R
- 8u_AC4R
- 16u_AC4R

IppStatus ippiLShiftC_<mod>(const Ipp<datatype>* pSrc, int srcStep, const Ipp32u value[4], Ipp<datatype>* pDst, int dstStep, IppiSize roiSize);

Supported values for mod:
- 8u_C4R
- 16u_C4R
- 32s_C4R

Case 3: In-place operation on one-channel data

IppStatus ippiLShiftC_<mod>(Ipp32u value, Ipp<datatype>* pSrcDst, int srcDstStep, IppiSize roiSize);

Supported values for mod:
- 8u_C1IR
- 16u_C1IR
- 32s_C1IR

Case 4: In-place operation on multi-channel data

IppStatus ippiLShiftC_<mod>(const Ipp32u value[3], Ipp<datatype>* pSrcDst, int srcDstStep, IppiSize roiSize);

Supported values for mod:
- 8u_C3IR
- 16u_C3IR
- 32s_C3IR
- 8u_AC4IR
- 16u_AC4IR
- 32s_AC4IR

IppStatus ippiLShiftC_<mod>(const Ipp32u value[4], Ipp<datatype>* pSrcDst, int srcDstStep, IppiSize roiSize);

Supported values for mod:
- 8u_C4IR
- 16u_C4IR
- 32s_C4IR

Include Files

ippi.h
Domain Dependencies

Headers: ippcore.h, ippvm.h, ipps.h

Libraries: ippcore.lib, ippvm.lib, ipps.lib

Parameters

- **pSrc**
  Pointer to the source image ROI.

- **srcStep**
  Distance in bytes between starts of consecutive lines in the source image.

- **value**
  The number of bits to shift (constant vector in case of multi-channel images).

- **pDst**
  Pointer to the destination image ROI.

- **dstStep**
  Distance in bytes between starts of consecutive lines in the destination image.

- **pSrcDst**
  Pointer to the source and destination image ROI for the in-place operation.

- **srcDstStep**
  Distance in bytes between starts of consecutive lines in the source and destination image for the in-place operation.

- **roiSize**
  Size of the source and destination ROI in pixels.

Description

This function operates with ROI (see Regions of Interest in Intel IPP).

This function changes the intensity of pixels in the source image ROI by shifting the bits in each pixel value by **value** bits to the left. In case of multi-channel data, each color channel can have its own shift value. The positions vacated after shifting the bits are filled with zeros. Values obtained as a result of left shift operations are not saturated. To get saturated values, use multiplication functions instead.

Note that the functions with the AC4 descriptor do not process alpha channels.

Return Values

- **ippStsNoErr**
  Indicates no error. Any other value indicates an error or a warning.

- **ippStsNullPtrErr**
  Indicates an error condition if one of the specified pointers is NULL.

- **ippStsSizeErr**
  Indicates an error condition if **roiSize** has a field with zero or negative value.

- **ippStsStepErr**
  Indicates an error condition if **srcStep**, **dstStep**, or **srcDstStep** has a zero or negative value.

Example

The code example below illustrates the use of left shift function.

```c
IppStatus lshift( void ) {
    Ipp8u img[8*8] = { 1, 0x7F, 0xFE };
    IppiSize roi = { 8, 8 };
    IppStatus st = ippiLShiftC_8u_C1IR( 1, img, 8, roi );
    printf( "%02x %02x %02x\n", img[0], img[1], img[2] );
    return st;
}
```
Alpha Composition

The Intel IPP provides functions that composite two image buffers using either the opacity (alpha) channel in the images or provided alpha values.

These functions operate on image buffers with 8-bit or 16-bit data in RGB or RGBA format. In all compositing operations a resultant pixel in destination buffer $p_{Dst}$ is created by overlaying a pixel from the foreground image buffer $p_{Src1}$ over a pixel from the background image buffer $p_{Src2}$. The supported types of images combining by using alpha values are listed in the table below.

<table>
<thead>
<tr>
<th>Type</th>
<th>Color Components</th>
<th>Output Pixel</th>
<th>Description in Imaging Terms</th>
</tr>
</thead>
<tbody>
<tr>
<td>OVER</td>
<td>$a_{A}*A+(1-a_{A})*a_{B}*B$</td>
<td>$a_{A}+(1-a_{A})*a_{B}$</td>
<td>$A$ occludes $B$</td>
</tr>
<tr>
<td>IN</td>
<td>$a_{A}<em>A</em>a_{B}$</td>
<td>$a_{A}*a_{B}$</td>
<td>$A$ within $B$. $A$ acts as a matte for $B$. $A$ shows only where $B$ is visible.</td>
</tr>
<tr>
<td>OUT</td>
<td>$a_{A}<em>A</em>(1-a_{B})$</td>
<td>$a_{A}*(1-a_{B})$</td>
<td>$A$ outside $B$. NOT-$B$ acts as a matte for $A$. $A$ shows only where $B$ is not visible.</td>
</tr>
<tr>
<td>ATOP</td>
<td>$a_{A}<em>A</em>a_{B}+(1-a_{A})*a_{B}*B$</td>
<td>$a_{A}*a_{B}+(1-a_{A})*a_{B}$</td>
<td>Combination of ($A$ IN $B$) and ($B$ OUT $A$). $B$ is both background and matte for $A$.</td>
</tr>
<tr>
<td>XOR</td>
<td>$a_{A}<em>A</em>(1-a_{B})+(1-a_{A})*a_{B}*B$</td>
<td>$a_{A}*(1-a_{B})+(1-a_{A})*a_{B}$</td>
<td>Combination of ($A$ OUT $B$) and ($B$ OUT $A$). $A$ and $B$ mutually exclude each other.</td>
</tr>
<tr>
<td>PLUS</td>
<td>$a_{A}*A+a_{B}*B$</td>
<td>$a_{A}+a_{B}$</td>
<td>Blend without precedence</td>
</tr>
</tbody>
</table>

In the formulas above, the input image buffers are denoted as $A$ and $B$ for simplicity. The Greek letter $\alpha$ with subscripts denotes the normalized (scaled) alpha value in the range 0 to 1. It is related to the integer alpha value $\text{alpha}$ as:

$$\alpha = \frac{\text{alpha}}{\max_{\text{val}}}$$

where $\max_{\text{val}}$ is 255 for 8-bit or 65535 for 16-bit unsigned pixel data.

For the ippiAlphaComp function that operates on 4-channel RGBA buffers only, $a_{A}$ and $a_{B}$ are the normalized alpha values of the two input image buffers, respectively.

For the ippiAlphaCompC function, $a_{A}$ and $a_{B}$ are the normalized constant alpha values that are passed as parameters to the function.

Note that in formulas for computing the resultant color channel values, $A$ and $B$ stand for the pixel color components of the respective input image buffers.

To save a significant amount of computation for some of the alpha compositing operations, use functions AlphaPremul, AlphaPremulC for pre-multiplying color channel values by the alpha values. This reduces the number of multiplications required in the compositing operations, which is especially efficient for repeated compositing of an image.

The type of composition operation that is performed by the function AlphaComp and AlphaCompC is specified by the parameter $\text{alphaType}$, the table below lists its possible values.
### Possible Values of alphaType Parameter

<table>
<thead>
<tr>
<th>Operation types</th>
<th>Parameter Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>OVER</td>
<td>ippAlphaOver</td>
</tr>
<tr>
<td></td>
<td>ippAlphaOverPremul</td>
</tr>
<tr>
<td>IN</td>
<td>ippAlphaIn</td>
</tr>
<tr>
<td></td>
<td>ippAlphaInPremul</td>
</tr>
<tr>
<td>OUT</td>
<td>ippAlphaOut</td>
</tr>
<tr>
<td></td>
<td>ippAlphaOutPremul</td>
</tr>
<tr>
<td>ATOP</td>
<td>ippAlphaATop</td>
</tr>
<tr>
<td></td>
<td>ippAlphaATopPremul</td>
</tr>
<tr>
<td>XOR</td>
<td>ippAlphaXor</td>
</tr>
<tr>
<td></td>
<td>ippAlphaXorPremul</td>
</tr>
<tr>
<td>PLUS</td>
<td>ippAlphaPlus</td>
</tr>
<tr>
<td></td>
<td>ippAlphaPlusPremul</td>
</tr>
</tbody>
</table>

### AlphaComp

*Combines two images using alpha (opacity) values of both images.*

**Syntax**

**Case 1: Not-in-place operation**

```c
IppStatus ippiAlphaComp_<mod>(const Ipp<datatype>* pSrc1, int src1Step, const Ipp<datatype>* pSrc2, int src2Step, Ipp<datatype>* pDst, int dstStep, IppiSize roiSize, IppiAlphaType alphaType);
```

**Supported values for mod:**

- 8u_AC1R
- 16u_AC1R
- 8u_AC4R
- 16u_AC4R

```c
IppStatus ippiAlphaComp_<mod>(const Ipp<datatype>* const pSrc1[4], int src1Step, const Ipp<datatype>* const pSrc2[4], int src2Step, Ipp<datatype>* const pDst[4], int dstStep, IppiSize roiSize, IppiAlphaType alphaType);
```

**Supported values for mod:**

- 8u_AP4R
- 16u_AP4R
Case 2: In-place operation

IppStatus ippiAlphaComp_<mod>(const Ipp<datatype>* pSrc, int srcStep, Ipp<datatype>* pSrcDst, int srcDstStep, IppiSize roiSize, IppiAlphaType alphaType);

Supported values for mod:

8u_AC4IR 16u_AC4IR

IppStatus ippiAlphaComp_<mod>(const Ipp<datatype>* const pSrc[4], int srcStep, Ipp<datatype>* const pSrcDst[4], int srcDstStep, IppiSize roiSize, IppiAlphaType alphaType);

Supported values for mod:

8u_AP4IR 16u_AP4IR

Include Files

ippi.h

Domain Dependencies

Headers: ippcore.h, ippvm.h, ipps.h
Libraries: ippcore.lib, ippvm.lib, ipps.lib

Parameters

pSrc, pSrc1, pSrc2

Pointers to the source image ROI for pixel-order data. An array of pointers to ROI in the separate source color planes in case of planar data.

tsrcStep, srct1Step, srct2Step

Distances, in bytes, between the starting points of consecutive lines in the source images.

pSrcDst

Pointer to the source and destination buffer or an array of pointers to separate source and destination color planes for in-place operation.

pDst

Pointer to the destination image ROI for pixel-order data. An array of pointers to ROI in the separate destination color planes in case of planar data.

srcDstStep

Distance, in bytes, between the starting points of consecutive lines in the source and destination image for in-place operation.

dstStep

Distance, in bytes, between the starting points of consecutive lines in the destination image.

roiSize

Size of the source and destination ROI, in pixels.
alphaType

The composition type to perform. See Table “Possible Values of the Parameter alphaType” for the type value and description.

Description

This function operates with ROI (see Regions of Interest in Intel IPP).

This function performs an image compositing operation on RGBA images using alpha values of both images. The compositing is done by overlaying pixels \((r_A, g_A, b_A, a_A)\) from the foreground image \(pSrc1\) with pixels \((r_B, g_B, b_B, a_B)\) from the background image \(pSrc2\) to produce pixels \((r_C, g_C, b_C, a_C)\) in the resultant image \(pDst\). The alpha values are assumed to be normalized to the range \([0..1]\).

The type of the compositing operation is indicated by the alphaType parameter. Use Table "Possible Values of the Parameter alphaType" to choose a valid alphaType value depending on the required composition type. For example, the resulting pixel color components for the OVER operation (see Table "Types of Image Composing Operations") are computed as follows:

\[
\begin{align*}
    r_C &= a_A * r_A + (1 - a_A) * a_B * r_B \\
    g_C &= a_A * g_A + (1 - a_A) * a_B * g_B \\
    b_C &= a_A * b_A + (1 - a_A) * a_B * b_B \\
    a_C &= a_A + (1 - a_A) * a_B
\end{align*}
\]

The resulting (normalized) alpha value is computed as

\[
a_C = a_A + (1 - a_A) * a_B
\]

This function can be used for unsigned pixel data only.

Return Values

- **ippStsNoErr**: Indicates no error. Any other value indicates an error or a warning.
- **ippStsNullPtrErr**: Indicates an error condition if one of the specified pointers is NULL.
- **ippStsSizeErr**: Indicates an error condition if roiSize has a field with zero or negative value.

AlphaCompC

Combines two images using constant alpha values.

Syntax

**Case 1: Not-in-place operation**

\[
\text{IppStatus ippiAlphaCompC}_{\text{<mod>}}\left(\text{const Ipp<datatype>* pSrc1, int src1Step, Ipp<datatype> alpha1, const Ipp<datatype>* pSrc2, int src2Step, Ipp<datatype> alpha2, Ipp<datatype>* pDst, int dstStep, IppiSize roiSize, IppiAlphaType alphaType}\right);
\]

Supported values for **mod**:

- 8u_C1R
- 8u_C3R
- 8u_C4R
- 8u_AC4R
- 16u_C1R
- 16u_C3R
- 16u_C4R
- 16u_AC4R
- 16s_C1R
- 32u_C1R
32s_C1R
32f_C1R

IppStatus ippiAlphaCompC_<mod>(const Ipp<datatype>* const pSrc1[4], int src1Step, Ipp<datatype> alpha1, const Ipp<datatype>* const pSrc2[4], int src2Step, Ipp<datatype> alpha2, Ipp<datatype>* const pDst[4], int dstStep, IppSize roiSize, IppAlphaType alphaType);

Supported values for <mod>:
8u_AP4R 16u_AP4R

Case 2: In-place operation
IppStatus ippiAlphaCompC_<mod>(const Ipp<datatype>* pSrc, int srcStep, Ipp<datatype> alpha1, Ipp<datatype>* pSrcDst, int srcDstStep, Ipp<datatype> alpha2, IppSize roiSize, IppAlphaType alphaType);

Supported values for <mod>:
8u_C1IR 16u_C1IR 16s_C1IR 32s_C1IR 32u_C1IR 32f_C1IR
8u_C3IR 16u_C3IR
8u_C4IR 16u_C4IR
8u_AC4IR 16u_AC4IR

Include Files
ippi.h

Domain Dependencies
Headers: ippcore.h, ippvm.h, ipps.h
Libraries: ippcore.lib, ippvm.lib, ipps.lib

Parameters
pSrc1, pSrc2

src1Step, src2Step

Pointers to the source image ROI for pixel-order data. An array of pointers to ROI in the separate source color planes in case of planar data.
Distances, in bytes, between the starting points of consecutive lines in the source images.
pSrcDst

Pointer to the source and destination buffer or an array of pointers to separate source and destination color planes for in-place operation.

pDst

Pointer to the destination image ROI for pixel-order data. An array of pointers to ROI in the separate destination color planes in case of planar data.

srcDstStep

Distance, in bytes, between the starting points of consecutive lines in the source and destination image for in-place operation.

dstStep

Distance, in bytes, between the starting points of consecutive lines in the destination image.

roiSize

Size of the source and destination ROI in pixels.

alpha1, alpha2

Constant alpha values to use for the compositing operation.

alphaType

The composition type to perform. See Table “Possible Values of the Parameter alphaType” for the type value and description.

Description

This function operates with ROI (see Regions of Interest in Intel IPP).

This function performs an image compositing operation on one-channel image buffers, three-channel RGB and four-channel RGBA image buffers and on planar images, using constant alpha values alpha1 and alpha2. These values are passed to the function as parameters.

The compositing is done by overlaying pixels from the foreground image ROI pSrc1 with pixels from the background image ROI pSrc2 to produce pixels in the resultant image ROI pDst. The alpha values are normalized to the range [0..1].

The type of the compositing operation is indicated by the alphaType parameter. Use Table “Possible Values of the Parameter alphaType” to choose a valid alphaType value depending on the required composition type.

For example, the resulting pixel color components for the OVER operation (see Table “Types of Image Composing Operations”) are computed as follows:

\[
\begin{align*}
    r_C &= a_1 \times r_A + (1 - a_1) \times a_2 \times r_B \\
    g_C &= a_1 \times g_A + (1 - a_1) \times a_2 \times g_B \\
    b_C &= a_1 \times b_A + (1 - a_1) \times a_2 \times b_B
\end{align*}
\]

where \(a_1, a_2\) are the normalised alpha values alpha1, alpha2.

This function can be used for unsigned pixel data only.

Return Values

ippStsNoErr

Indicates no error. Any other value indicates an error or a warning.

ippStsNullPtrErr

Indicates an error condition if one of the specified pointers is NULL.

ippStsSizeErr

Indicates an error condition if roiSize has a field with zero or negative value.

Example

The code example below shows how to use alpha composition function.

```c
IppStatus acomp( void ) {
    Ipp8u imga[8*8], imgb[8*8], imgc[8*8];
```
IppiSize roi = { 8, 8 };  
IppStatus st;  
ippiImageRamp_8u_C1R( imga, 8, roi, 0, 1, ippAxsHorizontal );  
ippiImageRamp_8u_C1R( imgb, 8, roi, 0, 2, ippAxsHorizontal );  
st = ippiAlphaCompC_8u_C1R( imga, 8, 255/3, imgb, 8, 255, imgc, 8, roi, ippAlphaOver );  
printf( "over: a=%d,A=255/3; b=%d,B=255; c=%d // 
c=a*A+b*(1-A)*B\n",imga[6],imgb[6],imgc[6] );  
return st;  
}

Output

over: a=6,A=255/3; b=12,B=255; c=10 // c=a*A+b*B*(1-A)

**AlphaPremul**

Pre-multiplies pixel values of an image by its alpha values.

**Syntax**

**Case 1: Not-in-place operation**

IppStatus ippiAlphaPremul_<mod>(const Ipp<datatype>* pSrc, int srcStep, Ipp<datatype>* pDst, int dstStep, IppSize roiSize);

Supported values for **mod**:

- 8u_AC4R
- 16u_AC4R

IppStatus ippiAlphaPremul_<mod>(const Ipp<datatype>* const pSrc[4], int srcStep, Ipp<datatype>* const pDst[4], int dstStep, IppSize roiSize);

Supported values for **mod**:

- 8u_AP4R
- 16u_AP4R

**Case 2: In-place operation**

IppStatus ippiAlphaPremul_<mod>(Ipp<datatype>* pSrcDst, int srcDstStep, IppSize roiSize);

Supported values for **mod**:

- 8u_AC4IR
- 16u_AC4IR

IppStatus ippiAlphaPremul_<mod>(Ipp<datatype>* const pSrcDst[4], int srcDstStep, IppSize roiSize);

Supported values for **mod**:

- 8u_AP4IR
- 16u_AP4IR

**Include Files**

ippi.h
Domain Dependencies

**Headers:** ippcore.h, ippvm.h, ipps.h

**Libraries:** ippcore.lib, ippvm.lib, ipps.lib

Parameters

- **pSrc**  
  Pointer to the source image ROI for pixel-order data. An array of pointers to ROI in the separate source color planes in case of planar data.

- **srcStep**  
  Distance in bytes between starts of consecutive lines in the source image.

- **pDst**  
  Pointer to the destination image ROI for pixel-order data. An array of pointers to ROI in the separate destination color planes in case of planar data.

- **dstStep**  
  Distance in bytes between starts of consecutive lines in the destination image.

- **pSrcDst**  
  Pointer to the source and destination buffer or an array of pointers to separate source and destination color planes for the in-place operation.

- **srcDstStep**  
  Distance in bytes between starts of consecutive lines in the source and destination image for the in-place operation.

- **roiSize**  
  Size of the source and destination ROI in pixels.

Description

This function operates with ROI (see Regions of Interest in Intel IPP).

This function converts a RGBA source image (pixel order or planar) to the pre-multiplied alpha form. If \((r, g, b, a)\) are the red, green, blue, and alpha values of a pixel, then new pixel values are \((r \times a, g \times a, b \times a, a)\) after execution of this function. Here \(a\) is the pixel normalized alpha value in the range 0 to 1.

The function `ippiAlphaPremul` can be used for unsigned pixel data only.

Return Values

- **ippStsNoErr**  
  Indicates no error. Any other value indicates an error or a warning.

- **ippStsNullPtrErr**  
  Indicates an error condition if one of the specified pointers is NULL.

- **ippStsSizeErr**  
  Indicates an error condition if `roiSize` has a field with zero or negative value.

**AlphaPremulC**

*Pre-multiplies pixel values of an image using constant alpha (opacity) values.*
Syntax

Case 1: Not-in-place operation

IppStatus ippiAlphaPremulC_<mod>(const Ipp<datatype>* pSrc, int srcStep, Ipp<datatype> alpha, Ipp<datatype>* pDst, int dstStep, IppiSize roiSize);

Supported values for mod:

8u_C1R  16u_C1R  
8u_C3R  16u_C3R  
8u_C4R  16u_C4R  
8u_AC4R 16u_AC4R

IppStatus ippiAlphaPremulC_<mod>(const Ipp<datatype>* const pSrc[4], int srcStep, Ipp<datatype> alpha, Ipp<datatype>* const pDst[4], int dstStep, IppiSize roiSize);

Supported values for mod:

8u_AP4R 16u_AP4R

Case 2: In-place operation

IppStatus ippiAlphaPremulC_<mod>(Ipp<datatype> alpha, Ipp<datatype>* pSrcDst, int srcDstStep, IppiSize roiSize);

Supported values for mod:

8u_C1IR 16u_C1IR  
8u_C3IR 16u_C3IR  
8u_C4IR 16u_C4IR  
8u_AC4IR 16u_AC4IR

IppStatus ippiAlphaPremulC_<mod>(Ipp<datatype> alpha, Ipp<datatype>* const pSrcDst[4], int srcDstStep, IppiSize roiSize);

Supported values for mod:

8u_AP4IR 16u_AP4IR

Include Files

ippi.h

Domain Dependencies

Headers: ippcore.h, ippvm.h, ipps.h

Libraries: ippcore.lib, ippvm.lib, ipps.lib
**Parameters**

- **pSrc**
  Pointer to the source image ROI for pixel-order data. An array of pointers to ROI in the separate source color planes in case of planar data.

- **srcStep**
  Distance in bytes between starts of consecutive lines in the source image.

- **pDst**
  Pointer to the destination image ROI for pixel-order data. An array of pointers to separate ROI in the destination color planes in case of planar data.

- **dstStep**
  Distance in bytes between starts of consecutive lines in the destination image.

- **pSrcDst**
  Pointer to the source and destination image ROI or an array of pointers to ROI in the separate source and destination color planes for the in-place operation.

- **srcDstStep**
  Distance in bytes between starts of consecutive lines in the source and destination image for the in-place operation.

- **roiSize**
  Size of the source and destination ROI in pixels.

- **alpha**
  Global alpha value used for pre-multiplying pixel values.

**Description**

This function operates with ROI (see Regions of Interest in Intel IPP).

This function converts either a one-, three-, four-channel image or planar RGBA image to the pre-multiplied alpha form, using the global alpha value \( \alpha \). For one-, three-, four-channel image buffers, pixel values in each channel are multiplied by \( \alpha \); for RGBA (pixel order and planar) images with \((r,g,b,a)\) pixel values, new pixel values are \((r \times \alpha, g \times \alpha, b \times \alpha, a)\) after execution of this function. Here \( a \) is the normalized \( \alpha \) value in the range 0 to 1.

The function `ippiAlphaPremulC` can be used for unsigned pixel data only.

**Return Values**

- **ippStsNoErr**
  Indicates no error. Any other value indicates an error or a warning.

- **ippStsNullPtrErr**
  Indicates an error condition if one of the specified pointers is NULL.

- **ippStsSizeErr**
  Indicates an error condition if \( \text{roiSize} \) has a field with zero or negative value.
This chapter describes image processing functions that perform different types of image color conversion. The Intel IPP software supports the following image color conversions:

- Color models conversion
- Conversion from color to gray scale and vice versa
- Different types of format conversion:
  - from pixel-order to planar format and vice versa
  - changing number of channels or planes
  - changing sampling format
  - altering order of samples or planes
- Gamma correction
- Reduction from high bit resolution color to low bit resolution color
- Intensity transformation using lookup tables
- Color twist
- Color keying

All Intel IPP color conversion functions perform point operations on pixels of the source image. For a given destination pixel, the resultant channel values are computed using channel values of the corresponding source pixel only, and not involving any neighborhood pixels. Thus, the rectangular region of interest (ROI, see Regions of Interest in Intel IPP) used in function operations may extend to the size of the whole image.

This chapter starts with introductory material that discusses color space models essential for understanding of the Intel IPP color conversion functions.

For more information about color spaces and color conversion techniques, see [Jack01], [Rogers85], and [Foley90].

**Gamma Correction**

Gamma correction of images is used to optimize the usage of data type depth when encoding an image by taking advantage of the non-linear manner in which humans perceive light and color. This non-linearity must be compensated to achieve correct color reproduction. To do this, luminance of each of the linear red, green, and blue components is reduced to a non-linear form using an inverse transformation. This process is called gamma correction.

The Intel IPP functions use the following basic equations to convert an RGB image to a gamma-corrected R'G'B' image:

For $R, G, B < 0.018$

\[
R' = 4.5R \\
G' = 4.5G \\
B' = 4.5B
\]

For $R, G, B \geq 0.018$

\[
R' = 1.099R^{0.45} - 0.099 \\
G' = 1.099G^{0.45} - 0.099 \\
B' = 1.099B^{0.45} - 0.099
\]

Note that the channel intensity values are normalized to fit in the range [0..1]. The gamma value is equal to $1/0.45 = 2.22$ in conformity with ITU Rec.709 specification (see [ITU709]).
CIE Chromaticity Diagram and Color Gamut

Figure CIE xyY Chromaticity Diagram and Color Gamut presents a diagram of all visible colors. It is called a chromaticity diagram and was developed as a result of the experimental investigations performed by CIE (International Commission on Illumination, http://members.eunet.at/cie). The diagram presents visible colors as a function of \( x \) (red) and \( y \) (green) components called chromaticity coordinates. Positions of various spectrum colors (from violet to red) are indicated as the points of a tongue-shaped curve called spectrum locus. The straight line connecting the ends of the curve is called the purple line. The point of equal energy represents the CIE standard for white light. Any point within the diagram represents some mixture of spectrum colors. The pure or fully saturated colors lie on the spectrum locus. A straight-line segment joining any two points in the diagram defines all color variations that can be obtained by additively combining these two colors. A triangle with vertices at any three points determine the gamut of colors that can be obtained by combining corresponding three colors.

The structure of the human eye that distinguishes three different stimuli, establishes the three-dimensional nature of color. The color may be described with a set of three parameters called tristimulus values, or components. These values may, for example, be dominant wavelength, purity, and luminance, or so-called primary colors: red, green, and blue.

The chromaticity diagram exhibits that the gamut of any three fixed colors cannot enclose all visible colors. For example, Figure CIE xyY Chromaticity Diagram and Color Gamut shows schematically the gamut of reproducible colors for the RGB primaries of a typical color CRT monitor, CMYK color printing, and for the NTSC television.

CIE xyY Chromaticity Diagram and Color Gamut

Color Models

The purpose of a color model is to facilitate the specification of colors in some standard generally accepted way. In essence, a color model is a specification of a 3-D coordinate system and a subspace within that system where each color is represented by a single point.
Each industry that uses color employs the most suitable color model. For example, the RGB color model is used in computer graphics, YUV or YCbCr are used in video systems, PhotoYCC* is used in PhotoCD* production and so on. Transferring color information from one industry to another requires transformation from one set of values to another. Intel IPP provides a wide number of functions to convert different color spaces to RGB and vice versa.

**RGB Color Model**

In the RGB model, each color appears as a combination of red, green, and blue. This model is called additive, and the colors are called primary colors. The primary colors can be added to produce the secondary colors of light (see Figure "Primary and Secondary Colors for RGB and CMYK Models") - magenta (red plus blue), cyan (green plus blue), and yellow (red plus green). The combination of red, green, and blue at full intensities makes white.

**Primary and Secondary Colors for RGB and CMYK Models**

![Primary and Secondary Colors for RGB and CMYK Models](image)

The color subspace of interest is a cube shown in Figure "RGB and CMY Color Models" (RGB values are normalized to 0..1), in which RGB values are at three corners; cyan, magenta, and yellow are the three other corners, black is at their origin; and white is at the corner farthest from the origin.

The gray scale extends from black to white along the diagonal joining these two points. The colors are the points on or inside the cube, defined by vectors extending from the origin.

Thus, images in the RGB color model consist of three independent image planes, one for each primary color. As a rule, the Intel IPP color conversion functions operate with non-linear gamma-corrected images R'G'B'.

The importance of the RGB color model is that it relates very closely to the way that the human eye perceives color. RGB is a basic color model for computer graphics because color displays use red, green, and blue to create the desired color. Therefore, the choice of the RGB color space simplifies the architecture and design of the system. Besides, a system that is designed using the RGB color space can take advantage of a large number of existing software routines, because this color space has been around for a number of years.

**RGB and CMY Color Models**

![RGB and CMY Color Models](image)
However, RGB is not very efficient when dealing with real-world images. To generate any color within the RGB color cube, all three RGB components need to be of equal pixel depth and display resolution. Also, any modification of the image requires modification of all three planes.

**CMYK Color Model**

The CMYK color model is a subset of the RGB model and is primarily used in color print production. CMYK is an acronym for cyan, magenta, and yellow along with black (noted as K). The CMYK color space is subtractive, meaning that cyan, magenta yellow, and black pigments or inks are applied to a white surface to subtract some color from white surface to create the final color. For example (see Figure "Primary and Secondary Colors for RGB and CMYK Models"), cyan is white minus red, magenta is white minus green, and yellow is white minus blue. Subtracting all colors by combining the CMY at full saturation should, in theory, render black. However, impurities in the existing CMY inks make full and equal saturation impossible, and some RGB light does filter through, rendering a muddy brown color. Therefore, the black ink is added to CMY. The CMY cube is shown in Figure "RGB and CMY Color Models", in which CMY values are at three corners; red, green, and blue are the three other corners, white is at the origin; and black is at the corner farthest from the origin.

**YUV Color Model**

The YUV color model is the basic color model used in analogue color TV broadcasting. Initially YUV is the re-coding of RGB for transmission efficiency (minimizing bandwidth) and for downward compatibility with black-and-white television. The YUV color space is "derived" from the RGB space. It comprises the luminance (Y) and two color difference (U, V) components. The luminance can be computed as a weighted sum of red, green and blue components; the color difference, or chrominance, components are formed by subtracting luminance from blue and from red.

The principal advantage of the YUV model in image processing is decoupling of luminance and color information. The importance of this decoupling is that the luminance component of an image can be processed without affecting its color component. For example, the histogram equalization of the color image in the YUV format may be performed simply by applying histogram equalization to its Y component.

There are many combinations of YUV values from nominal ranges that result in invalid RGB values, because the possible RGB colors occupy only part of the YUV space limited by these ranges. Figure "RGB Colors Cube in the YUV Color Space" shows the valid color block in the YUV space that corresponds to the RGB color cube RGB values that are normalized to [0..1]).

The $Y'U'V'$ notation means that the components are derived from gamma-corrected $R'G'B'$. Weighted sum of these non-linear components forms a signal representative of luminance that is called $luma_{Y'}$. ($Luma$ is often loosely referred to as luminance, so you need to be careful to determine whether a particular author assigns a linear or non-linear interpretation to the term luminance).

The Intel IPP functions use the following basic equation ([Jack01]) to convert between gamma-corrected $R'G'B'$ and $Y'U'V'$ models:

\[
Y' = 0.299*R' + 0.587*G' + 0.114*B' \\
U' = -0.147*R' - 0.289*G' + 0.436*B' = 0.492*(B' - Y') \\
V' = 0.615*R' - 0.515*G' - 0.100*B' = 0.877*(R' - Y') \\
R' = Y' + 1.140*V' \\
G' = Y' - 0.394*U' - 0.581*V'
\]
There are several YUV sampling formats such as 4:4:4, 4:2:2, and 4:2:0 that are supported by the Intel IPP color conversion functions and are described later in this chapter in Image Downsampling.

**YCbCr and YCCK Color Models**

The YCbCr color space is used for component digital video and was developed as part of the ITU-R BT.601 Recommendation. YCbCr is a scaled and offset version of the YUV color space.

The Intel IPP functions use the following basic equations [Jack01] to convert between $R'G'B'$ in the range 0-255 and $Y'C'b'C'r'$ (this notation means that all components are derived from gamma-corrected $R'G'B'$):

- $Y' = 0.257*R' + 0.504*G' + 0.098*B' + 16$
- $Cb' = -0.148*R' - 0.291*G' + 0.439*B' + 128$
- $Cr' = 0.439*R' - 0.368*G' - 0.071*B' + 128$
- $R' = 1.164*(Y'-16) + 1.596*(Cr'-128)$
- $G' = 1.164*(Y'-16) - 0.813*(Cr'-128) - 0.392*(Cb'-128)$
- $B' = 1.164*(Y'-16) + 2.017*(Cb'-128)$

Possible RGB colors occupy only part of the YCbCr color space (see Figure "RGB Colors Cube in the YCbCr Space") limited by the nominal ranges, therefore there are many YCbCr combinations that result in invalid RGB values.
There are several YCbCr sampling formats such as 4:4:4, 4:2:2, 4:1:1, and 4:2:0, which are supported by the Intel IPP color conversion functions and are described later in this chapter in Image Downsampling.

**RGB Colors Cube in the YCbCr Space**

---

**PhotoYCC Color Model**

The Kodak® PhotoYCC® was developed for encoding Photo CD® image data. It is based on both the ITU Recommendations 601 and 709, using luminance-chrominance representation of color like in BT.601 YCbCr and BT.709 ([ITU709]). This model comprises luminance (Y) and two color difference, or chrominance (C1, C2) components. The PhotoYCC is optimized for the color photographic material, and provides a color gamut that is greater than the one that can currently be displayed.

The Intel IPP functions use the following basic equations [Jack01] to convert non-linear gamma-corrected R'G'B' to Y'C'C':

\[
Y' = 0.213 * R' + 0.419 * G' + 0.081 * B' \\
C1' = -0.131 * R' - 0.256 * G' + 0.387 * B' + 0.612 \\
C2' = 0.373 * R' - 0.312 * B' + 0.537
\]

The equations above are given on the assumption that R', G', and B' values are normalized to the range [0..1].

Since the PhotoYCC model attempts to preserve the dynamic range of film, decoding PhotoYCC images requires selection of a color space and range appropriate for the output device. Thus, the decoding equations are not always the exact inverse of the encoding equations. The following equations [Jack01] are used in Intel IPP to generate R'G'B' values for driving a CRT display and require a unity relationship between the luma in the encoded image and the displayed image:

\[
R' = 0.981 * Y + 1.315 * (C2 - 0.537) \\
G' = 0.981 * Y - 0.311 * (C1 - 0.612) - 0.669 * (C2 - 0.537) \\
B' = 0.981 * Y + 1.601 * (C1 - 0.612)
\]

The equations above are given on the assumption that source Y, C1 and C2 values are normalized to the range [0..1], and the display primaries have the chromaticity values in accordance with [ITU709] specifications.
The possible RGB colors occupy only part of the YCC color space (see Figure "RGB Colors in the YCC Color Space") limited by the nominal ranges, therefore there are many YCC combinations that result in invalid RGB values.

**RGB Colors in the YCC Color Space**

---

**YCoCg Color Models**

The YCoCg color model was developed to increase the effectiveness of the image compression [Malvar03]. This color model comprises the luminance \(Y\) and two color difference components (\(C_0 -\) offset orange, \(C_g -\) offset green).

The Intel IPP functions use the following simple basic equations [Malvar03] to convert between RGB and YCoCg:

\[
Y = \frac{R}{4} + \frac{G}{2} + \frac{B}{4} \\
C_0 = \frac{R}{2} - \frac{B}{2} \\
C_g = -\frac{R}{4} + \frac{G}{2} - \frac{B}{4} \\
R = Y + C_0 - C_g \\
G = Y + C_g \\
B = Y - C_0 - C_g
\]

A variation of this color space which is called YCoCg-R, enables transformation reversibility with a smaller dynamic range requirements than does YCoCg [Malvar03-1].
The possible RGB colors occupy only part of the YCoCg color space (see Figure "RGB Color Cube in the YCoCg Color Space") limited by the nominal ranges, therefore there are many YCoCg combinations that result in invalid RGB values.

**RGB Color Cube in the YCoCg Color Space**

---

**HSV, and HLS Color Models**

The HLS (hue, lightness, saturation) and HSV (hue, saturation, value) color models were developed to be more "intuitive" in manipulating with color and were designed to approximate the way humans perceive and interpret color.

*Hue* defines the color itself. The values for the hue axis vary from 0 to 360 beginning and ending with red and running through green, blue and all intermediary colors.

*Saturation* indicates the degree to which the hue differs from a neutral gray. The values run from 0, which means no color saturation, to 1, which is the fullest saturation of a given hue at a given illumination.

Intensity component - *lightness* (HLS) or *value* (HSV), indicates the illumination level. Both vary from 0 (black, no light) to 1 (white, full illumination). The difference between the two is that maximum saturation of hue (S=1) is at *value* V=1 (full illumination) in the HSV color model, and at *lightness* L=0.5 in the HLS color model.

The HSV color space is essentially a cylinder, but usually it is represented as a cone or hexagonal cone (hexcone) as shown in the Figure "HSV Solid", because the hexcone defines the subset of the HSV space with valid RGB values. The *value* V is the vertical axis, and the vertex V=0 corresponds to black color. Similarly, a color solid, or 3D-representation, of the HLS model is a double hexcone (Figure "HSV Solid") with *lightness* as the axis, and the vertex of the second hexcone corresponding to white.
Both color models have intensity component decoupled from the color information. The HSV color space yields a greater dynamic range of saturation. Conversions from RGBtoHSV/RGBtoHSV and vice-versa in Intel IPP are performed in accordance with the respective pseudocode algorithms [Rogers85] given in the descriptions of corresponding conversion functions.

**HSV Solid**

![HSV Solid Diagram](Image)

**HLS Solid**

![HLS Solid Diagram](Image)
CIE XYZ Color Model

The XYZ color space is an international standard developed by the CIE (Commission Internationale de l'Eclairage). This model is based on three hypothetical primaries, XYZ, and all visible colors can be represented by using only positive values of X, Y, and Z. The CIE XYZ primaries are hypothetical because they do not correspond to any real light wavelengths. The Y primary is intentionally defined to match closely to luminance, while X and Z primaries give color information. The main advantage of the CIE XYZ space (and any color space based on it) is that this space is completely device-independent. The chromaticity diagram in Figure "CIE xyY Chromaticity Diagram and Color Gamut" is in fact a two-dimensional projection of the CIE XYZ sub-space. Note that arbitrarily combining X, Y, and Z values within nominal ranges can easily lead to a "color" outside of the visible color spectrum.

The position of the block of RGB-representable colors in the XYZ space is shown in Figure "RGB Colors Cube in the XYZ Color Space".

**RGB Colors Cube in the XYZ Color Space**

![RGB Colors Cube in the XYZ Color Space](image)

Intel IPP functions use the following basic equations [Rogers85], to convert between gamma-corrected R'G'B' and CIE XYZ models:

\[
X = 0.412453 \cdot R' + 0.35758 \cdot G' + 0.180423 \cdot B'
\]

\[
Y = 0.212671 \cdot R' + 0.71516 \cdot G' + 0.072169 \cdot B'
\]

\[
Z = 0.019334 \cdot R' + 0.119193 \cdot G' + 0.950227 \cdot B'
\]

The equations for \(X, Y, Z\) calculation are given on the assumption that \(R', G', B'\) values are normalized to the range \([0..1]\).

\[
R' = 3.240479 \cdot X - 1.53715 \cdot Y - 0.498535 \cdot Z
\]

\[
G' = -0.969256 \cdot X + 1.875991 \cdot Y + 0.041556 \cdot Z
\]

\[
B' = 0.055648 \cdot X - 0.204043 \cdot Y + 1.057311 \cdot Z
\]

The equations for \(R', G', B'\) calculation are given on the assumption that \(X, Y, Z\) values are in the range \([0..1]\).

CIE LUV and CIE Lab Color Models

The CIE LUV and CIE Lab color models are considered to be perceptually uniform and are referred to as uniform color models. Both are uniform derivations from the standard CIE XYZ space. "Perceptually uniform" means that two colors that are equally distant in the color space are equally distant perceptually. To
accomplish this approach, a uniform chromaticity scale (UCS) diagram was proposed by CIE (Figure "CIE $u', v'$ Uniform Chromaticity Scale Diagram"). The UCS diagram uses a mathematical formula to transform the XYZ values or x, y coordinates (Figure "CIE xyY Chromaticity Diagram and Color Gamut"), to a new set of values that present a visually more accurate two-dimensional model. The Y lightness scale is replaced with a new scale called L that is approximately uniformly spaced but is more indicative of the actual visual differences. Chrominance components are U and V for CIE LUV, and $a$ and $b$ (referred to also respectively as red/blue and yellow/blue chrominances) in CIE Lab. Both color spaces are derived from the CIE XYZ color space.

**CIE $u', v'$ Uniform Chromaticity Scale Diagram**

The CIE LUV color space is derived from CIE XYZ as follows ([Rogers85]),

$L = 116. \times (Y/Y_n)^{1/3} - 16.$

$U = 13. \times L \times (u - u_n)$

$V = 13. \times L \times (v - v_n)$

where

$u = 4.\times X / (X + 15.\times Y + 3.\times Z)$

$v = 9.\times Y / (X + 15.\times Y + 3.\times Z)$

$u_n = 4.\times x_n / (-2.\times x_n + 12.\times y_n + 3.)$

$v_n = 9.\times y_n / (-2.\times x_n + 12.\times y_n + 3.)$

Inverse conversion is performed in accordance with equations:

$Y = Y_n \times ((L + 16.) / 116.)^3.$

$X = -9.\times X_n / ((u - 4.)\times v - u \times v)$

$Z = (9.\times Y - 15.\times v\times Y - v\times X) / 3. \times v$

where

$u = U / (13.\times L) + u_n$

$v = V / (13.\times L) + v_n$

and $u_n, v_n$ are defined above.
Here $x_n = 0.312713$, $y_n = 0.329016$ are the CIE chromaticity coordinates of the D65 white point ([ITU709]), and $Y_n = 1.0$ is the luminance of the D65 white point. The values of the L component are in the range [0..100], U component in the range [-134..220], and V component in the range [-140..122].

The RGB-representable colors occupy only part of the LUV color space (see Figure 6-12) limited by the nominal ranges, therefore there are many LUV combinations that result in invalid RGB values.

**RGB Color Cube in the CIE LUV Color Space**

The CIE Lab color space is derived from CIE XYZ as follows:

$L = 116. \cdot (Y/Y_n)^{1/3} - 16 \text{ for } Y/Y_n > 0.008856$

$L = 903.3 \cdot (Y/Y_n)^{1/3} \text{ for } Y/Y_n \leq 0.008856$

$a = 500. \cdot [f(X/X_n) - f(Y/Y_n)]$

$b = 200. \cdot [f(Y/Y_n) - f(Z/Z_n)]$

where

$f(t) = t^{1/3} - 16 \text{ for } t > 0.008856$

$f(t) = 7.787 \cdot t + 16/116 \text{ for } t \leq 0.008856$

Here $Y_n = 1.0$ is the luminance, and $X_n = 0.950455$, $Z_n = 1.088753$ are the chrominances for the D65 white point.

The values of the L component are in the range [0..100], $a$ and $b$ component values are in the range [-128..127].

Inverse conversion is performed in accordance with equations:

$Y = Y_n \cdot p^3.$

$X = X_n \cdot (p + a/500.)^3.$

$Z = Z_n \cdot (p - b/200.)^3.$

where

$p = (L + 16)/116.$
Image Downsampling

Conventionally, digital color images are represented by setting specific values of the color space coordinates for each pixel. Color spaces with decoupled luminance and chrominance coordinates (YUV type) allow the number of bits required for acceptable color description of an image to be reduced. This reduction is based on greater sensitivity of the human eye to changes in luminance than to changes in chrominance. The idea behind this approach is to set individual value of luminance component to each pixel, while assigning the same color (chrominance components) to certain groups of pixels (sometimes called macropixels) in accordance with some specific rules. This process is called downsampling and there are different sampling formats depending on the underlying scheme.

The following sampling formats are supported by the Intel IPP image processing functions (excluding the JPEG functions):

4:4:4 YUV (YCbCr) - conventional format, no downsampling, Y, U(Cb), V(Cr) components are sampled at every pixel. If each component takes 8 bits, than every pixel requires 24 bits. This format is often denoted as YUV (YCbCr) with the 4:4:4 descriptor omitted.

4:2:2 YUV (YCbCr) - uses the 2:1 horizontal downsampling. It means that the Y component is sampled at each pixel, while U(Cb) and V(Cr) components are sampled every 2 pixels in horizontal direction. If each component takes 8 bits, the pair of pixels requires 32 bits.

4:1:1 YCbCr - uses the 4:1 horizontal downsampling. It means that the Y component is sampled at each pixel, while Cb and Cr components are sampled every 4 pixels horizontally. If each component takes 8 bits, each four horizontal pixels require 48 bits.

4:2:0 YUV (YCbCr) - uses the 2:1 horizontal downsampling and the 2:1 vertical downsampling. Y is sampled at each pixel, U(Cb) and V(Cr) are sampled at every block of 2x2 pixels. If each component takes 8 bits, each four-pixel block requires 48 bits.

In JPEG compression, downsampling has specific distinctive features and is denoted in a slightly different way. In JPEG domain, sampling formats determine the structure of minimal coded units, or MCUs. Therefore, the Intel IPP functions specific for a JPEG codec, support the following sampling formats:

4:4:4 YCbCr - for every 8x8 block of Y samples, there is one 8x8 block of each Cb and Cr samples.

4:2:2 YCbCr - for every two horizontal 8x8 blocks of Y samples, there is one 8x8 block of each Cb and Cr samples.

4:1:1 YCbCr - for every four (two in horizontal and two in vertical direction) 8x8 blocks of Y samples, there is one 8x8 block of each Cb and Cr samples.

Structure of the corresponding MCU for each of these sampling formats is shown in Figure MCU Structure for Different JPEG Sampling Formats.

**MCU Structure for Different JPEG Sampling Formats**
RGB Image Formats

In addition to the 24-bit-per-pixel RGB/BGR image formats, the Intel IPP color conversion functions support 32-bit-per-pixel RGB/BGR formats, which include three RGB channels plus alpha channel. For 24-bit formats, each color is one byte, every pixel is three bytes. For 32-bit formats, each color is one byte and alpha component is one byte, which yields four bytes per pixel. Memory layout for these formats is given in Table "Pixel-Order Image Formats".

For 16-bit formats, every pixel is two bytes and each color occupies a specified number of bits. The figure below shows all the supported 16-bit-per-pixel formats and their memory layout (bit order):

16-bit pixel formats

<table>
<thead>
<tr>
<th>Image Format</th>
<th>Number of Channels</th>
<th>Channel Values Order</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>RGB</td>
<td>3</td>
<td>R0 G0 B0 R1 G1 B1 R2 G2 B2</td>
<td>ippiRGBTToYUV_8u_C3R</td>
</tr>
<tr>
<td>RGB444</td>
<td></td>
<td></td>
<td>ippiYCbCrToRGB444_8u16u_C3R</td>
</tr>
<tr>
<td>RGB555</td>
<td></td>
<td></td>
<td>ippiYCbCrToRGB555_8u16u_C3R</td>
</tr>
<tr>
<td>RGB565</td>
<td></td>
<td></td>
<td>ippiYCbCrToRGB565_8u16u_C3R</td>
</tr>
<tr>
<td>BGR</td>
<td>4</td>
<td>R0 G0 B0 A0 R1 G1 B1 A1</td>
<td>ippiRGBTToYUV_8u_C4R</td>
</tr>
<tr>
<td>BGR444</td>
<td>3</td>
<td>B0 G0 R0 B1 G1 R1 R2 G2 B2</td>
<td>ippiYCbCrToBGR_8u_P3C3R</td>
</tr>
</tbody>
</table>

Pixel and Planar Image Formats

Data storage for an image can be pixel-oriented or planar-oriented (planar). For images in pixel order, all channel values for each pixel are clustered and stored consecutively. Their layout depends on the color model and downsampling scheme.

Table “Pixel-Order Image Formats” lists all pixel-order image formats that are supported by the Intel IPP color conversion functions and shows the corresponding channel values order (here, group of underlined symbols represents one pixel and symbol A denotes alpha-channel value). The last column of this table gives an example of an Intel IPP color conversion function that uses the respective image format.

<table>
<thead>
<tr>
<th>Image Format</th>
<th>Number of Channels</th>
<th>Channel Values Order</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>RGB</td>
<td>3</td>
<td>R0 G0 B0 R1 G1 B1 B2</td>
<td>ippiRGBTToYUV_8u_C3R</td>
</tr>
<tr>
<td>RGB444</td>
<td></td>
<td></td>
<td>ippiYCbCrToRGB444_8u16u_C3R</td>
</tr>
<tr>
<td>RGB555</td>
<td></td>
<td></td>
<td>ippiYCbCrToRGB555_8u16u_C3R</td>
</tr>
<tr>
<td>RGB565</td>
<td></td>
<td></td>
<td>ippiYCbCrToRGB565_8u16u_C3R</td>
</tr>
<tr>
<td>BGR</td>
<td>4</td>
<td>R0 G0 B0 A0 R1 G1 B1 A1</td>
<td>ippiRGBTToYUV_8u_C4R</td>
</tr>
<tr>
<td>BGR444</td>
<td>3</td>
<td>B0 G0 R0 B1 G1 R1 R2 G2 B2</td>
<td>ippiYCbCrToBGR_8u_P3C3R</td>
</tr>
</tbody>
</table>

218
### Planar Image Formats

<table>
<thead>
<tr>
<th>Image Format</th>
<th>Number of Planes</th>
<th>Planes Layout</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>RGB</td>
<td>3</td>
<td>R G B</td>
<td>see Figure below, a</td>
</tr>
<tr>
<td>YUV</td>
<td>3</td>
<td>Y U V</td>
<td>see Figure below, a</td>
</tr>
<tr>
<td>4:2:2 YUV</td>
<td>3</td>
<td>Y U V</td>
<td>see Figure below, b</td>
</tr>
</tbody>
</table>

Planar image formats supported by the Intel IPP color conversion functions are listed in Table "Planar Image Formats" along with examples of the Intel IPP functions using that format. Planes layout and their relative sizes are shown in Figure Plane Size and Layout: 3-planes Images and Figure Plane Size and Layout: 2-planes Images.
<table>
<thead>
<tr>
<th>Image Format</th>
<th>Numbe r of Planes</th>
<th>Planes Layout</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>4:2:0 YUV</td>
<td>3</td>
<td>Y U V</td>
<td>see Figure below, d</td>
</tr>
<tr>
<td>YCbCr</td>
<td>3</td>
<td>Y Cb Cr</td>
<td>see Figure below, a</td>
</tr>
<tr>
<td>4:2:2 YCbCr</td>
<td>3</td>
<td>Y Cb Cr</td>
<td>see Figure below, b</td>
</tr>
<tr>
<td>4:1:1 YCbCr</td>
<td>3</td>
<td>Y Cb Cr</td>
<td>see Figure below, c</td>
</tr>
<tr>
<td>4:1:1 YCbCr</td>
<td>2</td>
<td>Y CbCr</td>
<td>see Figure below for 2-planes images, a</td>
</tr>
<tr>
<td>4:2:0 YCbCr</td>
<td>3</td>
<td>Y Cb Cr</td>
<td>see Figure below, d</td>
</tr>
<tr>
<td>4:2:0 YCbCr</td>
<td>2</td>
<td>Y CbCr</td>
<td>see Figure below for 2-planes images, b</td>
</tr>
<tr>
<td>Image Format</td>
<td>Number of Planes</td>
<td>Planes Layout</td>
<td>Example</td>
</tr>
<tr>
<td>--------------</td>
<td>------------------</td>
<td>---------------</td>
<td>---------</td>
</tr>
<tr>
<td>4:2:0 YCrCb</td>
<td>3</td>
<td>Y Cr Cb</td>
<td>see Figure below, d ippiYCrCb420ToYCbCr422_8u_P3R</td>
</tr>
</tbody>
</table>

**Plane Size and Layout - 3-planes Images**

![3-plane image layout diagram]

- **a)** no downsampling, or sampling 4:4:4
- **b)** sampling 4:2:2
- **c)** sampling 4:1:1
- **d)** sampling 4:2:0

**Plane Size and Layout - 2-planes Images**

![2-plane image layout diagram]

- **a)** sampling 4:1:1
- **b)** sampling 4:2:0
Color Model Conversion

RGBToYUV

Converts an RGB image to the YUV color model.

Syntax

Case 1: Operation on pixel-order data

IppStatus ippiRGBToYUV_<mod>(const Ipp8u* pSrc, int srcStep, Ipp8u* pDst, int dstStep, IppiSize roiSize);

Supported values for mod:

8u_C3R  8u_AC4R

Case 2: Operation on planar data

IppStatus ippiRGBToYUV_8u_P3R(const Ipp8u* pSrc[3], int srcStep, Ipp8u* pDst[3], int dstStep, IppiSize roiSize);

Case 3: Conversion from pixel-order to planar data

IppStatus ippiRGBToYUV_8u_C3P3R(const Ipp8u* pSrc, int srcStep, Ipp8u* pDst[3], int dstStep, IppiSize roiSize);

IppStatus ippiRGBToYUV_8u_AC4P4R(const Ipp8u* pSrc, int srcStep, Ipp8u* pDst[4], int dstStep, IppiSize roiSize);

Include Files

ippcc.h

Domain Dependencies

Headers: ippcore.h, ippvm.h, ipps.h, ippi.h
Libraries: ippcore.lib, ippvm.lib, ipps.lib, ippi.lib

Parameters

pSrc

Pointer to the source image ROI for pixel-order data. An array of pointers to the source image ROI in separate color planes in case of planar data.

srcStep

Distance in bytes between starts of consecutive lines in the source image.

pDst

Pointer to the destination ROI for pixel-order data. An array of pointers to destination buffers in separate color planes in case of planar data.

dstStep

Distance in bytes between starts of consecutive lines in the destination image.

roiSize

Size of the source and destination ROI in pixels.

Description

This function operates with ROI (see Regions of Interest in Intel IPP).
This function converts the gamma-corrected R'G'B' image pSrc to the Y'U'V' image pDst (see Figure Converting an RGB image to YUV) according to the following formulas:

\[
Y' = 0.299*R' + 0.587*G' + 0.114*B' \\
U' = -0.147*R' - 0.289*G' + 0.436*B' = 0.492*(B' - Y') \\
V' = 0.615*R' - 0.515*G' - 0.106*B' = 0.877*(R' - Y')
\]

For digital RGB values in the range [0..255], Y' has the range [0..255], U varies in the range [-112..+112], and V in the range [-157..+157]. To fit in the range of [0..255], a constant value 128 is added to computed U and V values, and V is then saturated.

**Converting an RGB image to YUV**

![source image](image1.png) ![destination image](image2.png)

**Return Values**

- ippStsNoErr: Indicates no error. Any other value indicates an error or a warning.
- ippStsNullPtrErr: Indicates an error condition if pSrc or pDst pointer is NULL.
- ippStsSizeErr: Indicates an error condition if roiSize has a field with a zero or negative value.

**Example**

The code example below shows how to use the functionippiRGBToYUV_8u_C3R.

```c
#define nChannels 3

int main ()
{
    Ipp 8 u src [3*3* nChannels ] = {
        255, 0, 0, 255, 0, 0, 255, 0, 0,
        0, 255, 0, 0, 255, 0, 0, 255, 0,
        0, 0, 255, 0, 0, 255, 0, 0, 255};
    Ipp 8 u dst [3*3* nChannels ];
    IppSize roiSize = { 3, 3 };;
    IppStatus st = ippStsNoErr;
    int srcStep = 3* nChannels ;
    int dstStep = 3* nChannels ;
    st =ippiRGBToYUV_8 u _ C 3 R ( src , srcStep , dst , dstStep , roiSize );
    if ( st == ippStsNoErr)
    {
        printf("\n *************  passed ************* \n");
    }else{
        printf("\n *************  failed ************* \t");
    }
    return 0;
}
```
YUVToRGB

Converts a YUV image to the RGB color model.

Syntax

Case 1: Operation on pixel-order data

```
IppStatus ippiYUVToRGB_<mod>(const Ipp8u* pSrc, int srcStep, Ipp8u* pDst, int dstStep, IppiSize roiSize);
```

Supported values for `mod`:

- 8u_C3R
- 8u_AC4R

```
IppStatus ippiYUVToRGB_8u_C3C4R(const Ipp8u* pSrc, int srcStep, Ipp8u* pDst, int dstStep, IppiSize roiSize, Ipp8u* aval);
```

Case 2: Operation on planar data

```
IppStatus ippiYUVToRGB_8u_P3R(const Ipp8u* pSrc[3], int srcStep, Ipp8u* pDst[3], int dstStep, IppiSize roiSize);
```

Case 3: Conversion from planar to pixel-order data

```
IppStatus ippiYUVToRGB_8u_P3C3R(const Ipp8u* pSrc[3], int srcStep, Ipp8u* pDst, int dstStep, IppiSize roiSize);
```

Include Files

ippcc.h

Domain Dependencies

Headers: ippcore.h, ippvm.h, ipps.h,ippi.h

Libraries: ippcore.lib, ippvm.lib, ipps.lib,ippi.lib

Parameters

- **pSrc**: Pointer to the source buffer for pixel-order data. An array of pointers to separate source color planes in case of planar data.
- **srcStep**: Distance in bytes between starts of consecutive lines in the source image.
- **pDst**: Pointer to the destination buffer for pixel-order data. An array of pointers to separate destination color planes in case of planar data.
- **dstStep**: Distance in bytes between starts of consecutive lines in the destination image.
aval

Constant value to create the fourth channel.

roiSize

Size of the source and destination ROI in pixels.

Description

This function operates with ROI (see Regions of Interest in Intel IPP).

This function converts the Y'U'V' image pSrc to the gamma-corrected R'G'B' image pDst according to the following formulas:

\[
\begin{align*}
R' &= Y' + 1.140*V' \\
G' &= Y' - 0.394*U' - 0.581*V' \\
B' &= Y' + 2.032*U'
\end{align*}
\]

Return Values

ippStsNoErr

Indicates no error. Any other value indicates an error or a warning.

ippStsNullPtrErr

Indicates an error condition if pSrc or pDst is NULL.

ippStsSizeErr

Indicates an error condition if roiSize has a field with a zero or negative value.

RGBToYUV422

Converts an RGB image to the YUV color model; uses 4:2:2 sampling.

Syntax

Case 1: Operation on pixel-order data

IppStatus ippiRGBToYUV422_8u_C3C2R(const Ipp8u* pSrc, int srcStep, Ipp8u* pDst, int dstStep, IppiSize roiSize);

Case 2: Operation on planar data with ROI

IppStatus ippiRGBToYUV422_8u_P3R(const Ipp8u* pSrc[3], int srcStep, Ipp8u* pDst[3], int dstStep[3], IppiSize roiSize);

Case 3: Operation on planar data without ROI

IppStatus ippiRGBToYUV422_8u_P3(const Ipp8u* pSrc[3], Ipp8u* pDst[3], IppiSize imgSize);

Case 4: Conversion from pixel-order to planar data with ROI

IppStatus ippiRGBToYUV422_8u_C3P3R(const Ipp8u* pSrc, int srcStep, Ipp8u* pDst[3], int dstStep[3], IppiSize roiSize);

Case 5: Conversion from pixel-order to planar data without ROI

IppStatus ippiRGBToYUV422_8u_C3P3(const Ipp8u* pSrc, Ipp8u* pDst[3], IppiSize imgSize);

Include Files

ippcc.h

Domain Dependencies

Headers: ippcore.h, ippvm.h, ipps.h,ippi.h

Libraries: ippcore.lib, ippvm.lib, ipps.lib,ippi.lib
Parameters

**pSrc**
Pointer to the source image buffer for pixel-order image. An array of pointers to the source image buffer in each color plane for planar image.

**srcStep**
Distance in bytes between starts of consecutive lines in the source image for operations with ROI.

**pDst**
Pointer to the destination image buffer for pixel-order image. An array of pointers to the destination image buffer in each color plane for planar image.

**dstStep**
Distance in bytes between starts of consecutive lines in the destination image for operations with ROI. An array of three values for planar image.

**roiSize**
Size of the source and destination ROI in pixels.

**imgSize**
Size of the source and destination images in pixels for operations without ROI.

Description

This function converts the gamma-corrected R'G'B' image **pSrc** to the Y'U'V' image **pDst** with 4:2:2 sampling format, according to the same formulas as the function **ippiRGBToYUV** does. For more details on this sampling format, see Table "Pixel-Order Image Formats" and Table "Planar Image Formats".

For digital RGB values in the range [0..255], Y' has the range [0..255], U varies in the range [-112..+112], and V in the range [-157..+157]. To fit in the range of [0..255], the constant value 128 is added to computed U and V values, and V is then saturated.

Some function flavors operate with ROI (see Regions of Interest in Intel IPP). The function flavors that does not use ROI operate on the assumption that both the source and destination images have the same size and occupy a contiguous memory area, which means that image rows are not padded with zeroes. In this case the step parameters are not needed.

Return Values

**ippStsNoErr**
Indicates no error. Any other value indicates an error or a warning.

**ippStsNullPtrErr**
Indicates an error condition if **pSrc** or **pDst** pointer is NULL.

**ippStsSizeErr**
Indicates an error condition if **roiSize** or **imgSize** has a field with a zero or negative value.

YUV422ToRGB

Converts a YUV image with the 4:2:2 sampling to the RGB color model.

Syntax

**Case 1: Operation on pixel-order data**

```c
IppStatus ippiYUV422ToRGB_8u_C2C3R(const Ipp8u* pSrc, int srcStep, Ipp8u* pDst, int dstStep, IppiSize roiSize);
```

**Case 2: Operation on planar data with ROI**

```c
IppStatus ippiYUV422ToRGB_8u_P3R(const Ipp8u* pSrc[3], int srcStep[3], Ipp8u* pDst[3], int dstStep, IppiSize roiSize);
```
Case 3: Operation on planar data without ROI

IppStatus ippiYUV422ToRGB_8u_P3(const Ipp8u* pSrc[3], Ipp8u* pDst[3], IppiSize imgSize);

Case 4: Conversion from planar to pixel-order data with ROI

IppStatus ippiYUV422ToRGB_<mod>(const Ipp8u* pSrc[3], int srcStep[3], Ipp8u* pDst, int dstStep, IppiSize roiSize);

Supported values for mod:

8u_P3C3R  8u_P3AC4R

Case 5: Conversion from planar to pixel-order data without ROI

IppStatus ippiYUV422ToRGB_8u_P3C3(const Ipp8u* pSrc[3], Ipp8u* pDst, IppiSize imgSize);

Include Files

ippcc.h

Domain Dependencies

Headers: ippcore.h, ippvm.h, ipps.h,ippi.h
Libraries: ippcore.lib, ippvm.lib, ipps.lib,ippi.lib

Parameters

pSrc

Pointer to the source image buffer for pixel-order image. An array of pointers to the source image buffer in each color plane for planar image.

srcStep

Distance in bytes between starts of consecutive lines in the source image for operations with ROI. An array of three values in case of planar image.

pDst

Pointer to the destination image buffer for pixel-order image. An array of pointers to the destination image buffers in each color plane for planar image.

dstStep

Distance in bytes between starts of consecutive lines in the destination image for operations with ROI.

roiSize

Size of the source and destination ROI in pixels.

imgSize

Size of the source and destination images in pixels for operations without ROI.

Description

This function converts the $Y'U'V'$ image $pSrc$ to the gamma-corrected $R'G'B'$ image $pDst$ according to the same formulas as the function ippiYUVToRGB does. The difference is that ippiYUV422ToRGB4:2:0 sampling the input data to be in 4:2:2 sampling format (see Table “Pixel-Order Image Formats” and Table “Planar Image Formats” for more details).

The function ippiYUV422ToRGB_P3AC4R additionally creates an alpha channel in the destination image with alpha values set to zero.

Some function flavors operates with ROI (see Regions of Interest in Intel IPP). The function flavors that do not use ROI operate on the assumption that both the source and destination images have the same size and occupy a contiguous memory area, which means that image rows are not padded with zeroes. In this case the step arguments are not needed.
Return Values

ippStsNoErr  
Indicates no error. Any other value indicates an error or a 
warning.

ippStsNullPtrErr  
Indicates an error condition if $pSrc$ or $pDst$ is NULL.

ippStsSizeErr  
Indicates an error condition if $roiSize$ or $imgSize$ has a field 
with a zero or negative value.

RGBToYUV420

Converts an RGB image to the 4:2:0 YUV image.

Syntax

Case 1: Operation on planar data with ROI

$$\text{IppStatus ippiRGBToYUV420}_8u_\text{P3R}(\text{const Ipp8u* } pSrc[3], \text{int srcStep, Ipp8u* } pDst[3], \text{int } dstStep[3], \text{IppiSize roiSize});$$

Case 2: Operation on planar data without ROI

$$\text{IppStatus ippiRGBToYUV420}_8u_\text{P3}(\text{const Ipp8u*} pSrc[3], \text{Ipp8u* } pDst[3], \text{IppiSize imgSize});$$

Case 3: Conversion from pixel-order to planar data with ROI

$$\text{IppStatus ippiRGBToYUV420}_8u_\text{C3P3R}(\text{const Ipp8u* } pSrc, \text{int srcStep, Ipp8u* } pDst[3], \text{int } dstStep[3], \text{IppiSize roiSize});$$

Case 4: Conversion from pixel-order to planar data without ROI

$$\text{IppStatus ippiRGBToYUV420}_8u_\text{C3P3}(\text{const Ipp8u* } pSrc, \text{Ipp8u* } pDst[3], \text{IppiSize imgSize});$$

Include Files

ippcc.h

Domain Dependencies

Headers: ippcore.h, ippvm.h, ipps.h,ippi.h
Libraries: ippcore.lib, ippvm.lib, ipps.lib,ippi.lib

Parameters

$pSrc$  
Pointer to the source image buffer for pixel-order image. An array of 
pointers to the source image buffers in each color plane for planar 
image.

$srcStep$  
Distance in bytes between starts of consecutive lines in the source 
image for operations with ROI.

$pDst$  
An array of pointers to the destination image buffers in each color 
plane.

dstStep  
An array of distances in bytes between starts of consecutive lines in 
each plane of the destination image for operations with ROI.

$roiSize$  
Size of the source and destination ROI in pixels.

$imgSize$  
Size of the source and destination images in pixels for operations 
without ROI.
Description
This function converts the gamma-corrected R'G'B' image pSrc to the Y'U'V' image pDst with the 4:2:0 sampling (see Table “Planar Image Formats” for more details). The conversion is performed in the accordance with the same formulas as the functionippiRGBToYUV does.

For digital RGB values in the range [0..255], Y' has the range [0..255], U varies in the range [-112..+112], and V in the range [-157..+157]. To fit in the range of [0..255], a constant value 128 is added to computed U and V values, and V is then saturated.

Some function flavors operates with ROI see Regions of Interest in Intel IPP).

The function flavors that does not use ROI operate on the assumption that both the source and destination images have the same size and occupy a contiguous memory area, which means that image rows are not padded with zeroes. In this case the step parameters are not needed.

roiSize.width (imgSize.width) and roiSize.height (imgSize.height) should be multiples of 2. Otherwise, the function reduces their original values to the nearest multiples of 2, performs operation, and returns warning message.

Return Values

<table>
<thead>
<tr>
<th>Status Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ippStsNoErr</td>
<td>Indicates no error. Any other value indicates an error or a warning.</td>
</tr>
<tr>
<td>ippStsNullPtrErr</td>
<td>Indicates an error condition if pSrc or pDst is NULL.</td>
</tr>
<tr>
<td>ippStsSizeErr</td>
<td>Indicates an error condition if roiSize or imgSize has a field with a zero or negative value.</td>
</tr>
<tr>
<td>ippStsDoubleSize</td>
<td>Indicates a warning if roiSize or imgSize has a field that is not a multiple of 2.</td>
</tr>
</tbody>
</table>

YUV420ToRGB

Converts a YUV image that has 4:2:0 sampling format to the RGB image.

Syntax

Case 1: Operation on planar data with ROI
IppStatus ippiYUV420ToRGB_8u_P3R(const Ipp8u* pSrc[3], int srcStep[3], Ipp8u* pDst[3], int dstStep, IppiSize roiSize);

Case 2: Operation on planar data without ROI
IppStatus ippiYUV420ToRGB_8u_P3(const Ipp8u* pSrc[3], Ipp8u* pDst[3], IppiSize imgSize);

Case 3: Conversion from planar to pixel-order data with ROI
IppStatus ippiYUV420ToRGB_<mod>(const Ipp8u* pSrc[3], int srcStep[3], Ipp8u* pDst, int dstStep, IppiSize roiSize);

Supported values for mod:

- 8u_P3C3R
- 8u_P3AC4R

Case 4: Conversion from planar to pixel-order data without ROI
IppStatus ippiYUV420ToRGB_8u_P3C3(const Ipp8u* pSrc[3], Ipp8u* pDst, IppiSize imgSize);
Include Files
ippcc.h

Domain Dependencies
Headers: ippcore.h, ippvm.h, ipps.h,ippi.h
Libraries: ippcore.lib, ippvm.lib, ipps.lib,ippi.lib

Parameters

\begin{itemize}
\item \texttt{pSrc} An array of pointers to the source image buffers in each color plane.
\item \texttt{srcStep} An array of distances in bytes between starts of consecutive lines in each source image planes for operations with ROI.
\item \texttt{pDst} Pointer to the destination image buffer for pixel-order images. An array of pointers to the destination image buffers in each color plane for planar images.
\item \texttt{dstStep} Distance in bytes between starts of consecutive lines in the destination image for operations with ROI.
\item \texttt{roiSize} Size of the source and destination ROI in pixels.
\item \texttt{imgSize} Size of the source and destination images in pixels for operations without ROI.
\end{itemize}

Description

This function converts the Y'U'V' image \texttt{pSrc} to the gamma-corrected R'G'B' image \texttt{pDst} according to the same formulas as the function \texttt{ippiYUVToRGB} does. The difference is that \texttt{ippiYUV420ToRGB4:2:0} sampling the input data to be in 4:2:2 sampling format (see Table "Planar Image Formats" for more details).

The function \texttt{ippiYUV420ToRGB_P3AC4R} additionally creates an alpha channel in the destination image with alpha values set to zero.

Some function flavors operates with ROI see Regions of Interest in Intel IPP ).

The function flavors that does not use ROI operate on the assumption that both the source and destination images have the same size and occupy a contiguous memory area, which means that image rows are not padded with zeroes. In this case the step parameters are not needed. \texttt{roiSize.width(imgSize.width)} and \texttt{roiSize.height(imgSize.height)} should be multiples of 2. Otherwise, the function reduces their original values to the nearest multiples of 2, performs operation, and returns warning message.

Return Values

\begin{itemize}
\item \texttt{ippStsNoErr} Indicates no error. Any other value indicates an error or a warning.
\item \texttt{ippStsNullPtrErr} Indicates an error condition if \texttt{pSrc} or \texttt{pDst} pointer is NULL.
\item \texttt{ippStsSizeErr} Indicates an error condition if \texttt{roiSize} or \texttt{imgSize} has a field with a zero or negative value.
\item \texttt{ippStsDoubleSize} Indicates a warning if \texttt{roiSize} or \texttt{imgSize} has a field that is not a multiple of 2.
\end{itemize}
BGRToYUV420
Converts an BGR image to the YUV color model; uses 4:2:0 sampling

Syntax
IppStatus ippiBGRToYUV420_8u_AC4P3R(const Ipp8u* pSrc, int srcStep, Ipp8u* pDst[3], int dstStep[3], IppiSize roiSize);

Include Files
ippcc.h

Domain Dependencies
Headers: ippcore.h, ippvm.h, ipps.h, ippi.h
Libraries: ippcore.lib, ippvm.lib, ipps.lib, ippi.lib

Parameters
pSrc
Pointer to the source image ROI.

srcStep
Distance in bytes between starts of consecutive lines in the source image.

pDst
An array of pointers to the destination image buffers in each color plane.

dstStep
An array of distances in bytes between starts of consecutive lines in each plane of the destination image.

roiSize
Size of the source and destination ROI in pixels.

Description
This function operates with ROI (see Regions of Interest in Intel IPP).

This function converts the gamma-corrected B'G'R' image pSrc to the Y'U'V' image pDst with the 4:2:0 sampling (see Table “Planar Image Formats” for more details). The function uses the same formulas as the function ippiRGBToYUV does.

For digital BGR values in the range [0..255], Y' varies in the range [0..255], U - in the range [-112..+112], and V - in the range [-157..+157]. To fit in the range of [0..255], a constant value 128 is added to the computed U and V values, and V is then saturated.

roiSize.width and roiSize.height should be multiples of 2. If not the function reduces their original values to the nearest multiples of 2, performs operation, and returns warning message.

Return Values
ippStsNoErr
Indicates no error. Any other value indicates an error or a warning.

ippStsNullPtrErr
Indicates an error condition if pSrc or pDst is NULL.

ippStsSizeErr
Indicates an error condition if roiSize or imgSize has a field with a zero or negative value.

ippStsDoubleSize
Indicates a warning if roiSize or imgSize has a field that is not a multiple of 2.
YUV420ToBGR
Converts a YUV image that has 4:2:0 sampling to the BGR image.

Syntax
IppStatus ippiYUV420ToBGR_8u_P3C3R(const Ipp8u* pSrc[3], int srcStep[3], Ipp8u* pDst, int dstStep, IppiSize roiSize);

Include Files
ippcc.h

Domain Dependencies
Headers: ippcore.h, ippvm.h, ipps.h,ippi.h
Libraries: ippcore.lib, ippvm.lib, ipps.lib,ippi.lib

Parameters
pSrc  An array of pointers to ROI in each color plane in the source image.
srcStep An array of distances in bytes between starts of consecutive lines in the source image planes.
pDst  Pointer to the destination image ROI.
dstStep Distance in bytes between starts of consecutive lines in the destination image.
roiSize Size of the source and destination ROI in pixels.

Description
This function operates with ROI (see Regions of Interest in Intel IPP).
This function converts the Y'U'V' image pSrc to the gamma-corrected B'G'R' image pDst according to the same formulas as the function ippiYUVToRGB does. The input data must be presented in the 4:2:0 sampling format (see Table "Planar Image Formats" for more details).
roiSize.width and roiSize.height should be multiples of 2. Otherwise, the function reduces their original values to the nearest multiples of 2, performs operation, and returns warning message.

Return Values
ippStsNoErr Indicates no error. Any other value indicates an error or a warning.
ippStsNullPtrErr Indicates an error condition if pSrc or pDst is NULL.
ippStsSizeErr Indicates an error condition if roiSize has a field with a zero or negative value.
ippStsDoubleSize Indicates a warning if roiSize has a field that is not a multiple of 2.

YUV422v210ToRGB, YUV422v210ToBGR
Converts a YUV422 (v210) image to a RGB/BGR image for ITU-R BT.709 HDTV signal.
Syntax

IppStatus ippiYUV422v210ToRGB_709HDTV_32u16u_C3(const Ipp32u* pSrc, int srcStep, Ipp16u* pDst, int dstStep, IppiSize roiSize);
IppStatus ippiYUV422v210ToBGR_709HDTV_32u16u_C3(const Ipp32u* pSrc, int srcStep, Ipp16u* pDst, int dstStep, IppiSize roiSize);

Include Files

ippcc.h

Domain Dependencies

Headers: ippcore.h, ippvm.h, ipps.h, ippi.h
Libraries: ippcore.lib, ippvm.lib, ipps.lib, ippi.lib

Parameters

pSrc  
Pointer to the source image ROI.

srcStep  
Distance, in bytes, between starts of consecutive lines in the source image.

pDst  
Pointer to the destination image ROI.

dstStep  
Distance, in bytes, between starts of consecutive lines in the destination image.

roiSize  
Size of the source and destination ROI in pixels.

Description

This function operates with ROI (see Regions of Interest in Intel IPP).

This function converts the YUV image pSrc, packed in the 4:2:2 sampling format, to the gamma-corrected RGB/BGR image pDst for digital component video signals in compliance with the ITU-R BT.709 Recommendation [ITU709] for high-definition TV (HDTV). The source YUV image has the following sequence of bytes: UYV|YUY|VYU|YVY, ... . The conversion is performed according to the following formulas:

\[
\begin{align*}
R &= Y + 1.540*(V - 512) \\
G &= Y - 0.459*(V - 512) - 0.183*(U - 512) \\
B &= Y + 1.816*(U - 512)
\end{align*}
\]

The output RGB/BGR values are saturated to the range R [0..31], G [0..63], B [0..31].

Return Values

ippStsNoErr  
Indicates no error. Any other value indicates an error or a warning.

ippStsNullPtrErr  
Indicates an error condition if pSrc or pDst is NULL.

ippStsSizeErr  
Indicates an error condition if roiSize has a field with a zero or negative value.

YUV422v210ToGray

Converts a YUV422 (v210) image to a grayscale image for ITU-R BT.709 HDTV signal
Syntax

IppStatus ippiYUV422v210ToGray_709HDTV_32u16u_C3C1(const Ipp32u* pSrc, int srcStep, Ipp16u* pDst, int dstStep, IppiSize roiSize);

Include Files

ippcc.h

Domain Dependencies

Headers: ippcore.h, ippvm.h, ipps.h,ippi.h
Libraries: ippcore.lib, ippvm.lib, ipps.lib,ippi.lib

Parameters

pSrc Pointer to the source image ROI.
srcStep Distance, in bytes, between starts of consecutive lines in the source image.
pDst Pointer to the destination image ROI.
dstStep Distance, in bytes, between starts of consecutive lines in the destination image.
roiSize Size of the source and destination ROI in pixels.

Description

This function operates with ROI (see Regions of Interest in Intel IPP).
This function converts the YUV image pSrc, packed in the 4:2:2 sampling format, to the grayscale 16U_C1 image pDst for digital component video signals in compliance with the ITU-R BT.709 Recommendation [ITU709] for high-definition TV (HDTV). The source YUV image has the following sequence of bytes: UY|YUY|VYU|YVY, ...

Return Values

ippStsNoErr Indicates no error. Any other value indicates an error or a warning.
ippStsNullPtrErr Indicates an error condition if pSrc or pDst is NULL.
ippStsSizeErr Indicates an error condition if roiSize has a field with a zero or negative value.

RGBToYCbCr

Converts an RGB image to the YCbCr color model.

Syntax

Case 1: Operation on pixel-order data

IppStatus ippiRGBToYCbCr_<mod>(const Ipp8u* pSrc, int srcStep, Ipp8u* pDst, int dstStep, IppSize roiSize);

Supported values for mod:

8u_C3R  8u_AC4R
Case 2: Operation on planar data

IppStatus ippiRGBToYCbCr_8u_P3R(const Ipp8u* pSrc[3], int srcStep, Ipp8u* pDst[3], int dstStep, IppSize roiSize);

Case 3: Conversion from pixel-order to planar data

IppStatus ippiRGBToYCbCr_<mod>(const Ipp8u* pSrc, int srcStep, Ipp8u* pDst[3], int dstStep, IppSize roiSize);

Supported values for mod:

8u_C3P3R  8u_AC4P3R

Include Files
ippcc.h

Domain Dependencies
Headers: ippcore.h, ippvm.h, ipps.h,ippi.h
Libraries: ippcore.lib, ippvm.lib, ipps.lib,ippi.lib

Parameters

\textbf{pSrc} \hspace{2cm} \text{Pointer to the source image ROI for a pixel-order image. An array of pointers to ROI in each separate source color planes for planar images.}

\textbf{srcStep} \hspace{2cm} \text{Distance in bytes between starts of consecutive lines in the source image.}

\textbf{pDst} \hspace{2cm} \text{Pointer to the destination image ROI. Array of pointers to ROI in the separate destination color planes for planar images.}

\textbf{dstStep} \hspace{2cm} \text{Distance in bytes between starts of consecutive lines in the destination image.}

\textbf{roiSize} \hspace{2cm} \text{Size of the source and destination ROI in pixels.}

Description

This function operates with ROI (see Regions of Interest in Intel IPP).

This function converts the gamma-corrected $R'G'B'$ image \texttt{pSrc} with values in the range $[0..255]$ to the $Y'Cb'Cr'$ image \texttt{pDst} according to the following formulas:

\begin{align*}
Y' &= 0.257*R' + 0.504*G' + 0.098*B' + 16 \\
Cb' &= -0.148*R' - 0.291*G' + 0.439*B' + 128 \\
Cr' &= 0.439*R' - 0.368*G' - 0.071*B' + 128
\end{align*}

In the $YCbCr$ model, $Y$ is defined to have a nominal range $[16..235]$, while $Cb$ and $Cr$ are defined to have a range $[16..240]$, with the value of 128 as corresponding to zero.

Both the source and destination images have the same bit depth.

Return Values

\textbf{ippStsNoErr} \hspace{2cm} \text{Indicates no error. Any other value indicates an error or a warning.}

\textbf{ippStsNullPtrErr} \hspace{2cm} \text{Indicates an error condition if \texttt{pSrc} or \texttt{pDst} is NULL.}
Indicates an error condition if roiSize has a field with a zero or negative value.

**YCbCrToRGB**

*Converts a YCbCr image to the RGB color model.*

**Syntax**

**Case 1: Operation on pixel-order data**

IppStatus ippiYCbCrToRGB_<mod>(const Ipp8u* pSrc, int srcStep, Ipp8u* pDst, int dstStep, IppiSize roiSize);

Supported values for mod:

- 8u_C3R
- 8u_AC4R

**Case 2: Operation on planar data**

IppStatus ippiYCbCrToRGB_8u_P3R(const Ipp8u* pSrc[3], int srcStep, Ipp8u* pDst[3], int dstStep, IppiSize roiSize);

IppStatus ippiYCbCrToRGB_8u_P3C3R(const Ipp8u* pSrc[3], int srcStep, Ipp8u* pDst, int dstStep, IppiSize roiSize);

**Case 3: Conversion from planar to pixel-order data**

IppStatus ippiYCbCrToRGB_8u_P3C4R(const Ipp8u* pSrc[3], int srcStep, Ipp8u* pDst, int dstStep, IppiSize roiSize, Ipp8u aval);

**Include Files**

ippcc.h

**Domain Dependencies**

*Headers:* ippcore.h, ippvm.h, ipps.h,ippi.h

*Libraries:* ippcore.lib, ippvm.lib, ipps.lib,ippi.lib

**Parameters**

- **pSrc**  
  Pointer to the source image ROI for pixel-order image. Array of pointers to the ROI in each separate source color planes for planar images.

- **srcStep**  
  Distance in bytes between starts of consecutive lines in the source image.

- **pDst**  
  Pointer to the destination image ROI. Array of pointers to the ROI in the separate destination color planes for planar images.

- **dstStep**  
  Distance in bytes between starts of consecutive lines in the destination image.

- **roiSize**  
  Size of the source and destination ROI in pixels.

- **aval**  
  Constant value to create the fourth channel.

**Description**

This function operates with ROI (see Regions of Interest in Intel IPP).
This function converts the Y'Cb'Cr' image `pSrc` to the 24-bit gamma-corrected R'G'B' image `pDst`. The following formulas are used for conversion:

\[
\begin{align*}
R' &= 1.164 \times (Y' - 16) + 1.596 \times (Cr' - 128) \\
G' &= 1.164 \times (Y' - 16) - 0.813 \times (Cr' - 128) - 0.392 \times (Cb' - 128) \\
B' &= 1.164 \times (Y' - 16) + 2.017 \times (Cb' - 128)
\end{align*}
\]

The output R'G'B' values are saturated to the range [0..255].

The fourth channel is created by setting channel values to the constant value `aval`.

**Return Values**

- `ippStsNoErr`: Indicates no error. Any other value indicates an error or a warning.
- `ippStsNullPtrErr`: Indicates an error condition if `pSrc` or `pDst` pointer is NULL.
- `ippStsSizeErr`: Indicates an error condition if `roiSize` has a field with a zero or negative value.

### YCbCrToBGR

**Converts a YCbCr image to the BGR color model.**

**Syntax**

```c
IppStatus ippiYCbCrToBGR_8u_P3C3R(const Ipp8u* pSrc[3], int srcStep, Ipp8u* pDst, int dstStep, IppiSize roiSize);
IppStatus ippiYCbCrToBGR_8u_P3C4R(const Ipp8u* pSrc[3], int srcStep, Ipp8u* pDst, int dstStep, IppiSize roiSize, Ipp8u aval);
```

**Include Files**

`ippcc.h`

**Domain Dependencies**

**Headers:** `ippcore.h`, `ippvm.h`, `ipps.h`, `ippi.h`

**Libraries:** `ippcore.lib`, `ippvm.lib`, `ipps.lib`, `ippi.lib`

**Parameters**

- `pSrc`: An array of pointers to ROI in each separate source color planes.
- `srcStep`: Distance in bytes between starts of consecutive lines in the source image.
- `pDst`: Pointer to the destination image ROI.
- `dstStep`: Distance in bytes between starts of consecutive lines in the destination image.
- `roiSize`: Size of the source and destination ROI in pixels.
- `aval`: Constant value to create the fourth channel.

**Description**

This function operates with ROI (see Regions of Interest in Intel IPP).
This function converts the Y'Cb'Cr' image *pSrc* to the 24-bit gamma-corrected B'G'R' image *pDst* according to the same formulas as the function *ippiYCbCrToRGB* does. The output B'G'R' values are saturated to the range [0..255].

The fourth channel is created by setting channel values to the constant value *aval*.

**Return Values**

- **ippStsNoErr**
  Indicates no error. Any other value indicates an error or a warning.
- **ippStsNullPtrErr**
  Indicates an error condition if *pSrc* or *pDst* is NULL.
- **ippStsSizeErr**
  Indicates an error condition if *roiSize.width* is less than 4 or *roiSize.height* is less than 1.

---

**YCbCrToBGR_709CSC**

*Converts a YCbCr image to the BGR image for ITU-R BT.709 CSC signal.*

**Syntax**

```c
IppStatus ippiYCbCrToBGR_709CSC_8u_P3C3R(const Ipp8u* pSrc[3], int srcStep, Ipp8u* pDst, int dstStep, IppiSize roiSize);
IppStatus ippiYCbCrToBGR_709CSC_8u_P3C4R(const Ipp8u* pSrc[3], int srcStep, Ipp8u* pDst, int dstStep, IppiSize roiSize, Ipp8u aval);
```

**Include Files**

`ippcc.h`

**Domain Dependencies**

**Headers:** `ippcore.h`, `ippvm.h`, `ipps.h`, `ippi.h`

**Libraries:** `ippcore.lib`, `ippvm.lib`, `ipps.lib`, `ippi.lib`

**Parameters**

- **pSrc**
  An array of pointers to ROI in separate planes of the source image.

- **srcStep**
  An array of distances in bytes between starts of consecutive lines in the source image planes.

- **pDst**
  Pointer to the destination image ROI.

- **dstStep**
  Distance in bytes between starts of consecutive lines in the destination image.

- **roiSize**
  Size of the source and destination ROI in pixels.

- **aval**
  Constant value to create fourth channel.

**Description**

This function operates with ROI (see Regions of Interest in Intel IPP).

This function converts a planar Y'Cb'Cr' image *pSrc* to the three- or four-channel gamma-corrected B'G'R' image *pDst* for digital component video signals complied with the ITU-R BT.709 Recommendation [ITU709] for computer systems consideration (CSC). The conversion is performed according to the following formulas [Jack01]:

---

[etu709]

[Jack01]:

---
R' = 1.164*(Y' - 16) + 1.793*(Cr' - 128)
G' = 1.164*(Y' - 16) - 0.534*(Cr' - 128) - 0.213*(Cb' - 128)
B' = 1.164*(Y' - 16) + 2.115*(Cb' - 128)

The output R'G'B' values are saturated to the range [0..255].
The fourth channel is created by setting channel values to the constant value aval.

Return Values

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ippStsNoErr</td>
<td>Indicates no error. Any other value indicates an error or a warning.</td>
</tr>
<tr>
<td>ippStsNullPtrErr</td>
<td>Indicates an error condition if pSrc or pDst is NULL.</td>
</tr>
<tr>
<td>ippStsSizeErr</td>
<td>Indicates an error condition if roiSize has a field with a zero or negative value.</td>
</tr>
</tbody>
</table>

RGBToYCbCr422

Converts an RGB image to the YCbCr image with 4:2:2 sampling.

Syntax

Case 1: Operation on pixel-order data

IppStatus ippiRGBToYCbCr422_8u_C3C2R(const Ipp8u* pSrc, int srcStep, Ipp8u* pDst, int dstStep, IppiSize roiSize);

Case 2: Conversion from pixel-order to planar data

IppStatus ippiRGBToYCbCr422_8u_C3P3R(const Ipp8u* pSrc, int srcStep, Ipp8u* pDst[3], int dstStep[3], IppiSize roiSize);

Case 2: Conversion from planar to pixel-order data

IppStatus ippiRGBToYCbCr422_8u_P3C2R(const Ipp8u* pSrc[3], int srcStep, Ipp8u* pDst, int dstStep, IppiSize roiSize);

Include Files

ippcc.h

Domain Dependencies

Headers: ippcore.h, ippvm.h, ipps.h,ippi.h
Libraries: ippcore.lib, ippvm.lib, ipps.lib,ippi.lib

Parameters

pSrc

Pointer to the source image ROI for pixel-order image. An array of pointers to ROI in each separate source color planes for planar images.

srcStep

Distance in bytes between starts of consecutive lines in the source image.

pDst

Pointer to the destination image ROI for pixel-order image. An array of pointer to ROI in each separate planes for the planar destination image.
**dstStep**
Distance in bytes between starts of consecutive lines in the destination image.

**roiSize**
Size of the source and destination ROI in pixels.

**Description**
This function operates with ROI (see Regions of Interest in Intel IPP). This function converts the gamma-corrected R'G'B' image `pSrc` to the Y'Cb'C' image `pDst` with 4:2:2 sampling (see Table “Pixel-Order Image Formats” and Table “Planar Image Formats” for more details). The conversion is performed according to the same formulas as the function `ippiRGBToYCbCr` does.

The converted buffer for pixel-order image has the reduced bit depth of a 16 bits per pixel, whereas the source buffer has 24 bit depth.

**Return Values**
- `ippStsNoErr`: Indicates no error. Any other value indicates an error or a warning.
- `ippStsNullPtrErr`: Indicates an error condition if `pSrc` or `pDst` is NULL.
- `ippStsSizeErr`: Indicates an error condition if `roiSize` has a field with a zero or negative value.

**YCbCr422ToRGB**
Converts an YCbCr image with the 4:2:2 sampling to the RGB image.

**Syntax**

**Case 1: Operation on pixel-order data**
```c
IppStatus ippiYCbCr422ToRGB_8u_C2C3R(const Ipp8u* pSrc, int srcStep, Ipp8u* pDst, int dstStep, IppiSize roiSize);
IppStatus ippiYCbCr422ToRGB_8u_C2C4R(const Ipp8u* pSrc, int srcStep, Ipp8u* pDst, int dstStep, IppiSize roiSize, Ipp8u* aval);
```

**Case 2: Conversion from pixel-order to planar data**
```c
IppStatus ippiYCbCr422ToRGB_8u_C2P3R(const Ipp8u* pSrc, int srcStep, Ipp8u* pDst[3], int dstStep, IppiSize roiSize);
```

**Case 3: Conversion from planar to pixel-order data**
```c
IppStatus ippiYCbCr422ToRGB_8u_P3C3R(const Ipp8u* pSrc[3], int srcStep[3], Ipp8u* pDst, int dstStep, IppiSize roiSize);
```

**Include Files**
```c
ippcc.h
```

**Domain Dependencies**
- **Headers**: ippcore.h, ippvm.h, ipps.h, ippi.h
- **Libraries**: ippcore.lib, ippvm.lib, ipps.lib, ippi.lib
Parameters

- **pSrc**: Pointer to the source image ROI for pixel-order image. An array of pointers to ROI in each separate source planes for planar images. Distance, in bytes, between the starting points of consecutive lines in the source image.
- **srcStep**: Distance, in bytes, between the starting points of consecutive lines in the source image.
- **pDst**: Pointer to the ROI in the destination pixel-order image. An array of pointers to ROI in each planes of the destination planar image.
- **dstStep**: Distance, in bytes, between the starting points of consecutive lines in the destination image.
- **roiSize**: Size of the source and destination ROI in pixels.
- **aval**: Constant value to create the fourth channel.

Description

This function operates with ROI (see Regions of Interest in Intel IPP).

This function converts the \( Y'Cb'Cr' \) image \( pSrc \) with the 4:2:2 sampling (see Table "Pixel-Order Image Formats" and Table "Planar Image Formats" for more details) to the gamma-corrected \( R'G'B' \) image \( pDst \) according to the same formulas as the function \( \text{ippiYCbCrToRGB} \) does. The output \( R'G'B' \) values are saturated to the range \([0..255]\).

Return Values

- **ippStsNoErr**: Indicates no error. Any other value indicates an error or a warning.
- **ippStsNullPtrErr**: Indicates an error condition if \( pSrc \) or \( pDst \) is NULL.
- **ippStsSizeErr**: Indicates an error condition if \( roiSize \) has a field with a zero or negative value.

Example

The code example below demonstrates how to use the \( \text{ippiYCbCr422ToRGB\_8u\_C2C4R} \) function.

```c
const int WIDTH = 2;
const int HEIGHT = 2;

Ipp8u pSrc[WIDTH * HEIGHT * 2] =
{
    236,50,236,80,
    236,50,236,80,
};
Ipp8u pDstRGB[(WIDTH * HEIGHT) * 4];
int srcStep = WIDTH * 2, dstStep = WIDTH * 4;
IppiSize roiSize = {WIDTH, HEIGHT};
Ipp8u alphaValue = 0xFF;
IppStatus status = ippiYCbCr422ToRGB_8u_C2C4R(pSrc, srcStep, pDstRGB, dstStep, roiSize, alphaValue);
if ( status == ippStsNoErr)
    printf("\n *************  passed *************\n");
else
    printf("\n *************  failed *************\n");
```
**RGBToYCrCb422**

Converts 24-bit per pixel RGB image to 16-bit per pixel YCrCb image

**Syntax**

**Case 1: Operation on pixel-order data**

```c
IppStatus ippiRGBToYCrCb422_8u_C3C2R(const Ipp8u* pSrc, int srcStep, Ipp8u* pDst, int dstStep, IppiSize roiSize);
```

**Case 2: Conversion from planar to pixel-order data**

```c
IppStatus ippiRGBToYCrCb422_8u_P3C2R(const Ipp8u* pSrc[3], int srcStep, Ipp8u* pDst, int dstStep, IppiSize roiSize);
```

**Include Files**

ippcc.h

**Domain Dependencies**

*Headers:* ippcore.h, ippvm.h, ipps.h,ippi.h

*Libraries:* ippcore.lib, ippvm.lib, ipps.lib,ippi.lib

**Parameters**

- **pSrc**
  Pointer to the source image ROI for pixel-order image. An array of pointers to ROI in each separate source color planes for planar images.

- **srcStep**
  Distance in bytes between starts of consecutive lines in the source image.

- **pDst**
  Pointer to the destination image ROI.

- **dstStep**
  Distance in bytes between starts of consecutive lines in the destination image.

- **roiSize**
  Size of the source and destination ROI in pixels.

**Description**

This function operates with ROI (see Regions of Interest in Intel IPP). This function converts the gamma-corrected R'G'B' image \( pSrc \) to the Y'Cb'Cr' image \( pDst \) according to the same formulas as the function `ippiRGBToYCrCb422` does. The difference is that `ippiRGBToYCrCb422` uses 4:2:2 sampling format for the converted image (see Table "Pixel-Order Image Formats" and Table "Planar Image Formats" for more details).

The converted buffer has the reduced bit depth of 16 bits per pixel, whereas the source buffer has 24 bit depth.

**Return Values**

- **ippStsNoErr**
  Indicates no error. Any other value indicates an error or a warning.

- **ippStsNullPtrErr**
  Indicates an error condition if \( pSrc \) or \( pDst \) is NULL.

- **ippStsSizeErr**
  Indicates an error condition if \( roiSize \) has a field with a zero or negative value.
YCrCb422ToRGB, YCrCb422ToBGR

Convert 16-bit per pixel YCrCb image to 24 or 32-bit per pixel RGB or BGR image.

Syntax

Case 1: Operation on pixel-order data

IppStatusippiYCrCb422ToRGB_8u_C2C3R(const Ipp8u* pSrc, int srcStep, Ipp8u* pDst, int dstStep, IppiSize roiSize);
IppStatusippiYCrCb422ToRGB_8u_C2C4R(const Ipp8u* pSrc, int srcStep, Ipp8u* pDst, int dstStep, IppiSize roiSize, Ipp8u aval);
IppStatusippiYCrCb422ToBGR_8u_C2C3R(const Ipp8u* pSrc, int srcStep, Ipp8u* pDst, int dstStep, IppiSize roiSize);
IppStatusippiYCrCb422ToBGR_8u_C2C4R(const Ipp8u* pSrc, int srcStep, Ipp8u* pDst, int dstStep, IppiSize roiSize, Ipp8u aval);

Case 2: Conversion from pixel-order to planar data

IppStatusippiYCrCb422ToRGB_8u_C2P3R(const Ipp8u* pSrc, int srcStep, Ipp8u* pDst[3], int dstStep, IppiSize roiSize);

Include Files

ippcc.h

Domain Dependencies

Headers: ippcore.h, ippvm.h, ipps.h,ippi.h
Libraries: ippcore.lib, ippvm.lib, ipps.lib,ippi.lib

Parameters

pSrc  Pointer to the source image ROI for pixel-order image. An array of pointers to ROI in each separate source plane for planar images.
srcStep  Distance, in bytes, between the starting points of consecutive lines in the source image.
Pdst  Pointer to the ROI in the destination pixel-order image. An array of pointers to ROI in each plane of the destination planar image.
dstStep  Distance, in bytes, between the starting points of consecutive lines in the destination image.
roiSize  Size of the source and destination ROI in pixels.
aval  Constant value to create the fourth channel.

Description

This function operates with ROI (see Regions of Interest in Intel IPP).

This function converts the Y'Cr'C'b' image pSrc, packed in 4:2:2 sampling format (see Table "Pixel-Order Image Formats" and Table "Planar Image Formats" for more details) to the 24-bit gamma-corrected R'G'B' or B'G'R' image pDst according to the same formulas as the functionippiYCbCrToRGBdoes. The output R'G'B' values are saturated to the range [0..255]. Y'Cr'C'b' image with 4:2:2 sampling is also known as YVYU format.
Return Values

ippStsNoErr   Indicates no error. Any other value indicates an error or a warning.
ippStsNullPtrErr   Indicates an error condition if pSrc or pDst is NULL.
ippStsSizeErr   Indicates an error condition if roiSize has a field with a zero or negative value.

Example
The code example below demonstrates how to use the ippiYCrCb422ToRGB_8u_C2C4R function.

```c
#define WIDTH 2
#define HEIGHT 2
Ipp8u pSrc[WIDTH * HEIGHT * 2] =
{ 236,50,236,80,
  236,50,236,80,
};
Ipp8u pDstRGB[(WIDTH * HEIGHT) * 4];
int srcStep = WIDTH * 2, dstStep  = WIDTH * 4;
IppiSize roiSize = {WIDTH, HEIGHT};
Ipp8u alphaValue = 0xFF;
IppStatus status = ippiYCrCb422ToRGB_8u_C2C4R(pSrc, srcStep, pDstRGB, dstStep, roiSize, alphaValue);
if ( status == ippStsNoErr)
    printf("\n *************  passed *************\n");
else
    printf("\n *************  failed *************\n");
```

BGRToYCbCr422
Converts 24-bit per pixel BGR image to 16-bit per pixel YCbCr image.

Syntax

Case 1: Operation on pixel-order data

IppStatus ippiBGRToYCbCr422_8u_C3C2R(const Ipp8u* pSrc, int srcStep, Ipp8u* pDst, int dstStep, IppiSize roiSize);
IppStatus ippiBGRToYCbCr422_8u_AC4C2R(const Ipp8u* pSrc, int srcStep, Ipp8u* pDst, int dstStep, IppiSize roiSize);

Case 2: Conversion from pixel-order to planar data

IppStatus ippiBGRToYCbCr422_8u_C3P3R(const Ipp8u* pSrc, int srcStep, Ipp8u* pDst[3], int dstStep[3], IppiSize roiSize);
IppStatus ippiBGRToYCbCr422_8u_AC4P3R(const Ipp8u* pSrc, int srcStep, Ipp8u* pDst[3], int dstStep[3], IppiSize roiSize);

Include Files

ippcc.h

Domain Dependencies

Headers: ippcore.h, ippvm.h, ipps.h, ippi.h
Libraries: ippcore.lib, ippvm.lib, ipps.lib,ippi.lib

Parameters

pSrc Pointer to the source mage ROI.
srcStep Distance in bytes between starts of consecutive lines in the source image.
pDst Pointer to the ROI in the destination pixel-order image. An array of pointers to ROI in each planes of the destination planar image.
dstStep Distance in bytes between starts of consecutive lines in the destination pixel-order image. An array of distances in bytes for each plane of the destination planar image.
roiSize Size of the source and destination ROI in pixels.

Description

This function operates with ROI (see Regions of Interest in Intel IPP). This function converts a three- or four-channel gamma-corrected B'G'R' image pSrc to the two-channel or three-planes Y'Cb'Cr' image pDst according to the same formulas as the function ippiRGBToYCbCr does. The difference is that ippiBGRToYCbCr422 uses the 4:2:2 sampling format (see Table “Pixel-Order Image Formats” and Table “Planar Image Formats” for more details).

Return Values

ippStsNoErr Indicates no error. Any other value indicates an error or a warning.
ippStsNullFtrErr Indicates an error condition if pSrc or pDst is NULL.
ippStsSizeErr Indicates an error condition if roiSize.width is less than 2 or roiSize.height is less than 1.

YCbCr422ToBGR

Converts a YCbCr image with 4:2:2 sampling to the BGR image.

Syntax

Case 1: Operation on pixel-order data
IppStatus ippiYCbCr422ToBGR_8u_C2C3R(const Ipp8u* pSrc, int srcStep, Ipp8u* pDst, int dstStep, IppiSize roiSize);
IppStatus ippiYCbCr422ToBGR_8u_C2C4R(const Ipp8u* pSrc, int srcStep, Ipp8u* pDst, int dstStep, IppiSize roiSize, Ipp8u* aval);

Case 2: Conversion from planar to pixel-order data
IppStatus ippiYCbCr422ToBGR_8u_P3C3R(const Ipp8u* pSrc[3], int srcStep[3], Ipp8u* pDst, int dstStep, IppiSize roiSize);

Include Files

ippcc.h

Domain Dependencies

Headers: ippcore.h, ippvm.h, ipps.h,ippi.h
Libraries: ippcore.lib, ippvm.lib, ipps.lib,ippi.lib

Parameters

pSrc
Pointer to the source image ROI. An array of pointers to the ROI in each separate plane of the source planar image.

srcStep
Distance in bytes between starts of consecutive lines in the source image. An array of such distances in bytes for each plane of the source planar image.

pDst
Pointer to the destination image ROI.

dstStep
Distance in bytes between starts of consecutive lines in the destination image.

roiSize
Size of the source and destination ROI in pixels.

aval
Constant value to create the fourth channel.

Description

This function operates with ROI (see Regions of Interest in Intel IPP).

This function converts the Y'Cb'Cr' image pSrc with 4:2:2 sampling (see Table "Pixel-Order Image Formats" and Table "Planar Image Formats" for more details) to the gamma-corrected B'G'R' image pDst according to the same formulas as the functionippiYCbCrToRGB does.

The output B'G'R' values are saturated to the range [0..255].

The fourth channel is created by setting channel values to the constant value aval.

Return Values

ippStsNoErr
Indicates no error. Any other value indicates an error or a warning.

ippStsNullPtrErr
Indicates an error condition if pSrc or pDst is NULL.

ippStsSizeErr
Indicates an error condition if roiSize.width is less than 2 or roiSize.height is less than 1.

YCbCr422ToGray

Converts an interlaced 4:2:2 YCbCr or YCrCb image to gray-scale extracting luminance (Y) component.

Syntax

IppStatus ippiYCbCr422ToGray_8u_C2C1R(const Ipp8u* pSrc, int srcStep, Ipp8u* pDst, int dstStep, IppiSize roiSize);

Include Files

ippcc.h

Domain Dependencies

Headers: ippcore.h, ippvm.h, ipps.h,ippi.h
Libraries: ippcore.lib, ippvm.lib, ipps.lib,ippi.lib
Parameters

\textbf{pSrc} \hspace{2cm} Pointer to the source image ROI.

\textbf{srcStep} \hspace{2cm} Distance, in bytes, between the starting points of consecutive lines in the source image.

\textbf{pDst} \hspace{2cm} Pointer to the destination image ROI.

\textbf{dstStep} \hspace{2cm} Distance, in bytes, between the starting points of consecutive lines in the destination image.

\textbf{roiSize} \hspace{2cm} Size of the source and destination ROI in pixels.

Description

This function operates with ROI (see Regions of Interest in Intel IPP).

This function converts an interlaced Y'Cb'Cr' or Y'Cr'Cb' image \texttt{pSrc} with the 4:2:2 sampling (see Table "Pixel-Order Image Formats" and Table "Planar Image Formats" for more details) to the gray-scale image \texttt{pDst} extracting luminance (Y) component.

Y'Cb'Cr' image with 4:2:2 sampling is also known as YUY2 format, and Y'Cr'Cb' as YVYU.

Return Values

\textbf{ippStsNoErr} \hspace{2cm} Indicates no error. Any other value indicates an error or a warning.

\textbf{ippStsNullPtrErr} \hspace{2cm} Indicates an error condition if \texttt{pSrc} or \texttt{pDst} is NULL.

\textbf{ippStsSizeErr} \hspace{2cm} Indicates an error condition if \texttt{roiSize} has a field with a zero or negative value.

Example

The code example below demonstrates how to use the \texttt{ippiYCbCr422ToGray_8u_C2C1R} function.

```c
const int WIDTH = 2;
const int HEIGHT = 2;

Ipp8u pSrc[WIDTH * HEIGHT * 2] = {
    190,70,191,80,
    200,71,201,81,
};
Ipp8u pDst[WIDTH * HEIGHT];
int srcStep = WIDTH * 2, dstStep = WIDTH;
IppiSize roiSize = {WIDTH, HEIGHT};
IppStatus status = ippiYCbCr422ToGray_8u_C2C1R(pSrc, srcStep, pDst, dstStep, roiSize);
if ( status == ippStsNoErr)
    printf("PASS: \n\%3d \%3d \n\%3d \%3d\n", pDst[0], pDst[1], pDst[2], pDst[3]);
else
    printf("FAIL: status = \%d\n", status);
```

Result:

```
PASS:
190 191
200 201
```
RGBToCbYCr422, RGBToCbYCr422Gamma
Convert 24-bit per pixel RGB image to 16-bit per pixel CbYCr image.

Syntax
IppStatus ippiRGBToCbYCr422_8u_C3C2R(const Ipp8u* pSrc, int srcStep, Ipp8u* pDst, int dstStep, IppiSize roiSize);
IppStatus ippiRGBToCbYCr422Gamma_8u_C3C2R(const Ipp8u* pSrc, int srcStep, Ipp8u* pDst, int dstStep, IppiSize roiSize);

Include Files
ippcc.h

Domain Dependencies
Headers: ippcore.h, ippvm.h, ipps.h,ippi.h
Libraries: ippcore.lib, ippvm.lib, ipps.lib,ippi.lib

Parameters
pSrc  
Pointer to the source image ROI.
srcStep  
Distance in bytes between starts of consecutive lines in the source image.
pDst  
Pointer to the destination image ROI.
dstStep  
Distance in bytes between starts of consecutive lines in the destination image.
roiSize  
Size of the source and destination ROI in pixels.

Description
These functions operate with ROI (see Regions of Interest in Intel IPP).
The function ippiRGBToCbYCr422 converts the gamma-corrected R'G'B' image pSrc to the Cb'Y'Cr' image pDst according to the same formulas as the function ippiRGBToYCbCr does.
The function ippiRGBToCbYCr422Gamma performs gamma-correction of the source RGB image pSrc according to the same formula as the function ippiGammaFwd does, and then converts it to the Cb'Y'Cr' image pDst according to the same formulas as the function ippiRGBToYCbCr does.
The functions ippiRGBToCbYCr422 and ippiRGBToCbYCr422Gamma use 4:2:2 sampling format for the converted image.
A CbYCr image has the following sequence of bytes: Cb0Y0Cr0Y1, Cb1Y2Cr1Y3, ....

Return Values
ippStsNoErr  
Indicates no error. Any other value indicates an error or a warning.
ippStsNullPtrErr  
Indicates an error condition if pSrc or pDst is NULL.
ippStsSizeErr  
Indicates an error condition if roiSize has a field with a zero or negative value.
**CbYCr422ToRGB**
*Converts 16-bit per pixel CbYCr image to 24-bit per pixel RGB image.*

**Syntax**

```
IppStatus ippiCbYCr422ToRGB_8u_C2C3R(const Ipp8u* pSrc, int srcStep, Ipp8u* pDst, int dstStep, IppiSize roiSize);
```

**Include Files**

ippcc.h

**Domain Dependencies**

Headers: ippcore.h, ippvm.h, ipps.h,ippi.h

Libraries: ippcore.lib, ippvm.lib, ipps.lib,ippi.lib

**Parameters**

- `pSrc`: Pointer to the source image ROI.
- `srcStep`: Distance in bytes between starts of consecutive lines in the source image.
- `pDst`: Pointer to the destination image ROI.
- `dstStep`: Distance in bytes between starts of consecutive lines in the destination image.
- `roiSize`: Size of the source and destination ROI in pixels

**Description**

This function operates with ROI (see Regions of Interest in Intel IPP).

This function converts the \( \text{Cb'}Y'\text{Cr'} \) image `pSrc`, packed in the 4:2:2 sampling format, to the 24-bit gamma-corrected \( \text{R'}\text{G'}B' \) image `pDst` according to the same formulas as the function `ippiYCbCrToRGB` does.

A CbYCr image has the following sequence of bytes: \( \text{Cb}0\text{Y}0\text{Cr}0\text{Y}1, \text{Cb}1\text{Y}2\text{Cr}1\text{Y}3, ... \).

The output \( \text{R'}\text{G'}B' \) values are saturated to the range \([0..255] \).

**Return Values**

- `ippStsNoErr`: Indicates no error. Any other value indicates an error or a warning.
- `ippStsNullPtrErr`: Indicates an error condition if `pSrc` or `pDst` is NULL.
- `ippStsSizeErr`: Indicates an error condition if `roiSize` has a field with a zero or negative value.

**BGRToCbYCr422**
*Converts 32-bit per pixel BGR image to 16-bit per pixel CbYCr image.*
Syntax

IppStatus ippiBGRToCbYCr422_8u_AC4C2R(const Ipp8u* pSrc, int srcStep, Ipp8u* pDst, int dstStep, IppiSize roiSize);

Include Files

ippcc.h

Domain Dependencies

Headers: ippcore.h, ippvm.h, ipps.h,ippi.h
Libraries: ippcore.lib, ippvm.lib, ipps.lib,ippi.lib

Parameters

- pSrc
  Pointer to the source image ROI.
- srcStep
  Distance in bytes between starts of consecutive lines in the source image.
- pDst
  Pointer to the destination image ROI.
- dstStep
  Distance in bytes between starts of consecutive lines in the destination image.
- roiSize
  Size of the source and destination ROI in pixels.

Description

This function operates with ROI (see Regions of Interest in Intel IPP).

This function converts the four-channel gamma-corrected B'G'R' image pSrc to the two-channel Cb'Y'Cr' image pDst according to the same formulas as the function ippiRGBToYCbCr does. The function ippiBGRToCbYCr422 uses 4:2:2 sampling format for the converted image. An alpha-channel information is lost.

An CbYCr image has the following sequence of bytes: Cb0Y0Cr0Y1, Cb1Y2Cr1Y3, ... .

Return Values

- ippStsNoErr
  Indicates no error. Any other value indicates an error or a warning.
- ippStsNullPtrErr
  Indicates an error condition if pSrc or pDst is NULL.
- ippStsSizeErr
  Indicates an error condition if roiSize has a field with a zero or negative value.

BGRToCbYCr422_709HDTV

Converts BGR image to 16-bit per pixel CbYCr image for ITU-R BT.709 HDTV signal.

Syntax

IppStatus ippiBGRToCbYCr422_709HDTV_8u_C3C2R(const Ipp8u* pSrc, int srcStep, Ipp8u* pDst, int dstStep, IppiSize roiSize);
IppStatus ippiBGRToCbYCr422_709HDTV_8u_AC4C2R(const Ipp8u* pSrc, int srcStep, Ipp8u* pDst, int dstStep, IppiSize roiSize);
Include Files
ippcc.h

Domain Dependencies
Headers: ippcore.h, ippvm.h, ipps.h,ippi.h
Libraries: ippcore.lib, ippvm.lib, ipps.lib,ippi.lib

Parameters

\texttt{pSrc}  
Pointer to the source image ROI.

\texttt{srcStep}  
Distance in bytes between starts of consecutive lines in the source image.

\texttt{pDst}  
Pointer to the destination image ROI.

\texttt{dstStep}  
Distance in bytes between starts of consecutive lines in the destination image.

\texttt{roiSize}  
Size of the source and destination ROI in pixels.

Description
This function operates with ROI (see Regions of Interest in Intel IPP).

This function converts the three- or four-channel gamma-corrected \texttt{B'G'R'} image \texttt{pSrc} to the two-channel \texttt{Cb'Y'Cr'} image \texttt{pDst} for digital component video signals complied with the ITU-R BT.709 Recommendation [ITU709] for high-definition TV (HDTV). The source image pixel values are in the range \([16..235]\). The conversion is performed according to the following formulas [Jack01]:

\[
Y' = 0.213*R' + 0.715*G' + 0.072*B' \\
Cb' = -0.117*R' - 0.394*G' + 0.511*B' + 128 \\
Cr' = 0.511*R' - 0.464*G' - 0.047*B' + 128
\]

The values of \texttt{Y'} of the destination image are in the range \([16..235]\), the values of \texttt{Cb'}, \texttt{Cr'} are in the range \([16..240]\). They should be saturated at the 1 and 254 levels.

The function \texttt{ippiBGRToCbYCr422_709HDTV} uses the \texttt{4:2:2} sampling format for the converted image. The alpha-channel information is lost.

A \texttt{CbYCr} image has the following sequence of bytes: \texttt{Cb0Y0Cr0Y1, Cb1Y2Cr1Y3, ...}.

Return Values

\texttt{ippStsNoErr}  
Indicates no error. Any other value indicates an error or a warning.

\texttt{ippStsNullPtrErr}  
Indicates an error condition if \texttt{pSrc} or \texttt{pDst} pointer is \texttt{NULL}.

\texttt{ippStsSizeErr}  
Indicates an error condition if \texttt{roiSize} has a field with a zero or negative value.

\texttt{CbYCr422ToBGR}  
\textit{Converts 16-bit per pixel CbYCr image to four channel BGR image.}
Syntax
IppStatus ippiCbYCr422ToBGR_8u_C2C4R(const Ipp8u* pSrc, int srcStep, Ipp8u* pDst, int dstStep, IppiSize roiSize, Ipp8u aval);

Include Files
ippcc.h

Domain Dependencies
Headers: ippcore.h, ippvm.h, ipps.h,ippi.h
Libraries: ippcore.lib, ippvm.lib, ipps.lib,ippi.lib

Parameters
pSrc Pointer to the source image ROI.
srcStep Distance in bytes between starts of consecutive lines in the source image.
pDst Pointer to the destination image ROI.
dstStep Distance in bytes between starts of consecutive lines in the destination image.
roiSize Size of the source and destination ROI in pixels.
aval Constant value to create the fourth channel.

Description
This function operates with ROI (see Regions of Interest in Intel IPP).
This function converts the Cb'Y'Cr' image pSrc, packed in 4:2:2 sampling format, to the four channel gamma-corrected B'G'R' image pDst according to the same formulas as the function ippiYCbCrToRGB does.
A CbYCr image has the following sequence of bytes: Cb0Y0Cr0Y1, Cb1Y2Cr1Y3, ...
The output B'G'R' values are saturated to the range [0..255].
The fourth channel is created by setting channel values to the constant value aval.

Return Values
ippStsNoErr Indicates no error. Any other value indicates an error or a warning.
ippStsNullPtrErr Indicates an error condition if pSrc or pDst is NULL.
ippStsSizeErr Indicates an error condition if roiSize has a field with a zero or negative value.

CbYCr422ToBGR_709HDTV
Converts 16-bit per pixel CbYCr image to the BGR image for ITU-R BT.709 HDTV signal.

Syntax
IppStatus ippiCbYCr422ToBGR_709HDTV_8u_C2C3R(const Ipp8u* pSrc, int srcStep, Ipp8u* pDst, int dstStep, IppiSize roiSize);
IppStatus ippiCbYCr422ToBGR_709HDTV_8u_C2C4R(const Ipp8u* pSrc, int srcStep, Ipp8u* pDst, int dstStep, IppiSize roiSize, Ipp8u aval);

Include Files

ippcc.h

Domain Dependencies

Headers: ippcore.h, ippvm.h, ipps.h,ippi.h
Libraries: ippcore.lib, ippvm.lib, ipps.lib,ippi.lib

Parameters

pSrc
    Pointer to the source image ROI.

srcStep
    Distance in bytes between starts of consecutive lines in the source image.

pDst
    Pointer to the destination image ROI.

dstStep
    Distance in bytes between starts of consecutive lines in the destination image.

roiSize
    Size of the source and destination ROI in pixels.

aval
    Constant value to create the fourth channel.

Description

This function operates with ROI (see Regions of Interest in Intel IPP).

This function converts the \texttt{Cb'Y'Cr'} image \texttt{pSrc}, packed in \texttt{4:2:2} sampling format, to the three- or four-channel gamma-corrected \texttt{B'G'R'} image \texttt{pDst} for digital component video signals complied with the ITU-R BT.709 Recommendation [ITU709] for high-definition TV (HDTV). A source \texttt{CbYCr} image has the following sequence of bytes: \texttt{Cb0Y0Cr0Y1}, \texttt{Cb1Y2Cr1Y3}, ... . The values of \texttt{Y'} are in the range \([16..235]\), the values of \texttt{Cb'}, \texttt{Cr'} are in the range \([16..240]\). The conversion is performed according to the following formulas [Jack01]:

\[
R' = Y' + 1.540*(Cr' - 128)
G' = Y' - 0.459*(Cr' - 128) - 0.183*(Cb' - 128)
B' = Y' + 1.816*(Cb' - 128)
\]

The destination image pixel values have a nominal range \([16..235]\). The resulting \texttt{R'G'B'} values should be saturated at the 0 and 255 levels.

The output \texttt{B'G'R'} values are saturated to the range \([0..255]\).

The fourth channel is created by setting channel values to the constant value \texttt{aval}.

Return Values

ippStsNoErr
    Indicates no error. Any other value indicates an error or a warning.

ippStsNullPtrErr
    Indicates an error condition if \texttt{pSrc} or \texttt{pDst} is NULL.

ippStsSizeErr
    Indicates an error condition if \texttt{roiSize} has a field with a zero or negative value.
RGBToYCbCr420

Converts an RGB image to the YCbCr color model; uses 4:2:0 sampling.

Syntax

IppStatus ippiRGBToYCbCr420_8u_C3P3R(const Ipp8u* pSrc, int srcStep, Ipp8u* pDst[3], int dstStep[3], IppiSize roiSize);
IppStatus ippiRGBToYCbCr420_8u_C3P2R(const Ipp8u* pSrc, int srcStep, Ipp8u* pDst, int dstStep, Ipp8u* pDstCbCr, int dstCbCrStep, IppiSize roiSize);
IppStatus ippiRGBToYCbCr420_8u_C4P2R(const Ipp8u* pSrc, int srcStep, Ipp8u* pDst, int dstStep, Ipp8u* pDstCbCr, int dstCbCrStep, IppiSize roiSize);

Include Files

ippcc.h

Domain Dependencies

Headers: ippcore.h, ippvm.h, ipps.h,ippi.h
Libraries: ippcore.lib, ippvm.lib, ipps.lib,ippi.lib

Parameters

pSrc
Pointer to the source image ROI.

srcStep
Distance in bytes between starts of consecutive lines in the source image.

pDst
An array of pointers to ROI in the separate planes of the destination image for three-plane image.

dstStep
Array of distances in bytes between starts of consecutive lines in each plane of the destination image for three-plane image.

pDstY
Pointer to ROI in the luminance plane of the destination image for two-plane image.

dstStep, dstYStep
Distance in bytes between starts of consecutive lines in the luminance plane for two-plane image.

pDstCbCr
Pointer to ROI in the interleaved chrominance plane of the destination image for two-plane image.

dstCbCrStep
Distance in bytes between starts of consecutive lines in the interleaved chrominance plane for two-plane image.

roiSize
Size of the source and destination ROI in pixels.

Description

This function operates with ROI (see Regions of Interest in Intel IPP).

This function converts the gamma-corrected R'G'B' image pSrc to the Y'Cb'Cr' image according to the same formulas as the function ippiRGBToYCbCr does. The difference is that ippiRGBToYCbCr420 uses 4:2:0 sampling format for the converted image (see Table "Planar Image Formats" for more details).

roiSize.width should be multiple of 2, and roiSize.height should be multiple of 2 (for three-plane image) or 4 (for two-plane image). If not the function reduces their original values to the nearest multiples of 2 or 4 correspondingly, performs operation, and returns warning message.
Return Values

ippStsNoErr
Indicates no error. Any other value indicates an error or a warning.

ippStsNullPtrErr
Indicates an error condition if pSrc or pDst is NULL.

ippStsSizeErr
Indicates an error condition if roiSize has a field with value less than 2.

ippStsDoubleSize
Indicates a warning if roiSize.width is not a multiple of 2, or if roiSize.height is not a multiple of 2 (for three-plane image) or 4 (for two-plane image).

YCbCr420ToRGB, YCbCr420ToBGR
Convert a YCbCr image that has 4:2:0 sampling format to the RGB or BGR color model.

Syntax

IppStatus ippiYCbCr420ToRGB_8u_P3C3R(const Ipp8u* pSrc[3], int srcStep[3], Ipp8u* pDst, int dstStep, IppSize roiSize);
IppStatus ippiYCbCr420ToRGB_8u_P2C3R(const Ipp8u* pSrcY, int srcYStep, const Ipp8u* pSrcCbCr, int srcCbCrStep, Ipp8u* pDst, int dstStep, IppSize roiSize);
IppStatus ippiYCbCr420ToRGB_8u_P2C4R(const Ipp8u* pSrcY, int srcYStep, const Ipp8u* pSrcCbCr, int srcCbCrStep, Ipp8u* pDst, int dstStep, IppSize roiSize, Ipp8u aval);
IppStatus ippiYCbCr420ToBGR_8u_P2C3R(const Ipp8u* pSrcY, int srcYStep, const Ipp8u* pSrcCbCr, int srcCbCrStep, Ipp8u* pDst, int dstStep, IppSize roiSize);
IppStatus ippiYCbCr420ToBGR_8u_P2C4R(const Ipp8u* pSrcY, int srcYStep, const Ipp8u* pSrcCbCr, int srcCbCrStep, Ipp8u* pDst, int dstStep, IppSize roiSize, Ipp8u aval);

Include Files

ippcc.h

Domain Dependencies

Headers: ippcore.h, ippvm.h, ipps.h, ippi.h
Libraries: ippcore.lib, ippvm.lib, ipps.lib, ippi.lib

Parameters

pSrc
An array of pointers to ROI in separate planes of the three-plane source image.

srcStep
An array of distances in bytes between starts of consecutive lines in each plane of the three-plane source image.

pSrcY
Pointer to ROI in the luminance plane of the two-plane source image.

srcYStep
Distance in bytes between starts of consecutive lines in the luminance plane of the two-plane source image.

pSrcCbCr
Pointer to ROI in the interleaved chrominance plane of the two-plane source image.
**Description**

This function operates with ROI (see Regions of Interest in Intel IPP).

This function converts the Y’Cb’Cr’ image `pSrc` to the gamma-corrected R'G'B' or B'G'R' image `pDst` according to the same formulas as the function `ippiYCbCrToRGB` does. The difference is that the `ippiYCbCr420ToRGB` and `ippiYCbCr420ToBGR` functions use the input data in the 4:2:0 sampling format, in which the number of Cb and Cr samples is reduced by half in both vertical and horizontal directions (see Table "Planar Image Formats" for more details). Two-plane Y’Cb’Cr’ image with 4:2:0 sampling is also known as NV12 format.

The value of `roiSize.width` and `roiSize.height` must be a multiple of 2. Otherwise, the function reduces original values to the nearest multiples of 2, performs operation, and returns a warning message.

**Return Values**

- `ippStsNoErr`: Indicates no error. Any other value indicates an error or a warning.
- `ippStsNullPtrErr`: Indicates an error condition if `pSrc` or `pDst` is NULL.
- `ippStsSizeErr`: Indicates an error condition if `roiSize` has a field with value less than 2.
- `ippStsDoubleSize`: Indicates a warning if `roiSize` has a field that is not a multiple of 2.

**Example**

The code example below demonstrates how to use the `ippiYCbCr420ToRGB_8u_P2C4R` function.

```c
#define WIDTH 4
#define HEIGHT 4

Ipp8u pSrcY[WIDTH * HEIGHT] =
{
    236,236,236,236,
    236,236,236,236,
    236,236,236,236,
    236,236,236,236
};
Ipp8u pSrcCbCr[WIDTH * HEIGHT / 2] =
{
    128,128,128,128,
    128,128,128,128
};
Ipp8u pDstRGB[(WIDTH * HEIGHT) * 4];
int srcYStep = WIDTH, srcCbCrStep = WIDTH, dstStep = WIDTH * 4;
IppiSize roiSize = {WIDTH, HEIGHT};
Ipp8u alphaValue = 0xFF;
```
IppStatus status = ippiYCbCr420ToRGB_8u_P2C4R(pSrcY, srcYStep, pSrcCbCr, srcCbCrStep,
pDstRGB, dstStep, roiSize, alphaValue);
if ( status == ippStsNoErr)
    printf("\n *************  passed *************\n");
else
    printf("\n *************  failed *************\n");

RGBToYCrCb420
Converts an RGB image to the YCrCb image with
4:2:0 sampling format.

Syntax
IppStatus ippiRGBToYCrCb420_8u_AC4P3R(const Ipp8u* pSrc, int srcStep, Ipp8u* pDst[3],
int dstStep[3], IppiSize roiSize);

Include Files
ippcc.h

Domain Dependencies
Headers: ippcore.h, ippvm.h, ipps.h,ippi.h
Libraries: ippcore.lib, ippvm.lib, ipps.lib,ippi.lib

Parameters

pSrc
Pointer to the source image ROI.

srcStep
Distance in bytes between starts of consecutive lines in the source image.

pDst
Array of pointers to ROI in separate planes of the destination image.

dstStep
Array of distances in bytes between starts of consecutive lines in the destination image planes.

roiSize
Size of the source and destination ROI in pixels.

Description
This function operates with ROI (see Regions of Interest in Intel IPP).
This function converts a four-channel gamma-corrected R'G'B' image pSrc to the planar Y'Cr'Cb' image pDst with the 4:2:0 sampling (see Table “Planar Image Formats” for more details). The conversion is performed according to the same formulas as the function ippiRGBToYCbCr does.
roiSize.width and roiSize.height should be multiples of 2. If not the function reduces their original values to the nearest multiples of 2, performs the operation, and returns a warning message.

Return Values

ippStsNoErr
Indicates no error. Any other value indicates an error or a warning.

ippStsNullPtrErr
Indicates an error condition if pSrc or pDst is NULL.

ippStsSizeErr
Indicates an error condition if roiSize has a field with a zero or negative value.
Indicates a warning if roiSize has a field that is not a multiple of 2.

YCrCb420ToRGB, YCrCb420ToBGR
Convert a YCrCb image with the 4:2:0 sampling to the RGB or BGR image.

Syntax

IppStatus ippiYCrCb420ToRGB_8u_P3C4R (const Ipp8u* pSrc[3], int srcStep[3], Ipp8u* pDst, int dstStep, IppiSize roiSize, Ipp8u aval);
IppStatus ippiYCrCb420ToRGB_8u_P2C4R (const Ipp8u* pSrcY, int srcYStep, const Ipp8u* pSrcCrCb, int srcCrCbStep, Ipp8u* pDst, int dstStep, IppiSize roiSize, Ipp8u aval);
IppStatus ippiYCrCb420ToRGB_8u_P2C3R (const Ipp8u* pSrcY, int srcYStep, const Ipp8u* pSrcCrCb, int srcCrCbStep, Ipp8u* pDst, int dstStep, IppiSize roiSize);
IppStatus ippiYCrCb420ToBGR_8u_P2C3R (const Ipp8u* pSrcY, int srcYStep, const Ipp8u* pSrcCrCb, int srcCrCbStep, Ipp8u* pDst, int dstStep, IppiSize roiSize, Ipp8u aval);
IppStatus ippiYCrCb420ToBGR_8u_P2C4R (const Ipp8u* pSrcY, int srcYStep, const Ipp8u* pSrcCrCb, int srcCrCbStep, Ipp8u* pDst, int dstStep, IppiSize roiSize, Ipp8u aval);
IppStatus ippiYCrCb420ToBGR_Filter_8u_P3C4R (const Ipp8u* pSrc[3], int srcStep[3], Ipp8u* pDst, int dstStep, IppiSize roiSize, Ipp8u aval);

Include Files

ippcc.h

Domain Dependencies

Headers: ippcore.h, ippvm.h, ipps.h,ippi.h
Libraries: ippcore.lib, ippvm.lib, ipps.lib,ippi.lib

Parameters

pSrc
An array of pointers to ROI in separate planes of the three-plane source image.
srcStep
An array of distances, in bytes, between the starting points of consecutive lines in each plane of the three-plane source image.
pSrcY
Pointer to ROI in the luminance plane of the two-plane source image.
srcYStep
Distance, in bytes, between the starting points of consecutive lines in the luminance plane of the two-plane source image.
pSrcCrCb
Pointer to ROI in the interleaved plane chrominance plane of the two-plane source image.
srcCrCbStep
Distance, in bytes, between the starting points of consecutive lines in the interleaved chrominance plane of the two-plane source image.
pDst
Pointer to the destination image ROI.
dstStep
Distance, in bytes, between the starting points of consecutive lines in the destination image.
roiSize

Size of the source and destination ROI, in pixels.

aval

Constant value to create the fourth channel.

Description

This function operates with ROI.

This function converts the Y'Cr'Cb' image pSrc to the gamma-corrected R'G'B' or B'G'R' image pDst according to the same formulas as the ippiippiYCbCrToRGB function does. The difference is that the ippiYCrCb420ToRGB and ippiYCrCb420ToBGR functions use the source data in the 4:2:0 sampling format, in which the number of Cb and Cr samples is reduced by half in both vertical and horizontal directions (see Table "Planar Image Formats" for more details). Two-plane Y'Cr'Cb image with 4:2:0 sampling is also known as NV21 format.

The value of roiSize.width and roiSize.height must be a multiple of 2. Otherwise, the function reduces original values of roiSize.width and roiSize.height to the nearest multiples of 2, performs operation, and returns a warning.

Return Values

ippStsNoErr

Indicates no error. Any other value indicates an error or a warning.

ippStsNullPtrErr

Indicates an error when pSrc or pDst is NULL.

ippStsSizeErr

Indicates an error when roiSize has a field with a value less than 2.

ippStsDoubleSize

Indicates a warning if roiSize has a field that is not a multiple of 2.

Example

The code example below demonstrates how to use the ippiYCrCb420ToRGB_8u_P2C4R function.

```c
static void sampleNV21ToRGBA()
{
    Ipp8u pY[4*4]=
    {
        236,236,236,236,
        236,236,236,236,
        236,236,236,236,
        236,236,236,236
    };
    Ipp8u pCbCr[4*2]=
    {
        128,128,128,128,
        128,128,128,128
    };
    Ipp8u pRGB[(4*4)*4];
    int YStep = 4, CbCrStep = 4, rgbStep = 4*4;
    IppSize roiSize = {4,4};
    Ipp8u alpha = 0xFF;
    IppStatus status = ippiYCrCb420ToRGB_8u_P2C4R(pY, YStep, pCbCr, CbCrStep, pRGB, rgbStep, roiSize, alpha);
    if ( status == ippStsNoErr)
        printf("\n *************  passed *************\n");
    else
        printf("\n *************  failed *************\t\n");
}
```
See Also
Regions of Interest in Intel IPP

**BGRToYCbCr420**
Converts a BGR image to the YCbCr image with 4:2:0 sampling format.

**Syntax**

IppStatus ippiBGRToYCbCr420_8u_C3P3R(const Ipp8u* pSrc, int srcStep, Ipp8u* pDst[3], int dstStep[3], IppiSize roiSize);

IppStatus ippiBGRToYCbCr420_8u_C3P2R(const Ipp8u* pRGB, int rgbStep, Ipp8u* pY, int yStep, Ipp8u* pCbCr, int cbCrStep, IppiSize roiSize);

IppStatus ippiBGRToYCbCr420_8u_AC4P3R(const Ipp8u* pSrc, int srcStep, Ipp8u* pDst[3], int dstStep[3], IppiSize roiSize);

IppStatus ippiBGRToYCbCr420_8u_AC4P2R(const Ipp8u* pRGB, int rgbStep, Ipp8u* pY, int yStep, Ipp8u* pCbCr, int cbCrStep, IppiSize roiSize);

**Include Files**
ippcc.h

**Domain Dependencies**
Headers: ippcore.h, ippvm.h, ipps.h,ippi.h
Libraries: ippcore.lib, ippvm.lib, ipps.lib,ippi.lib

**Parameters**

- **pSrc, pRGB**
  Pointer to the source image ROI.

- **pY**
  Pointer to the image Y plane.

- **srcStep, rgbStep**
  Distance, in bytes, between the starting points of consecutive lines in the source image.

- **yStep**
  Distance, in bytes, between the starting points of consecutive lines in the image Y plane.

- **pDst, pCbCr**
  An array of pointers to ROI in separate planes of the destination image.

- **dstStep, cbCrStep**
  An array of distances, in bytes, between the starting points of consecutive lines in the destination image planes.

- **roiSize**
  Size of the source and destination ROI in pixels.

**Description**
This function operates with ROI (see Regions of Interest in Intel IPP).

This function converts a three- or four-channel gamma-corrected B'G'R' image to the planar Y'Cb'Cr' image according to the same formulas as the function ippiRGBToYCbCr does. The difference is that ippiBGRToYCbCr420 uses 4:2:0 sampling format (see Table "Planar Image Formats" for more details). roiSize.width and roiSize.height should be multiples of 2. If not the function reduces their original values to the nearest multiples of 2, performs operation, and returns warning message.
Return Values

ippStsNoErr Indicates no error. Any other value indicates an error or a warning.

ippStsNullPtrErr Indicates an error condition if pSrc (pRGB) or pDst (pCbCr) is NULL.

ippStsSizeErr Indicates an error condition if roiSize.width is less than 2 or roiSize.height is less than 2.

ippStsDoubleSize Indicates a warning if roiSize has a field that is not a multiple of 2.

BGRToYCbCr420_709CSC
Converts a BGR image to the YCbCr image with 4:2:0 sampling for ITU-R BT.709 CSC signal.

Syntax

IppStatus ippiBGRToYCbCr420_709CSC_8u_C3P3R(const Ipp8u* pSrc, int srcStep, Ipp8u* pDst[3], int dstStep[3], IppiSize roiSize);
IppStatus ippiBGRToYCbCr420_709CSC_8u_C3P2R(const Ipp8u* pSrc, int srcStep, Ipp8u* pDstY, int dstYStep, Ipp8u* pDstCbCr, int dstCbCrStep, IppiSize roiSize);
IppStatus ippiBGRToYCbCr420_709CSC_8u_AC4P3R(const Ipp8u* pSrc, int srcStep, Ipp8u* pDst[3], int dstStep[3], IppiSize roiSize);
IppStatus ippiBGRToYCbCr420_709CSC_8u_AC4P2R(const Ipp8u* pSrc, int srcStep, Ipp8u* pDstY, int dstYStep, Ipp8u* pDstCbCr, int dstCbCrStep, IppiSize roiSize);

Include Files

ippcc.h

Domain Dependencies

Headers: ippcore.h, ippvm.h, ipps.h, ippi.h
Libraries: ippcore.lib, ippvm.lib, ipps.lib, ippi.lib

Parameters

pSrc Pointer to the source image ROI.
srcStep Distance in bytes between starts of consecutive lines in the source image.

pDst, pDstY, pDstCbCr An array of pointers to ROI in separate planes of the destination image.
dstStep, dstYStep, dstCbCrStep An array of distances in bytes between starts of consecutive lines in the destination image planes.
roiSize Size of the source and destination ROI in pixels.

Description

This function operates with ROI (see Regions of Interest in Intel IPP).
This function converts a three- or four-channel gamma-corrected B’G’R’ image \( p_{Src} \) to the planar \( Y'Cb'Cr' \) image \( p_{Dst} \) for digital component video signals complied with the ITU-R BT.709 Recommendation [ITU709] for computer systems consideration (CSC). The source image pixel values are in the range \([0..255]\). The conversion is performed according to the following formulas [Jack01]:

\[
Y' = 0.183*R' + 0.614*G' + 0.062*B' + 16
\]

\[
Cb' = -0.101*R' - 0.338*G' + 0.439*B' + 128
\]

\[
Cr' = 0.439*R' - 0.399*G' - 0.040*B' + 128
\]

The destination image \( p_{Dst} \) has 4:2:0 sampling format (see Table “Planar Image Formats” for more details).

The values of \( roiSize.width \) and \( roiSize.height \) should be multiples of 2. Otherwise, the function reduces their original values to the nearest multiples of 2, performs operation, and returns a warning message.

Return Values

- ippStsNoErr: Indicates no error. Any other value indicates an error or a warning.
- ippStsNullPtrErr: Indicates an error condition if \( p_{Src} \) or \( p_{Dst} \) is NULL.
- ippStsSizeErr: Indicates an error condition if \( roiSize.width \) is less than 2 or \( roiSize.height \) is less than 2.
- ippStsDoubleSize: Indicates a warning if \( roiSize \) has a field that is not a multiple of 2.

**BGRToYCbCr420_709HDTV**

Converts a BGR image to the YCbCr image with 4:2:0 sampling for ITU-R BT.709 HDTV signal.

**Syntax**

```c
IppStatus ippiBGRToYCbCr420_709HDTV_8u_AC4P3R(const Ipp8u* pSrc, int srcStep, Ipp8u* pDst[3], int dstStep[3], IppiSize roiSize);
```

**Include Files**

ippcc.h

**Domain Dependencies**

Headers: ippcore.h, ippvm.h, ipps.h,ippi.h

Libraries: ippcore.lib, ippvm.lib, ipps.lib,ippi.lib

**Parameters**

- \( p_{Src} \): Pointer to the source image ROI.
- \( srcStep \): Distance in bytes between starts of consecutive lines in the source image.
- \( p_{Dst} \): An array of pointers to ROI in separate planes of the destination image.
- \( dstStep \): An array of distances in bytes between starts of consecutive lines in the destination image planes.
**roiSize**
Size of the source and destination ROI in pixels.

**Description**
This function operates with ROI (see Regions of Interest in Intel IPP).

This function converts a four-channel gamma-corrected B'G'R' image `pSrc` to the planar Y'Cb'Cr' image `pDst` for digital component video signals complied with the ITU-R BT.709 Recommendation [ITU709] for high-definition TV (HDTV). The source image pixel values are in the range [16..235]. The conversion is performed according to the following formulas [Jack01]:

\[
Y' = 0.213 \cdot R' + 0.715 \cdot G' + 0.072 \cdot B'
\]

\[
Cb' = -0.117 \cdot R' - 0.394 \cdot G' + 0.511 \cdot B' + 128
\]

\[
Cr' = 0.511 \cdot R' - 0.464 \cdot G' - 0.047 \cdot B' + 128
\]

The values of \(Y'\) of the destination image are in the range [16..235], the values of \(Cb', Cr'\) are in the range [16..240]. They should be saturated at the 1 and 254 levels.

The destination image `pDst` has the 4:2:0 sampling format (see Table “Planar Image Formats” for more details).

The values of `roiSize.width` and `roiSize.height` should be multiples of 2. Otherwise, the function reduces their original values to the nearest multiples of 2, performs the operation, and returns a warning message.

**Return Values**
- `ippStsNoErr` Indicates no error. Any other value indicates an error or a warning.
- `ippStsNullPtrErr` Indicates an error condition if `pSrc` or `pDst` is NULL.
- `ippStsSizeErr` Indicates an error condition if `roiSize.width` is less than 2 or `roiSize.height` is less than 2.
- `ippStsDoubleSize` Indicates a warning if `roiSize` has a field that is not a multiple of 2.

**BGRToYCrCb420_709CSC**
Converts a BGR image to the YCrCb image with 4:2:0 sampling for ITU-R BT.709 CSC signal.

**Syntax**

```c
IppStatus ippiBGRToYCrCb420_709CSC_8u_AC4P3R(const Ipp8u* pSrc, int srcStep, Ipp8u* pDst[3], int dstStep[3], IppiSize roiSize);
IppStatus ippiBGRToYCrCb420_709CSC_8u_C3P3R(const Ipp8u* pSrc, int srcStep, Ipp8u* pDst[3], int dstStep[3], IppiSize roiSize);
```

**Include Files**

ippcc.h

**Domain Dependencies**

**Headers:** ippcore.h, ippvm.h, ipps.h,ippi.h

**Libraries:** ippcore.lib, ippvm.lib, ipps.lib,ippi.lib
Parameters

pSrc  
Pointer to the source image ROI.

srcStep  
Distance in bytes between starts of consecutive lines in the source image.

pDst  
An array of pointers to ROI in separate planes of the destination image.

dstStep  
An array of distances in bytes between starts of consecutive lines in the destination image planes.

roiSize  
Size of the source and destination ROI in pixels.

Description

This function operates with ROI (see Regions of Interest in Intel IPP).

This function converts a three- or four-channel gamma-corrected B'G'R' image pSrc to the planar Y'Cr'Cb' image pDst for digital component video signals complied with the ITU-R BT.709 Recommendation [ITU709] for computer systems consideration (CSC). The source image pixel values are in the range [0..255]. The conversion is performed according to the following formulas [Jack01]:

\[
Y' = 0.183*R' + 0.614*G' + 0.062*B' + 16 \\
Cb' = -0.101*R' - 0.338*G' + 0.439*B' + 128 \\
Cr' = 0.439*R' - 0.399*G' - 0.040*B' + 128
\]

The destination image pDst has 4:2:0 sampling format (see Table “Planar Image Formats” for more details).

The values of roiSize.width and roiSize.height should be multiples of 2. Otherwise, the function reduces their original values to the nearest multiples of 2, performs the operation, and returns a warning message.

Return Values

ippStsNoErr  
Indicates no error. Any other value indicates an error or a warning.

ippStsNullPtrErr  
Indicates an error condition if pSrc or pDst is NULL.

ippStsSizeErr  
Indicates an error condition if roiSize.width is less than 2 or roiSize.height is less than 2.

ippStsDoubleSize  
Indicates a warning if roiSize has a field that is not a multiple of 2.

YCbCr420ToBGR

Converts a YCbCr image with the 4:2:0 sampling to the BGR image.

Syntax

IppStatus ippiYCbCr420ToBGR_8u_P3C3R(const Ipp8u* pSrc[3], int srcStep[3], Ipp8u* pDst, int dstStep, IppiSize roiSize);

IppStatus ippiYCbCr420ToBGR_8u_P3C4R(const Ipp8u* pSrc[3], int srcStep[3],Ipp8u* pDst, int dstStep, IppiSize roiSize, Ipp8u aval);
Include Files
ippcc.h

Domain Dependencies
Headers: ippcore.h, ippvm.h, ipps.h,ippi.h
Libraries: ippcore.lib, ippvm.lib, ipps.lib,ippi.lib

Parameters

- **pSrc**: An array of pointers to ROI in separate planes of the source image.
- **srcStep**: An array of distances in bytes between starts of consecutive lines in the source image planes.
- **pDst**: Pointer to the destination image ROI.
- **dstStep**: Distance in bytes between starts of consecutive lines in the destination image.
- **roiSize**: Size of the source and destination ROI in pixels.
- **aval**: Constant value to create fourth channel.

Description

This function operates with ROI (see Regions of Interest in Intel IPP).

This function converts the Y'Cb'Cr' image **pSrc** with the 4:2:0 sampling (see Table "Planar Image Formats" for more details) to the gamma-corrected three- or four-channel B'G'R' image **pDst**. The conversion is performed according to the same formulas as the function **ippiYCbCrToRGB** does.

Fourth channel is created by setting channel values to the constant value **aval**.

The values of **roiSize.width** and **roiSize.height** should be multiples of 2. If not the function reduces their original values to the nearest multiples of 2, performs operation, and returns warning message.

Return Values

- **ippStsNoErr**: Indicates no error. Any other value indicates an error or a warning.
- **ippStsNullPtrErr**: Indicates an error condition if **pSrc** or **pDst** is NULL.
- **ippStsSizeErr**: Indicates an error condition if **roiSize** has a field with a zero or negative value.
- **ippStsDoubleSize**: Indicates a warning if **roiSize** has a field that is not a multiple of 2.

**YCbCr420ToBGR_709CSC**

Converts a YCbCr image with 4:2:0 sampling to the BGR image for ITU-R BT.709 CSC signal.

Syntax

```c
IppStatus ippiYCbCr420ToBGR_709CSC_8u_P3C3R(const Ipp8u* pSrc[3], int srcStep[3], Ipp8u* pDst, int dstStep, IppiSize roiSize);
```

Include Files
ippcc.h
Domain Dependencies

Headers: ippcore.h, ippvm.h, ipps.h,ippi.h
Libraries: ippcore.lib, ippvm.lib, ipps.lib,ippi.lib

Parameters

pSrc  An array of pointers to ROI in separate planes of the source image.
srcStep  An array of distances in bytes between starts of consecutive lines in the source image planes.
pDst  Pointer to the destination image ROI.
dstStep  Distance in bytes between starts of consecutive lines in the destination image.
roiSize  Size of the source and destination ROI in pixels.

Description

This function operates with ROI (see Regions of Interest in Intel IPP).

This function converts a planar Y'Cb'Cr' image pSrc to the three-channel gamma-corrected B'G'R' image pDst for digital component video signals complied with the ITU-R BT.709 Recommendation [ITU709] for computer systems consideration (CSC). The conversion is performed according to the following formulas [Jack01]:

\[
R' = 1.164*(Y' - 16) + 1.793*(Cr' - 128) \\
G' = 1.164*(Y' - 16) - 0.534*(Cr' - 128) - 0.213*(Cb' - 128) \\
B' = 1.164*(Y' - 16) + 2.115*(Cb' - 128)
\]

The output R'G'B' values are saturated to the range [0..255]. The source image pDst has the 4:2:0 sampling format (see Table “Planar Image Formats” for more details).

The values of roiSize.width and roiSize.height should be multiples of 2. Otherwise, the function reduces their original values to the nearest multiples of 2, performs the operation, and returns a warning message.

Return Values

ippStsNoErr  Indicates no error. Any other value indicates an error or a warning.
ippStsNullPtrErr  Indicates an error condition if pSrc or pDst is NULL.
ippStsSizeErr  Indicates an error condition if roiSize has a field with a zero or negative value.
ippStsDoubleSize  Indicates a warning if roiSize has a field that is not a multiple of 2.

YCbCr420ToBGR_709HDTV

Converts a YCbCr image with 4:2:0 sampling to the BGR image for ITU-R BT.709 HDTV signal.

Syntax

IppStatus ippiYCbCr420ToBGR_709HDTV_8u_P3C4R(const Ipp8u* pSrc[3], int srcStep[3], Ipp8u* pDst, int dstStep, IppiSize roiSize, Ipp8u aval);
Include Files
ippcc.h

Domain Dependencies
Headers: ippcore.h, ippvm.h, ipps.h,ippi.h
Libraries: ippcore.lib, ippvm.lib, ipps.lib,ippi.lib

Parameters

pSrc
An array of pointers to ROI in separate planes of the source image.

srcStep
An array of distances in bytes between starts of consecutive lines in the source image planes.

pDst
Pointer to the destination image ROI.

dstStep
Distance in bytes between starts of consecutive lines in the destination image.

roiSize
Size of the source and destination ROI in pixels.

aval
Constant value to create fourth channel.

Description
This function operates with ROI (see Regions of Interest in Intel IPP).

This function converts a planar Y'Cb'Cr' image pSrc to the four-channel gamma-corrected B'G'R' image pDst for digital component video signals complied with the ITU-R BT.709 Recommendation [ITU709] for high-definition TV (HDTV). The values of Y' are in the range [16..235], the values of Cb', Cr' are in the range [16..240]. The conversion is performed according to the following formulas [Jack01]:

R' = Y' + 1.540*(Cr' - 128)
G' = Y' - 0.459*(Cr' - 128) - 0.183*(Cb' - 128)
B' = Y' + 1.816*(Cb' - 128)

The destination image pixel values have a nominal range [16..235]. The resulting R'G'B' values should be saturated at the 0 and 255 levels.

The source image pDst has the 4:2:0 sampling format (see Table "Planar Image Formats" for more details).

The values of roiSize.width and roiSize.height should be multiples of 2. Otherwise the function reduces their original values to the nearest multiples of 2, performs operation, and returns a warning message.

Return Values

ippStsNoErr
Indicates no error. Any other value indicates an error or a warning.

ippStsNullPtrErr
Indicates an error condition if pSrc or pDst is NULL.

ippStsSizeErr
Indicates an error condition if roiSize has a field with a zero or negative value.

ippStsDoubleSize
Indicates a warning if roiSize has a field that is not a multiple of 2.
**BGRToYCrCb420**  
*Converts a BGR image to the YCrCb image with 4:2:0 sampling format.*

**Syntax**

```c
IppStatus ippiBGRToYCrCb420_8u_C3P3R(const Ipp8u* pSrc, int srcStep, Ipp8u* pDst[3], int dstStep[3], IppiSize roiSize);
IppStatus ippiBGRToYCrCb420_8u_AC4P3R(const Ipp8u* pSrc, int srcStep, Ipp8u* pDst[3], int dstStep[3], IppiSize roiSize);
```

**Include Files**

`ippcc.h`

**Domain Dependencies**

**Headers:** `ippcore.h`, `ippvm.h`, `ipps.h`, `ippi.h`

**Libraries:** `ippcore.lib`, `ippvm.lib`, `ipps.lib`, `ippi.lib`

**Parameters**

- `pSrc`: Pointer to the source image ROI.
- `srcStep`: Distance in bytes between starts of consecutive lines in the source image.
- `pDst`: An array of pointers to ROI in separate planes of the destination image.
- `dstStep`: An array of distances in bytes between starts of consecutive lines in the destination image planes.
- `roiSize`: Size of the source and destination ROI in pixels.

**Description**

This function operates with ROI (see Regions of Interest in Intel IPP).

This function converts a three- or four-channel gamma-corrected B'G'R' image `pSrc` to the planar Y'Cr'Cb' image `pDst` according to the same formulas as the function `ippiRGBToYCbCr` does. The destination image `pDst` has the 4:2:0 sampling format and the following order of pointers: Y-plane, Cr-plane, Cb-plane (see Table "Planar Image Formats" for more details).

**Return Values**

- `ippStsNoErr`: Indicates no error. Any other value indicates an error or a warning.
- `ippStsNullPtrErr`: Indicates an error condition if `pSrc` or `pDst` is NULL.
- `ippStsSizeErr`: Indicates an error condition if `roiSize.width` is less than 2 or `roiSize.height` is less than 2.

**BGRToYCbCr411**  
*Converts a BGR image to the YCbCr planar image that has a 4:1:1 sampling format.*
### Syntax

IppStatus ippiBGRToYCbCr411_8u_C3P3R(const Ipp8u* pSrc, int srcStep, Ipp8u* pDst[3], int dstStep[3], IppSize roiSize);

IppStatus ippiBGRToYCbCr411_8u_AC4P3R(const Ipp8u* pSrc, int srcStep, Ipp8u* pDst[3], int dstStep[3], IppSize roiSize);

### Include Files

ippcc.h

### Domain Dependencies

**Headers:** ippcore.h, ippvm.h, ipps.h, ippi.h

**Libraries:** ippcore.lib, ippvm.lib, ipps.lib, ippi.lib

### Parameters

- **pSrc**: Pointer to the source image ROI.
- **srcStep**: Distance in bytes between starts of consecutive lines in the source image.
- **pDst**: An array of pointers to ROI in separate planes of the destination image.
- **dstStep**: An array of distances in bytes between starts of consecutive lines in the destination image planes.
- **roiSize**: Size of the source and destination ROI in pixels.

### Description

This function operates with ROI (see Regions of Interest in Intel IPP).

This function converts a three- or four-channel gamma-corrected B'G'R' image `pSrc` to the planar Y'Cb'Cr' image `pDst` according to the same formulas as the function `ippiRGBToYCbCr` does. The difference is that `ippiBGRToYCbCr411` uses the 4:1:1 sampling format (see Table "Planar Image Formats" for more details).

### Return Values

- **ippStsNoErr**: Indicates no error. Any other value indicates an error or a warning.
- **ippStsNullPtrErr**: Indicates an error condition if `pSrc` or `pDst` is NULL.
- **ippStsSizeErr**: Indicates an error condition if `roiSize.width` is less than 4 or `roiSize.height` is less than 1.

### YCbCr411ToBGR

**Converts a YCbCr image that has 4:1:1 sampling format to the RGB color model.**

### Syntax

IppStatus ippiYCbCr411ToBGR_8u_C3P3R(const Ipp8u* pSrc[3], int srcStep[3], Ipp8u* pDst, int dstStep, IppSize roiSize);

IppStatus ippiYCbCr411ToBGR_8u_AC4P3R(const Ipp8u* pSrc[3], int srcStep[3], Ipp8u* pDst, int dstStep, IppSize roiSize, Ipp8u aval);
Include Files
ippcc.h

Domain Dependencies
Headers: ippcore.h, ippvm.h, ipps.h,ippi.h
Libraries: ippcore.lib, ippvm.lib, ipps.lib,ippi.lib

Parameters

pSrc
An array of pointers to ROI in separate planes of the source image.

srcStep
An array of distances in bytes between starts of consecutive lines in the source image planes.

pDst
Pointer to the destination image ROI.

dstStep
Distance in bytes between starts of consecutive lines in the destination image.

roiSize
Size of the source and destination ROI in pixels.

aval
Constant value to create fourth channel.

Description
This function operates with ROI (see Regions of Interest in Intel IPP).

This function converts the planar Y'Cb'Cr' image pSrc to the three- or four-channel image pDst. To compute gamma-corrected R'G'B' (B'G'R') channel values the above formulas are used. The difference is thatippiYCbCr411ToBGR uses the input data in the 4:1:1 sampling format (see Table "Planar Image Formats" for more details). Fourth channel is created by setting channel values to the constant value aval.

Return Values

ippStsNoErr
Indicates no error. Any other value indicates an error or a warning.

ippStsNullPtrErr
Indicates an error condition if pSrc or pDst is NULL.

ippStsSizeErr
Indicates an error condition if roiSize has a field with a zero or negative value.

RGBToXYZ
Converting an RGB image to the XYZ color model.

Syntax
IppStatus ippiRGBToXYZ_<mod>(const Ipp<datatype>* pSrc, int srcStep, Ipp<datatype>* pDst, int dstStep, IppiSize roiSize);

Supported values for mod:

<table>
<thead>
<tr>
<th>8u_C3R</th>
<th>16u_C3R</th>
<th>16s_C3R</th>
<th>32f_C3R</th>
</tr>
</thead>
<tbody>
<tr>
<td>8u_AC4R</td>
<td>16u_AC4R</td>
<td>16s_AC4R</td>
<td>32f_AC4R</td>
</tr>
</tbody>
</table>

Include Files
ippcc.h
Domain Dependencies

Headers: ippcore.h, ippvm.h, ipps.h,ippi.h
Libraries: ippcore.lib, ippvm.lib, ipps.lib,ippi.lib

Parameters

- **pSrc**: Pointer to the source image ROI.
- **srcStep**: Distance in bytes between starts of consecutive lines in the source image.
- **pDst**: Pointer to the destination image ROI.
- **dstStep**: Distance in bytes between starts of consecutive lines in the destination image.
- **roiSize**: Size of the source and destination ROI in pixels.

Description

This function operates with ROI (see Regions of Interest in Intel IPP).

This function converts the RGB image pSrc to the CIEXYZ image pDst according to the following basic equations:

\[
\begin{align*}
X &= 0.412453 \times R + 0.35758 \times G + 0.180423 \times B \\
Y &= 0.212671 \times R + 0.71516 \times G + 0.072169 \times B \\
Z &= 0.019334 \times R + 0.119193 \times G + 0.950227 \times B
\end{align*}
\]

The equations above are given on the assumption that R, G, and B values are normalized to the range [0..1]. In case of the floating-point data type, the input RGB values must already be in the range [0..1]. For integer data types, normalization is done by the conversion function internally.

The computed XYZ values are saturated if they fall out of range [0..1].

In case of integer function flavors, these values are then scaled to the full range of the destination data type (see Table "Image Data Types and Ranges" in Chapter 2).

Return Values

- **ippStsNoErr**: Indicates no error. Any other value indicates an error.
- **ippStsNullPtrErr**: Indicates an error condition if pSrc or pDst is NULL.
- **ippStsSizeErr**: Indicates an error condition if roiSize has a field with a zero or negative value.

XYZToRGB

*Converts an XYZ image to the RGB color model.*

Syntax

```
IppStatus ippiXYZToRGB_<mod>(const Ipp<datatype>* pSrc, int srcStep, Ipp<datatype>* pDst, int dstStep, IppiSize roiSize);
```

Supported values for mod:

- 8u_C3R, 16u_C3R, 16s_C3R, 32f_C3R
- 8u_AC4R, 16u_AC4R, 16s_AC4R, 32f_AC4R
Include Files
ippcc.h

Domain Dependencies
Headers: ippcore.h, ippvm.h, ipps.h,ippi.h
Libraries: ippcore.lib, ippvm.lib, ipps.lib,ippi.lib

Parameters
pSrc
Pointer to the source image ROI.
srcStep
Distance in bytes between starts of consecutive lines in the source image.
pDst
Pointer to the destination image ROI.
dstStep
Distance in bytes between starts of consecutive lines in the destination image.
roiSize
Size of the source and destination ROI in pixels.

Description
This function operates with ROI (see Regions of Interest in Intel IPP).
This function converts the CIEXYZ image pSrc to the RGB image pDst according to the following basic equations:
R = 3.240479 * X - 1.53715 * Y - 0.498535 * Z
G = -0.969256 * X + 1.875991 * Y + 0.041556 * Z
B = 0.055648 * X - 0.204043 * Y + 1.057311 * Z
The equations above are given on the assumption that X, Y, and Z values are in the range [0..1]. In case of the floating-point data type, the input XYZ values must already be in the range [0..1]. For integer data types, normalization is done by the conversion function internally.
The computed RGB values are saturated if they fall out of range [0..1].
In case of integer function flavors, these values are then scaled to the full range of the destination data type (see Table "Image Data Types and Ranges" in Chapter 2).

Return Values
ippStsNoErr
Indicates no error. Any other value indicates an error.
ippStsNullPtrErr
Indicates an error condition if pSrc or pDst is NULL.
ippStsSizeErr
Indicates an error condition if roiSize has a field with a zero or negative value.

RGBToLUV, BGRToLUV
Converts an RGB or BGR image to the LUV color model.
Syntax

Case 1: RGB to LUV

IppStatus ippiRGBToLUV_<mod>(const Ipp<datatype>* pSrc, int srcStep, Ipp<datatype>* pDst, int dstStep, IppiSize roiSize);

Supported values for mod:
- 8u_C3R
- 16u_C3R
- 16s_C3R
- 32f_C3R
- 8u_AC4R
- 16u_AC4R
- 16s_AC4R
- 32f_AC4R

Case 2: BGR to LUV

IppStatus ippiBGRToLUV_<mod>(const Ipp<datatype>* pSrc, int srcStep, Ipp<datatype>* pDst, int dstStep, IppiSize roiSize);

Supported values for mod:
- 8u_C3R
- 32f_C3R

Include Files

ippcc.h

Domain Dependencies

Headers: ippcore.h, ippvm.h, ipps.h,ippi.h
Libraries: ippcore.lib, ippvm.lib, ipps.lib,ippi.lib

Parameters

pSrc
- Pointer to the source image ROI.
srcStep
- Distance in bytes between starts of consecutive lines in the source image.
pDst
- Pointer to the destination image ROI.
dstStep
- Distance in bytes between starts of consecutive lines in the destination image.
roiSize
- Size of the source and destination ROI in pixels.

Description

This function operates with ROI (see Regions of Interest in Intel IPP).

This function converts the RGB or BGR image pSrc to the CIE LUV image pDst in two steps. First, the conversion is done into CIE XYZ format, using equations defined for the ippiRGBToXYZ function. After that, conversion to LUV image is performed in accordance with the following equations:

L = 116. * \( \frac{Y}{Y_n} \)^{1/3} - 16.
U = 13. * L * (u - u_n)
V = 13. * L * (v - v_n)

where
u = 4.*X / (X + 15.*Y + 3.*Z)
v = 9.*Y / (X + 15.*Y + 3.*Z)
u_n = 0.197839
\[ v_n = 0.468342 \]

The computed values of the \( L \) component are in the range \([0..100]\), \( U \) component in the range \([-134..220]\), and \( V \) component in the range \([-140..122]\).

The equations above are given on the assumption that \( R \), \( G \), and \( B \) values are normalized to the range \([0..1]\). In case of the floating-point data type, the input \( RGB \) values must already be in the range \([0..1]\). For integer data types, normalization is done by the conversion function internally.

In case of 8u data type, the computed \( L \), \( U \), and \( V \) values are quantized and converted to fit in the range \([0..IPP_MAX_8U]\) as follows:

\[ L = L \times \text{IPP_MAX}_8U / 100. \]
\[ U = (U + 134.) \times \text{IPP_MAX}_8U / 354. \]
\[ V = (V + 140.) \times \text{IPP_MAX}_8U / 262. \]

In case of 16u data type, the computed \( L \), \( U \), and \( V \) values are quantized and converted to fit in the range \([0..IPP_MAX_16U]\) as follows:

\[ L = L \times \text{IPP_MAX}_16U / 100. \]
\[ U = (U + 134.) \times \text{IPP_MAX}_16U / 354. \]
\[ V = (V + 140.) \times \text{IPP_MAX}_16U / 262. \]

In case of 16s data type, the computed \( L \), \( U \), and \( V \) values are quantized and converted to fit in the range \([\text{IPP_MIN}_16S..\text{IPP_MAX}_16S]\) as follows:

\[ L = L \times \text{IPP_MAX}_16U / 100. + \text{IPP_MIN}_16S \]
\[ U = (U + 134.) \times \text{IPP_MAX}_16U / 354. + \text{IPP_MIN}_16S \]
\[ V = (V + 140.) \times \text{IPP_MAX}_16U / 262. + \text{IPP_MIN}_16S \]

For 32f data type, no further conversion is done and \( L \), \( U \), and \( V \) components remain in the ranges \([0..100]\), \([-134..220]\), and \([-140..122]\), respectively.

### Return Values

- **ippStsNoErr**
  Indicates no error. Any other value indicates an error.
- **ippStsNullPtrErr**
  Indicates an error condition if \( pSrc \) or \( pDst \) is NULL.
- **ippStsSizeErr**
  Indicates an error condition if \( roiSize \) has a field with a zero or negative value.

### LUVToRGB, LUVToBGR

*Converts a LUV image to the RGB or BGR color model.*

#### Syntax

**Case 1: LUV to RGB**

\[
\text{IppStatus ippiLUVToRGB <mod>(const Ipp<datatype>* pSrc, int srcStep, Ipp<datatype>* pDst, int dstStep, IppiSize roiSize)};
\]

Supported values for \( \text{mod} \):

- 8u_C3R
- 16u_C3R
- 16s_C3R
- 32f_C3R
- 8u_AC4R
- 16u_AC4R
- 16s_AC4R
- 32f_AC4R
Case 2: LUV to BGR

IppStatus ippiLUVToBGR_<mod>(const Ipp<datatype>* pSrc, int srcStep, Ipp<datatype>* pDst, int dstStep, IppSize roiSize);

Supported values for mod:
8u_C3R 32f_C3R

Include Files
ippcc.h

Domain Dependencies
Headers: ippcore.h, ippvm.h, ipps.h, ippi.h
Libraries: ippcore.lib, ippvm.lib, ipps.lib, ippi.lib

Parameters
pSrc Pointer to the source image ROI.
srcStep Distance in bytes between starts of consecutive lines in the source image.
pDst Pointer to the destination image ROI.
dstStep Distance in bytes between starts of consecutive lines in the destination image.
roiSize Size of the source and destination ROI in pixels.

Description
This function operates with ROI (see Regions of Interest in Intel IPP).
This function converts the CIE LUV image pSrc to the RGB or BGR image pDst in two steps. First, the conversion is carried out into CIE XYZ format.
To accomplish it, LUV components are transformed back into their original range. This is done for different data types in the following way.

For 8u data type:
L = L * 100./ IPP_MAX_8U
U = (U * 354./ IPP_MAX_8U) - 134.
V = (V * 262./ IPP_MAX_8U) - 140.

For 16u data type:
L = L * 100./ IPP_MAX_16U
U = (U * 354./ IPP_MAX_16U) - 134.
V = (V * 262./ IPP_MAX_16U) - 140.

For 16s data type:
L = (L - IPP_MIN_16S)* 100./ IPP_MAX_16U
U = ((U - IPP_MIN_16S)* 354./ IPP_MAX_16U) - 134.
V = ((V - IPP_MIN_16S) * 262./ IPP_MAX_16U) - 140.

After that, conversion to XYZ format takes place as follows:
\begin{align*}
  Y &= Y_n \times \left( \frac{(L + 16.)}{116.} \right)^3. \\
  X &= -9. \times Y \times u \div \left( (u - 4.) \times v - u \times v \right) \\
  Z &= \frac{9. \times Y - 15 \times v \times Y - v \times X}{3. \times v} \\

  \text{where} \\
  u &= \frac{U}{13\times L} + u_n \\
  v &= \frac{V}{13\times L} + v_n \\

  \text{and} \\
  u_n &= \frac{4. \times x_n}{(-2. \times x_n + 12. \times y_n + 3.)} \\
  v_n &= \frac{9. \times y_n}{(-2. \times x_n + 12. \times y_n + 3.)}
\end{align*}

Here $x_n = 0.312713$, $y_n = 0.329016$ are the CIE chromaticity coordinates of the D65 white point, and $Y_n = 1.0$ is the luminance of the D65 white point.

After this intermediate conversion is done, the obtained $XYZ$ image is then converted to the destination $RGB$ or $BGR$ format using equations defined for the `ippiXYZToRGB` function.

### Return Values

- **ippStsNoErr**: Indicates no error. Any other value indicates an error.
- **ippStsNullPtrErr**: Indicates an error condition if $pSrc$ or $pDst$ is `NULL`.
- **ippStsSizeErr**: Indicates an error condition if $roiSize$ has a field with a zero or negative value.

**BGRToLab, RGBToLab**

*Converts a BGR or RGB image to the Lab color model.*

### Syntax

**Case 1: BGR to Lab**

```c
IppStatus ippiBGRTolab_8u_C3R(const Ipp8u* pSrc, int srcStep, Ipp8u* pDst, int dstStep, IppiSize roiSize);
IppStatus ippiBGRTolab_8u16u_C3R(const Ipp8u* pSrc, int srcStep, Ipp16u* pDst, int dstStep, IppiSize roiSize);
IppStatus ippiBGRTolab_32f_C3R(const Ipp32f* pSrc, int srcStep, Ipp32f* pDst, int dstStep, IppiSize roiSize);
```

**Case 2: RGB to Lab**

```c
IppStatus ippiRGBToLab_8u_C3R(const Ipp8u* pSrc, int srcStep, Ipp8u* pDst, int dstStep, IppiSize roiSize);
IppStatus ippiRGBToLab_32f_C3R(const Ipp32f* pSrc, int srcStep, Ipp32f* pDst, int dstStep, IppiSize roiSize);
IppStatus ippiRGBToLab_32f_P3R(const Ipp32f* pSrc[3], int srcStep[3], Ipp32f* pDst[3], int dstStep[3], IppiSize roiSize);
IppStatus ippiRGBToLab_64f_P3R(const Ipp64f* pSrc[3], int srcStep[3], Ipp64f* pDst[3], int dstStep[3], IppiSize roiSize);
```
Case 3: RGB to Lab with platform-aware functions

IppStatus ippiRGBToLab_32f_P3R_L(const Ipp32f* pSrc[3], IppSizeL srcStep[3], Ipp32f* pDst[3], IppSizeL dstStep[3], IppiSizeL roiSize);
IppStatus ippiRGBToLab_64f_P3R_L(const Ipp64f* pSrc[3], IppSizeL srcStep[3], Ipp64f* pDst[3], IppSizeL dstStep[3], IppiSizeL roiSize);

Case 4: RGB to Lab with TL functions based on the Platform Aware API

IppStatus ippiRGBToLab_32f_P3R_LT(const Ipp32f* pSrc[3], IppSizeL srcStep[3], Ipp32f* pDst[3], IppSizeL dstStep[3], IppiSizeL roiSize);
IppStatus ippiRGBToLab_64f_P3R_L(const Ipp64f* pSrc[3], IppSizeL srcStep[3], Ipp64f* pDst[3], IppSizeL dstStep[3], IppiSizeL roiSize);

Case 5: RGB to Lab with TL functions based on the Classic API

IppStatus ippiRGBToLab_32f_P3R_T(const Ipp32f* pSrc[3], int srcStep[3], Ipp32f* pDst[3], int dstStep[3], IppiSize roiSize);
IppStatus ippiRGBToLab_64f_P3R_T(const Ipp64f* pSrc[3], int srcStep[3], Ipp64f* pDst[3], int dstStep[3], IppiSize roiSize);

Include Files
ippcc.h
ippcc_l.h
ippcc_tl.h

Domain Dependencies
Headers: ippcore.h, ippvm.h, ipps.h,ippi.h
Libraries: ippcore.lib, ippvm.lib, ipps.lib,ippi.lib

Parameters
pSrc, pSrc[3]  
Pointer to the source image ROI.

srcStep, srcStep[3]  
Distance in bytes between starts of consecutive lines in the source image.

dstStep, dstStep[3]  
Distance in bytes between starts of consecutive lines in the destination image.

roiSize  
Size of the source and destination ROI in pixels.

Description
This function operates with ROI (see Regions of Interest in Intel IPP).
This function converts the BGR or RGB image pSrc to the CIE Lab image pDst, and vice versa. Conversion to Lab consists of two steps. First, the conversion is done into CIE XYZ format, using equations defined for the function ippiRGBtoXYZ. After that, conversion to the Lab image is performed in accordance with the following equations:
L = 116. * \((Y/Y_n) \)^{1/3} - 16 \text{ for } Y/Y_n > 0.008856
L = 903.3 * \((Y/Y_n) \)^{1/3} \text{ for } Y/Y_n \leq 0.008856
a = 500. * \left[f(X/X_n) - f(Y/Y_n)\right]
\[ b = 200. \cdot \left[ f\left(\frac{Y}{Y_n}\right) - f\left(\frac{Z}{Z_n}\right) \right] \]

where

\[ f(t) = \begin{cases} 
\frac{t^{1/3}}{t > (6/29)^3} \\
\frac{1}{3} \left( \frac{1}{6} t + \frac{1}{29} \right) & t \leq (6/29)^3 
\end{cases} \]

Here \( Y_n = 1.0, X_n = 0.950455, Z_n = 1.088753 \) for the D65 white point with the CIE chromaticity coordinates \( x_n = 0.312713, y_n = 0.329016 \).

The equations above are given on the assumption that initial B, G, R values are normalized to the range [0..1]. The computed values of the \( L \) component are in the range [0..100], \( a \) and \( b \) component values are in the range [-128..127].

These values are quantized and scaled to the 8-bit range of 0 to 255 for \( 8u \_C3 \) flavors:

\[ L = L \cdot 255./100. \]
\[ a = (a + 128.) \]
\[ b = (a + 128.) \]

or to the 16-bit range of 0 to 65535 for \( ippBGRToLab\_8u16u\_C3R \):

\[ L = L \cdot 65535./100. \]
\[ a = (a + 128.) \cdot 255 \]
\[ b = (a + 128.) \cdot 255 \]

**Return Values**

- \( ippStsNoErr \) Indicates no error. Any other value indicates an error.
- \( ippStsNullPtrErr \) Indicates an error condition if \( pSrc, pSrc[3], pDst, \) or \( pDst[3] \) is NULL.
- \( ippStsSizeErr \) Indicates an error condition if \( roiSize \) has a field with a zero or negative value.

### LabToBGR, LabToRGB

*Converting a Lab image to the BGR or RGB color model.*

**Syntax**

**Case 1: Lab to BGR**

\[
\text{IppStatus ippiLabToBGR\_8u\_C3R(const Ipp8u\* pSrc, int srcStep, Ipp8u\* pDst, int dstStep, IppiSize roiSize);} \\
\text{IppStatus ippiLabToBGR\_16u8u\_C3R(const Ipp16u\* pSrc, int srcStep, Ipp8u\* pDst, int dstStep, IppiSize roiSize);} \\
\text{IppStatus ippiLabToBGR\_32f\_C3R(const Ipp32f\* pSrc, int srcStep, Ipp32f\* pDst, int dstStep, IppiSize roiSize);} \\
\]
Case 2: Lab to RGB

IppStatusippiLabToRGB_8uC3R(const Ipp8u*pSrc, int srcStep, Ipp8u*pDst, int dstStep, IppiSize roiSize);

IppStatusippiLabToRGB_32fC3R(const Ipp32fpSrc, int srcStep, Ipp32fpDst, int dstStep, IppiSize roiSize);

IppStatusippiLabToRGB_32fP3R(const Ipp32fpSrc[3], int srcStep[3], Ipp32fpDst[3], int dstStep[3], IppiSize roiSize);

IppStatusippiLabToRGB_64fP3R(const Ipp64fpSrc[3], int srcStep[3], Ipp64fpDst[3], int dstStep[3], IppiSize roiSize);

Case 3: Lab to RGB with platform-aware functions

IppStatusippiLabToRGB_32fP3R_L(const Ipp32fpSrc[3], IppSizeL srcStep[3], Ipp32fpDst[3], IppSizeL dstStep[3], IppiSizeL roiSize);

IppStatusippiLabToRGB_64fP3R_L(const Ipp64fpSrc[3], IppSizeL srcStep[3], Ipp64fpDst[3], IppSizeL dstStep[3], IppiSizeL roiSize);

Case 4: Lab to RGB with TL functions based on the Platform Aware API

IppStatusippiLabToRGB_32fP3R_LT(const Ipp32fpSrc[3], IppSizeL srcStep[3], Ipp32fpDst[3], IppSizeL dstStep[3], IppiSizeL roiSize);

IppStatusippiLabToRGB_64fP3R_LT(const Ipp64fpSrc[3], IppSizeL srcStep[3], Ipp64fpDst[3], IppSizeL dstStep[3], IppiSizeL roiSize);

Case 5: Lab to RGB with TL functions based on the Classic API

IppStatusippiLabToRGB_32fP3R_T(const Ipp32fpSrc[3], int srcStep[3], Ipp32fpDst[3], int dstStep[3], IppiSize roiSize);

IppStatusippiLabToRGB_64fP3R_T(const Ipp64fpSrc[3], int srcStep[3], Ipp64fpDst[3], int dstStep[3], IppiSize roiSize);

Include Files

ippcc.h
ippcc_l.h
ippcc_tl.h

Domain Dependencies

Headers: ippcore.h, ippvm.h, ipps.h,ippi.h
Libraries: ippcore.lib, ippvm.lib, ipps.lib,ippi.lib

Parameters

pSrc, pSrc[3]  
Pointer to the source image ROI.

srcStep, srcStep[3]  
Distance in bytes between starts of consecutive lines in the source image.

pDst, pDst[3]  
Pointer to the destination image ROI.

dstStep, dstStep[3]  
Distance in bytes between starts of consecutive lines in the destination image.

roiSize  
Size of the source and destination ROI in pixels.
**Description**

This function operates with ROI (see Regions of Interest in Intel IPP).

This function converts the CIE Lab image \textit{pSrc} to the BGR or RGB image \textit{pDst} in two steps. First, the conversion is carried out into CIE XYZ format.

To accomplish it, Lab components are transformed back into their original range. This is done for different data types in the following way.

For 8u data type:

\[
\begin{align*}
L &= L * 100./255. \\
a &= a - 128. \\
b &= b - 128.
\end{align*}
\]

For 16u data type:

\[
\begin{align*}
L &= L * 100./65535. \\
a &= (a/255. - 128.) \\
b &= (b/255.) - 128.)
\end{align*}
\]

After that, conversion to XYZ format takes place as follows:

\[
\begin{align*}
Y &= Y_n^* P^3. \\
X &= X_n^* (P + a/500.)^3. \\
Z &= Z_n^* (P - b/200.)^3.
\end{align*}
\]

where

\[
P = (L +16)/116.
\]

After this intermediate conversion is done, the obtained XYZ image is then converted to the destination BGR or RGB format using equations defined for the \texttt{ippiXYZToRGB} function.

**Return Values**

- \texttt{ippStsNoErr} 
  Indicates no error. Any other value indicates an error.
- \texttt{ippStsNullPtrErr} 
  Indicates an error condition if \textit{pSrc}, \textit{pSrc[3]}, \textit{pDst}, or \textit{pDst[3]} is NULL.
- \texttt{ippStsSizeErr} 
  Indicates an error condition if \textit{roiSize} has a field with a zero or negative value.

**RGBToYCC**

Converts an RGB image to the YCC color model.

**Syntax**

\[
\begin{align*}
\text{IppStatus ippiRGBToYCC\_mod}(\text{const Ipp<datatype>\* pSrc, int srcStep, Ipp<datatype>\* pDst, int dstStep, IppSize roiSize});
\end{align*}
\]

Supported values for \texttt{mod}:

\[
\begin{align*}
8u\_C3R & & 16u\_C3R & & 16s\_C3R & & 32f\_C3R \\
8u\_AC4R & & 16u\_AC4R & & 16s\_AC4R & & 32f\_AC4R
\end{align*}
\]
Include Files

ippcc.h

Domain Dependencies

Headers: ippcore.h, ippvm.h, ipps.h,ippi.h
Libraries: ippcore.lib, ippvm.lib, ipps.lib,ippi.lib

Parameters

pSrc

Pointer to the source image ROI.

tsrcStep

Distance in bytes between starts of consecutive lines in the source image.

pDst

Pointer to the destination image ROI.

tdstStep

Distance in bytes between starts of consecutive lines in the destination image.

roiSize

Size of the source and destination ROI in pixels.

Description

This function operates with ROI (see Regions of Interest in Intel IPP).

This function converts the gamma-corrected R'B'G' image pSrc to the PhotoY'C'C' image pDst according to the following basic equations:

\[
Y' = 0.299*R' + 0.587*G' + 0.114*B' \\
C1' = -0.299*R' - 0.587*G' + 0.886*B' = B' - Y \\
C2' = 0.701*R' - 0.587*G' - 0.114*B' = R' - Y
\]

The equations above are given on the assumption that R', G', and B' values are normalized to the range [0..1]. In case of the floating-point data type, the input R'G'B' values must already be in the range [0..1]. For integer data types, normalization is done by the conversion function internally.

The computed Y', C1', C2' values are then quantized and converted to fit in the range [0..1] as follows:

\[
Y' = 1. / 1.402 * Y' \\
C1' = 111.4 / 255. * C1' + 156. / 255. \\
C2' = 135.64 /255. * C2' + 137. / 255.
\]

In case of integer function flavors, these values are then scaled to the full range of the destination data type (see Table "Image Data Types and Ranges" in Chapter 2).

Return Values

ippStsNoErr

Indicates no error. Any other value indicates an error.

ippStsNullPtrErr

Indicates an error condition if pSrc or pDst is NULL.

ippStsSizeErr

Indicates an error condition if roiSize has a field with a zero or negative value.

YCCToRGB

Converts a YCC image to the RGB color model.
Syntax

IppStatus ippiYCCToRGB_<mod>(const Ipp<datatype>* pSrc, int srcStep, Ipp<datatype>* pDst, int dstStep, IppiSize roiSize);

Supported values for mod:

<p>| | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>8u_C3R</td>
<td>16u_C3R</td>
<td>16s_C3R</td>
<td>32f_C3R</td>
</tr>
<tr>
<td>8u_AC4R</td>
<td>16u_AC4R</td>
<td>16s_AC4R</td>
<td>32f_AC4R</td>
</tr>
</tbody>
</table>

Include Files

ippcc.h

Domain Dependencies

Headers: ippcore.h, ippvm.h, ipps.h, ippi.h

Libraries: ippcore.lib, ippvm.lib, ipps.lib, ippi.lib

Parameters

pSrc

Pointer to the source image ROI.

srcStep

Distance in bytes between starts of consecutive lines in the source image.

pDst

Pointer to the destination image ROI.

dstStep

Distance in bytes between starts of consecutive lines in the destination image.

roiSize

Size of the source and destination ROI in pixels.

Description

This function operates with ROI (see Regions of Interest in Intel IPP).

This function converts the PhotoY'C'C' image pSrc to the R'B'G' image pDst. The function ippiYCCToRGB first restores normal luminance and chrominance data as:

\[
Y' = 1.3584 \times Y' \\
C1' = 2.2179 \times (C1' - 156./255.) \\
C2' = 1.8215 \times (C2' - 137./255.)
\]

The equations above are given on the assumption that source Y, C1, and C2 values are normalized to the range [0..1]. In case of the floating-point data type, the input YCC values must already be in the range [0..1]. For integer data types, normalization is done by the conversion function internally.

After that, YCC data are transformed into RGB format according to the following basic equations:

\[
R' = Y' + C2' \\
G' = Y' - 0.194 \times C1' - 0.509 \times C2' \\
B' = Y' + C1'
\]

In case of integer function flavors, the computed R'B'G' values are then scaled to the full range of the destination data type (see Table “Image Data Types and Ranges” in Chapter 2).

Return Values

ippStsNoErr

Indicates no error. Any other value indicates an error.
Indicates an error condition if \( pSrc \) or \( pDst \) is NULL.
Indicates an error condition if \( roiSize \) has a field with a zero or negative value.

**RGBToHLS**

*Converts an RGB image to the HLS color model.*

**Syntax**

\[
\text{IppStatus ippiRGBToHLS}_<\text{mod}>(\text{const Ipp<datatype>* } pSrc, \text{ int srcStep}, \text{ Ipp<datatype>* } pDst, \text{ int dstStep}, \text{ IppiSize roiSize});
\]

**Supported values for** \( \text{mod} \):

- 8u_C3R
- 16u_C3R
- 16s_C3R
- 32f_C3R
- 8u_AC4R
- 16u_AC4R
- 16s_AC4R
- 32f_AC4R

**Include Files**

ippcc.h

**Domain Dependencies**

*Headers:* ippcore.h, ippvm.h, ipps.h,ippi.h

*Libraries:* ippcore.lib, ippvm.lib, ipps.lib,ippi.lib

**Parameters**

- \( pSrc \)
- Pointer to the source image ROI.

- \( srcStep \)
- Distance in bytes between starts of consecutive lines in the source image.

- \( pDst \)
- Pointer to the destination image ROI.

- \( dstStep \)
- Distance in bytes between starts of consecutive lines in the destination image.

- \( roiSize \)
- Size of the source and destination ROI in pixels.

**Description**

This function operates with ROI (see Regions of Interest in Intel IPP).

This function converts the R'B'G' image \( pSrc \) to the HLS image \( pDst \). For function flavors operating on the floating point data, source RGB values must be in the range \([0..1]\).

The conversion algorithm from RGB to HLS can be represented in pseudocode as follows:

```c
// Lightness:
M1 = \max(R,G,B); M2 = \min(R,G,B); L = (M1+M2)/2

// Saturation:
if M1 = M2 then // achromatics case
  S = 0
  H = 0
else // chromatics case
  if L <= 0.5 then
    S = (M1-M2) / (M1+M2)
  else
    S = (M1-M2) / (2-M1-M2)
```

Image Color Conversion
// Hue:
Cr = (M1-R) / (M1-M2)
Cg = (M1-G) / (M1-M2)
Cb = (M1-B) / (M1-M2)
if R = M1 then H = Cb - Cg               //change R=M2 to R=M1
if G = M1 then H = 2 + Cr - Cb         //change G=M2 to G=M1
if B = M1 then H = 4 + Cg - Cr         //change B=M2 to B=M1
H = 60*H
if H < 0 then H = H + 360

For floating point function flavors, the computed \( H, \ L, \ S \) values are scaled to the range \([0..1]\). In case of integer function flavors, these values are scaled to the full range of the destination data type (Table "Image Data Types and Ranges").

**Return Values**

- ippStsNoErr: Indicates no error. Any other value indicates an error.
- ippStsNullPtrErr: Indicates an error condition if \( pSrc \) or \( pDst \) is NULL.
- ippStsSizeErr: Indicates an error condition if \( roiSize \) has a field with a zero or negative value.

**HLSToRGB**

**Converts an HLS image to the RGB color model.**

**Syntax**

```cpp
IppStatus ippiHLSToRGB_<mod>(const Ipp<datatype>* pSrc, int srcStep, Ipp<datatype>* pDst, int dstStep, IppiSize roiSize);
```

Supported values for mod:

<table>
<thead>
<tr>
<th>8u_C3R</th>
<th>16u_C3R</th>
<th>16s_C3R</th>
<th>32f_C3R</th>
</tr>
</thead>
<tbody>
<tr>
<td>8u_AC4R</td>
<td>16u_AC4R</td>
<td>16s_AC4R</td>
<td>32f_AC4R</td>
</tr>
</tbody>
</table>

**Include Files**

ippcc.h

**Domain Dependencies**

Headers: ippcore.h, ippvm.h, ipps.h,ippi.h

Libraries: ippcore.lib, ippvm.lib, ipps.lib,ippi.lib

**Parameters**

- \( pSrc \): Pointer to the source image ROI.
- \( srcStep \): Distance in bytes between starts of consecutive lines in the source image.
- \( pDst \): Pointer to the destination image ROI.
- \( dstStep \): Distance in bytes between starts of consecutive lines in the destination image.
- \( roiSize \): Size of the source and destination ROI in pixels.
Description

This function operates with ROI (see Regions of Interest in Intel IPP).

This function converts the HLS image \( pSrc \) to the R'B'G' image \( pDst \). For function flavors operating on the floating point data, source HLS values must be in the range \([0..1]\). The conversion algorithm from HLS to RGB can be represented in pseudocode as follows:

\[
\begin{align*}
\text{if } L \leq 0.5 & \text{ then } M2 = L \cdot (1 + S) \\
\text{else } & M2 = L + S - L \cdot S \\
M1 & = 2 \cdot L - M2 \text{ if } S = 0 \\
& \text{if } h \leq 60 \text{ then } R = G = B = L \\
& \text{if } h > 360 \text{ then } h = h - 360 \\
& \text{else if } h < 60 \text{ then } R = (M1 + (M2 - M1) \cdot h / 60) \\
& \text{else if } h < 180 \text{ then } G = (M1 + (M2 - M1) \cdot h / 240) / 60 \\
& \text{else if } h < 240 \text{ then } B = M1 \\
& \text{else if } h < 60 \text{ then } R = (M2 - 120) \cdot h / 60 \\
& \text{else if } h < 180 \text{ then } G = (M2 - 120) \cdot h / 60 \\
& \text{else if } h < 240 \text{ then } B = M1 \\
& \text{else } h += 360 \\
& \text{else if } h < 60 \text{ then } R = M1 \\
& \text{else if } h < 180 \text{ then } G = M1 \\
& \text{else if } h < 240 \text{ then } B = M1 \\
& \text{else } R = M1 \\
& \text{else } G = M1 \\
& \text{else } B = M1 \\
\end{align*}
\]

For floating point function flavors, the computed \( R', G', B' \) values are scaled to the range \([0..1]\). In case of integer function flavors, these values are scaled to the full range of the destination data type (see Table “Image Data Types and Ranges” in Chapter 2).

Return Values

- **ippStsNoErr**
  Indicates no error. Any other value indicates an error.
- **ippStsNullPtrErr**
  Indicates an error condition if \( pSrc \) or \( pDst \) is NULL.
- **ippStsSizeErr**
  Indicates an error condition if \( roiSize \) has a field with a zero or negative value.

BGRToHLS

*Converts a BGR image to the HLS color model.*

Syntax

\[
\begin{align*}
\text{IppStatus ippiBGRToHLS}_8u\_AC4R(\text{const Ipp8u* } pSrc, \text{ int srcStep, Ipp8u* } pDst, \text{ int dstStep, IppiSize roiSize}); \\
\text{IppStatus ippiBGRToHLS}_8u\_C3P3R(\text{const Ipp8u* } pSrc, \text{ int srcStep, Ipp8u* } pDst[3], \text{ int dstStep, IppiSize roiSize}); \\
\text{IppStatus ippiBGRToHLS}_8u\_AC4P4R(\text{const Ipp8u* } pSrc, \text{ int srcStep, Ipp8u* } pDst[4], \text{ int dstStep, IppiSize roiSize}); \\
\text{IppStatus ippiBGRToHLS}_8u\_P3C3R(\text{const Ipp8u* } pSrc[3], \text{ int srcStep, Ipp8u* } pDst, \text{ int dstStep, IppiSize roiSize}); \\
\text{IppStatus ippiBGRToHLS}_8u\_AP4C4R(\text{const Ipp8u* } pSrc[4], \text{ int srcStep, Ipp8u* } pDst, \text{ int dstStep, IppiSize roiSize}); \\
\text{IppStatus ippiBGRToHLS}_8u\_P3R(\text{const Ipp8u* } pSrc[3], \text{ int srcStep, Ipp8u* } pDst[3], \text{ int dstStep, IppiSize roiSize}); \\
\text{IppStatus ippiBGRToHLS}_8u\_AP4R(\text{const Ipp8u* } pSrc[4], \text{ int srcStep, Ipp8u* } pDst[4], \text{ int dstStep, IppiSize roiSize}); \\
\end{align*}
\]

Include Files

ippcc.h
Domain Dependencies

Headers: ippcore.h, ippvm.h, ipps.h,ippi.h
Libraries: ippcore.lib, ippvm.lib, ipps.lib,ippi.lib

Parameters

pSrc  Pointer to the ROI in the pixel-order source image. An array of pointers to ROI in each plane in the planar source image.
srcStep  Distance in bytes between starts of consecutive lines in the source image.

pDst  Pointer to the ROI in the pixel-order destination image. An array of pointers to ROI in each plane in the planar destination image.
dstStep  Distance in bytes between starts of consecutive lines in the destination image.
roiSize  Size of the source and destination ROI in pixels.

Description

This function operates with ROI (see Regions of Interest in Intel IPP).

This function converts the B'G'R' image pSrc to the HLS image pDst according to the same formula as the function ippiRGBToHLS does.

Return Values

ippStsNoErr  Indicates no error. Any other value indicates an error.
ippStsNullPtrErr  Indicates an error condition if pSrc or pDst is NULL.
ippStsSizeErr  Indicates an error condition if roiSize has a field with a zero or negative value.

HLSToBGR

Converts an HLS image to the RGB color model.

Syntax

IppStatus ippiHLSToBGR_8u_C3P3R(const Ipp8u* pSrc, int srcStep, Ipp8u* pDst[3], int dstStep, IppSize roiSize);
IppStatus ippiHLSToBGR_8u_AC4P4R(const Ipp8u* pSrc, int srcStep, Ipp8u* pDst[4], int dstStep, IppSize roiSize);
IppStatus ippiHLSToBGR_8u_AP4R(const Ipp8u* pSrc[4], int srcStep, Ipp8u* pDst[4], int dstStep, IppSize roiSize);
IppStatus ippiHLSToBGR_8u_AP4C4R(const Ipp8u* pSrc[4], int srcStep, Ipp8u* pDst, int dstStep, IppSize roiSize);
IppStatus ippiHLSToBGR_8u_P3R(const Ipp8u* pSrc[3], int srcStep, Ipp8u* pDst[3], int dstStep, IppSize roiSize);
IppStatus ippiHLSToBGR_8u_AP4C4R(const Ipp8u* pSrc[4], int srcStep, Ipp8u* pDst, int dstStep, IppSize roiSize);
Include Files
ippcc.h

Domain Dependencies
Headers: ippcore.h, ippvm.h, ipps.h,ippi.h
Libraries: ippcore.lib, ippvm.lib, ipps.lib,ippi.lib

Parameters

- pSrc: Pointer to the ROI in the pixel-order source image. An array of pointers to ROI in each plane in the planar source image.
- srcStep: Distance in bytes between starts of consecutive lines in the source image.
- pDst: Pointer to the ROI in the pixel-order destination image. An array of pointers to ROI in each plane in the planar destination image.
- dstStep: Distance in bytes between starts of consecutive lines in the destination image.
- roiSize: Size of the source and destination ROI in pixels.

Description
This function operates with ROI (see Regions of Interest in Intel IPP).
This function converts the HLS image pSrc to the B'G'R' image pDst according to the same formula as the function ippiHLSToRGB does.

Return Values
- ippStsNoErr: Indicates no error. Any other value indicates an error.
- ippStsNullPtrErr: Indicates an error condition if pSrc or pDst is NULL.
- ippStsSizeErr: Indicates an error condition if roiSize has a field with a zero or negative value.

RGBToHSV
Converting an RGB image to the HSV color model.

Syntax
IppStatus ippiRGBToHSV_<mod>(const Ipp<datatype>* pSrc, int srcStep, Ipp<datatype>* pDst, int dstStep, IppiSize roiSize);

Supported values for mod:
- 8u_C3R
- 16u_C3R
- 8u_AC4R
- 16u_AC4R

Include Files
ippcc.h

Domain Dependencies
Headers: ippcore.h, ippvm.h, ipps.h,ippi.h
Libraries: ippcore.lib, ippvm.lib, ipps.lib,ippi.lib

Parameters

- **pSrc**: Pointer to the source image ROI.
- **srcStep**: Distance in bytes between starts of consecutive lines in the source image.
- **pDst**: Pointer to the destination image ROI.
- **dstStep**: Distance in bytes between starts of consecutive lines in the destination image.
- **roiSize**: Size of the source and destination ROI in pixels.

Description

This function operates with ROI (see Regions of Interest in Intel IPP).

This function converts the RGB image **pSrc** to the HSV image **pDst**.

The conversion algorithm from RGB to HSV can be represented in pseudocode as follows:

```
// Value: V = max(R,G,B); // Saturation: temp = min(R,G,B); if V = 0 then // achromatics
case S = 0:// chromatics case
H = (V - temp)/V // Hue: Cr = (V - R) / (V - temp) Cg = (V - G) / (V - temp) Cb = (V - B) / (V - temp)
if R = V then H = Cb - Cg if G = V then H = 2 + Cr - Cb if B = V then H = 4 + Cg - Cr
H = 60*H if H < 0 then H = H + 360
```

The computed H, S, V values are scaled to the full range of the destination data type (see Table "Image Data Types and Ranges" in Chapter 2).

Return Values

- **ippStsNoErr**: Indicates no error. Any other value indicates an error.
- **ippStsNullPtrErr**: Indicates an error condition if **pSrc** or **pDst** is NULL.
- **ippStsSizeErr**: Indicates an error condition if **roiSize** has a field with a zero or negative value.

**HSVToRGB**

*Converts an HSV image to the RGB color model.*

Syntax

```
IppStatus ippiHSVToRGB_<mod>(const Ipp<datatype>* pSrc, int srcStep, Ipp<datatype>* pDst, int dstStep, IppiSize roiSize);
```

Supported values for **mod**:

- 8u_C3R
- 16u_C3R
- 8u_AC4R
- 16u_AC4R

Include Files

ippcc.h

Domain Dependencies

Headers: ippcore.h, ippvm.h, ipps.h,ippi.h

Libraries: ippcore.lib, ippvm.lib, ipps.lib,ippi.lib
Parameters

**pSrc**
Pointer to the source image ROI.

**srcStep**
Distance in bytes between starts of consecutive lines in the source image.

**pDst**
Pointer to the destination image ROI.

**dstStep**
Distance in bytes between starts of consecutive lines in the destination image.

**roiSize**
Size of the source and destination ROI in pixels.

Description

This function operates with ROI (see Regions of Interest in Intel IPP).

This function converts a HSV image `pSrc` to the image `R'G'B'` `pDst`.

The conversion algorithm from HSV to RGB can be represented in pseudocode as follows:

```
if (S == 0) then
  R = G = B = V
else
  if (H == 360) then
    H = 0
  else
    H = H / 60
  I = floor(H)
  F = H - I
  M = V * (1 - S)
  N = V * (1 - S * F)
  K = V * (1 - S * (1 - F))

  if (I == 0) then
    R = V; G = K; B = M
  if (I == 1) then
    R = N; G = V; B = M
  if (I == 2) then
    R = M; G = V; B = K
  if (I == 3) then
    R = M; G = N; B = V
  if (I == 4) then
    R = K; G = M; B = V
  if (I == 5) then
    R = V; G = M; B = N
```

The computed `R', G', B'` values are scaled to the full range of the destination data type (see Table "Image Data Types and Ranges" in Chapter 2).

Return Values

- **ippStsNoErr**: Indicates no error. Any other value indicates an error.
- **ippStsNullPtrErr**: Indicates an error condition if `pSrc` or `pDst` is NULL.
- **ippStsSizeErr**: Indicates an error condition if `roiSize` has a field with a zero or negative value.

RGBToYCoCg

*Converts a RGB image to the YCoCg color model.*

Syntax

```
IppStatus ippiRGBToYCoCg_8u_C3P3R(const Ipp8u* pSrc, int srcStep, Ipp8u* pDst[3], int dstStep, IppiSize roi);
```

Include Files

ippcc.h

Domain Dependencies

- **Headers**: ippcore.h, ippvm.h, ipps.h,ippi.h
- **Libraries**: ippcore.lib, ippvm.lib, ipps.lib,ippi.lib

Parameters

**pSrc**
Pointer to the source image ROI.
**Description**
This function operates with ROI (see Regions of Interest in Intel IPP).

This function converts a RGB image `pSrc` to the YCoCg image `pDst` according to the following formulas:

\[
Y = ((R + 2G + B) + 2)/4 \\
Co = ((R - B) + 1)/2 \\
Cg = ((-R + 2G - B) + 2)/4
\]

**Return Values**

- **ippStsNoErr**: Indicates no error. Any other value indicates an error.
- **ippStsNullPtrErr**: Indicates an error condition if one of the specified pointers is NULL.

**YCoCgToRGB**
*Converts a YCoCg image to the RGB image.*

**Syntax**

```c
IppStatus ippiYCoCgToRGB_8u_P3C3R(const Ipp8u* pSrc[3], int srcStep, Ipp8u* pDst, int dstStep, IppiSize roi);
```

**Include Files**

ippcc.h

**Domain Dependencies**

Headers: ippcore.h, ippvm.h, ipps.h,ippi.h

Libraries: ippcore.lib, ippvm.lib, ipps.lib,ippi.lib

**Parameters**

- **pSrc**: Array of pointers to the source image ROI in each plane.
- **srcStep**: Distance in bytes between starts of consecutive lines in the source image.
- **pDst**: Pointer to the destination image ROI.
- **dstStep**: Distance in bytes between starts of consecutive lines in the destination image.
- **roi**: Size of the source and destination ROI in pixels.

**Description**

This function operates with ROI (see Regions of Interest in Intel IPP).
This function converts the YCoCg image \( pSrc \) to the RGB image \( pDst \) according to the following formulas:

\[
\begin{align*}
R &= Y + Co - Cg \\
G &= Y + Cg \\
B &= Y - Co - Cg
\end{align*}
\]

**Return Values**

- ippStsNoErr: Indicates no error. Any other value indicates an error.
- ippStsNullPtrErr: Indicates an error condition if one of the specified pointers is NULL.

**BGRToYCoCg**

Converts a 24-bit BGR image to the YCoCg color model.

**Syntax**

IppStatus ippiBGRToYCoCg_8u16s_C3P3R(const Ipp8u* \( pBGR \), int \( bgrStep \), Ipp16s* \( pYCC \)[3], int \( yccStep \), IppiSize \( roiSize \));

IppStatus ippiBGRToYCoCg_8u16s_C4P3R(const Ipp8u* \( pBGR \), int \( bgrStep \), Ipp16s* \( pYCC \)[3], int \( yccStep \), IppiSize \( roiSize \));

**Include Files**

ippcc.h

**Domain Dependencies**

Headers: ippcore.h, ippvm.h, ipp.h, ippi.h

Libraries: ippcore.lib, ippvm.lib, ipp.lib, ippi.lib

**Parameters**

- \( pBGR \): Pointer to the source image ROI.
- \( bgrStep \): Distance in bytes between starts of consecutive lines in the source image.
- \( pYCC \): Array of pointers to the destination image ROI in each plane.
- \( yccStep \): Distance in bytes between starts of consecutive lines in the destination image.
- \( roiSize \): Size of the source and destination ROI in pixels.

**Description**

This function operates with ROI (see Regions of Interest in Intel IPP).

This function converts the 24-bit BGR image \( pBGR \) to the YCoCg image \( pYCC \) according to the following formulas:

\[
\begin{align*}
Y &= ((R + 2*G + B) + 2)/4 \\
Co &= ((R - B) + 1)/2 \\
Cg &= (( - R + 2*G - B) + 2)/4
\end{align*}
\]
Return Values

- ippStsNoErr: Indicates no error. Any other value indicates an error.
- ippStsNullPtrErr: Indicates an error condition if one of the specified pointers is NULL.

SBGRToYCoCg

Converts a 48-bit BGR image to the YCoCg color model.

Syntax

IppStatus ippiSBGRToYCoCg_<mod>(const Ipp16s* pBGR, int bgrStep, Ipp<dstDatatype>* pYCC[3], int yccStep, IppiSize roiSize);

Supported values for mod:

- 16s_C3P3R
- 16s32s_C3P3R
- 16s_C4P3R
- 16s32s_C4P3R

Include Files

ippcc.h

Domain Dependencies

Headers: ippcore.h, ippvm.h, ipps.h,ippi.h
Libraries: ippcore.lib, ippvm.lib, ipps.lib,ippi.lib

Parameters

- pBGR: Pointer to the source image ROI.
- bgrStep: Distance in bytes between starts of consecutive lines in the source image.
- pYCC: Array of pointers to the destination image ROI in each plane.
- yccStep: Distance in bytes between starts of consecutive lines in the destination image.
- roiSize: Size of the source and destination ROI in pixels.

Description

This function operates with ROI (see Regions of Interest in Intel IPP).

This function converts the 48-bit BGR image pBGR to the YCoCg image pYCC according to the following formulas:

\[
\begin{align*}
Y &= \frac{(R + 2\times G + B) + 2}{4} \\
Co &= \frac{(R - B) + 1}{2} \\
Cg &= \frac{(\ - R + 2\times G - B) + 2}{4}
\end{align*}
\]

Return Values

- ippStsNoErr: Indicates no error. Any other value indicates an error.
Indicates an error condition if one of the specified pointers is NULL.

**YCoCgToBGR**

*Converts a YCoCg image to the 24-bit BGR image.*

**Syntax**

IppStatus ippiYCoCgToBGR_16s8u_P3C3R(const Ipp16s* pYCC[3], int yccStep, Ipp8u* pBGR, int bgrStep, IppiSize roiSize);

IppStatus ippiYCoCgToBGR_16s8u_P3C4R(const Ipp16s* pYCC[3], int yccStep, Ipp8u* pBGR, int bgrStep, IppiSize roiSize, Ipp8u aval);

**Include Files**

ippcc.h

**Domain Dependencies**

*Headers:* ippcore.h, ippvm.h, ipps.h, ippi.h

*Libraries:* ippcore.lib, ippvm.lib, ipps.lib, ippi.lib

**Parameters**

- **pYCC**
  - Array of pointers to the source image ROI in each plane.
- **yccStep**
  - Distance in bytes between starts of consecutive lines in the source image.
- **pBGR**
  - Pointer to the destination image ROI.
- **bgrStep**
  - Distance in bytes between starts of consecutive lines in the destination image.
- **roiSize**
  - Size of the source and destination ROI in pixels.
- **aval**
  - Constant value to create the fourth channel.

**Description**

This function operates with ROI (see Regions of Interest in Intel IPP).

This function converts the YCoCg image *pYCC* to the 24-bit BGR image *pBGR* according to the following formulas:

- \( R = Y + Co - Cg \)
- \( G = Y + Cg \)
- \( B = Y - Co - Cg \)

The fourth channel is created by setting channel values to the constant value *aval*.

**Return Values**

- **ippStsNoErr**
  - Indicates no error. Any other value indicates an error.
- **ippStsNullPtrErr**
  - Indicates an error condition if one of the specified pointers is NULL.
YCoCgToSBGR

Converts a YCoCg image to the 48-bit BGR image.

Syntax

Case 1: Conversion to 3-channel image

IppStatus ippiYCoCgToSBGR_16s_P3C3R(const Ipp16s* pYCC[3], int yccStep, Ipp16s* pBGR, int bgrStep, IppiSize roiSize);
IppStatus ippiYCoCgToSBGR_32s16s_P3C3R(const Ipp32s* pYCC[3], int yccStep, Ipp16s* pBGR, int bgrStep, IppiSize roiSize);

Case 2: Conversion to 4-channel image

IppStatus ippiYCoCgToSBGR_16s_P3C4R(const Ipp16s* pYCC[3], int yccStep, Ipp16s* pBGR, int bgrStep, IppiSize roiSize, Ipp16s aval);
IppStatus ippiYCoCgToSBGR_32s16s_P3C4R(const Ipp32s* pYCC[3], int yccStep, Ipp16s* pBGR, int bgrStep, IppiSize roiSize, Ipp16s aval);

Include Files

ippcc.h

Domain Dependencies

Headers: ippcore.h, ippvm.h, ipps.h,ippi.h
Libraries: ippcore.lib, ippvm.lib, ipps.lib,ippi.lib

Parameters

pYCC

Array of pointers to the source image ROI in each plane.
yccStep

Distance in bytes between starts of consecutive lines in the source image.
pBGR

Pointer to the destination image ROI.
bgrStep

Distance in bytes between starts of consecutive lines in the destination image.
roiSize

Size of the source and destination ROI in pixels.
aval

Constant value to create the fourth channel.

Description

This function operates with ROI (see Regions of Interest in Intel IPP).

This function converts the YCoCg image pYCC to the 48-bit BGR image pBGR according to the following formulas:

\[ R = Y + Co - Cg \]
\[ G = Y + Cg \]
\[ B = Y - Co - Cg \]

The fourth channel is created by setting channel values to the constant value aval.
Return Values

ippStsNoErr
Indicates no error. Any other value indicates an error.

ippStsNullPtrErr
Indicates an error condition if one of the specified pointers is NULL.

BGRToYCoCg_Rev
Converts a 24-bit BGR image to the YCoCg-R color model.

Syntax

IppStatus ippiBGRToYCoCg_Rev_8u16s_C3P3R(const Ipp8u* pBGR, int bgrStep, Ipp16s* pYCC[3], int yccStep, IppiSize roiSize);
IppStatus ippiBGRToYCoCg_Rev_8u16s_C4P3R(const Ipp8u* pBGR, int bgrStep, Ipp16s* pYCC[3], int yccStep, IppiSize roiSize);

Include Files

ippcc.h

Domain Dependencies

Headers: ippcore.h, ippvm.h, ipps.h,ippi.h
Libraries: ippcore.lib, ippvm.lib, ipps.lib,ippi.lib

Parameters

pBGR
Pointer to the source image ROI.

bgrStep
Distance in bytes between starts of consecutive lines in the source image.

pYCC
Array of pointers to the destination image ROI in each plane.

yccStep
Distance in bytes between starts of consecutive lines in the destination image.

roiSize
Size of the source and destination ROI in pixels.

Description

This function operates with ROI (see Regions of Interest in Intel IPP).

This function converts the 24-bit BGR image pSrc to the YCoCg-R image pDst according to the following formulas:

\[
\begin{align*}
C_o &= R - B \\
t &= B + (C_o \gg 1) \\
C_g &= G - t \\
Y &= t + (C_g \gg 1)
\end{align*}
\]

Return Values

ippStsNoErr
Indicates no error. Any other value indicates an error.
Indicates an error condition if one of the specified pointers is NULL.

**SBGRToYCoCg_Rev**

*Converts a 48-bit BGR image to the YCoCg-R color model.*

**Syntax**

```
IppStatus ippiSBGRToYCoCg_Rev_<mod>(const Ipp16s* pBGR, int bgrStep, Ipp<dstDatatype>* pYCC[3], int yccStep, IppiSize roiSize);
```

Supported values for `mod`:

- 16s_C3P3R
- 16s32s_C3P3R
- 16s_C4P3R
- 16s32s_C4P3R

**Include Files**

`ippcc.h`

**Parameters**

- `pBGR`:
  - Pointer to the source image ROI.
- `bgrStep`:
  - Distance in bytes between starts of consecutive lines in the source image.
- `pYCC`:
  - Array of pointers to the destination image ROI in each plane.
- `yccStep`:
  - Distance in bytes between starts of consecutive lines in the destination image.
- `roiSize`:
  - Size of the source and destination ROI in pixels.

**Description**

This function operates with ROI (see Regions of Interest in Intel IPP).

This function converts the 48-bit BGR image `pBGR` to the YCoCg-R image `pYCC` according to the following formulas:

\[
\begin{align*}
C_0 &= R - B \\
t &= B + (C_0 >> 1) \\
C_g &= G - t \\
Y &= t + (C_g >> 1)
\end{align*}
\]

**Return Values**

- `ippStsNoErr`:
  - Indicates no error. Any other value indicates an error.
- `ippStsNullPtrErr`:
  - Indicates an error condition if one of the specified pointers is NULL.

---

**YCoCgToBGR_Rev**

*Converts a YCoCg-R image to the 24-bit BGR image.*
Syntax
IppStatus ippiYCoCgToBGR_Rev_16s8u_P3C3R(const Ipp16s* pYCC[3], int yccStep, Ipp8u* pBGR, int bgrStep, IppiSize roiSize);
IppStatus ippiYCoCgToBGR_Rev_16s8u_P3C4R(const Ipp16s* pYCC[3], int yccStep, Ipp8u* pBGR, int bgrStep, IppiSize roiSize, Ipp8u aval);

Include Files
ippcc.h

Domain Dependencies
Headers: ippcore.h, ippvm.h, ipps.h, ippi.h
Libraries: ippcore.lib, ippvm.lib, ipps.lib, ippi.lib

Parameters
pYCC Array of pointers to the source image ROI in each plane.
yccStep Distance in bytes between starts of consecutive lines in the source image.
pBGR Pointer to the destination image ROI.
bgrStep Distance in bytes between starts of consecutive lines in the destination image.
roiSize Size of the source and destination ROI in pixels.
aval Constant value to create the fourth channel.

Description
This function operates with ROI (see Regions of Interest in Intel IPP).
This function converts the YCoCg-R image pYCC to the 24-bit BGR image pBGR according to the following formulas:
\[ t = Y - (Cg >> 1) \]
\[ G = Cg + t \]
\[ B = t - (Co >> 1) \]
\[ R = B + Co \]
The fourth channel is created by setting channel values to the constant value aval.

Return Values
ippStsNoErr Indicates no error. Any other value indicates an error.
ippStsNullPtrErr Indicates an error condition if one of the specified pointers is NULL.

YCoCgToSBGR_Rev
Converts a YCoCg-R image to the 48-bit BGR image.
Syntax

Case 1: Conversion to 3-channel image.

```c
IppStatus ippiYCoCgToSBGR_16s_P3C3R(const Ipp16s* pYCC[3], int yccStep, Ipp16s* pBGR, int bgrStep, IppiSize roiSize);
IppStatus ippiYCoCgToSBGR_32s16s_P3C3R(const Ipp32s* pYCC[3], int yccStep, Ipp16s* pBGR, int bgrStep, IppiSize roiSize);
```

Case 2: Conversion to 4-channel image

```c
IppStatus ippiYCoCgToSBGR_16s_P3C4R(const Ipp16s* pYCC[3], int yccStep, Ipp16s* pBGR, int bgrStep, IppiSize roiSize, Ipp16s aval);
IppStatus ippiYCoCgToSBGR_32s16s_P3C4R(const Ipp32s* pYCC[3], int yccStep, Ipp16s* pBGR, int bgrStep, IppiSize roiSize, Ipp16s aval);
```

Include Files

ippcc.h

Domain Dependencies

Headers: ippcore.h, ippvm.h, ipps.h,ippi.h
Libraries: ippcore.lib, ippvm.lib, ipps.lib,ippi.lib

Parameters

- **pYCC**: Array of pointers to the source image ROI in each plane.
- **yccStep**: Distance in bytes between starts of consecutive lines in the source image.
- **pBGR**: Pointer to the destination image ROI.
- **bgrStep**: Distance in bytes between starts of consecutive lines in the destination image.
- **roiSize**: Size of the source and destination ROI in pixels.
- **aval**: Constant value to create the fourth channel.

Description

This function operates with ROI (see Regions of Interest in Intel IPP).

This function converts the YCoCg-R image **pYCC** to the 48-bit BGR image **pBGR** according to the following formulas:

- \( t = Y - (C_g >> 1) \)
- \( G = C_g + t \)
- \( B = t - (C_o >> 1) \)
- \( R = B + C_o \)

The fourth channel is created by setting channel values to the constant value **aval**.

Return Values

- **ippStsNoErr**: Indicates no error. Any other value indicates an error.
Indicates an error condition if one of the specified pointers is NULL.

Color - Gray Scale Conversions

GrayToRGB
Converts a gray scale image to RGB/BGR by copying luminance component to color components.

Syntax
IppStatus ippiGrayToRGB_<mod>(const Ipp<datatype>* pSrc, int srcStep, Ipp<datatype>* pDst, int dstStep, IppiSize roiSize);

Supported values for mod:
8u_C1C3R 16u_C1C3R 32f_C1C3R

IppStatus ippiGrayToRGB_<mod>(const Ipp<datatype>* pSrc, int srcStep, Ipp<datatype>* pDst, int dstStep, IppiSize roiSize, Ipp<datatype> aval);

Supported values for mod:
8u_C1C4R 16u_C1C4R 32f_C1C4R

Include Files
ippcc.h

Domain Dependencies
Headers: ippcore.h, ippvm.h, ipps.h,ippi.h
Libraries: ippcore.lib, ippvm.lib, ipps.lib,ippi.lib

Parameters
pSrc
Pointer to the source image ROI.
srcStep
Distance, in bytes, between the starting points of consecutive lines in the source image.
pDst
Pointer to the destination image ROI.
dstStep
Distance, in bytes, between the starting points of consecutive lines in the destination image.
roiSize
Size of the source and destination ROI, in pixels.
aval
Constant value to create the fourth channel.

Description
This function operates with ROI (see Regions of Interest in Intel IPP).
This function converts a gray scale image to an RGB/BGR image by copying luminance component to color components.
### Return Values

- **ippStsNoErr**: Indicates no error. Any other value indicates an error.
- **ippStsNullPtrErr**: Indicates an error when *pSrc* or *pDst* is NULL.
- **ippStsSizeErr**: Indicates an error condition if *roiSize* has a field with a zero or negative value.

### Example

The code example below demonstrates how to use the `ippiGrayToRGB_8u_C1C4R` function.

```c
const int WIDTH = 2;
const int HEIGHT = 1;

Ipp8u pSrc[WIDTH * HEIGHT] = {
    113, 113,
};
Ipp8u pDst[WIDTH * HEIGHT * 4];
int srcStep = WIDTH, dstStep = WIDTH * 4;
IppiSize roiSize = {WIDTH, HEIGHT};
IppStatus status = ippiGrayToRGB_8u_C1C4R(pSrc, srcStep, pDst, dstStep, roiSize, 0xFF);
if (status == ippStsNoErr) {
    printf("PASS:\n(%3d %3d %3d %3d), (%3d %3d %3d %3d)\n", pDst[0], pDst[1], pDst[2],
            pDst[3], pDst[4], pDst[5], pDst[6], pDst[7]);
} else
    printf("FAIL: status = %d\n", status);
```

### Result:

**PASS:**

(113 113 113 255), (113 113 113 255)

### See Also

- **Regions of Interest in Intel IPP**
- **RGBToGray**

**Converts an RGB image to gray scale using fixed transform coefficients.**

### Syntax

```c
IppStatus ippiRGBToGray_<mod>(const Ipp<datatype>* pSrc, int srcStep, Ipp<datatype>* pDst, int dstStep, IppiSize roiSize);
```

### Supported values for *mod*

<table>
<thead>
<tr>
<th>8u_C3C1R</th>
<th>16u_C3C1R</th>
<th>16s_C3C1R</th>
<th>32f_C3C1R</th>
</tr>
</thead>
<tbody>
<tr>
<td>8u_AC4C1R</td>
<td>16u_AC4C1R</td>
<td>16s_AC4C1R</td>
<td>32f_AC4C1R</td>
</tr>
</tbody>
</table>

### Include Files

`ippcc.h`

### Domain Dependencies

**Headers**: `ippcore.h`, `ippvm.h`, `ipps.h`, `ippi.h`

**Libraries**: `ippcore.lib`, `ippvm.lib`, `ipps.lib`, `ippi.lib`
Parameters

- **pSrc**: Pointer to the source image ROI.
- **srcStep**: Distance in bytes between starts of consecutive lines in the source image.
- **pDst**: Pointer to the destination image ROI.
- **dstStep**: Distance in bytes between starts of consecutive lines in the destination image.
- **roiSize**: Size of the source and destination ROI in pixels.

Description

This function operates with ROI (see Regions of Interest in Intel IPP).

Conversion from RGB image to gray scale (see figure Converting an RGB Image to Gray Scale) uses the following basic equation to compute luma from nonlinear gamma-corrected red, green, and blue values: $Y' = 0.299 \times R' + 0.587 \times G' + 0.114 \times B'$. Note that the transform coefficients conform to the standard for the NTSC red, green, and blue CRT phosphors.

**Converting an RGB Image to Gray Scale**

Return Values

- **ippStsNoErr**: Indicates no error. Any other value indicates an error.
- **ippStsNullPtrErr**: Indicates an error condition if *pSrc* or *pDst* pointer is NULL.
- **ippStsSizeErr**: Indicates an error condition if *roiSize* has a field with a zero or negative value.

Example

The code example below demonstrates how to use the function *ippiRGBToGray_8u_C3C1R*.

```c
Ipp8u src[12*3] = { 255, 0, 0, 255, 0, 0, 255, 0, 0, 255, 0, 0,
                    0, 255, 0, 0, 255, 0, 0, 255, 0, 0, 255, 0, 0,
                    0, 0, 255, 0, 0, 255, 0, 0, 255, 0, 0, 255, 0,
                    0, 0, 255, 0, 0, 255, 0, 0, 255, 0, 0, 255,
                    0, 0, 255, 0, 0, 255, 0, 0, 255, 0, 0, 255};
Ipp8u dst[4*3];
IppiSize srcRoi = { 4, 3 };
ippiRGBToGray_8u_C3C1R ( src, 12, dst, 4, srcRoi );
```

Result:

```
255 0 0 255 0 0 255 0 0 255 0 0
0 255 0 0 255 0 0 255 0 0 255 0 0 src
0 0 255 0 0 255 0 0 255 0 0 255
```
ColorToGray
Converting an RGB image to gray scale using custom transform coefficients.

Syntax

IppStatus ippiColorToGray\_mod(const Ipp\_datatype* pSrc, int srcStep, Ipp\_datatype* pDst, int dstStep, IppiSize roiSize, const Ipp32f coeffs[3]);

Supported values for mod:

- 8u\_C3C1R
- 16u\_C3C1R
- 16s\_C3C1R
- 32f\_C3C1R

IppStatus ippiColorToGray\_mod(const Ipp\_datatype* pSrc, int srcStep, Ipp\_datatype* pDst, int dstStep, IppiSize roiSize, const Ipp64f coeffs[3]);

Supported values for mod:

- 64f\_C3C1R
- 64f\_AC4C1R

Include Files

ippcc.h

Domain Dependencies

Headers: ippcore.h, ippvm.h, ipps.h, ippi.h
Libraries: ippcore.lib, ippvm.lib, ipps.lib, ippi.lib

Parameters

- pSrc
  Pointer to the source image ROI.
- srcStep
  Distance in bytes between starts of consecutive lines in the source image.
- pDst
  Pointer to the destination image ROI.
- dstStep
  Distance in bytes between starts of consecutive lines in the destination image.
- roiSize
  Size of the source and destination ROI in pixels.
- coeffs
  Transform coefficients.

Description

This function operates with ROI (see Regions of Interest in Intel IPP).

This function uses the following equation to convert an RGB image to gray scale:

\[ Y = coeffs[0] \times R + coeffs[1] \times G + coeffs[2] \times B, \]
where the *coeffs* array contains user-defined transform coefficients which must be non-negative and satisfy the condition

\[ coeffs[0] + coeffs[1] + coeffs[2] \leq 1. \]

**Return Values**

- **ippStsNoErr**: Indicates no error. Any other value indicates an error.
- **ippStsNullPtrErr**: Indicates an error condition if *pSrc* or *pDst* is NULL.
- **ippStsSizeErr**: Indicates an error condition if *roiSize* has a field with a zero or negative value.

**CFAToBGRA**

Restores the RGB image from the gray-scale CFA image using the VNG algorithm.

**Syntax**

```c
IppStatus ippiCFAToBGRA_VNG_8u_C1C4R_L(const Ipp8u* pSrc, IppiRectL srcRoiL, IppSizeL srcSizeL, IppSizeL srcStepL, Ipp32f scale[4], Ipp8u* pDst, IppSizeL dstStepL, IppiBayerGrid grid);
IppStatus ippiCFAToBGRA_VNG_16u_C1C4R_L(const Ipp16u* pSrc, IppiRectL srcRoiL, IppSizeL srcSizeL, IppSizeL srcStepL, Ipp32f scale[4], Ipp16u* pDst, IppSizeL dstStepL, IppiBayerGrid grid);
```

**Include Files**

ippcc_l.h

**Domain Dependencies**

**Headers**: ippcore.h, ippvm.h, ipps.h, ippi.h

**Libraries**: ippcore.lib, ippvm.lib, ipps.lib, ippi.lib

**Parameters**

- **pSrc**: Pointer to the source image.
- **srcRoiL**: Region of interest in the source image (of the IppRectL type).
- **srcSizeL**: Size of the source image.
- **srcStepL**: Distance, in bytes, between the starting points of consecutive lines in the source image.
- **scale[4]**: Coefficients by which the resulting RGB channels are multiplied after interpolation. Be default, equal to 1.0.
- **pDst**: Pointer to the destination image.
- **dstStepL**: Distance, in bytes, between the starting points of consecutive lines in the destination image.
- **grid**: Specifies the configuration of the Bayer grid in the source image. The function copies 2-pixel width border pixels from the internal neighborhood pixels. The following values are possible:

  ippiBayerBGGR
ippiBayerRGGB
ippiBayerGBRG
ippiBayerGRBG

Description
This function operates with ROI (see Regions of Interest in Intel IPP).
This function transforms the one-channel gray-scale image \( p_{Src} \) that is produced by applying the color filter array (CFA) to 24-bit three-channel RGB image using the Variable Number of Gradients (VNG) demosaicing algorithm.

Return Values
- ippStsNoErr: Indicates no error. Any other value indicates an error.
- ippStsNullPtrErr: Indicates an error condition if one of the specified pointers is NULL.
- ippStsSizeErr: Indicates an error condition if the \( srcSizeL \) has a field that is less than 5, or if the \( roiSizeL \) has a field with a negative or zero value.
- ippStsBadArgErr: Indicates an error condition if \( grid \) has an illegal value.

CFAToRGB
Restores the RGB image from the gray-scale CFA image.

Syntax
IppStatus ippiCFAToRGB_8u_C1C3R(const Ipp8u* \( p_{Src} \), IppiRect \( srcRoi \), IppiSize \( srcSize \), int \( srcStep \), Ipp8u* \( p_{Dst} \), int \( dstStep \), IppiBayerGrid \( grid \), int \( interpolation \));
IppStatus ippiCFAToRGB_16u_C1C3R(const Ipp16u* \( p_{Src} \), IppiRect \( srcRoi \), IppiSize \( srcSize \), int \( srcStep \), Ipp16u* \( p_{Dst} \), int \( dstStep \), IppiBayerGrid \( grid \), int \( interpolation \));

Include Files
ippcc.h

Domain Dependencies
Headers: ippcore.h, ippvm.h, ipps.h, ippi.h
Libraries: ippcore.lib, ippvm.lib, ipps.lib, ippi.lib

Parameters
- \( p_{Src} \): Pointer to the source image origin.
- \( srcSize \): Size of the source image.
- \( srcRoi \): Region of interest in the source image (of the IppiRect type).
- \( srcStep \): Distance in bytes between starts of consecutive lines in the source image.
- \( p_{Dst} \): Pointer to the destination image.
Distance in bytes between starts of consecutive lines in the destination image.

Specifies the configuration of the Bayer grid in the source image. The following values are possible:

- ippiBayerBGGR
- ippiBayerRGGB
- ippiBayerGBRG
- ippiBayerGRBG

Interpolation method, reserved, must be 0.

Description

This function operates with ROI (see Regions of Interest in Intel IPP).

This function transforms the one-channel gray-scale image `pSrc` that is produced by applying the color filter array (CFA) - an array of Bayer filters, to 24-bit three-channel RGB image. The order of the color component in the source image - Bayer grid - is specified by the parameter `grid`. Four possible values of this parameter correspond to the allowed variants of the Bayer grid (see Figure below).

**Possible Configurations of the Bayer Grids**

![Possible Configurations of the Bayer Grids](image)

Each element of the source image contains an intensity value for only one color component, two others are interpolated using neighbor elements. R and B values are interpolated linearly from the nearest neighbors of the same color. When interpolating R and B values on green pixel, the average values of the two nearest neighbors (above and below, or left and right) of the same colors are used. When interpolating R or B values on the blue or red pixel respectively, the average values of the four nearest blue (red) pixels cornering the red (blue) pixel are used. G values are interpolated using an adaptive interpolation [Sak98] from a pair of nearest neighbors (vertical or horizontal) and taking into account the correlation in the red (or blue) component. The pair is chosen depending on the values of the difference between the red (blue) pixels in the vertical and horizontal directions. If the difference is smaller in the vertical direction - a vertical pair of green pixels is used, if it is smaller in the horizontal direction - a horizontal pair is used. If the difference is the same, all four neighbors are used.

This interpolation requires border pixels for the input pixels near the horizontal or vertical edge of the image. The function uses the mirrored border of two edge rows or columns of the input image. In this case the G values is calculated as the average of four nearest green pixels.

Return Values

- ippStsNoErr: Indicates no error. Any other value indicates an error.
- ippStsNullPtrErr: Indicates an error condition if one of the specified pointer is NULL.
ippStsSizeErr
Indicates an error condition if the srcSize has a field that is less than 2, or if the roiSize has a field with negative or zero value.

ippStsBadArgErr
Indicates an error condition if grid has an illegal value.

DemosaicAHD
Restores the RGB image from the gray-scale CFA image using AHD algorithm.

Syntax
IppStatus ippiDemosaicAHD_8u_C1C3R(const Ipp8u* pSrc, IppiRect srcRoi, IppiSize srcSize, int srcStep, Ipp8u* pDst, int dstStep, IppiBayerGrid grid, Ipp8u* pTmp, int tmpStep);
IppStatus ippiDemosaicAHD_16u_C1C3R(const Ipp16u* pSrc, IppiRect srcRoi, IppiSize srcSize, int srcStep, Ipp16u* pDst, int dstStep, IppiBayerGrid grid, Ipp16u* pTmp, int tmpStep);

Include Files
ippcc.h

Domain Dependencies
Headers: ippcore.h, ippvm.h, ipps.h,ippi.h
Libraries: ippcore.lib, ippvm.lib, ipps.lib,ippi.lib

Parameters
pSrc
Pointer to the source image.
srcRoi
Region of interest in the source image (of the IppiRect type).
srcSize
Size of the source image.
srcStep
Distance in bytes between starts of consecutive lines in the source image.
pDst
Pointer to the destination image.
dstStep
Distance in bytes between starts of consecutive lines in the destination image.
grid
Specifies the configuration of the Bayer grid in the source image. The following values are possible (see Figure Possible Configurations of the Bayer Grids):
ippiBayerBGGR
ippiBayerRGGB
ippiBayerGBRG
ippiBayerGRBG
pTmp
Pointer to the temporary image of (srcRoi.width + 6, 30) size.
tmpStep
Distance in bytes between starts of consecutive lines in the temporary image.
Description

This function operates with ROI (see Regions of Interest in Intel IPP).

This function transforms the one-channel gray-scale image \( p_{Src} \) that is produced by applying the color filter array (CFA) to 24-bit three-channel RGB image using the adaptive homogeneity-directed demosaicing (AHD) algorithm \cite{Hir05}. The algorithm requires the temporary image \( p_{Tmp} \) of size \( srcRoi.width + 6.30 \).

The type of the Bayer grid (see Figure Possible Configurations of the Bayer Grids) is specified by the parameter \( grid \).

Return Values

- \texttt{ippStsNoErr} Indicates no error. Any other value indicates an error.
- \texttt{ippStsNullPtrErr} Indicates an error condition if one of the specified pointer is NULL.
- \texttt{ippStsSizeErr} Indicates an error condition if the \( srcSize \) has a field that is less than 5, or if the \( roiSize \) has a field with negative or zero value.
- \texttt{ippStsBadArgErr} Indicates an error condition if \( grid \) has an illegal value.

Format Conversion

This section describes Intel IPP functions that perform image color conversion without changing the color space. These functions convert pixel-order images to planar format and vice versa, change the number of channels or planes, alter sampling formats and sequences of samples and planes. Several functions additionally perform filtering - deinterlacing and upsampling.

Intel IPP format conversion functions are specified mainly in the YCbCr color space, but as they do not transform color model they may be used to perform described types of conversion for any other color spaces with decoupled luminance and chrominance coordinates (YUV type).

YCbCr422

\textit{Converts 4:2:2 YCbCr image.}

Syntax

\begin{verbatim}
IppStatus ippiYCbCr422_8u_C2P3R(const Ipp8u* pSrc, int srcStep, Ipp8u* pDst[3], int dstStep[3], IppiSize roiSize);
IppStatus ippiYCbCr422_8u_P3C2R(const Ipp8u* pSrc[3], int srcStep[3], Ipp8u* pDst, int dstStep, IppiSize roiSize);
\end{verbatim}

Include Files

ippcc.h

Domain Dependencies

Headers: ippcore.h, ippvm.h, ipps.h,ippi.h
Libraries: ippcore.lib, ippvm.lib, ipps.lib,ippi.lib
Parameters

pSrc
Pointer to the ROI in the pixel-order source image. Array of pointers to the ROI in each plane of the planar source image.

srcStep
Distance in bytes between starts of consecutive lines in the source image. Array of distance values for the source image planes.

pDst
Pointer to the ROI in the pixel-order destination image. Array of pointers to the ROI in each plane of the planar destination image.

dstStep
Distance in bytes between starts of consecutive lines in the destination image. Array of distance values for the destination image planes.

roiSize
Size of the source and destination ROI in pixels.

Description

This function operates with ROI (see Regions of Interest in Intel IPP).

This function converts the 4:2:2 two-channel source image pSrc to the 4:2:2 three-plane image pDst and vice versa (see Table "Pixel-Order Image Formats" and Table "Planar Image Formats" for more details on 4:2:2 planar and pixel-order formats).

Return Values

ippStsNoErr
Indicates no error. Any other value indicates an error.

ippStsNullPtrErr
Indicates an error condition if any of the specified pointers is NULL.

ippStsSizeErr
Indicates an error condition if roiSize.width of the first plane is less than 2, or roiSize.height is less than or equal to zero.

YCbCr422ToYCrCb422

Converts 4:2:2 YCbCr image to 4:2:2 YCrCb image.

Syntax

IppStatus ippiYCbCr422ToYCrCb422_8u_C2R(const Ipp8u* pSrc, int srcStep, Ipp8u* pDst, int dstStep, IppiSize roiSize);

IppStatus ippiYCbCr422ToYCrCb422_8u_P3C2R(const Ipp8u* pSrc[3], int srcStep[3], Ipp8u* pDst, int dstStep, IppiSize roiSize);

Include Files

ippcc.h

Domain Dependencies

Headers: ippcore.h, ippvm.h, ipps.h,ippi.h

Libraries: ippcore.lib, ippvm.lib, ipps.lib,ippi.lib

Parameters

pSrc
Pointer to the ROI in the pixel-order source image. Array of pointers to the ROI in each plane of the planar source image.
**Description**

This function operates with ROI (see Regions of Interest in Intel IPP).

This function converts 4:2:2 YCbCr source image pSrc to the 4:2:2 YCrCb two-channel image pDst that has the following sequence of samples: Y0, Cr0, Y1, Cb0, Y2, Cr1, Y3, Cb1, ... (see Table "Pixel-Order Image Formats"). The source image can be either two-channel or three-plane (see Table "Pixel-Order Image Formats" and Table "Planar Image Formats").

**Return Values**

ippStsNoErr Indicates no error. Any other value indicates an error.

ippStsNullPtrErr Indicates an error condition if any of the specified pointers is NULL.

ippStsSizeErr Indicates an error condition if roiSize.width is less than 2.

**YCbCr422ToCbYCr422**

*Converts 4:2:2 YCbCr image to 4:2:2 CbYCr image.*

**Syntax**

IppStatus ippiYCbCr422ToCbYCr422_8u_C2R(const Ipp8u* pSrc, int srcStep, Ipp8u* pDst, int dstStep, IppiSize roiSize);

IppStatus ippiYCbCr422ToCbYCr422_8u_P3C2R(const Ipp8u* pSrc[3], int srcStep[3], Ipp8u* pDst, int dstStep, IppiSize roiSize);

**Include Files**

ippcc.h

**Domain Dependencies**

Headers: ippcore.h, ippvm.h, ipps.h,ippi.h

Libraries: ippcore.lib, ippvm.lib, ipps.lib,ippi.lib

**Parameters**

- **pSrc**
  
  Pointer to the ROI in the pixel-order source image. Array of pointers to the ROI in each plane of the planar source image.

- **srcStep**
  
  Distance in bytes between starts of consecutive lines in the source image. Array of distance values for the source image planes.

- **pDst**
  
  Pointer to the destination image ROI.

- **dstStep**
  
  Distance in bytes between starts of consecutive lines in the destination image.
**roiSize**

Size of the source and destination ROI in pixels; its width should be multiple of 2.

**Description**

This function operates with ROI (see Regions of Interest in Intel IPP).

This function converts \(4:2:2\) YCbCr source image \(pSrc\) to the \(4:2:0\) CbYCr two-channel image \(pDst\) that has the following sequence of samples: \(Cb0, Y0, Cr0, Y1, Cb1, Y2, Cr1, Y3, Cb3, ...\) (see Table "Pixel-Order Image Formats"). The source image can be either two-channel or three-plane (see Table "Pixel-Order Image Formats" and Table "Planar Image Formats").

**Return Values**

- **ippStsNoErr** Indicates no error. Any other value indicates an error.
- **ippStsNullPtrErr** Indicates an error condition if any of the specified pointers is NULL.
- **ippStsSizeErr** Indicates an error condition if roiSize.width of the first plane is less than 2, or roiSize.height is less than or equal to zero.

**YCbCr422ToYCbCr420**

Converts YCbCr image from \(4:2:2\) sampling format to \(4:2:0\) format.

**Syntax**

**Case 1: Operation on planar data**

```c
IppStatus ippiYCbCr422ToYCbCr420_8u_P3R(const Ipp8u* pSrc[3], int srcStep[3], Ipp8u* pDst[3], int dstStep[3], IppiSize roiSize);
IppStatus ippiYCbCr422ToYCbCr420_8u_P3P2R(const Ipp8u* pSrc[3], int srcStep[3], Ipp8u* pDstY, int dstYStep, Ipp8u* pDstCbCr, int dstCbCrStep, IppiSize roiSize);
```

**Case 2: Conversion from pixel-order to planar data**

```c
IppStatus ippiYCbCr422ToYCbCr420_8u_C2P3R(const Ipp8u* pSrc, int srcStep, Ipp8u* pDst[3], int dstStep[3], IppiSize roiSize);
IppStatus ippiYCbCr422ToYCbCr420_8u_C2P2R(const Ipp8u* pSrc, int srcStep, Ipp8u* pDstY, int dstYStep, Ipp8u* pDstCbCr, int dstCbCrStep, IppiSize roiSize);
```

**Include Files**

ippcc.h

**Domain Dependencies**

Headers: ippcore.h, ippvm.h, ipps.h,ippi.h

Libraries: ippcore.lib, ippvm.lib, ipps.lib,ippi.lib

**Parameters**

- **pSrc** Pointer to the ROI in the pixel-order source image. Array of pointers to the ROI in each plane of the planar source image.
- **srcStep** Distance in bytes between starts of consecutive lines in the source image. Array of distance values for the source image planes.
**Description**

This function operates with ROI (see Regions of Interest in Intel IPP).

This function converts the 4:2:2 image \( pSrc \) to the 4:2:0 image. The source image can be two-channel or three-plane, destination image always is planar with two or three planes (see Table "Pixel-Order Image Formats" and Table "Planar Image Formats"). Two-plane image contains luminance samples \( Y_0, Y_1, Y_2, \ldots \) in the first plane \( pDstY \), and interleaved chrominance samples \( Cb_0, Cr_0, Cb_1, Cr_1, \ldots \) in the second plane \( pDstCbCr \).

**Return Values**

- ippStsNoErr: Indicates no error. Any other value indicates an error.
- ippStsNullPtrErr: Indicates an error condition if any of the specified pointers is NULL.
- ippStsSizeErr: Indicates an error condition if any field of the \( roiSize \) is less than 2.

**Example**

The code example below shows how to use the function \( ippiYCbCr422ToYCbCr420_8u_C2P3R \).

```c
{
    Ipp8u*   ImageI420[3];
    int      stepI420[3];
    Ipp8u*   ImageYUY2;
    int      stepYUY2;
    IppiSize roiSize = { 1024, 768};
    ImageI420[0] = ippiMalloc_8u_C1( roiSize.width, roiSize.height, &stepI420[0]);
    ImageI420[1] = ippiMalloc_8u_C1( roiSize.width, roiSize.height, &stepI420[1]);
    ImageYUY2 = ippiMalloc_8u_C2( roiSize.width, roiSize.height, &stepYUY2);
    ippiYCbCr422ToYCbCr420_8u_C2P3R( ImageYUY2, stepYUY2, ImageI420, stepI420, roiSize);

    ippiFree(ImageI420[0]);
    ippiFree(ImageI420[1]);
    ippiFree(ImageI420[2]);
    ippiFree(ImageYUY2);
}
```
**YCbCr422To420_Interlace**  
*Converts interlaced YCbCr image from 4:2:2 sampling format to 4:2:0 format.*

**Syntax**

```c
IppStatus ippiYCbCr422To420_Interlace_8u_P3R(const Ipp8u* pSrc[3], int srcStep[3], Ipp8u* pDst[3], int dstStep[3], IppiSize roiSize);
```

**Include Files**

ippcc.h

**Domain Dependencies**

**Headers:** ippcore.h, ippvm.h, ipps.h,ippi.h  
**Libraries:** ippcore.lib, ippvm.lib, ipps.lib,ippi.lib

**Parameters**

- **pSrc**  
  Array of pointers to the ROI in each plane for source image.

- **srcStep**  
  Array of distances in bytes between starts of consecutive lines in each plane for the source image.

- **pDst**  
  Array of pointers to the ROI in each plane for destination image.

- **dstStep**  
  Array of distances in bytes between starts of consecutive lines in each plane for destination image.

- **roiSize**  
  Size of the source and destination ROI in pixels, its width must be multiple of 2, and height must be multiple of 4.

**Description**

This function operates with ROI (see Regions of Interest in Intel IPP).  
This function converts the interlaced 4:2:2 image `pSrc` to the 4:2:0 image `pDst` (see Table "Planar Image Formats").

The conversion is performed in accordance with the following formulas:

\[
Y_{1_{\text{dest}}} = Y_{1_{\text{src}}},
\]

\[
Cb_{0(Cr0)}_{\text{dest}} = \frac{3 \times Cb_{0(Cr0)}_{\text{src}} + Cb_{2(Cr2)}_{\text{src}} + 2}{4};
\]

\[
Cb_{1(Cr1)}_{\text{dest}} = \frac{Cb_{1(Cr1)}_{\text{src}} + 3 \times Cb_{3(Cr3)}_{\text{src}} + 2}{4};
\]

**Return Values**

- **ippStsNoErr**  
  Indicates no error. Any other value indicates an error.

- **ippStsNullPtrErr**  
  Indicates an error condition if any of the specified pointers is NULL.

- **ippStsSizeErr**  
  Indicates an error condition if `roiSize.width` is less than 2, or `roiSize.height` is less than 4.

- **ippStsDoubleSize**  
  Indicates a warning if `roiSize.width` is not multiple of 2, or `roiSize.height` is not multiple of 4.
**YCbCr422ToYCrCb420**  
*Converts 4:2:2 YCbCr image to 4:2:0 YCrCb image.*

**Syntax**

IppStatus ippiYCbCr422ToYCrCb420_8u_C2P3R(const Ipp8u* pSrc, int srcStep, Ipp8u* pDst[3], int dstStep[3], IppiSize roiSize);

IppStatus ippiYCbCr422ToYCrCb420_8u_C2P2R(const Ipp8u* pSrc, int srcStep, Ipp8u* pDstY, int dstYStep, Ipp8u* pDstCrCb, int dstUVStep, IppiSize roiSize);

**Include Files**

ippcc.h

**Domain Dependencies**

Headers: ippcore.h, ippvm.h, ipps.h,ippi.h

Libraries: ippcore.lib, ippvm.lib, ipps.lib,ippi.lib

**Parameters**

- **pSrc**  
  Pointer to the source image ROI.

- **srcStep**  
  Distance in bytes between starts of consecutive lines in the source image.

- **pDst**  
  Array of pointers to the ROI in each plane of the destination image.

- **pDstY**  
  Pointer to the destination image Y plane.

- **dstStep**  
  Array of distances, in bytes, between the starting points of consecutive lines in the destination image planes.

- **dstYStep**  
  Array of distances, in bytes, between the starting points of consecutive lines in the destination image Y plane.

- **dstUVStep**  
  Array of distances, in bytes, between the starting points of consecutive lines in the destination image UV plane.

- **roiSize**  
  Size of the source and destination ROI in pixels, height and width should be multiple of 2.

**Description**

This function operates with ROI (see Regions of Interest in Intel IPP).

This function converts the 4:2:2 two-channel image *pSrc* that has the following sequence of samples: Y0, Cb0, Y1, Cr0, Y2, Cb1, Y3, Cr1, ... to the 4:2:0 three-plane image *pDst* with the following order of pointers: Y-plane, Cr-plane, Cb-plane (see Table “Pixel-Order Image Formats” and Table “Planar Image Formats”).

**Return Values**

- **ippStsNoErr**  
  Indicates no error. Any other value indicates an error.

- **ippStsNullPtrErr**  
  Indicates an error condition if any of the specified pointers is NULL.

- **ippStsSizeErr**  
  Indicates an error condition if any field of the *roiSize* is less than 2.
YCbCr422ToYCbCr411
Converts YCbCr image from 4:2:2 sampling format to 4:1:1 format.

Syntax

Case 1: Operation on planar data
IppStatus ippiYCbCr422ToYCbCr411_8u_P3R(const Ipp8u* pSrc[3], int srcStep[3], Ipp8u* pDst[3], int dstStep[3], IppiSize roiSize);
IppStatus ippiYCbCr422ToYCbCr411_8u_P3P2R(const Ipp8u* pSrc[3], int srcStep[3], Ipp8u* pDstY, int dstYStep, Ipp8u* pDstCbCr, int dstCbCrStep, IppiSize roiSize);

Case 2: Conversion from pixel-order to planar data
IppStatus ippiYCbCr422ToYCbCr411_8u_C2P3R(const Ipp8u* pSrc, int srcStep, Ipp8u* pDst[3], int dstStep[3], IppiSize roiSize);
IppStatus ippiYCbCr422ToYCbCr411_8u_C2P2R(const Ipp8u* pSrc, int srcStep, Ipp8u* pDstY, int dstYStep, Ipp8u* pDstCbCr, int dstCbCrStep, IppiSize roiSize);

Include Files
ippcc.h

Domain Dependencies
Headers: ippcore.h, ippvm.h, ipps.h, ippi.h
Libraries: ippcore.lib, ippvm.lib, ipps.lib, ippi.lib

Parameters

pSrc
Pointer to the ROI in the pixel-order source image. Array of pointers to the ROI in each plane of the planar source image.

srcStep
Distance in bytes between starts of consecutive lines in the source image. Array of distance values for the source image planes.

pDst
Array of pointers to the ROI in each plane for a three-plane destination image.

dstStep
Array of distances in bytes between starts of consecutive lines in each plane for a three-plane destination image.

pDstY
Pointer to the ROI in the luminance plane for a two-plane destination image.

dstYStep
Distance in bytes between starts of consecutive lines in the luminance plane of a destination image.

pDstCbCr
Pointer to the ROI in the interleaved chrominance plane for a two-plane destination image.

dstCbCrStep
Distance in bytes between starts of consecutive lines in the chrominance plane of a destination image.

roiSize
Size of the ROI in pixels.

Description
This function operates with ROI (see Regions of Interest in Intel IPP).
This function converts the 4:2:2 image \( p\text{Src} \) to the 4:1:1 image. The source image can be two-channel or three-plane (see Table "Pixel-Order Image Formats" for more details), destination image always is planar with two or three planes (see Table "Planar Image Formats" for more details). The two-plane image contains luminance samples \( Y0, Y1, Y2, \ldots \) in the first plane \( p\text{Dst}Y \), and interleaved chrominance samples \( \text{Cb}0, \text{Cr}0, \text{Cb}1, \text{Cr}1, \ldots \) in the second plane \( p\text{DstCbCr} \).

The value of the fields of the \( \text{roiSize} \) have certain limitations:
- its width should be multiple of 4 and cannot be less than 4 for operation on two-channel images;
- its width should be multiple of 4 and cannot be less than 4, and its height should be multiple of 2 and cannot be less than 2 for three-plane to two-plane image conversion;
- both height and width should be multiple of 2 and cannot be less than 2 for operation on three-plane images.

**Return Values**

- \( \text{ippStsNoErr} \): Indicates no error. Any other value indicates an error.
- \( \text{ippStsNullPtrErr} \): Indicates an error condition if any of the specified pointers is NULL.
- \( \text{ippStsSizeErr} \): Indicates an error condition if corresponding fields of the \( \text{roiSize} \) is less than specified above values.

**YCrCb422ToYCbCr422**

**Converts 4:2:2 YCrCb image to 4:2:2 YCbCr image.**

**Syntax**

\[
\text{IppStatus ippiYCrCb422ToYCbCr422\_8u\_C2P3R(const Ipp8u* } p\text{Src, int } src\text{Step, Ipp8u* } p\text{Dst}[3], \text{int } dst\text{Step}[3], \text{IppiSize } roi\text{Size});
\]

**Include Files**

ippcc.h

**Domain Dependencies**

Headers: ippcore.h, ippvm.h, ipps.h,ippi.h

Libraries: ippcore.lib, ippvm.lib, ipps.lib,ippi.lib

**Parameters**

- \( p\text{Src} \): Pointer to the source image ROI.
- \( src\text{Step} \): Distance in bytes between starts of consecutive lines in the source image.
- \( p\text{Dst} \): Array of pointers to the ROI in each plane of the destination image.
- \( dst\text{Step} \): Array of distances in bytes between starts of consecutive lines in the destination image planes.
- \( roi\text{Size} \): Size of the ROI in pixels, its width should be multiple of 2.

**Description**

This function operates with ROI (see Regions of Interest in Intel IPP).
This function converts the 4:2:2\text{YCrCb} two-channel image \textit{pSrc} (see Table “Pixel-Order Image Formats”) to the 4:2:0\text{YCbCr} three-plane image \textit{pDst} (see Table “Planar Image Formats”).

Return Values

- \textit{ippStsNoErr}
  - Indicates no error. Any other value indicates an error.
- \textit{ippStsNullPtrErr}
  - Indicates an error condition if any of the specified pointers is NULL.
- \textit{ippStsSizeErr}
  - Indicates an error condition if \textit{roiSize.width} is less than 2.

\textbf{YCrCb422ToYCbCr420}

\textit{Converrs 4:2:2 \textit{YCrCb} image to 4:2:0 \textit{YCbCr} image.}

Syntax

\begin{verbatim}
IppStatus ippiYCrCb422ToYCbCr420_8u_C2P3R(const Ipp8u* pSrc, int srcStep, ipp8u* pDst[3], int dstStep[3], IppiSize roiSize);
\end{verbatim}

Include Files

ippcc.h

Domain Dependencies

Headers: ippcore.h, ippvm.h, ipps.h,ippi.h
Libraries: ippcore.lib, ippvm.lib, ipps.lib,ippi.lib

Parameters

\begin{itemize}
  \item \textit{pSrc}
  \begin{itemize}
    \item Pointer to the source image ROI.
  \end{itemize}
  \item \textit{srcStep}
  \begin{itemize}
    \item Distance in bytes between starts of consecutive lines in the source image.
  \end{itemize}
  \item \textit{pDst}
  \begin{itemize}
    \item Array of pointers to the ROI in each plane of the destination image.
  \end{itemize}
  \item \textit{dstStep}
  \begin{itemize}
    \item Array of distances in bytes between starts of consecutive lines in the destination image planes.
  \end{itemize}
  \item \textit{roiSize}
  \begin{itemize}
    \item Size of the ROI in pixels, height and width should be multiple of 2.
  \end{itemize}
\end{itemize}

Description

This function operates with ROI (see Regions of Interest in Intel IPP).

This function converts the 4:2:2\text{YCrCb} two-channel image \textit{pSrc} (see Table “Pixel-Order Image Formats”) to the 4:2:0\text{YCbCr} three-plane image \textit{pDst} (see Table “Planar Image Formats”).

Return Values

- \textit{ippStsNoErr}
  - Indicates no error. Any other value indicates an error.
- \textit{ippStsNullPtrErr}
  - Indicates an error condition if any of the specified pointers is NULL.
- \textit{ippStsSizeErr}
  - Indicates an error condition if any field of the \textit{roiSize} is less than 2.
**YCrCb422ToYCbCr411**  
*Converts 4:2:2 YCrCb image to 4:1:1 YCbCr image.*

**Syntax**

```c
IppStatus ippiYCrCb422ToYCbCr411_8u_C2P3R(const Ipp8u* pSrc, int srcStep, Ipp8u* pDst[3], int dstStep[3], IppiSize roiSize);
```

**Include Files**

ippcc.h

**Domain Dependencies**

**Headers:** ippcore.h, ippvm.h, ipps.h,ippi.h  
**Libraries:** ippcore.lib, ippvm.lib, ipps.lib,ippi.lib

**Parameters**

- `pSrc`  
  Pointer to the source image ROI.
- `srcStep`  
  Distance in bytes between starts of consecutive lines in the source image.
- `pDst`  
  Array of pointers to the ROI in each plane of the destination image.
- `dstStep`  
  Array of distances in bytes between starts of consecutive lines in the destination image planes.
- `roiSize`  
  Size of the ROI in pixels, its width should be multiple of 4.

**Description**

This function operates with ROI (see Regions of Interest in Intel IPP).

This function converts the 4:2:2 YCrCb two-channel image `pSrc` (see Table "Pixel-Order Image Formats") to the 4:1:1 YCbCr three-plane image `pDst` (see Table "Planar Image Formats").

**Return Values**

- `ippStsNoErr`  
  Indicates no error. Any other value indicates an error.
- `ippStsNullPtrErr`  
  Indicates an error condition if any of the specified pointers is NULL.
- `ippStsSizeErr`  
  Indicates an error condition if `roiSize.width` is less than 4.

**CbYCr422ToYCbCr422**  
*Converts 4:2:2 CbYCr image to 4:2:2 YCbCr image.*

**Syntax**

```c
IppStatus ippiCbYCr422ToYCbCr422_8u_C2R(const Ipp8u* pSrc, int srcStep, Ipp8u* pDst, int dstStep, IppiSize roiSize);
IppStatus ippiCbYCr422ToYCbCr422_8u_C2P3R(const Ipp8u* pSrc, int srcStep, Ipp8u* pDst[3], int dstStep[3], IppiSize roiSize);
```
Include Files
ippcc.h

Domain Dependencies
Headers: ippcore.h, ippvm.h, ipps.h,ippi.h
Libraries: ippcore.lib, ippvm.lib, ipps.lib,ippi.lib

Parameters
pSrc
Pointer to the source image ROI.
srcStep
Distance in bytes between starts of consecutive lines in the source image.
pDst
Pointer to the ROI in the pixel-order destination image. Array of pointers to the ROI in each plane of the planar destination image.
dstStep
Distance in bytes between starts of consecutive lines in the destination image. Array of distance values for the destination image planes.
roiSize
Size of the ROI in pixels, its width should be multiple of 2.

Description
This function operates with ROI (see Regions of Interest in Intel IPP).
This function converts the 4:2:2CbYCr two-channel image pSrc to the 4:2:2 YCbCr two-channel or three-plane image pDst (see Table “Pixel-Order Image Formats” and Table “Planar Image Formats”). The source image has the following sequence of samples: Cb0, Y0, Cr0, Y1, Cb1, Y2, Cr1, Y3, Cb2, ... . Two-channel destination image has different sequence of samples: Y0, Cb0, Y1, Cr0, Y2, Cb1, Y3, Cr1, Y4, ....

Return Values
ppStsNoErr
Indicates no error. Any other value indicates an error.
ippStsNullPtrErr
Indicates an error condition if any of the specified pointers is NULL.
ippStsSizeErr
Indicates an error condition if roiSize.width is less than 2.

CbYCr422ToYCbCr420
Converts 4:2:2 CbYCr image to 4:2:0 YCbCr image.

Syntax
IppStatus ippiCbYCr422ToYCbCr420_8u_C2P3R(const Ipp8u* pSrc, int srcStep, Ipp8u* pDst[3], int dstStep[3], IppiSize roiSize);
IppStatus ippiCbYCr422ToYCbCr420_8u_C2P2R(const Ipp8u* pSrc, int srcStep, Ipp8u* pDstY, int dstYStep, Ipp8u* pDstCbCr, int dstCbCrStep, IppiSize roiSize);

Include Files
ippcc.h

Domain Dependencies
Headers: ippcore.h, ippvm.h, ipps.h,ippi.h
Libraries: ippcore.lib, ippvm.lib, ipp.lib,ippi.lib

Parameters

\[ pSrc \]  
Pointer to the source image ROI.

\[ srcStep \]  
Distance in bytes between starts of consecutive lines in the source image.

\[ pDst \]  
Array of pointers to the ROI in each plane for a three-plane destination image.

\[ dstStep \]  
Array of distances in bytes between starts of consecutive lines in each plane for a three-plane destination image.

\[ pDstY \]  
Pointer to the ROI in the luminance plane for a two-plane destination image.

\[ dstYStep \]  
Distance in bytes between starts of consecutive lines in the luminance plane of a destination image.

\[ pDstCbCr \]  
Pointer to the ROI in the interleaved chrominance plane for a two-plane destination image.

\[ dstCbCrStep \]  
Distance in bytes between starts of consecutive lines in the chrominance plane of a destination image.

\[ roiSize \]  
Size of the source and destination ROI in pixels, height and width should be multiple of 2.

Description

This function operates with ROI (see Regions of Interest in Intel IPP).

This function converts the 4:2:2 CbYCr two-channel image \( pSrc \) to the 4:2:0 YCbCr two- or three-plane image \( pDst \) (see Table "Pixel-Order Image Formats" and Table "Planar Image Formats"). The source image has the following sequence of samples: \( Cb_0, Y_0, Cr_0, Y_1, Cb_1, Y_2, Cr_1, Y_3, Cb_2, ... \). Three-plane destination image has the following order of pointers: \( Y \)-plane, \( Cb \)-plane, \( Cr \)-plane. Two-plane destination image contains luminance samples \( Y_0, Y_1, Y_2, ... \) in the first plane \( pDstY \), and interleaved chrominance samples \( Cb_0, Cr_0, Cb_1, Cr_1, ... \) in the second plane \( pDstCbCr \).

Return Values

- \( ippStsNoErr \): Indicates no error. Any other value indicates an error.
- \( ippStsNullPtrErr \): Indicates an error condition if any of the specified pointers is NULL.
- \( ippStsSizeErr \): Indicates an error condition if any field of the \( roiSize \) is less than 2.

\textbf{CbYCr422ToYCbCr420_Interlace}

Converts interlaced 4:2:2 CbYCr image to 4:2:0 YCbCr image.

\textbf{Syntax}

\begin{verbatim}
IppStatus ippiCbYCr422ToYCbCr420_Interlace_8u_C2P3R(const Ipp8u* pSrc, int srcStep, Ipp8u* pDst[3], int dstStep[3], IppSize roiSize);
\end{verbatim}

\textbf{Include Files}

ippcc.h
Domain Dependencies

Headers: ippcore.h, ippvm.h, ipps.h,ippi.h
Libraries: ippcore.lib, ippvm.lib, ipps.lib,ippi.lib

Parameters

pSrc
Pointer to the source image ROI.

srcStep
Distance in bytes between starts of consecutive lines in the source image.

pDst
Array of pointers to the ROI in each plane for destination image.

dstStep
Array of distances in bytes between starts of consecutive lines in each plane for a three-plane destination image.

roiSize
Size of the source and destination ROI in pixels, its width must be multiple of 2, and height must be multiple of 4.

Description

This function operates with ROI (see Regions of Interest in Intel IPP).

This function converts interlaced 4:2:2CbYCr two-channel image pSrc to the 4:2:0YCbCr three-plane image pDst (see Table "Pixel-Order Image Formats" and Table "Planar Image Formats"). The source image has the following sequence of samples: Cb0, Y0, Cr0, Y1, Cb1, Y2, Cr1, Y3, Cb2, ... . Three-plane destination image has the following order of pointers: Y-plane, Cb-plane, Cr-plane.

The conversion is performed in accordance with the following formulas:

\[ Y_{1_{dest}} = Y_{1_{src}}; \]
\[ Cb0(Cr0)_{dest} = \frac{3*Cb0(Cr0)_{src} + Cb2(Cr2)_{src} + 2}{4}; \]
\[ Cb1(Cr1)_{dest} = \frac{Cb1(Cr1)_{src} + 3*Cb3(Cr3)_{src} + 2}{4}; \]

Return Values

ippStsNoErr
Indicates no error. Any other value indicates an error.

ippStsNullPtrErr
Indicates an error condition if any of the specified pointers is NULL.

ippStsSizeErr
Indicates an error condition if roiSize.width is less than 2, or roiSize.height is less than 4.

ippStsDoubleSize
Indicates a warning if roiSize.width is not multiple of 2, or roiSize.height is not multiple of 4.

CbYCr422ToYCrCb420

Converts 4:2:2 CbYCr image to 4:2:0 YCrCb image.

Syntax

IppStatus ippiCbYCr422ToYCrCb420_8u_C2P3R(const Ipp8u* pSrc, int srcStep, Ipp8u* pDst[3], int dstStep[3], IppiSize roiSize);

Include Files

ippcc.h
Domain Dependencies

Headers: ippcore.h, ippvm.h, ipps.h,ippi.h
Libraries: ippcore.lib, ippvm.lib, ipps.lib,ippi.lib

Parameters

- **pSrc**: Pointer to the source image ROI.
- **srcStep**: Distance in bytes between starts of consecutive lines in the source image.
- **pDst**: Array of pointers to the ROI in each plane of the destination image.
- **dstStep**: Array of distances in bytes between starts of consecutive lines in the destination image planes.
- **roiSize**: Size of the source and destination ROI in pixels, height and width should be multiple of 2.

Description

This function operates with ROI (see Regions of Interest in Intel IPP).

This function converts the 4:2:2CbYCr two-channel image pSrc to the 4:2:0YCrCb three-plane image pDst. The source image has the following sequence of samples: Cb0, Y0, Cr0, Y1, Cb1, Y2, Cr1, Y3, Cb2, ... . The destination image has the following order of pointers: Y-plane, Cr-plane, Cb-plane (see Table “Pixel-Order Image Formats” and Table “Planar Image Formats”).

Return Values

- **ippStsNoErr**: Indicates no error. Any other value indicates an error.
- **ippStsNullPtrErr**: Indicates an error condition if any of the specified pointers is NULL.
- **ippStsSizeErr**: Indicates an error condition if any field of the roiSize is less than 2.

__CbYCr422ToYCbCr411__

_Converts 4:2:2 CbYCr image to 4:1:1 YCbCr image._

Syntax

IppStatus ippiCbYCr422ToYCbCr411_8u_C2P3R(const Ipp8u* pSrc, int srcStep, Ipp8u* pDst[3], int dstStep[3], IppiSize roiSize);

Include Files

ippcc.h

Domain Dependencies

Headers: ippcore.h, ippvm.h, ipps.h,ippi.h
Libraries: ippcore.lib, ippvm.lib, ipps.lib,ippi.lib

Parameters

- **pSrc**: Pointer to the source image ROI.
srcStep
Distance in bytes between starts of consecutive lines in the source image.

pDst
Array of pointers to the ROI in each plane of the destination image.

dstStep
Array of distances in bytes between starts of consecutive lines in the destination image planes.

roiSize
Size of the ROI in pixels, its width should be multiple of 4.

Description
This function operates with ROI (see Regions of Interest in Intel IPP).

This function converts the 4:2:2CbYCr two-channel image pSrc to the 4:1:1YCbCr three-plane image pDst.
The source image has the following sequence of samples: Cb0, Y0, Cr0, Y1, Cb1, Y2, Cr1, Y3, Cb2, ... . The destination image has the following order of pointers: Y-plane, Cb-plane, Cr-plane (see Table "Pixel-Order Image Formats" and Table "Planar Image Formats").

Return Values
ippStsNoErr
Indicates no error. Any other value indicates an error.

ippStsNullPtrErr
Indicates an error condition if any of the specified pointers is NULL.

ippStsSizeErr
Indicates an error condition if roiSize.width is less than 4.

YCbCr420
Converts 4:2:0 YCbCr image.

Syntax
IppStatus ippiYCbCr420_8u_P3P2R(const Ipp8u* pSrc[3], int srcStep[3], Ipp8u* pDstY, int dstYStep, Ipp8u* pDstCbCr, int dstCbCrStep, IppiSize roiSize);

IppStatus ippiYCbCr420_8u_P2P3R(const Ipp8u* pSrcY, int srcYStep, const Ipp8u* pSrcCbCr, int srcCbCrStep, Ipp8u* pDst[3], int dstStep[3], IppiSize roiSize);

Include Files
ippcc.h

Domain Dependencies
Headers: ippcore.h, ippvm.h, ipps.h,ippi.h
Libraries: ippcore.lib, ippvm.lib, ipps.lib,ippi.lib

Parameters
pSrc
Array of pointers to the ROI in each plane for a three-plane source image.

srcStep
Array of distances in bytes between starts of consecutive lines in each plane for a three-plane source image.

pSrcY
Pointer to the ROI in the luminance plane for a two-plane source image.

srcYStep
Distance in bytes between starts of consecutive lines in the luminance plane of a source image.
pSrcCbCr

Pointer to the ROI in the interleaved chrominance plane for a two-plane source image.

csrcCrStep

Distance in bytes between starts of consecutive lines in the interleaved chrominance plane of the source image.

pDst

Array of pointers to the ROI in each plane for a three-plane destination image.

dstStep

Array of distances in bytes between starts of consecutive lines in each plane for a three-plane destination image.

pDstY

Pointer to the ROI in the luminance plane for a two-plane destination image.

dstYStep

Distance in bytes between starts of consecutive lines in the luminance plane of a destination image.

pDstCbCr

Pointer to the ROI in the interleaved chrominance plane for a two-plane destination image.

dstCbCrStep

Distance in bytes between starts of consecutive lines in the chrominance plane of a destination image.

roiSize

Size of the source and destination ROI in pixels, height and width should be multiple of 2.

Description

This function operates with ROI (see Regions of Interest in Intel IPP).

This function converts the 4:2:0 three-plane (see Table "Planar Image Formats") source image pSrc to the 4:2:0 two-plane image and vice versa. Two-plane image contains luminance samples Y0, Y1, Y2, .. in the first plane, and interleaved chrominance samples Cb0, Cr0, Cb1, Cr1,... in the second plane.

Return Values

IppStsNoErr

Indicates no error. Any other value indicates an error.

IppStsNullPtrErr

Indicates an error condition if any of the specified pointers is NULL.

IppStsSizeErr

Indicates an error condition if any field of the roiSize is less than 2.

YCbCr420ToYCbCr422

Converts YCbCr image from 4:2:0 sampling format to 4:2:2 format.

Syntax

IppStatus ippiYCbCr420ToYCbCr422_8u_P3R(const Ipp8u* pSrc[3], int srcStep[3], Ipp8u* pDst[3], int dstStep[3], IppiSize roiSize);

IppStatus ippiYCbCr420ToYCbCr422_8u_P2P3R(const Ipp8u* pSrcY, int srcYStep, const Ipp8u* pSrcCbCr, int srcCbCrStep, Ipp8u* pDst[3], int dstStep[3], IppiSize roiSize);

IppStatus ippiYCbCr420ToYCbCr422_8u_P2C2R(const Ipp8u* pSrcY, int srcYStep, const Ipp8u* pSrcCbCr, int srcCbCrStep, Ipp8u* pDst, int dstStep, IppiSize roiSize);

Include Files

ippcc.h
Domain Dependencies

Headers: ippcore.h, ippvm.h, ipps.h,ippi.h

Libraries: ippcore.lib, ippvm.lib, ipps.lib,ippi.lib

Parameters

\textbf{pSrc}  
Array of pointers to the ROI in each plane for a three-plane source image.

\textbf{srcStep}  
Array of distances in bytes between starts of consecutive lines in each plane for a three-plane source image.

\textbf{pSrcY}  
Pointer to the ROI in the luminance plane for a two-plane source image.

\textbf{srcYStep}  
Distance in bytes between starts of consecutive lines in the luminance plane of a source image.

\textbf{pSrcCbCr}  
Pointer to the ROI in the interleaved chrominance plane for a two-plane source image.

\textbf{srcCbCrStep}  
Distance in bytes between starts of consecutive lines in the interleaved chrominance plane of the source image.

\textbf{pDst}  
Pointer to the ROI in the pixel-order destination image. Array of pointers to the ROI in each plane of the planar destination image.

\textbf{dstStep}  
Distance in bytes between starts of consecutive lines in the destination image. Array of distance values for the destination image planes.

\textbf{roiSize}  
Size of the ROI in pixels, height and width should be multiple of 2.

Description

This function operates with ROI (see Regions of Interest in Intel IPP).

This function converts the 4:2:0 planar source image \textit{pSrc} to the 4:2:2 image \textit{pDst}. The source image can be two- or three-plane image (see Table “Planar Image Formats”). The first plane of the two-plane source image \textit{pSrcY} contains luminance samples \(Y_0, Y_1, Y_2, \ldots\), the second plane \textit{pSrcCbCr} contains interleaved chrominance samples \(C_{b0}, C_{r0}, C_{b1}, C_{r1}, \ldots\). The destination image \textit{pDst} can be three-plane or two-channel (see Table “Pixel-Order Image Formats” and Table “Planar Image Formats”).

Return Values

\textbf{ippStsNoErr}  
Indicates no error. Any other value indicates an error.

\textbf{ippStsNullPtrErr}  
Indicates an error condition if any of the specified pointers is NULL.

\textbf{ippStsSizeErr}  
Indicates an error condition if any field of the \textit{roiSize} is less than 2.

\textbf{YCbCr420ToYCbCr422\_Filter}  
Convert 4:2:0 image to 4:2:2 image with additional filtering.

Syntax

\begin{verbatim}
IppStatus ippiYCbCr420ToYCbCr422\_Filter_8u\_P3R(const Ipp8u* pSrc[3], int srcStep[3],
Ipp8u* pDst[3], int dstStep[3], IppiSize roiSize);
\end{verbatim}
IppStatus ippiYCbCr420ToYCbCr422_Filter_8u_P2P3R(const Ipp8u* pSrcY, int srcYStep, const Ipp8u* pSrcCbCr, int srcCbCrStep, Ipp8u* pDst[3], int dstStep[3], IppSize roiSize);
IppStatus ippiYCbCr420ToYCbCr422_Filter_8u_P2C2R(const Ipp8u* pSrcY, int srcYStep, const Ipp8u* pSrcCbCr, int srcCbCrStep, Ipp8u* pDst, int dstStep, IppSize roiSize, int layout);

**Include Files**
ippcc.h

**Domain Dependencies**

**Headers:** ippcore.h, ippvm.h, ipps.h, ippi.h

**Libraries:** ippcore.lib, ippvm.lib, ipps.lib, ippi.lib

**Parameters**

- **pSrc**
  Array of pointers to the ROI in each plane for a three-plane source image.

- **srcStep**
  Array of distances in bytes between starts of consecutive lines in each plane for a three-plane source image.

- **pSrcY**
  Pointer to the ROI in the luminance plane for a two-plane source image.

- **srcYStep**
  Distance in bytes between starts of consecutive lines in the luminance plane of a source image.

- **pSrcCbCr**
  Pointer to the ROI in the interleaved chrominance plane for a two-plane source image.

- **srcCbCrStep**
  Distance in bytes between starts of consecutive lines in the interleaved chrominance plane of the source image.

- **pDst**
  Pointer to the ROI in the pixel-order destination image. Array of pointers to the ROI in each plane of the planar destination image.

- **dstStep**
  Distance in bytes between starts of consecutive lines in the destination image. Array of distance values for the destination image planes.

- **roiSize**
  Size of the ROI in pixels.

- **layout**
  Slice layout. Possible values:
  - IPP_UPPER
  - IPP_CENTER
  - IPP_LOWER
  - IPP_LOWER && IPP_UPPER
  - IPP_UPPER && IPP_CENTER
  - Image is not sliced

**Description**

This function operates with ROI (see Regions of Interest in Intel IPP).

This function converts the 4:2:0 planar source image `pSrc` to the 4:2:2 image `pDst` and performs additional filtering. The source image can be two- or three-plane image (see Table "Planar Image Formats"). The first plane of the two-plane source image `pSrcY` contains luminance samples `Y0, Y1, Y2, ...`, the second
plane pSrcCbCr contains interleaved chrominance samples Cb0, Cr0, Cb1, Cr1, .... . The destination image pDst can be three-plane or two-channel (see Table “Pixel-Order Image Formats” and Table “Planar Image Formats”).

The function flavorsippiYCbCr420ToYCbCr422_Filter_8u_P3R andippiYCbCr420ToYCbCr422_Filter_8u_P2P3R additionally perform the vertical upsampling using a Catmull-Rom interpolation (cubic convolution interpolation). In this case roiSize.width should be multiple of 2, and roiSize.height should be multiple of 8.

The functionippiYCbCr420ToYCbCr422_Filter_8u_P2C2R additionally performs deinterlace filtering. Commonly it is used to process images that are divided into slices. In this case slice layout should be specified, since the function processes the first (upper), last (lowermost), and intermediate (middle) slices differently. The height of slices should be a multiple of 16.

Caution
The image slices should be processed exactly in the following order: the first slice, intermediate slices, the last slice.

The function may be applied to a not-sliced image as well. In this case roiSize.width and roiSize.height should be multiple of 2.

Return Values

ippStsNoErr Indicates no error. Any other value indicates an error.
ippStsNullPtrErr Indicates an error condition if any of the specified pointers is NULL.
ippStsSizeErr Indicates an error condition if roiSize.width has wrong value.

YCbCr420To422_Interlace
Converts interlaced YCbCr image from 4:2:0 sampling format to 4:2:2 format.

Syntax

IppStatus ippiYCbCr420To422_Interlace_8u_P3R(const Ipp8u* pSrc[3], int srcStep[3], Ipp8u* pDst[3], int dstStep[3], IppSize roiSize);

Include Files

ippcc.h

Domain Dependencies

Headers: ippcore.h, ippvm.h, ipps.h, ippi.h
Libraries: ippcore.lib, ippvm.lib, ipps.lib, ippi.lib

Parameters

pSrc
Array of pointers to the ROI in each plane for source image.

srcStep
Array of distances in bytes between starts of consecutive lines in each plane for the source image.

pDst
Array of pointers to the ROI in each plane for destination image.
**dstStep**
Array of distances in bytes between starts of consecutive lines in each plane for the destination image.

**roiSize**
Size of the source and destination ROI in pixels, its width must be multiple of 2, and height must be multiple of 4.

**Description**
This function operates with ROI (see Regions of Interest in Intel IPP).

This function converts the interlaced planar 4:2:0 source image `pSrc` to the 4:2:2 image `pDst`. Three-plane image has the following order of pointers: Y-plane, Cb-plane, Cr-plane.

The conversion is performed in accordance with the following formulas:

\[
Y_{\text{dest}} = Y_{\text{src}};
\]

\[
Cb_{0}(Cr_{0})_{\text{dest}} = (5*Cb_{0}(Cr_{0})_{\text{src}} + 3*Cb_{2}(Cr_{2})_{\text{src}} + 4)/8;
\]

\[
Cb_{1}(Cr_{1})_{\text{dest}} = (7*Cb_{1}(Cr_{1})_{\text{src}} + Cb_{3}(Cr_{3})_{\text{src}} + 4)/8;
\]

\[
Cb_{2}(Cr_{2})_{\text{dest}} = (Cb_{0}(Cr_{0})_{\text{src}} + 7*Cb_{2}(Cr_{2})_{\text{src}} + 4)/8;
\]

\[
Cb_{3}(Cr_{3})_{\text{dest}} = (3*Cb_{1}(Cr_{1})_{\text{src}} + 5*Cb_{3}(Cr_{3})_{\text{src}} + 4)/8;
\]

**Return Values**

- `ippStsNoErr` Indicates no error. Any other value indicates an error.
- `ippStsNullPtrErr` Indicates an error condition if any of the specified pointers is NULL.
- `ippStsSizeErr` Indicates an error condition if `roiSize.width` is less than 2, or `roiSize.height` is less than 4.
- `ippStsDoubleSize` Indicates a warning if `roiSize.width` is not multiple of 2, or `roiSize.height` is not multiple of 4.

**YCbCr420ToCbYCr422**
*Converts 4:2:0 YCbCr image to 4:2:2 CbYCr image.*

**Syntax**

```c
IppStatus ippiYCbCr420ToCbYCr422_8u_P2C2R(const Ipp8u* pSrcY, int srcYStep, const Ipp8u* pSrcCbCr, int srcCbCrStep, Ipp8u* pDst, int dstStep, IppiSize roiSize);
```

**Include Files**

`ippcc.h`

**Domain Dependencies**

Headers: `ippcore.h`, `ippvm.h`, `ipps.h`, `ippi.h`

Libraries: `ippcore.lib`, `ippvm.lib`, `ipps.lib`, `ippi.lib`

**Parameters**

- `pSrcY` Pointer to the ROI in the luminance plane of the source image.
- `srcYStep` Distance in bytes between starts of consecutive lines in the luminance plane of the source image.
**Description**
This function operates with ROI (see Regions of Interest in Intel IPP).

This function converts the planar 4:2:0 two-plane source image to the pixel-order 4:2:2 two-channel image. The first plane of the source image pSrcY contains luminance samples \( Y_0, Y_1, Y_2, \ldots \), the second plane pSrcCbCr contains interleaved chrominance samples \( Cb_0, Cr_0, Cb_1, Cr_1, \ldots \). The destination image pDst has the following sequence of samples: \( Cb_0, Y_0, Cr_0, Y_1, Cb_1, Y_2, Cr_1, Y_3, Cb_2, \ldots \).

**Return Values**
- **ippStsNoErr**: Indicates no error. Any other value indicates an error.
- **ippStsNullPtrErr**: Indicates an error condition if any of the specified pointers is NULL.
- **ippStsSizeErr**: Indicates an error condition if any field of the roiSize is less than 2.

**YCbCr420ToCbYCr422_Interlace**
Converts interlaced 4:2:0 YCbCr image to 4:2:2 CbYCr image.

**Syntax**
```c
IppStatus ippiYCbCr420ToCbYCr422_Interlace_8u_P3C2R(const Ipp8u* pSrc[3], int srcStep[3], Ipp8u* pDst, int dstStep, IppiSize roiSize);
```

**Include Files**
ippcc.h

**Domain Dependencies**
Headers: ippcore.h, ippvm.h, ipp.h, ippi.h
Libraries: ippcore.lib, ippvm.lib, ipp.slib, ippi.lib

**Parameters**
- **pSrc**: Array of pointers to the ROI in each plane for source image.
- **srcStep**: Array of distances in bytes between starts of consecutive lines in each plane for the source image.
- **pDst**: Pointer to the destination image ROI.
**Description**

This function operates with ROI (see Regions of Interest in Intel IPP).

This function converts the interlaced planar 4:2:0 image to the pixel-order 4:2:2 two-channel image. Three-plane source image has the following order of pointers: Y-plane, Cb-plane, Cr-plane. The destination image `pDst` has the following sequence of samples: Cb0, Y0, Cb1, Y1, Cr1, Y2, Cb2, ...

The conversion is performed in accordance with the following formulas:

\[
\begin{align*}
Y_{\text{dest}} &= Y_{\text{src}}; \\
Cb_0(Cr_0)_{\text{dest}} &= (5*Cb_0(Cr_0)_{\text{src}} + 3*Cb_2(Cr_2)_{\text{src}} + 4)/8; \\
Cb_1(Cr_1)_{\text{dest}} &= (7*Cb_1(Cr_1)_{\text{src}} + Cb_3(Cr_3)_{\text{src}} + 4)/8; \\
Cb_2(Cr_2)_{\text{dest}} &= (Cb_0(Cr_0)_{\text{src}} + 7*Cb_2(Cr_2)_{\text{src}} + 4)/8; \\
Cb_3(Cr_3)_{\text{dest}} &= (3*Cb_1(Cr_1)_{\text{src}} + 5*Cb_3(Cr_3)_{\text{src}} + 4)/8;
\end{align*}
\]

**Return Values**

- **ippStsNoErr**: Indicates no error. Any other value indicates an error.
- **ippStsNullPtrErr**: Indicates an error condition if any of the specified pointers is NULL.
- **ippStsSizeErr**: Indicates an error condition if `roiSize.width` is less than 2, or `roiSize.height` is less than 4.
- **ippStsDoubleSize**: Indicates a warning if `roiSize.width` is not multiple of 2, or `roiSize.height` is not multiple of 4.

---

**YCbCr420ToYCrCb420**

*Converts 4:2:0 YCbCr image to 4:2:0 YCrCb image.*

**Syntax**

```c
IppStatus ippiYCbCr420ToYCrCb420_8u_P2P3R(const Ipp8u* pSrcY, int srcYStep, const Ipp8u* pSrcCbCr, int srcCbCrStep, Ipp8u* pDst[3], int dstStep[3], IppiSize roiSize);
```

**Include Files**

`ippcc.h`

**Domain Dependencies**

Headers: `ippcore.h`, `ippvm.h`, `ipps.h`, `ippi.h`

Libraries: `ippcore.lib`, `ippvm.lib`, `ipps.lib`, `ippi.lib`

**Parameters**

- **pSrcY**: Pointer to the ROI in the luminance plane of the source image.
- **srcYStep**: Distance in bytes between starts of consecutive lines in the luminance plane of the source image.
pSrcCbCr

Pointer to the ROI in the interleaved chrominance plane of the source image.

srcCbCrStep

Distance in bytes between starts of consecutive lines in the interleaved chrominance plane of the source image.

pDst

Array of pointers to the ROI in each plane of the destination image.

dstStep

Array of distances in bytes between starts of consecutive lines in the destination image planes.

roiSize

Size of the source and destination ROI in pixels, height and width should be multiple of 2.

Description

This function operates with ROI (see Regions of Interest in Intel IPP).

This function converts the 4:2:0 two-plane source image pSrc to the 4:2:0 three-plane image pDst. The first plane of the source image pSrcY contains luminance samples Y0, Y1, Y2, ..., the second plane pSrcCbCr contains interleaved chrominance samples Cb0, Cr0, Cb1, Cr1, .... The destination image pDst has the following order of pointers: Y-plane, Cr-plane, Cb-plane (see Table "Planar Image Formats").

Return Values

ippStsNoErr
Indicates no error. Any other value indicates an error.

ippStsNullPtrErr
Indicates an error condition if any of the specified pointers is NULL.

ippStsSizeErr
Indicates an error condition if any field of the roiSize is less than 2.

YCbCr420ToYCrCb420_Filter

Convert 4:2:0 YCbCr image to 4:2:0 YCrCb image with deinterlace filtering.

Syntax

IppStatus ippiYCbCr420ToYCrCb420_Filter_8u_P2P3R(const Ipp8u* pSrcY, int srcYStep, const Ipp8u* pSrcCbCr, int srcCbCrStep, Ipp8u* pDst[3], int dstStep[3], IppiSize roiSize, int layout);

Include Files

ippcc.h

Domain Dependencies

Headers: ippcore.h, ippvm.h, ipps.h,ippi.h
Libraries: ippcore.lib, ippvm.lib, ipps.lib,ippi.lib

Parameters

pSrcY
Pointer to the ROI in the luminance plane of the source image.

srcYStep
Distance in bytes between starts of consecutive lines in the luminance plane of the source image.

pSrcCbCr
Pointer to the ROI in the interleaved chrominance plane of the source image.
**Description**

This function operates with ROI (see Regions of Interest in Intel IPP).

This function converts the 4:2:0 two-plane source image to the 4:2:0 three-plane image. The first plane of the source image \( p_{Src\ Y} \) contains luminance samples \( Y_0, Y_1, Y_2, \ldots \), the second plane \( p_{Src\ CbCr} \) contains interleaved chrominance samples \( Cb_0, Cr_0, Cb_1, Cr_1, \ldots \). The destination image \( p_{Dst} \) has the following order of pointers: \( Y \)-plane, \( Cr \)-plane, \( Cb \)-plane. The function additionally performs deinterlace filtering. Commonly it is used to process sliced images. In this case the slice \( layout \) should be specified, since the function processes the first (upper), last (lowermost), and intermediate (middle) slices differently. The height of slices should be a multiple of 16. The function may be applied to a not-sliced image as well.

**Caution**

The image slices should be processed exactly in the following order: the first slice, intermediate slices, the last slice.

**Return Values**

- **ippStsNoErr**: Indicates no error. Any other value indicates an error.
- **ippStsNullPtrErr**: Indicates an error condition if any of the specified pointers is NULL.
- **ippStsSizeErr**: Indicates an error condition if any field of the \( roi\ Size \) is less than 2.

**YCbCr420ToYCbCr411**

*Converts YCbCr image from 4:2:0 sampling format to 4:1:1 format.*

**Syntax**

```c
IppStatus ippiYCbCr420ToYCbCr411_8u_P3P2R(const Ipp8u* pSrc[3], int srcStep[3], Ipp8u* pDstY, int dstYStep, Ipp8u* pDstCbCr, int dstCbCrStep, IppiSize roiSize);
```
IppStatusippiYCbCr420ToYCbCr411_8u_P2P3R(const Ipp8u* pSrcY, int srcYStep, const Ipp8u* pSrcCbCr, int srcCbCrStep, Ipp8u* pDst[3], int dstStep[3], IppiSize roiSize);
IppStatusippiYCbCr420To411_8u_P3R(const Ipp8u* pSrc[3], int srcStep[3], Ipp8u* pDst[3], int dstStep[3], IppiSize roiSize);
IppStatusippiYCbCr420To1620_8u_P3R(const Ipp8u* pSrc[3], int srcStep[3], Ipp8u* pDst[3], int dstStep[3], IppiSize roiSize);
IppStatusippiYCbCr1620To420_8u_P3R(const Ipp8u* pSrc[3], int srcStep[3], Ipp8u* pDst[3], int dstStep[3], IppiSize roiSize);

Include Files
ippcc.h

Domain Dependencies
Headers: ippcore.h, ippvm.h, ipps.h,ippi.h
Libraries: ippcore.lib, ippvm.lib, ipps.lib,ippi.lib

Parameters

pSrc
Array of pointers to the ROI in each plane for a three-plane source image.

srcStep
Array of distances in bytes between starts of consecutive lines in each plane for a three-plane source image.

pSrcY
Pointer to the ROI in the luminance plane for a two-plane source image.

srcYStep
Distance in bytes between starts of consecutive lines in the luminance plane of a source image.

pSrcCbCr
Pointer to the ROI in the interleaved chrominance plane for a two-plane source image.

srcCbCrStep
Distance in bytes between starts of consecutive lines in the interleaved chrominance plane of the source image.

pDst
Array of pointers to the ROI in each plane for a three-plane destination image.

dstStep
Array of distances in bytes between starts of consecutive lines in each plane for a three-plane destination image.

pDstY
Pointer to the ROI in the luminance plane for a two-plane destination image.

dstYStep
Distance in bytes between starts of consecutive lines in the luminance plane of a destination image.

pDstCbCr
Pointer to the ROI in the interleaved chrominance plane for a two-plane destination image.

dstCbCrStep
Distance in bytes between starts of consecutive lines in the chrominance plane of a destination image.

roiSize
Size of the ROI in pixels, its width should be multiple of 4, its height should be multiple of 2.

Description
This function operates with ROI (see Regions of Interest in Intel IPP).
This function converts the 4:2:0 source image to the 4:1:1 destination image. The source two-plane image is converted to destination three-plane image and vice versa. The first plane of the two-plane image contains luminance samples Y0, Y1, Y2, ..., the second plane contains interleaved chrominance samples Cb0, Cr0, Cb1, Cr1, .... The three-plane image has the following order of pointers: Y-plane, Cb-plane, Cr-plane (see Table “Planar Image Formats”).

Return Values

- ippStsNoErr: Indicates no error. Any other value indicates an error.
- ippStsNullPtrErr: Indicates an error condition if any of the specified pointers is NULL.
- ippStsSizeErr: Indicates an error condition if roiSize.width is less than 4 or roiSize.height is less than 2.

YCrCb420ToYCbCr422

Converts 4:2:0 YCrCb image to 4:2:2 YCbCr image.

Syntax

```c
IppStatus ippiYCrCb420ToYCbCr422_8u_P3R(const Ipp8u* pSrc[3], int srcStep[3], Ipp8u* pDst[3], int dstStep[3], IppiSize roiSize);
IppStatus ippiYCrCb420ToYCbCr422_8u_P3C2R(const Ipp8u* pSrc[3], int srcStep[3], Ipp8u* pDst, int dstStep, IppiSize roiSize);
```

Include Files

ippcc.h

Domain Dependencies

Headers: ippcore.h, ippvm.h, ipps.h,ippi.h
Libraries: ippcore.lib, ippvm.lib, ipps.lib,ippi.lib

Parameters

- `pSrc`: Array of pointers to the ROI in each plane of the source image.
- `srcStep`: Array of distances in bytes between starts of consecutive lines in each plane of the source image.
- `pDst`: Pointer to the ROI in the pixel-order destination image. Array of pointers to the ROI in each plane of the planar destination image.
- `dstStep`: Distance in bytes between starts of consecutive lines in the destination image. Array of distance values for the destination image planes.
- `roiSize`: Size of the ROI in pixels, height and width should be multiple of 2.

Description

This function converts the 4:2:0 YCrCb three-plane image `pSrc` to the 4:2:2 YCbCr three-plane or two-channel image `pDst` (see Table “Pixel-Order Image Formats” and Table “Planar Image Formats”).

This function operates with ROI (see Regions of Interest in Intel IPP).
Return Values

ippStsNoErr  Indicates no error. Any other value indicates an error.
ippStsNullPtrErr  Indicates an error condition if any of the specified pointers is NULL.
ippStsSizeErr  Indicates an error condition if any field of the roiSize is less than 2.

YCrCb420ToYCbCr422_Filter
Converts 4:2:0 YCrCb image to 4:2:2 YCbCr image with additional filtering.

Syntax
IppStatus ippiYCrCb420ToYCbCr422_Filter_8u_P3R(const Ipp8u* pSrc[3], int srcStep[3], Ipp8u* pDst[3], int dstStep[3], IppiSize roiSize);

Include Files
ippcc.h

Domain Dependencies
Headers: ippcore.h, ippvm.h, ipps.h,ippi.h
Libraries: ippcore.lib, ippvm.lib, ipps.lib,ippi.lib

Parameters
pSrc  Array of pointers to the ROI in each plane of the source image.
srcStep  Array of distances in bytes between starts of consecutive lines in each plane of the source image.
pDst  Array of pointers to the ROI in each plane of the destination image.
dstStep  Array of distances in bytes between starts of consecutive lines in each plane of the destination image.
roiSize  Size of the ROI in pixels, its width should be multiple of 2, its height should be multiple of 8.

Description
This function operates with ROI (see Regions of Interest in Intel IPP).
This function converts the 4:2:0YCrCb three-plane image pSrc to the 4:2:2YCbCr three-plane image pDst (see Table "Planar Image Formats").
Additionally, this function performs the vertical upsampling using a Catmull-Rom interpolation (cubic convolution interpolation).

Return Values
ippStsNoErr  Indicates no error. Any other value indicates an error.
ippStsNullPtrErr  Indicates an error condition if any of the specified pointers is NULL.
Indicates an error condition if roiSize.width is less than 2 or roiSize.height is less than 8.

YCrCb420ToCbYCr422
Converting 4:2:0 YCrCb image to 4:2:2 CbYCr image.

Syntax

IppStatus ippiYCrCb420ToCbYCr422_8u_P3C2R(const Ipp8u* pSrc[3], int srcStep[3], Ipp8u* pDst, int dstStep, IppiSize roiSize);

Include Files

ippcc.h

Domain Dependencies

Headers: ippcore.h, ippvm.h, ipps.h,ippi.h
Libraries: ippcore.lib, ippvm.lib, ipps.lib,ippi.lib

Parameters

pSrc
Array of pointers to the ROI in each plane of the source image.

srcStep
Array of distances in bytes between starts of consecutive lines in each plane of the source image.

pDst
Pointer to the destination image ROI.

dstStep
Distance in bytes between starts of consecutive lines in the destination image.

roiSize
Size of the source and destination ROI in pixels, height and width should be multiple of 2.

Description

This function operates with ROI (see Regions of Interest in Intel IPP).

This function converts the 4:2:0 YCrCb three-plane image pSrc (see Table “Planar Image Formats”) to the 4:2:2 CbYCr two-channel image pDst with the following sequence of samples: Cb0, Y0, Cr0, Y1, Cb1, Y2, Cr1, Y3, Cb2, ... (see Table “Pixel-Order Image Formats”).

Return Values

ippStsNoErr
Indicates no error. Any other value indicates an error.

ippStsNullPtrErr
Indicates an error condition if any of the specified pointers is NULL.

ippStsSizeErr
Indicates an error condition if any field of the roiSize is less than 2.

YCrCb420ToYCbCr420
Converting 4:2:0 YCrCb image to 4:2:0 YCbCr image.
Syntax

IppStatusippiYCrCb420ToYCbCr420_8u_P3P2R(const Ipp8u* pSrc[3], int srcStep[3], Ipp8u* pDstY, int dstYStep, Ipp8u* pDstCbCr, int dstCbCrStep, IppiSize roiSize);

Include Files

ippcc.h

Domain Dependencies

Headers: ippcore.h, ippvm.h, ipps.h,ippi.h

Libraries: ippcore.lib, ippvm.lib, ipps.lib,ippi.lib

Parameters

pSrc
srcStep
pDstY
dstYStep
pDstCbCr
dstCbCrStep
roiSize

Array of pointers to the ROI in each plane of the source image.

Array of distances in bytes between starts of consecutive lines in each plane of the source image.

Pointer to the ROI in the luminance plane of a destination image.

Distance in bytes between starts of consecutive lines in the luminance plane of the destination image.

Pointer to the ROI in the interleaved chrominance plane of the destination image.

Distance in bytes between starts of consecutive lines in the interleaved chrominance plane of a destination image.

Size of the source and destination ROI in pixels, height and width should be multiple of 2.

Description

This function operates with ROI (see Regions of Interest in Intel IPP).

This function converts the 4:2:0 YCrCb three-plane image pSrc (see Table “Planar Image Formats”) to the 4:2:0 YCbCr two-plane image that contains luminance samples Y0, Y1, Y2, .. in the first plane pDstY, and interleaved chrominance samples Cb0, Cr0, Cb1, Cr1, ... in the second plane pDstCbCr.

Return Values

ippStsNoErr

Indicates no error. Any other value indicates an error.

ippStsNullPtrErr

Indicates an error condition if any of the specified pointers is NULL.

ippStsSizeErr

Indicates an error condition if any field of the roiSize is less than 2.

YCrCb420ToYCbCr411

Converts 4:2:0 YCrCb image to 4:1:1 YCbCr image.

Syntax

IppStatusippiYCrCb420ToYCbCr411_8u_P3P2R(const Ipp8u* pSrc[3], int srcStep[3], Ipp8u* pDstY, int dstYStep, Ipp8u* pDstCbCr, int dstCbCrStep, IppiSize roiSize);
Include Files

ippcc.h

Domain Dependencies

Headers: ippcore.h, ippvm.h, ipps.h,ippi.h
Libraries: ippcore.lib, ippvm.lib, ipps.lib,ippi.lib

Parameters

pSrc
    Array of pointers to the ROI in each plane of the source image.
srcStep
    Array of distances in bytes between starts of consecutive lines in
each plane of the source image.
pDstY
    Pointer to the ROI in the luminance plane of a destination image.
dstYStep
    Distance in bytes between starts of consecutive lines in the
    luminance plane of the destination image.
pDstCbCr
    Pointer to the ROI in the interleaved chrominance plane of the
destination image.
dstCbCrStep
    Distance in bytes between starts of consecutive lines in the
    interleaved chrominance plane of a destination image.
roiSize
    Size of the ROI in pixels, its width should be multiple of 4, its height
    should be multiple of 2.

Description

This function operates with ROI (see Regions of Interest in Intel IPP).
This function converts the 4:2:0YCrCb three-plane image pSrc (see Table "Planar Image Formats") to the
4:1:1YCbCr two-plane image that contains luminance samples Y0, Y1, Y2, .. in the first plane pDstY, and
interleaved chrominance samples Cb0, Cr0, Cb1, Cr1, ... in the second plane pDstCbCr..

Return Values

ippStsNoErr
    Indicates no error. Any other value indicates an error.
ippStsNullPtrErr
    Indicates an error condition if any of the specified pointers is
    NULL.
ippStsSizeErr
    Indicates an error condition if roiSize.width is less than 4 or
    roiSize.height is less than 2.

YCbCr411

Converts 4:1:1 YCbCr image.

Syntax

IppStatus ippiYCbCr411_8u_P3P2R(const Ipp8u* pSrc[3], int srcStep[3], Ipp8u* pDstY, int
dstYStep, Ipp8u* pDstCbCr, int dstCbCrStep, IppiSize roiSize);
IppStatus ippiYCbCr411_8u_P2P3R(const Ipp8u* pSrcY, int srcYStep, const Ipp8u* pSrcCbCr,
    int srcCbCrStep, Ipp8u* pDst[3], int dstStep[3], IppiSize roiSize);

Include Files

ippcc.h
Domain Dependencies

Headers: ippcore.h, ippvm.h, ipps.h,ippi.h
Libraries: ippcore.lib, ippvm.lib, ipps.lib,ippi.lib

Parameters

- **pSrc**
  Array of pointers to the ROI in each plane for a three-plane source image.

- **srcStep**
  Array of distances in bytes between starts of consecutive lines in each plane for a three-plane source image.

- **pSrcY**
  Pointer to the ROI in the luminance plane for a two-plane source image.

- **srcYStep**
  Distance in bytes between starts of consecutive lines in the luminance plane of a source image.

- **pSrcCbCr**
  Pointer to the ROI in the interleaved chrominance plane for a two-plane source image.

- **srcCbCrStep**
  Distance in bytes between starts of consecutive lines in the interleaved chrominance plane of the source image.

- **pDst**
  Array of pointers to the ROI in each plane for a three-plane destination image.

- **dstStep**
  Array of distances in bytes between starts of consecutive lines in each plane for a three-plane destination image.

- **pDstY**
  Pointer to the ROI in the luminance plane for a two-plane destination image.

- **dstYStep**
  Distance in bytes between starts of consecutive lines in the luminance plane of a destination image.

- **pDstCbCr**
  Pointer to the ROI in the interleaved chrominance plane for a two-plane destination image.

- **dstCbCrStep**
  Distance in bytes between starts of consecutive lines in the chrominance plane of a destination image.

- **roiSize**
  Size of the ROI in pixels, its width should be multiple of 4.

Description

This function operates with ROI (see Regions of Interest in Intel IPP).

This function converts the 4:1:1 three-plane (see Table "Planar Image Formats") source image **pSrc** to the 4:1:1 two-plane image and vice versa. Two-plane image contains luminance samples \(Y_0, Y_1, Y_2, \ldots\) in the first plane, and interleaved chrominance samples \(Cb_0, Cr_0, Cb_1, Cr_1, \ldots\) in the second plane.

Return Values

- **ippStsNoErr**
  Indicates no error. Any other value indicates an error.

- **ippStsNullPtrErr**
  Indicates an error condition if any of the specified pointers is NULL.

- **ippStsSizeErr**
  Indicates an error condition if **roiSize.width** is less than 4.

**YCbCr411ToYCbCr422**

Converts 4:1:1 YCbCr image to 4:2:2 YCbCr image.
Syntax
IppStatusippiYCbCr411ToYCbCr422_8u_P3R(const Ipp8u*pSrc[3], int srcStep[3], Ipp8u*pDst[3], int dstStep[3], IppiSize roiSize);
IppStatusippiYCbCr411ToYCbCr422_8u_P3C2R(const Ipp8u*pSrc[3], int srcStep[3], Ipp8u*pDst, int dstStep, IppiSize roiSize);
IppStatusippiYCbCr411ToYCbCr422_8u_P2P3R(const Ipp8u*pSrcY, int srcYStep, const Ipp8u*pSrcCbCr, int srcCbCrStep, Ipp8u*pDst[3], int dstStep[3], IppiSize roiSize);
IppStatusippiYCbCr411ToYCbCr422_8u_P2C2R(const Ipp8u*pSrcY, int srcYStep, const Ipp8u*pSrcCbCr, int srcCbCrStep, Ipp8u*pDst, int dstStep, IppiSize roiSize);

Include Files
ippcc.h

Domain Dependencies
Headers: ippcore.h, ippvm.h, ipps.h, ippi.h
Libraries: ippcore.lib, ippvm.lib, ipps.lib, ippi.lib

Parameters

- **pSrc**
  Array of pointers to the ROI in each plane for a three-plane source image.

- **srcStep**
  Array of distances in bytes between starts of consecutive lines in each plane for a three-plane source image.

- **pSrcY**
  Pointer to the ROI in the luminance plane for a two-plane source image.

- **srcYStep**
  Distance in bytes between starts of consecutive lines in the luminance plane of a source image.

- **pSrcCbCr**
  Pointer to the ROI in the interleaved chrominance plane for a two-plane source image.

- **srcCbCrStep**
  Distance in bytes between starts of consecutive lines in the interleaved chrominance plane of the source image.

- **pDst**
  Pointer to the ROI in the pixel-order destination image. Array of pointers to the ROI in each plane of the planar destination image.

- **dstStep**
  Distance in bytes between starts of consecutive lines in the destination image. Array of distance values for the destination image planes.

- **roiSize**
  Size of the ROI in pixels, its width should be multiple of 4.

Description
This function operates with ROI (see Regions of Interest in Intel IPP).

This function converts 4:1:1 planar source image pSrc to the 4:2:2 image pDst. The first plane of the two-plane source image pSrcY contains luminance samples Y0, Y1, Y2, ..., the second plane pSrcCbCr contains interleaved chrominance samples Cb0, Cr0, Cb1, Cr1, .... The destination image pDst can be either three-plane (see Table "Planar Image Formats") or two-channel image (see Table "Pixel-Order Image Formats").

Return Values

- **ippStsNoErr**
  Indicates no error. Any other value indicates an error.
Indicates an error condition if any of the specified pointers is NULL.

Indicates an error condition if roiSize.width is less than 4.

**YCbCr411ToYCrCb422**

Converts 4:1:1 YCbCr image to 4:2:2 YCrCb image.

**Syntax**

IppStatus ippiYCbCr411ToYCrCb422_8u_P3R(const Ipp8u* pSrc[3], int srcStep[3], Ipp8u* pDst[3], int dstStep[3], IppiSize roiSize);

IppStatus ippiYCbCr411ToYCrCb422_8u_P3C2R(const Ipp8u* pSrc[3], int srcStep[3], Ipp8u* pDst, int dstStep, IppiSize roiSize);

**Include Files**

ippcc.h

**Domain Dependencies**

Headers: ippcore.h, ippvm.h, ipps.h, ippi.h

Libraries: ippcore.lib, ippvm.lib, ipps.lib, ippi.lib

**Parameters**

- **pSrc**: Array of pointers to the ROI in each plane for a three-plane source image.
- **srcStep**: Array of distances in bytes between starts of consecutive lines in each plane for a three-plane source image.
- **pDst**: Pointer to the ROI in the pixel-order destination image. Array of pointers to the ROI in each plane of the planar destination image.
- **dstStep**: Distance in bytes between starts of consecutive lines in the destination image. Array of distance values for the destination image planes.
- **roiSize**: Size of the ROI in pixels, its width should be multiple of 4.

**Description**

This function operates with ROI (see Regions of Interest in Intel IPP).

This function converts the 4:1:1 three-plane image pSrc to the 4:2:2 two-channel or three-plane image pDst with different order of components. The source image has the following order of pointers: Y-plane, Cb-plane, Cr-plane (see Table "Planar Image Formats"). The three-plane destination image has the following order of pointers: Y-plane, Cr-plane, Cb-plane (see Table "Planar Image Formats"), and two-channel destination image has the following sequence of samples: Y0, Cr0, Y1, Cb0, Y2, Cr1, Y3, Cb1, ... (see Table "Pixel-Order Image Formats").

**Return Values**

- **ippStsNoErr**: Indicates no error. Any other value indicates an error.
- **ippStsNullPtrErr**: Indicates an error condition if any of the specified pointers is NULL.
Indicates an error condition if roiSize.width is less than 4.

**YCbCr411ToYCbCr420, YCbCr411To420**  
*Converts 4:1:1 YCbCr image to 4:2:0 YCbCr image.*

**Syntax**

```c
IppStatus ippiYCbCr411ToYCbCr420_8u_P3R(const Ipp8u* pSrc[3], int srcStep[3], Ipp8u* pDst[3], int dstStep[3], IppiSize roiSize);
IppStatus ippiYCbCr411ToYCbCr420_8u_P3P2R(const Ipp8u* pSrc[3], int srcStep[3], Ipp8u* pDstY, int dstYStep, Ipp8u* pDstCbCr, int dstCbCrStep, IppiSize roiSize);
IppStatus ippiYCbCr411ToYCbCr420_8u_P2P3R(const Ipp8u* pSrcY, int srcYStep, const Ipp8u* pSrcCbCr, int srcCbCrStep, Ipp8u* pDst[3], int dstStep[3], IppiSize roiSize);
IppStatus ippiYCbCr411To420_8u_P3R(const Ipp8u* pSrc[3], int srcStep[3], Ipp8u* pDst[3], int dstStep[3], IppiSize roiSize);
```

**Include Files**  
ippcc.h

**Domain Dependencies**

*Headers:* ippcore.h, ippvm.h, ipps.h,ippi.h  
*Libraries:* ippcore.lib, ippvm.lib, ipps.lib,ippi.lib

**Parameters**

- **pSrc**: Array of pointers to the ROI in each plane for a three-plane source image.
- **srcStep**: Array of distances in bytes between starts of consecutive lines in each plane for a three-plane source image.
- **pSrcY**: Pointer to the ROI in the luminance plane for a two-plane source image.
- **srcYStep**: Distance in bytes between starts of consecutive lines in the luminance plane of the source image.
- **pSrcCbCr**: Pointer to the ROI in the interleaved chrominance plane for a two-plane source image.
- **srcCbCrStep**: Distance in bytes between starts of consecutive lines in the interleaved chrominance plane of the source image.
- **pDst**: Array of pointers to the ROI in each plane for a three-plane destination image.
- **dstStep**: Array of distances in bytes between starts of consecutive lines in each plane for a three-plane destination image.
- **pDstY**: Pointer to the ROI in the luminance plane for a two-plane destination image.
- **dstYStep**: Distance in bytes between starts of consecutive lines in the luminance plane of a destination image.
- **pDstCbCr**: Pointer to the ROI in the interleaved chrominance plane for a two-plane destination image.
**dstCbCrStep**  
Distance in bytes between starts of consecutive lines in the chrominance plane of a destination image.

**roiSize**  
Size of the ROI in pixels, its width should be multiple of 4, its height should be multiple of 2.

### Description

This function operates with ROI (see Regions of Interest in Intel IPP).

This function converts the 4:1:1 planar source image `pSrc` (see Table "Planar Image Formats") to the 4:2:0 planar image `pDst`. Both source and destination images can be three- or two-plane. Three-plane images have the following order of pointers: Y-plane, Cb-plane, Cr-plane (see Table "Planar Image Formats"). Two-plane images contain luminance samples `Y0, Y1, Y2, ..` in the first plane, and interleaved chrominance samples `Cb0, Cr0, Cb1, Cr1, ...` in the second plane.

### Return Values

- **ippStsNoErr**: Indicates no error. Any other value indicates an error.
- **ippStsNullPtrErr**: Indicates an error condition if any of the specified pointers is NULL.
- **ippStsSizeErr**: Indicates an error condition if `roiSize.width` is less than 4 or `roiSize.height` is less than 2.

### YCbCr411ToYCrCb420

**Converts 4:1:1 YCbCr image to 4:2:0 YCrCb image.**

### Syntax

```c
IppStatus ippiYCbCr411ToYCrCb420_8u_P2P3R(const Ipp8u* pSrcY, int srcYStep, const Ipp8u* pSrcCbCr, int srcCbCrStep, Ipp8u* pDst[3], int dstStep[3], IppiSize roiSize);
```

### Include Files

ippcc.h

### Domain Dependencies

**Headers**: ippcore.h, ippvm.h, ipps.h,ippi.h  
**Libraries**: ippcore.lib, ippvm.lib, ipps.lib,ippi.lib

### Parameters

- **pSrcY**: Pointer to the ROI in the luminance plane of the source image.  
- **srcYStep**: Distance in bytes between starts of consecutive lines in the luminance plane of the source image.  
- **pSrcCbCr**: Pointer to the ROI in the interleaved chrominance plane of the source image.  
- **srcCbCrStep**: Distance in bytes between starts of consecutive lines in the interleaved chrominance plane of the source image.  
- **pDst**: Array of pointers to the ROI in each plane of the destination image.  
- **dstStep**: Array of distances in bytes between starts of consecutive lines in each plane of the destination image.
roiSize

Size of the ROI in pixels, its width should be multiple of 4, its height should be multiple of 2.

Description

This function operates with ROI (see Regions of Interest in Intel IPP).

This function converts the 4:1:1 two-plane source image \( pSrc \) to the 4:2:0 three-plane image \( pDst \) with a different order of components. The first plane of the source image \( pSrcY \) contains luminance samples \( Y_0, Y_1, Y_2, \ldots \), the second plane \( pSrcCbCr \) contains interleaved chrominance samples \( Cb_0, Cr_0, Cb_1, Cr_1, \ldots \). The destination image has the following order of pointers: \( Y \)-plane, \( Cr \)-plane, \( Cb \)-plane (see Table “Planar Image Formats”),

Return Values

- \( ippStsNoErr \): Indicates no error. Any other value indicates an error.
- \( ippStsNullPtrErr \): Indicates an error condition if any of the specified pointers is NULL.
- \( ippStsSizeErr \): Indicates an error condition if roiSize.width is less than 4 or roiSize.height is less than 2.

Color Twist

Color twist conversion functions use values of all color channels of a source pixel to compute the resultant destination channel value. The destination channel value is obtained as the result of multiplying the corresponding row of the color-twist matrix by the vector of source pixel channel values.

For example, if \((r,g,b)\) is a source pixel, then the destination pixel values \((R,G,B)\) are computed as follows:

\[
R = t_{11}r + t_{12}g + t_{13}b + t_{14}
\]
\[
G = t_{21}r + t_{22}g + t_{23}b + t_{24}
\]
\[
B = t_{31}r + t_{32}g + t_{33}b + t_{34}
\]

where

\[
T = \begin{bmatrix}
    t_{11} & t_{12} & t_{13} & t_{14} \\
    t_{21} & t_{22} & t_{23} & t_{24} \\
    t_{31} & t_{32} & t_{33} & t_{34}
\end{bmatrix}
\]

is the color twist matrix. The color twist matrix used by the Intel IPP functions is a matrix of size 3x4, or 4x4 with floating-point elements. The matrix elements are specific for each particular type of color conversion.

ColorTwist

Applies a color twist matrix to an image with floating-point pixel values.
Syntax

Case 1: Not-in-place operation on pixel-order data

```c
IppStatus ippiColorTwist_<mod>(const Ipp32f* pSrc, int srcStep, Ipp32f* pDst, int dstStep, IppiSize roiSize, const Ipp32f twist[3][4]);
```

Supported values for mod:

- 32f_C3R
- 32f_AC4R

```c
IppStatus ippiColorTwist_32f_C4R(const Ipp32f* pSrc, int srcStep, Ipp32f* pDst, int dstStep, IppiSize roiSize, const Ipp32f twist[4][4]);
```

Case 2: Not-in-place operation on planar data

```c
IppStatus ippiColorTwist_32f_P3R(const Ipp32f* pSrc[3], int srcStep, Ipp32f* pDst[3], int dstStep, IppiSize roiSize, const Ipp32f twist[3][4]);
```

Case 3: In-place operation on pixel-order data

```c
IppStatus ippiColorTwist_<mod>(Ipp32f* pSrcDst, int srcDstStep, IppiSize roiSize, const Ipp32f twist[3][4]);
```

Supported values for mod:

- 32f_C3IR
- 32f_AC4IR

Case 4: In-place operation on planar data

```c
IppStatus ippiColorTwist_32f_IP3R(Ipp32f* pSrcDst[3], int srcDstStep, IppiSize roiSize, const Ipp32f twist[3][4]);
```

Include Files

- ippcc.h

Domain Dependencies

Headers: ippcore.h, ippvm.h, ipps.h,ippi.h

Libraries: ippcore.lib, ippvm.lib, ipps.lib,ippi.lib

Parameters

- **pSrc**
  - Pointer to the source image ROI.

- **srcStep**
  - Distance in bytes between starts of consecutive lines in the source image.

- **pDst**
  - Pointer to the destination image ROI.

- **dstStep**
  - Distance in bytes between starts of consecutive lines in the destination image.

- **pSrcDst**
  - Pointer to the source and destination image ROI for the in-place operation.

- **srcDstStep**
  - Distance in bytes between starts of consecutive lines in the source and destination image for the in-place operation.

- **roiSize**
  - Size of the source and destination ROI in pixels.

- **twist**
  - The array containing color-twist matrix elements.
Description
This function operates with ROI (see Regions of Interest in Intel IPP).

This function applies the color-twist matrix to all three color channels in the source image with floating-point pixel values to obtain the resulting data in the destination image. The destination channel value is obtained as the result of multiplying the corresponding row of the color-twist matrix by the vector of source pixel channel values.

Return Values
ippStsNoErr
Indicates no error. Any other value indicates an error.

ippStsNullPtrErr
Indicates an error condition if pSrc, pDst, or pSrcDst is NULL.

ippStsSizeErr
Indicates an error condition if roiSize has a field with a zero or negative value.

ColorTwist32f
Applies a color twist matrix to an image with integer pixel values.

Syntax
Case 1: Not-in-place operation on pixel-order data
IppStatus ippiColorTwist32f_<mod>(const Ipp<datatype>* pSrc, int srcStep, Ipp<datatype>* pDst, int dstStep, IppiSize roiSize, const Ipp32f twist[3][4]);

Supported values for mod:
8u_C3R 16u_C3R 16s_C3R
8u_AC4R 16u_AC4R 16s_AC4R

Case 2: Not-in-place operation on planar data
IppStatus ippiColorTwist32f_<mod>(const Ipp<datatype>* pSrc[3], int srcStep, Ipp<datatype>* pDst[3], int dstStep, IppiSize roiSize, const Ipp32f twist[3][4]);

Supported values for mod:
8u_P3R 16u_P3R 16s_P3R

Case 3: In-place operation on pixel-order data
IppStatus ippiColorTwist32f_<mod>(Ipp<datatype>* pSrcDst, int srcDstStep, IppiSize roiSize, const Ipp32f twist[3][4]);

Supported values for mod:
8u_C3IR 16u_C3IR 16s_C3IR
8u_AC4IR 16u_AC4IR 16s_AC4IR
Case 4: In-place operation on planar data

IppStatusippiColorTwist32f <mod>(Ipp<datatype>* pSrcDst[3], int srcDstStep, IppSize roiSize, const Ipp32f twist[3][4]);

Supported values for mod:

8u_IP3R   16u_IP3R   16s_IP3R

Include Files

ippcc.h

Domain Dependencies

Headers: ippcore.h, ippvm.h, ipps.h,ippi.h
Libraries: ippcore.lib, ippvm.lib,ipps.lib,ippi.lib

Parameters

pSrc  
  Pointer to the source image ROI.

srcStep  
  Distance, in bytes, between the starting points of consecutive lines in the source image.

pDst  
  Pointer to the destination image ROI.

dstStep  
  Distance, in bytes, between the starting points of consecutive lines in the destination image.

pSrcDst  
  Pointer to the source and destination image ROI for the in-place operation.

srcDstStep  
  Distance, in bytes, between the starting points of consecutive lines in the source and destination image for the in-place operation.

roiSize  
  Size of the source and destination ROI in pixels.

twist  
  The array containing color-twist matrix elements.

Description

This function operates with ROI (see Regions of Interest in Intel IPP).

This function applies the color-twist matrix to all three color channel values in the integer source image to obtain the resulting data in the destination image. For example, the conversion from the RGB to the YCbCr format can be done as

\[
Y = 0.299*R + 0.587*G + 0.114*B
\]

\[
Cb = -0.16874*R - 0.33126*G + 0.5*B + 0.5
\]

\[
Cr = 0.5*R - 0.41869*G - 0.08131*B + 0.5
\]

which can be described in terms of the following color twist matrix:

\[
\begin{bmatrix}
0.29900f & 0.58700f & 0.11400f & 0.000f \\
-0.16874f & -0.33126f & 0.50000f & 128.0f \\
0.50000f & -0.41869f & -0.08131f & 128.0f
\end{bmatrix}
\]

Color-twist matrices may also be used to perform many other color conversions.

Return Values

ippStsNoErr  
  Indicates no error. Any other value indicates an error.


### Color Keying

**CompColorKey**  
*Performs color keying of two images.*

**Syntax**

**Case 1: Operation on one-channel data**

```c
IppStatus ippiCompColorKey_<mod>(const Ipp<datatype>* pSrc1, int src1Step, const Ipp<datatype>* pSrc2, int src2Step, Ipp<datatype>* pDst, int dstStep, IppiSize roiSize, Ipp<datatype> colorKey);
```

Supported values for `mod`:

- 8u_C1R
- 16u_C1R
- 16s_C1R

**Case 2: Operation on multi-channel data**

```c
IppStatus ippiCompColorKey_<mod>(const Ipp<datatype>* pSrc1, int src1Step, const Ipp<datatype>* pSrc2, int src2Step, Ipp<datatype>* pDst, int dstStep, IppiSize roiSize, Ipp<datatype> colorKey[3]);
```

Supported values for `mod`:

- 8u_C3R
- 16u_C3R
- 16s_C3R

```c
IppStatus ippiCompColorKey_<mod>(const Ipp<datatype>* pSrc1, int src1Step, const Ipp<datatype>* pSrc2, int src2Step, Ipp<datatype>* pDst, int dstStep, IppiSize roiSize, Ipp<datatype> colorKey[4]);
```

Supported values for `mod`:

- 8u_C4R
- 16u_C4R
- 16s_C4R

**Include Files**

ippi.h

**Domain Dependencies**

- Headers: ippcore.h, ippvm.h, ipps.h
- Libraries: ippcore.lib, ippvm.lib, ipps.lib

**Parameters**

- `pSrc1`, `pSrc2`  
  Pointer to the source images ROI.

- `src1Step`, `src2Step`  
  Distances in bytes between starts of consecutive lines in the source images.
pDst

Pointer to the destination image ROI.

dstStep

Distance in bytes between starts of consecutive lines in the destination image.

roiSize

Size of the source and destination ROI in pixels.

colorKey

Value of the key color for 1-channel images, array of color values for multi-channel images.

Description

This function operates with ROI (see Regions of Interest in Intel IPP).

This function replaces all areas of the source image pSrc1 containing the specified key color colorKey with the corresponding pixels of the background image pSrc2 and stores the result in the destination image pDst.

The Figure Applying the Function ippiCompColorKey to Sample Images shows an example of how the function ippiCompColorKey works.

Applying the Function ippiCompColorKey to Sample Images

![Sample Images](image-url)

Return Values

ippStsNoErr

Indicates no error. Any other value indicates an error.

ippStsNullPtrErr

Indicates an error condition if one of the specified pointers is NULL.

ippStsSizeErr

Indicates an error condition if roiSize has a field with a zero or negative value.

ippStsStepErr

Indicates an error condition if one of the step values is less than or equal to 0.

AlphaCompColorKey

Performs color keying and alpha composition of two images.

Syntax

IppStatus ippiAlphaCompColorKey_8u_AC4R(const Ipp8u* pSrc1, int src1Step, Ipp8u alpha1, const Ipp8u* pSrc2, int src2Step, Ipp8u alpha2, Ipp8u* pDst, int dstStep, IppiSize roiSize, Ipp8u colorKey[4], IppiAlphaType alphaType);

Include Files

ippi.h
Domain Dependencies

Headers: ippcore.h, ippvm.h, ipps.h
Libraries: ippcore.lib, ippvm.lib, ipps.lib

Parameters

- **pSrc1, pSrc2**: Pointer to the source images ROI.
- **src1Step, src2Step**: Distances in bytes between starts of consecutive lines in the source images.
- **alpha1, alpha2**: Alpha value.
- **pDst**: Pointer to the destination image ROI.
- **dstStep**: Distance in bytes between starts of consecutive lines in the destination image.
- **roiSize**: Size of the source and destination ROI in pixels.
- **colorKey**: Array of color values.
- **alphaType**: The type of composition to perform (without pre-multiplying). See Table “Possible Values of alphaType Parameter” for more details.

Description

This function operates with ROI (see Regions of Interest in Intel IPP).

This function replaces all areas of the source image pSrc1 containing the specified key color colorKey with the corresponding pixels of the background image pSrc2 and additionally performs alpha composition (see supported Table “Types of Image Compositing Operations”) in accordance with the parameter alphaType.

Note the alpha channel in the pDst is not changed after color keying.

The parameter alphaType should not be set to the values intended for operations with pre-multiplying.

Return Values

- **ippStsNoErr**: Indicates no error. Any other value indicates an error.
- **ippStsNullPtrErr**: Indicates an error condition if one of the specified pointers is NULL.
- **ippStsSizeErr**: Indicates an error condition if roiSize has a field with a zero or negative value.
- **ippStsStepErr**: Indicates an error condition if one of the step values is less than or equal to 0.
- **ippStsAlphaTypeErr**: Indicates an error condition if alphaType specifies the unsupported type of composition.

Gamma Correction

**GammaFwd**

Performs gamma-correction of the source image with RGB data.
### Syntax

**Case 1: Not-in-place operation on integer pixel-order data**

```c
IppStatus ippiGammaFwd_<mod>(const Ipp<datatype>* pSrc, int srcStep, Ipp<datatype>* pDst, int dstStep, IppiSize roiSize);
```

**Supported values for** `mod`:

- 8u_C3R
- 16u_C3R
- 8u_AC4R
- 16u_AC4R

**Case 2: Not-in-place operation on integer planar data**

```c
IppStatus ippiGammaFwd_<mod>(const Ipp<datatype>* pSrc[3], int srcStep, Ipp<datatype>* pDst[3], int dstStep, IppiSize roiSize);
```

**Supported values for** `mod`:

- 8u_P3R
- 16u_P3R

**Case 3: Not-in-place operation on floating-point pixel-order data**

```c
IppStatus ippiGammaFwd_<mod>(const Ipp32f* pSrc, int srcStep, Ipp32f* pDst, int dstStep, IppiSize roiSize, Ipp32f vMin, Ipp32f vMax);
```

**Supported values for** `mod`:

- 32f_C3R
- 32f_AC4R

**Case 4: Not-in-place operation on floating-point planar data**

```c
IppStatus ippiGammaFwd_32f_P3R (const Ipp32f* pSrc[3], int srcStep, Ipp32f* pDst[3], int dstStep, IppiSize roiSize, Ipp32f vMin, Ipp32f vMax);
```

**Case 5: In-place operation on integer pixel-order data**

```c
IppStatus ippiGammaFwd_<mod>(Ipp<datatype>* pSrcDst, int srcDstStep, IppiSize roiSize);
```

**Supported values for** `mod`:

- 8u_C3IR
- 16u_C3IR
- 8u_AC4IR
- 16u_AC4IR

**Case 6: In-place operation on integer planar data**

```c
IppStatus ippiGammaFwd_<mod>(Ipp<datatype>* pSrcDst[3], int srcDstStep, IppiSize roiSize);
```

**Supported values for** `mod`:

- 8u_IP3R
- 16u_IP3R

**Case 7: In-place operation on floating-point pixel-order data**

```c
IppStatus ippiGammaFwd_<mod>(Ipp32f* pSrcDst, int srcDstStep, IppiSize roiSize, Ipp32f vMin, Ipp32f vMax);
```

**Supported values for** `mod`:

- 32f_C3IR
- 32f_AC4IR
Case 8: In-place operation on floating-point planar data

IppStatus ippiGammaFwd_32f_IP3R (Ipp32f* pSrcDst[3], int srcDstStep, IppSize roiSize, Ipp32f vMin, Ipp32f vMax);

Include Files
ippcc.h

Domain Dependencies
Headers: ippcore.h, ippvm.h, ipps.h,ippi.h
Libraries: ippcore.lib, ippvm.lib, ipps.lib,ippi.lib

Parameters
pSrc
Pointer to the ROI in the pixel-order source image. Array of pointers to the ROI in each plane of the planar source image.
srcStep
Distance in bytes between starts of consecutive lines in the source image.
pDst
Pointer to the ROI in the pixel-order destination image. Array of pointers to the ROI in each plane of the planar destination image.
dstStep
Distance in bytes between starts of consecutive lines in the destination image.
pSrcDst
Pointer to the source and destination image ROI for the in-place operation.
srcDstStep
Distance in bytes between starts of consecutive lines in the source and destination image for the in-place operation.
roiSize
Size of the source and destination ROI in pixels.
vMin, vMax
Minimum and maximum values of the input floating-point data.

Description
This function operates with ROI (see Regions of Interest in Intel IPP).
This function performs gamma-correction of the source image with RGB data. It uses the following basic equations to convert an RGB image to the gamma-corrected R'G'B' image:

for R,G,B < 0.018
R' = 4.5 * R
G' = 4.5 * G
B' = 4.5 * B

for R,G,B ≥ 0.018
R' = 1.099 * R^{0.45} - 0.099
G' = 1.099 * G^{0.45} - 0.099
B' = 1.099 * B^{0.45} - 0.099

Note that the channel intensity values are normalized to fit in the range of [0..1]. The gamma value is equal to 1/0.45 = 2.22 in conformity with [ITU709] specification.
Return Values

ippStsNoErr
 Indicates no error. Any other value indicates an error.

ippStsNullPtrErr
 Indicates an error condition if \( pSrc, pDst, \) or \( pSrcDst \) is NULL.

ippStsSizeErr
 Indicates an error condition if \( roiSize \) has a field with a zero or negative value.

ippStsGammaRangeErr
 Indicates an error condition if the input data bounds are incorrect, that is \( vMax \) is less than or equal to \( vMin \).

GammaInv

Converts a gamma-corrected R'G'B' image back to the original RGB image.

Syntax

Case 1: Not-in-place operation on integer pixel-order data

IppStatus ippiGammaInv_<mod>(const Ipp<datatype>* pSrc, int srcStep, Ipp<datatype>* pDst, int dstStep, IppiSize roiSize);

Supported values for \( mod \):

\[\begin{align*}
8u\_C3R & & 16u\_C3R \\
8u\_AC4R & & 16u\_AC4R
\end{align*}\]

Case 2: Not-in-place operation on integer planar data

IppStatus ippiGammaInv_<mod>(const Ipp<datatype>* pSrc[3], int srcStep, Ipp<datatype>* pDst[3], int dstStep, IppiSize roiSize);

Supported values for \( mod \):

\[\begin{align*}
8u\_P3R & & 16u\_P3R
\end{align*}\]

Case 3: Not-in-place operation on floating-point pixel-order data

IppStatus ippiGammaInv_<mod>(const Ipp32f* pSrc, int srcStep, Ipp32f* pDst, int dstStep, IppiSize roiSize, Ipp32f vMin, Ipp32f vMax);

Supported values for \( mod \):

\[\begin{align*}
32f\_C3R & & 32f\_AC4R
\end{align*}\]

Case 4: Not-in-place operation on floating-point planar data

IppStatus ippiGammaInv_32f_P3R (const Ipp32f* pSrc[3], int srcStep, Ipp32f* pDst[3], int dstStep, IppiSize roiSize, Ipp32f vMin, Ipp32f vMax);

Case 5: In-place operation on integer pixel-order data

IppStatus ippiGammaInv_<mod>(Ipp<datatype>* pSrcDst, int srcDstStep, IppiSize roiSize);

Supported values for \( mod \):

\[\begin{align*}
8u\_C3IR & & 16u\_C3IR \\
8u\_AC4IR & & 16u\_AC4IR
\end{align*}\]
Case 6: In-place operation on integer planar data

IppStatus ippiGammaInv_<mod>(Ipp<datatype>* pSrcDst[3], int srcDstStep, IppiSize roiSize);

Supported values for mod:

8u_IP3R   16u_IP3R

Case 7: In-place operation on floating-point pixel-order data

IppStatus ippiGammaInv_<mod>(Ipp32f* pSrcDst, int srcDstStep, IppiSize roiSize, Ipp32f vMin, Ipp32f vMax);

Supported values for mod:

32f_C3IR
32f_AC4IR

Case 8: In-place operation on floating-point planar data

IppStatus ippiGammaInv_32f_IP3R (Ipp32f* pSrcDst[3], int srcDstStep, IppiSize roiSize, Ipp32f vMin, Ipp32f vMax);

Include Files

ippcc.h

Domain Dependencies

Headers: ippcore.h, ippvm.h, ipps.h,ippi.h

Libraries: ippcore.lib, ippvm.lib, ipps.lib,ippi.lib

Parameters

pSrc Pointer to the ROI in the pixel-order source image. Array of pointers to the ROI in each plane of the planar source image.
srcStep Distance in bytes between starts of consecutive lines in the source image.

pDst Pointer to the ROI in the pixel-order destination image. Array of pointers to the ROI in each plane of the planar destination image.
dstStep Distance in bytes between starts of consecutive lines in the destination image.

pSrcDst Pointer to the source and destination image ROI for the in-place operation.
srcDstStep Distance in bytes between starts of consecutive lines in the source and destination image for the in-place operation.
roiSize Size of the source and destination ROI in pixels.
vMin, vMax Minimum and maximum values of the input floating-point data.

Description

This function operates with ROI (see Regions of Interest in Intel IPP).

This function converts a gamma-corrected $R'G'B'$ image back to the original RGB image. It uses the following equations:

for $R',G',B' < 0.0812$
\[
R = \frac{R'}{4.5}
\]
\[
G = \frac{G'}{4.5}
\]
\[
B = \frac{B'}{4.5}
\]

for \( R', G', B' \geq 0.0812 \)

\[
R = \left( \frac{R' + 0.099}{1.099} \right)^{2.22}
\]
\[
G = \left( \frac{G' + 0.099}{1.099} \right)^{2.22}
\]
\[
B = \left( \frac{B' + 0.099}{1.099} \right)^{2.22}
\]

Note that the channel intensity values are normalized to fit in the range of [0..1]. The gamma value is equal to \( 1/0.45 = 2.22 \) in conformity with [ITU709] specification.

**Return Values**

- ippStsNoErr: Indicates no error. Any other value indicates an error.
- ippStsNullPtrErr: Indicates an error condition if \( pSrc \), \( pDst \), or \( pSrcDst \) is NULL.
- ippStsSizeErr: Indicates an error condition if \( roiSize \) has a field with a zero or negative value.
- ippStsGammaRangeErr: Indicates an error condition if the input data bounds are incorrect, that is, \( vMax \) is less than or equal to \( vMin \).

**Intensity Transformation**

The functions described in this section perform different types of intensity transformation including reduction of the intensity levels in each channel of the image, intensity transformation using lookup tables, and mapping high dynamic range image (HDRI) into low dynamic range image (LDRI).

**ReduceBitsGetBufferSize**

*Computes the size of the work buffer for the ippiReduceBits function.*

**Syntax**

\[
\text{IppStatus ippiReduceBitsGetBufferSize(IppChannels ippChan, IppiSize roiSize, int noise, IppiDitherType dtype, int* pBufferSize);}
\]

**Include Files**

cppcc.h

**Domain Dependencies**

**Headers:** ippcore.h, ippvm.h, ipps.h, ippi.h

**Libraries:** ippcore.lib, ippvm.lib, ipps.lib, ippi.lib
Parameters

ippChan
Number of channels in the source images. Possible values: ippC1, ippC3, or ippAC4.

roiSize
Size, in pixels, of the source images.

noise
Number specifying the amount of noise added (as a percentage of the range [0..100]).

dtype
Type of dithering to be used. For the list of supported types, refer to the ippiReduceBits function description.

pBufferSize
Pointer to the computed value of the buffer size, in bytes.

Description
The function computes the size of the work buffer, in bytes, for the ippiReduceBits function and stores the result in the pBufferSize parameter.

Return Values

ippStsNoErr
Indicates no error. Any other value indicates an error or a warning.

ippStsNullPtrErr
Indicates an error when any of the specified pointers is NULL.

ippStsSizeErr
Indicates an error when roiSize is less than, or equal to zero.

ippStsChannelErr
Indicates an error when ippChan has an illegal value.

ippStsDataTypeErr
Indicates an error when noise is less than 0, or greater than 100.

ippStsDataTypeErr
Indicates an error when the specified dithering type is not supported.

See Also
ReduceBits Reduces the bit resolution of an image.

ReduceBits
Reduces the bit resolution of an image.

Syntax

Case 1: Operation on data of the same source and destination bit depths

IppStatus ippiReduceBits_<mod>(const Ipp<datatype>* pSrc, int srcStep, Ipp<datatype>* pDst, int dstStep, IppiSize roiSize, int noise, IppiDitherType dtype, int levels, Ipp8u* pBuffer);

Supported values for mod:

<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>8u_C1R</td>
<td>16u_C1R</td>
<td>16s_C1R</td>
</tr>
<tr>
<td>8u_C3R</td>
<td>16u_C3R</td>
<td>16s_C3R</td>
</tr>
<tr>
<td>8u_C4R</td>
<td>16u_C4R</td>
<td>16s_C4R</td>
</tr>
</tbody>
</table>
IppStatus ippiReduceBits_<mod>(const Ipp<datatype>* pSrc, int srcStep, Ipp<datatype>* pDst, int dstStep, IppiSize roiSize, int noise, IppiDitherType dtype, int levels, Ipp8u* pBuffer);

Supported values for mod:

8u_AC4R  16u_AC4R  16s_AC4R

Case 2: Operation on data of different source and destination bit depths

IppStatus ippiReduceBits_8u1u_C1R(const Ipp8u* pSrc, int srcStep, Ipp8u* pDst, int dstStep, int dstBitOffset, IppiSize roiSize, int noise, int seed, IppiDitherType dtype, Ipp8u threshold, Ipp8u* pBuffer);

IppStatus ippiReduceBits_<mod>(const Ipp<srcDatatype>* pSrc, int srcStep, Ipp<dstDatatype>* pDst, int dstStep, IppiSize roiSize, int noise, IppiDitherType dtype, int levels, Ipp8u* pBuffer);

Supported values for mod:

16u8u_C1R  16s8u_C1R  32f8u_C1R  32f16u_C1R  32f16s_C1R
16u8u_C3R  16s8u_C3R  32f8u_C3R  32f16u_C3R  32f16s_C3R
16u8u_C4R  16s8u_C4R  32f8u_C4R  32f16u_C4R  32f16s_C4R

IppStatus ippiReduceBits_<mod>(const Ipp<srcDatatype>* pSrc, int srcStep, Ipp<dstDatatype>* pDst, int dstStep, IppiSize roiSize, int noise, IppiDitherType dtype, int levels, Ipp8u* pBuffer);

Supported values for mod:

16u8u_AC4R  16s8u_AC4R  32f8u_AC4R  32f16u_AC4R  32f16s_AC4R

Include Files

ippcc.h

Domain Dependencies

Headers: ippcore.h, ippvm.h, ipps.h,ippi.h

Libraries: ippcore.lib, ippvm.lib, ipps.lib,ippi.lib

Parameters

pSrc
Pointer to the source image ROI.

srcStep
Distance in bytes between starts of consecutive lines in the source image.

pDst
Pointer to the destination image ROI.

dstStep
Distance in bytes between starts of consecutive lines in the destination image.

dstBitOffset
Offset (in bits) in the first byte of the destination image row.

roiSize
Size of the source and destination ROI in pixels.

noise
The number specifying the amount of noise added. This parameter is set as a percentage of range [0..100].
**seed**
The seed value used by the pseudo-random number generation, should be set to 0.

**dtype**
The type of dithering to be used. The following types are supported:

- ippDitherNone: No dithering is done
- ippDitherStucki: The Stucki's error diffusion dithering algorithm is used
- ippDitherFS: The Floid-Steinberg error diffusion dithering algorithm is used
- ippDitherJJN: The Jarvice-Judice-Ninke error diffusion dithering algorithm is used
- ippDitherBayer: The Bayer's threshold dithering algorithm is used

**levels**
The number of output levels for halftoning (dithering); can be varied in the range \([2..(1<<\text{depth})]\), where \(\text{depth}\) is the bit depth of the destination image.

**threshold**
Threshold level for Stucki's dithering for the functionippiReduceBits_8u1u_C1R.

**pBuffer**
Pointer to the buffer for internal calculations. To compute the size of the buffer, use the ReduceBitsGetBufferSize function.

**Description**
This function operates with ROI (see Regions of Interest in Intel IPP).

This function reduces the number of intensity levels in each channel of the source image \(p\text{Src}\) and places the results in respective channels of the destination image \(p\text{Dst}\). Note that for floating point source data type, RGB values must be in the range \([0..1]\).

The **levels** parameter sets the resultant number of intensity levels in each channel of the destination image.

If the **noise** value is greater than 0, some random noise is added to the threshold level used in computations. The amplitude of the noise signal is specified by the **noise** parameter set as a percentage of the destination image luminance range. For the 4x4 ordered dithering mode, the threshold value is determined by the dither matrix used, whereas for the error diffusion dithering mode the input threshold is set as half of the **range** value, where

\[
\text{range} = ((1<<\text{depth}) - 1)/(\text{levels} - 1)
\]

and \(\text{depth}\) is the bit depth of the source image.

For floating-point data type, \(\text{range} = 1.0/(\text{levels} - 1)\).

**8u to 1u conversion.** Source image is converted to a bitonal image. The functionippiReduceBits_8u1u_C1R supports only one dithering algorithm - Stucki's error diffusion. The destination image has a 8u data type, where each byte represents eight consecutive pixels of the bitonal image (1 bit per pixel). In this case, additional parameter **dstBitOffset** is required to specify the start position of the destination ROI buffer.

**Return Values**

- ippStsNoErr: Indicates no error. Any other value indicates an error.
- ippStsNullPtrErr: Indicates an error condition if \(p\text{Src}\) or \(p\text{Dst}\) is NULL.
 Indicates an error condition if roiSize has a field with a zero or negative value.
Indicates an error condition if noise has an illegal value.
Indicates an error condition if the specified dithering type is not supported.
Indicates an error condition if levels value is out of admissible range.
Indicates an error condition if memory allocation fails.

See Also
ReduceBitsGetBufferSize Computes the size of the work buffer for theippiReduceBits function.

LUT_GetSize
Computes the size of the LUT specification structure.

Syntax
IppStatus ippiLUT_GetSize(IppiInterpolationType interpolation, IppDataType dataType, IppChannels channels, IppSize roiSize, const int nLevels[], int* pSpecSize);

Include Files
ippi.h

Domain Dependencies
Headers: ippcore.h, ippvm.h, ipps.h
Libraries: ippcore.lib, ippvm.lib, ipps.lib

Parameters
interpolation
Interpolation algorithm, possible values are:
ippNearest Nearest neighbor interpolation.
ippCubic Cubic interpolation.
ippLinear Linear interpolation.

datatype
Data type of the image. Possible values are ipp8u, ipp16u, ipp16s, or ipp32f.

channels
Number of channels in the image. Possible values are: ippC1, ippC3, ippC4, or ippAC4.

roiSize
Size, in pixels, of the destination ROI.

nLevels
Number of levels, separate for each channel.

pSpecSize
Pointer to the computed size, in bytes, of the specification structure.

Description
This function computes the size of the specification structure for theippiLUT function. The result is stored in the pSpecSize parameter.
For an example on how to use this function, refer to the example provided with theippiLUT function description.
Return Values

<table>
<thead>
<tr>
<th>Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ippStsNoErr</td>
<td>Indicates no error.</td>
</tr>
<tr>
<td>ippStsNullPtrErr</td>
<td>Indicates an error when any of the specified pointers is NULL.</td>
</tr>
<tr>
<td>ippStsSizeErr</td>
<td>Indicates an error when roiSize has a field with a value less than 1.</td>
</tr>
<tr>
<td>ippStsChannelErr</td>
<td>Indicates an error when channel has an illegal value.</td>
</tr>
<tr>
<td>ippStsDataTypeErr</td>
<td>Indicates an error when dataType has an illegal value.</td>
</tr>
<tr>
<td>ippStsInterpolationErr</td>
<td>Indicates an error when interpolation has an illegal value.</td>
</tr>
</tbody>
</table>

See Also

LUTMODIFIED API. Maps an image by applying intensity transformation.

LUT_Init

Initializing the LUT specification structure.

Syntax

IppStatus ippiLUT_Init_8u(IppiInterpolationType interpolation, IppChannels channels, IppiSize roiSize, const Ipp32s* pValues[], const Ipp32s* pLevels[], int nLevels[], IppiLUT_Spec* pSpec);
IppStatus ippiLUT_Init_16u(IppiInterpolationType interpolation, IppChannels channels, IppiSize roiSize, const Ipp32s* pValues[], const Ipp32s* pLevels[], int nLevels[], IppiLUT_Spec* pSpec);
IppStatus ippiLUT_Init_16s(IppiInterpolationType interpolation, IppChannels channels, IppiSize roiSize, const Ipp32s* pValues[], const Ipp32s* pLevels[], int nLevels[], IppiLUT_Spec* pSpec);
IppStatus ippiLUT_Init_32f(IppiInterpolationType interpolation, IppChannels channels, IppiSize roiSize, const Ipp32f* pValues[], const Ipp32f* pLevels[], int nLevels[], IppiLUT_Spec* pSpec);

Include Files

ippi.h

Domain Dependencies

Headers: ippcore.h, ippvm.h, ipps.h
Libraries: ippcore.lib, ippvm.lib, ipps.lib

Parameters

interpolation

<table>
<thead>
<tr>
<th>Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ippNearest</td>
<td>Nearest neighbor interpolation.</td>
</tr>
<tr>
<td>ippCubic</td>
<td>Cubic interpolation.</td>
</tr>
<tr>
<td>ippLinear</td>
<td>Linear interpolation.</td>
</tr>
</tbody>
</table>

channels

Number of channels in the image. Possible values are: ippC1, ippC3, ippC4, or ippAC4.
roiSize
pValues
pLevels
nLevels
pSpec

Description
This function initializes the specification structure for the ippiLUT function. To compute the size of the structure, use the ippiLUTGetSize function.

Length of the pLevels and pValues arrays is defined by the nLevels parameter. Number of level and intensity values are nLevels-1.

The interpolation parameter defines the mapping algorithm for the LUT function:

- **ippiNearest**: Every source pixel $pSrc(x,y)$ from the range $[pLevels[k], pLevels[k+1])$ is mapped to the destination pixel $pDst(x,y)$ which value is equal to $pValues[k]$.
- **ippiLinear**: Every source pixel $pSrc(x, y)$ from the range $[pLevels[k], pLevels[k+1])$ is mapped to the destination pixel $pDst(x, y)$ which value is computed according to the following formula:

$$pDst(x, y) = pValues[k] + (pSrc(x, y) - pLevels[k]) \times (pValues[k+1] - pValues[k]) / (pLevels[k+1] - Levels[k])$$

- **ippiCubic**: Every source pixel $pSrc(x,y)$ from the range $[pLevels[k], pLevels[k+1])$ is mapped to the destination pixel $pDst(x,y)$ which value is computed as

$$pDst(x,y) = A \times pSrc(x,y)^3 + B \times pSrc(x,y)^2 + C \times pSrc(x,y) + D.$$  

The function operates on the assumption that the cubic polynomial curve passes through the following four points:

$$([pLevels[k-1], pLevels[k-1])$$
$$([pLevels[k], pLevels[k])$$
$$([pLevels[k+1], pLevels[k+1])$$
$$([pLevels[k+2], pLevels[k+2])$$

Based on that, coefficients $A$, $B$, $C$, $D$ are computed by solving the following set of linear equations:

$$A \times pLevels[k-1]^3 + B \times pLevels[k-1]^2 + C \times pLevels[k-1] + D = pValues[k-1]$$

$$A \times pLevels[k]^3 + B \times pLevels[k]^2 + C \times pLevels[k] + D = pValues[k]$$

$$A \times pLevels[k+1]^3 + B \times pLevels[k+1]^2 + C \times pLevels[k+1] + D = pValues[k+1]$$

$$A \times pLevels[k+2]^3 + B \times pLevels[k+2]^2 + C \times pLevels[k+2] + D = pValues[k+2]$$

Pixels in the pSrc image that are not in the range $[pLevels[0], pLevels[nLevels-1])$ are copied to the pDst image without any transformation.

For an example on how to use this function, refer to the example provided with the ippiLUT function description.

Return Values

ippiStsNoErr
Indicates no error.

ippiStsNullPtrErr
Indicates an error when any of the specified pointers is NULL.
ippiStsSizeErr  Indicates an error when roiSize has a field with a value less than 1.
ippiStsChannelErr Indicates an error when channel has an illegal value.
ippiStsLUTNofLevelsErr Indicates an error when nLevels is less than 2.
ippiStsInterpolationErr Indicates an error when interpolation has an illegal value.

See Also
LUTMODIFIED API. Maps an image by applying intensity transformation.
LUTGetSize Computes the size of the LUT specification structure.

LUT
MODIFIED API. Maps an image by applying intensity transformation.

Syntax
Case 1: Not-in-place operation on one-channel integer data
IppStatusippiLUT_<mod>(const Ipp<datatype>* pSrc, int srcStep, Ipp<datatype>* pDst, int dstStep, IppiSize roiSize, IppiLUT_Spec* pSpec);

Supported values for mod:
8u_C1R    16u_C1R    16s_C1R

Case 2: Not-in-place operation on multi-channel integer data
IppStatusippiLUT_<mod>(const Ipp<datatype>* pSrc, int srcStep, Ipp<datatype>* pDst, int dstStep, IppiSize roiSize, IppiLUT_Spec* pSpec);

Supported values for mod:
8u_C3R    16u_C3R    16s_C3R
8u_AC4R    16u_AC4R    16s_AC4R

ippiLUT_<mod>(const Ipp<datatype>* pSrc, int srcStep, Ipp<datatype>* pDst, int dstStep, IppiSize roiSize, IppiLUT_Spec* pSpec);

Supported values for mod:
8u_C4R    16u_C4R    16s_C4R

Case 3: Not-in-place operation on one-channel floating-point data
IppStatusippiLUT_32f_C1R(const Ipp32f* pSrc, int srcStep, Ipp32f* pDst, int dstStep, IppiSize roiSize, IppiLUT_Spec* pSpec);

Case 4: Not-in-place operation on multi-channel floating-point data
IppStatusippiLUT_<mod>(const Ipp32f* pSrc, int srcStep, Ipp32f* pDst, int dstStep, IppiSize roiSize, IppiLUT_Spec* pSpec);

Supported values for mod:
32f_C3R
32f_AC4R
Case 5: In-place operation on one-channel integer data
IppStatus ippiLUT_<mod>(Ipp<datatype>* pSrcDst, int srcDstStep, IppiSize roiSize, IppiLUT_Spec* pSpec);

Supported values for \texttt{mod}:
- \texttt{8u\_C1IR}
- \texttt{16u\_C1IR}
- \texttt{16s\_C1IR}

Case 6: In-place operation on multi-channel integer data
IppStatus ippiLUT_<mod>(Ipp<datatype>* pSrcDst, int srcDstStep, IppiSize roiSize, IppiLUT_Spec* pSpec);

Supported values for \texttt{mod}:
- \texttt{8u\_C3IR}
- \texttt{16u\_C3IR}
- \texttt{16s\_C3IR}
- \texttt{8u\_AC4IR}
- \texttt{16u\_AC4IR}
- \texttt{16s\_AC4IR}

IppStatus ippiLUT_<mod>(Ipp<datatype>* pSrcDst, int srcDstStep, IppiSize roiSize, IppiLUT_Spec* pSpec);

Supported values for \texttt{mod}:
- \texttt{8u\_C4IR}
- \texttt{16u\_C4IR}
- \texttt{16s\_C4IR}

Case 7: In-place operation on one-channel floating-point data
IppStatus ippiLUT_32f\_C1IR(Ipp32f* pSrcDst, int srcDstStep, IppiSize roiSize, IppiLUT_Spec* pSpec);

Case 8: In-place operation on multi-channel floating-point data
IppStatus ippiLUT_<mod>(Ipp32f* pSrcDst, int srcDstStep, IppiSize roiSize, IppiLUT_Spec* pSpec);

Supported values for \texttt{mod}:
- \texttt{32f\_C3IR}
- \texttt{32f\_AC4IR}

IppStatus ippiLUT_32f\_C4IR(Ipp32f* pSrcDst, int srcDstStep, IppiSize roiSize, IppiLUT_Spec* pSpec);

Include Files
ippi.h

Domain Dependencies
Headers: ippcore.h, ippvm.h, ipps.h
Libraries: ippcore.lib, ippvm.lib, ipps.lib

Parameters
\texttt{pSrc}  
Pointer to the source image ROI.

\texttt{srcStep}  
Distance, in bytes, between the starting points of consecutive lines in the source image.
**Description**

**Important** The API of this function has been modified in Intel IPP 9.0 release.

This function operates with ROI (see Regions of Interest in Intel IPP).

Before using this function, you need to compute the size of the specification structure using the `LUT_GetSize` function and initialize the structure using `LUT_Init`.

This function performs intensity transformation of the source image `pSrc` using the lookup table (LUT) specified by the arrays `pLevels`, `pValues`, and interpolation type specified in the `LUT_Init` function when `pSpec` is initialized.

The figure below shows particular curves that are used in all the `ippiLUT` function flavors for mapping. The level values are 0, 64, 128, 192, 256; the intensity values are 20, 60, 160, 180, 230.

![Curves](image.png)

**Return Values**

- **ippStsNoErr**
  Indicates no error. Any other value indicates an error or a warning.

- **ippStsNullPtrErr**
  Indicates an error when any of the specified pointers is `NULL`.

- **ippStsSizeErr**
  Indicates an error when `roiSize` has a field with a value less than 1.
ippStsStepErr  Indicates an error when srcStep, dstStep, or srcDstStep has a zero or negative value.
ippStsBadArgErr  Indicates an error when pSpec initialization is incorrect.

Example
The code example below demonstrates how to use LUT_GetSize, LUT_Init, and ippiLUT functions.

```c
#include "ippcore.h"
#include "ippi.h"
#include "t_genPRINT.h"

void func_LUTLinear()
{
    IppStatus status;
    Ipp32f pSrc[8*8];
    int srcStep = 8*sizeof(Ipp32f);
    IppSize roiSize = {8, 8};

    Ipp32f pDst[8*8];
    int dstStep = 8*sizeof(Ipp32f);
    Ipp32f pLevels[5] = {0.0, 0.128, 0.256, 0.512, 1.0};
    const Ipp32f *ppLevels[1] = { pLevels };
    Ipp32f pValues[5] = {0.2, 0.4, 0.6, 0.8, 1.0};
    const Ipp32f *ppValues[1] = { pValues };
    int nLevels[1] = {5};
    int specSize;
    IppiLUT_Spec* pSpec;

    status = ippiImageJaehne_32f_C1R( pSrc, srcStep, roiSize );
    printf_32f_2D( "pSrc:", pSrc, roiSize, roiSize.width*sizeof(Ipp32f), status );

    ippiLUT_GetSize( ippLinear, ipp32f, ippC1, roiSize, nLevels, &specSize );
    pSpec = (IppiLUT_Spec*)ippMalloc( specSize );
    ippiLUT_Init_32f( ippLinear, ipp32f, ippC1, roiSize, ppValues, ppLevels, nLevels, pSpec );
    status = ippiLUT_32f_C1R( pSrc, srcStep, pDst, dstStep, roiSize, pSpec );
    printf_32f_2D( "pDst:", pDst, roiSize, roiSize.width*sizeof(Ipp32f), status );
    ippFree( pSpec );
}

Result:

pSrc:
0.00 0.26 0.65 0.82 0.82 0.65 0.26 0.00
0.26 0.82 1.00 0.98 0.98 1.00 0.82 0.26
0.65 1.00 0.89 0.74 0.74 0.89 1.00 0.65
0.82 0.98 0.74 0.55 0.55 0.74 0.98 0.82
0.82 0.98 0.74 0.55 0.55 0.74 0.98 0.82
0.65 1.00 0.89 0.74 0.74 0.89 1.00 0.65
0.26 0.82 1.00 0.98 0.98 1.00 0.82 0.26
0.00 0.26 0.65 0.82 0.82 0.65 0.26 0.00

pDst:
0.20 0.61 0.85 0.93 0.93 0.85 0.61 0.20
0.61 0.93 1.00 0.99 0.99 1.00 0.93 0.61
0.85 1.00 0.95 0.89 0.89 0.95 1.00 0.85
```
See Also
Regions of Interest in Intel IPP
LUTGetSize Computes the size of the LUT specification structure.
LUTInit Initializes the LUT specification structure.

LUTPalette, LUTPaletteSwap
Maps an image by applying intensity transformation in accordance with a palette table.

Syntax
Case 1: Operations on one-channel data
IppStatusippiLUTPalette_<mod>(const Ipp<srcDatatype>* pSrc, int srcStep,
Ipp<dstDatatype>* pDst, int dstStep, IppiSize roiSize, const Ipp<dstDatatype>* pTable,
int nBitSize);

Supported values for mod:

  8u_C1R     16u_C1R
  8u32u_C1R  16u8u_C1R     
              16u32u_C1R

IppStatusippiLUTPalette_<mod>(const Ipp<srcDatatype>* pSrc, int srcStep, Ipp8u* pDst,
int dstStep, IppiSize roiSize, const Ipp8u* pTable, int nBitSize);

Supported values for mod:

  8u24u_C1R   16u24u_C1R

Case 2: Operations on multi-channel data
IppStatusippiLUTPalette_<mod>(const Ipp<datatype>* pSrc, int srcStep, Ipp<datatype>*
pDst, int dstStep, IppiSize roiSize, const Ipp<datatype>* const pTable[3], int
nBitSize);

Supported values for mod:

  8u_C3R     16u_C3R
  8u_AC4R    16u_AC4R

IppStatusippiLUTPalette_<mod>(const Ipp<datatype>* pSrc, int srcStep, Ipp<datatype>*
pDst, int dstStep, IppiSize roiSize, const Ipp<datatype>* const pTable[4], int
nBitSize);

Supported values for mod:

  8u_C4R     16u_C4R

Include Files
ippi.h
Domain Dependencies

Headers: ippcore.h, ippvm.h, ipps.h
Libraries: ippcore.lib, ippvm.lib, ipps.lib

Parameters

- **pSrc**: Pointer to the source image ROI.
- **srcStep**: Distance in bytes between starts of consecutive lines in the source image.
- **pDst**: Pointer to the destination image ROI.
- **dstStep**: Distance in bytes between starts of consecutive lines in the destination image.
- **roiSize**: Size of the source ROI in pixels.
- **pTable**: Pointer to the palette table, or an array of pointers to the palette tables for each source channel.
- **nBitSize**: Number of significant bits in the source image.

Description

This function operates with ROI (see Regions of Interest in Intel IPP).

The function `ippiLUTPalette` performs intensity transformation of the source image `pSrc` using the palette lookup table `pTable`. This table is a vector with $2^{nBitSize}$ elements that contain intensity values specified by the user. The function uses `nBitSize` lower bits of intensity value of each source pixel as an index in the `pTable` and assigns the correspondent intensity value from the table to the respective pixel in the destination image `pDst`. The number of significant bits `nBitSize` should be in the range $[1, 8]$ for functions that operate on `8u` source images, and $[1, 16]$ for functions that operate on `16u` source images.

Some function flavors that operate on the 3-channel source image additionally create a 4-th channel - alpha channel - in the destination image and place it at first position. The channel values of the alpha channel can be set to the arbitrary constant value `alphaValue`. If this value is less than 0 or greater than the upper boundary of the data range, the channel values are not set.

The function flavor `ippiLUTPaletteSwap` reverses the order of channels in the destination image.

Return Values

- **ippStsNoErr**: Indicates no error. Any other value indicates an error or a warning.
- **ippStsNullPtrErr**: Indicates an error when any of the specified pointers is NULL.
- **ippStsSizeErr**: Indicates an error condition if `roiSize` has a field with a zero or negative value.
- **ippStsOutOfRangeErr**: Indicates an error if `nBitSize` is out of the range.

Example

The code example below shows how to use the function `ippiLUTPalette_8u32u_C1R`.

```c
#include "ippcore.h"
#include "ippi.h"
#include "t_genPRINT.h"
```
void func_LUTPalette()
{
    IppStatus status;
    Ipp8u pSrc[8*8];
    int srcStep = 8*sizeof(Ipp8u);
    IppiSize roiSize = {8, 8};

    Ipp32u pDst[8*8];
    int dstStep = 8*sizeof(Ipp32f);
    int nBitSize = 3;
    Ipp32u pTable[8] ={1, 2, 3, 4, 5, 6, 7, 8} ;

    status = ippiImageJaehne_8u_C1R( pSrc, srcStep, roiSize);
    printf_8u_2D( "pSrc:", pSrc, roiSize, srcStep, status );

    status = ippiLUTPalette_8u32u_C1R( pSrc, srcStep, pDst, dstStep, roiSize, pTable, nBitSize);
    printf_32u_2D( "pDst:", pDst, roiSize, dstStep, status );
}

Result:

pSrc:
0 67 165 209 209 165 67 0
67 209 255 250 250 255 209 67
165 255 226 188 188 226 255 165
209 250 188 140 140 188 250 209
209 250 188 140 140 188 250 209
165 255 226 188 188 226 255 165
67 209 255 250 250 255 209 67
0 67 165 209 209 165 67 0

pDst:
1 4 6 2 2 6 4 1
4 2 8 3 3 8 2 4
6 8 3 5 5 3 8 6
2 3 5 5 5 5 3 2
2 3 5 5 5 5 3 2
6 8 3 5 5 3 8 6
4 2 8 3 3 8 2 4
1 4 6 2 2 6 4 1

ToneMapLinear, ToneMapMean
Maps an HDRI image to the LDRI image.

Syntax
IppStatus ippiToneMapLinear_32f8u_C1R(const Ipp32f* pSrc, int srcStep, Ipp8u* pDst, int dstStep, IppiSize roiSize);
IppStatus ippiToneMapMean_32f8u_C1R(const Ipp32f* pSrc, int srcStep, Ipp8u* pDst, int dstStep, IppiSize roiSize);

Include Files
ippcc.h

Domain Dependencies
Headers: ippcore.h, ippvm.h, ipps.h, ippi.h
Libraries: ippcore.lib, ippvm.lib, ipps.lib,ippi.lib

Parameters

\( pSrc \)  
Pointer to the ROI in the HDRI source image.

\( srcStep \)  
Distance in bytes between starts of consecutive lines in the source image.

\( pDst \)  
Pointer to the ROI in the LDRI destination image.

\( dstStep \)  
Distance in bytes between starts of consecutive lines in the destination image.

\( roiSize \)  
Size of the source and destination ROI in pixels.

Description

They both operate with ROI (see Regions of Interest in Intel IPP).

These functions convert the source high dynamic range image (HDRI) \( pSrc \) into low dynamic range image (LDRI) \( pDst \). Pixel values of the source image must be positive.

The function ippiToneMapLinear implements the Linear Scale-Factor method converting each source pixel \( pSrc[i] \) in accordance with the formula:

\[
pSrc[i] = pSrc[i]/Lmax, \quad Lmax = \max\{pSrc[i]\}.
\]

The function ippiToneMapMean implements the Mean Value method converting each source pixel \( pSrc[i] \) in accordance with the formula:

\[
pSrc[i] = 0.5\cdot pSrc[i]/Lave, \quad Lave = \text{average}\{pSrc[i]\}.
\]

If the value of \( Lmax \) or \( Lave \) is less than 0, then the function does not perform the operation and returns the warning message.

Return Values

\( ippStsNoErr \)  
Indicates no error. Any other value indicates an error or a warning.

\( ippStsNullPtrErr \)  
Indicates an error condition if \( pSrc \) or \( pDst \) is NULL.

\( ippStsSizeErr \)  
Indicates an error condition if \( roiSize \) has a field with a zero or negative value.

\( ippStsNoOperation \)  
Indicates a warning if the values of \( Lmax \) or \( Lave \) are less than 0.
Threshold and Compare Operations

This chapter describes the Intel® IPP image processing functions that operate on a pixel-by-pixel basis: threshold and compare functions.

Thresholding

The threshold functions change pixel values depending on whether they are less or greater than the specified threshold.

The type of comparison operation used to threshold pixel values is specified by the ippCmpOp parameter; this operation can be either “greater than” or “less than” (see Structures and Enumerators in Chapter 2 for more information). For some thresholding functions the type of comparison operation is fixed.

If an input pixel value satisfies the compare condition, the corresponding output pixel is set to the fixed value that is specific for a given threshold function flavor. Otherwise, it is either not changed, or set to another fixed value, which is defined in a particular function description.

For images with multi-channel data, the compare conditions should be set separately for each channel.

Threshold

Performs thresholding of pixel values in an image buffer.

Syntax

Case 1: Not-in-place operation on one-channel data

IppStatus ippiThreshold_<mod>(const Ipp<datatype>* pSrc, int srcStep, Ipp<datatype>* pDst, int dstStep, IppiSize roiSize, Ipp<datatype> threshold, IppCmpOp ippCmpOp);

Supported values for mod:

- 8u_C1R
- 16u_C1R
- 16s_C1R
- 32f_C1R

Case 2: Not-in-place operation on multi-channel data

IppStatus ippiThreshold_<mod>(const Ipp<datatype>* pSrc, int srcStep, Ipp<datatype>* pDst, int dstStep, IppiSize roiSize, const Ipp<datatype> threshold[3], IppCmpOp ippCmpOp);

Supported values for mod:

- 8u_C3R
- 16u_C3R
- 16s_C3R
- 32f_C3R
- 8u_AC4R
- 16u_AC4R
- 16s_AC4R
- 32f_AC4R
Case 3: In-place operation on one-channel data

IppStatusippiThreshold_<mod>(Ipp<datatype>* pSrcDst, int srcDstStep, IppiSize roiSize, Ipp<datatype> threshold, IppCmpOp ippCmpOp);

Supported values for mod:

8u_C1IR  16u_C1IR  16s_C1IR  32f_C1IR

Case 4: In-place operation on multi-channel data

IppStatusippiThreshold_<mod>(Ipp<datatype>* pSrcDst, int srcDstStep, IppiSize roiSize, const Ipp<datatype> threshold[3], IppCmpOp ippCmpOp);

Supported values for mod:

8u_C3IR  16u_C3IR  16s_C3IR  32f_C3IR
8u_AC4IR  16u_AC4IR  16s_AC4IR  32f_AC4IR

Include Files
ippi.h

Domain Dependencies
Headers: ippcore.h, ippvm.h, ipps.h
Libraries: ippcore.lib, ippvm.lib, ipps.lib

Parameters

pSrc  Pointer to the source image ROI.
srcStep  Distance in bytes between starts of consecutive lines in the source image.
Pdst  Pointer to the destination image ROI.
dstStep  Distance in bytes between starts of consecutive lines in the destination image.
PSrcDst  Pointer to the source and destination image ROI (for the in-place operation).
SrcDstStep  Distance in bytes between starts of consecutive lines in the source and destination image buffer (for the in-place operation).
roiSize  Size of the source and destination ROI in pixels.
threshold  The threshold level value to use for each pixel. In case of multi-channel data, an array of threshold values for each color channel is used.
ippCmpOp  The operation specified for comparing pixel values and the threshold. Comparison for either “less than” or “greater than” can be used.

Description

This function operates with ROI (see Regions of Interest in Intel IPP).

This function thresholds pixels in the source image pSrc using the specified level threshold. Pixel values in the source image are compared to the threshold value according to the type of comparison operation specified in the ippCmpOp. The following values for ippCmpOp are possible:
• **ippCmpLess** specifies the “less than” comparison and defines the **threshold** value as a lower bound. Comparison is performed by the following formula:

\[
p_{\text{Dst}}[n] = \begin{cases} 
\text{threshold}, & \text{if } p_{\text{Src}}[n] < \text{threshold} \\
p_{\text{Src}}[n], & \text{otherwise}
\end{cases}
\]

• **ippCmpGreater** specifies the “greater than” comparison and defines the **threshold** value as an upper bound. Comparison is performed by the following formula:

\[
p_{\text{Dst}}[n] = \begin{cases} 
\text{threshold}, & \text{if } p_{\text{Src}}[n] > \text{threshold} \\
p_{\text{Src}}[n], & \text{otherwise}
\end{cases}
\]

If the result of comparison is true, the corresponding output pixel is set to the **threshold** value. Otherwise, it is set to the source pixel value.

**Return Values**

- **ippStsNoErr** Indicates no error. Any other value indicates an error or a warning.
- **ippStsNullPtrErr** Indicates an error condition if one of the specified pointer is NULL.
- **ippStsSizeErr** Indicates an error condition if roiSize has a field with zero or negative value.
- **ippStsStepErr** Indicates an error condition if srcStep, dstStep, or srcDstStep has a zero or negative value.
- **ippStsNotSupportedModeErr** Indicates an error if the comparison mode is not supported.

**Example**

The code example below shows how to use the **ippiThreshold_8u_C1R** function.

```c
void func_threshold()
{
    IppiSize ROI = {5,4};
    Ipp8u src[9*4] = {1, 2, 4, 8, 16, 8, 4, 2, 1,
                      1, 2, 4, 8, 16, 8, 4, 2, 1,
                      1, 2, 4, 8, 16, 8, 4, 2, 1,
                      1, 2, 4, 8, 16, 8, 4, 2, 1};
    Ipp8u dst[9*4];
    Ipp8u threshold = 6;
    ippiThreshold_8u_C1R(src, 9, dst, 9, ROI, threshold, ippCmpGreater);
}
```

**Result:**

<table>
<thead>
<tr>
<th>dst</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 2 4 6 6 8 4 2 1</td>
</tr>
<tr>
<td>1 2 4 6 6 8 4 2 1</td>
</tr>
<tr>
<td>1 2 4 6 6 8 4 2 1</td>
</tr>
<tr>
<td>1 2 4 6 6 8 4 2 1</td>
</tr>
</tbody>
</table>
**Threshold_GT**

*Performs thresholding of pixel values in an image, using the comparison for "greater than".*

**Syntax**

**Case 1: Not-in-place operation on one-channel data**

\[
\text{IppStatus ippiThreshold_GT_<mod>}(\text{const Ipp<datatype>* pSrc, int srcStep, Ipp<datatype>* pDst, int dstStep, IppSize roiSize, Ipp<datatype> threshold});
\]

Supported values for \texttt{mod}:

- 8u\_C1R
- 16u\_C1R
- 16s\_C1R
- 32f\_C1R

**Case 2: Not-in-place operation on multi-channel data**

\[
\text{IppStatus ippiThreshold_GT_<mod>}(\text{const Ipp<datatype>* pSrc, int srcStep, Ipp<datatype>* pDst, int dstStep, IppSize roiSize, const Ipp<datatype> threshold[3]});
\]

Supported values for \texttt{mod}:

- 8u\_C3R
- 16u\_C3R
- 16s\_C3R
- 32f\_C3R
- 8u\_AC4R
- 16u\_AC4R
- 16s\_AC4R
- 32f\_AC4R

**Case 3: In-place operation on one-channel data**

\[
\text{IppStatus ippiThreshold_GT_<mod>}(\text{Ipp<datatype>* pSrcDst, int srcDstStep, IppSize roiSize, Ipp<datatype> threshold});
\]

Supported values for \texttt{mod}:

- 8u\_C1IR
- 16u\_C1IR
- 16s\_C1IR
- 32f\_C1IR

**Case 4: In-place operation on multi-channel data**

\[
\text{IppStatus ippiThreshold_GT_<mod>}(\text{Ipp<datatype>* pSrcDst, int srcDstStep, IppSize roiSize, const Ipp<datatype> threshold[3]});
\]

Supported values for \texttt{mod}:

- 8u\_C3IR
- 16u\_C3IR
- 16s\_C3IR
- 32f\_C3IR
- 8u\_AC4IR
- 16u\_AC4IR
- 16s\_AC4IR
- 32f\_AC4IR

**Include Files**

ippi.h

**Domain Dependencies**

*Headers*: ippcore.h, ippvm.h, ipps.h

*Libraries*: ippcore.lib, ippvm.lib, ipps.lib
**Parameters**

- **pSrc**: Pointer to the source image ROI.
- **srcStep**: Distance in bytes between starts of consecutive lines in the source image.
- **pDst**: Pointer to the destination image ROI.
- **dstStep**: Distance in bytes between starts of consecutive lines in the destination image.
- **pSrcDst**: Pointer to the source and destination image ROI (for the in-place operation).
- **srcDstStep**: Distance in bytes between starts of consecutive lines in the source and destination image buffer (for the in-place operation).
- **roiSize**: Size of the source and destination ROI in pixels.
- **threshold**: The threshold level value to use for each pixel. In case of multi-channel data, an array of threshold values for each color channel is used.

**Description**

This function operates with ROI (see Regions of Interest in Intel IPP).

This function performs thresholding of pixels in the source image `pSrc` using the specified level `threshold`. Pixel values in the source image are compared to the `threshold` value for “greater than”.

If the result of the compare is true, the corresponding output pixel is set to the `threshold` value. Otherwise, it is set to the source pixel value.

**Return Values**

- **ippStsNoErr**: Indicates no error. Any other value indicates an error or a warning.
- **ippStsNullPtrErr**: Indicates an error condition if one of the specified pointer is NULL.
- **ippStsSizeErr**: Indicates an error condition if `roiSize` has a field with zero or negative value.
- **ippStsStepErr**: Indicates an error condition if `srcStep`, `dstStep`, or `srcDstStep` has a zero or negative value.

**Threshold_LT**

*Performs thresholding of pixel values in an image buffer, using the comparison for “less than”.*

**Syntax**

**Case 1: Not-in-place operation on one-channel data**

```c
IppStatus ippiThreshold_LT_<mod>(const Ipp<datatype>* pSrc, int srcStep, Ipp<datatype>* pDst, int dstStep, IppSize roiSize, Ipp<datatype> threshold);
```

Supported values for `mod`:

- `8u_C1R`
- `16u_C1R`
- `16s_C1R`
- `32f_C1R`
Case 2: Not-in-place operation on multi-channel data

IppStatus ippiThreshold_LT_<mod>(const Ipp<datatype>* pSrc, int srcStep, Ipp<datatype>* pDst, int dstStep, IppiSize roiSize, const Ipp<datatype> threshold[3]);

Supported values for mod:

8u_C3R     16u_C3R     16s_C3R     32f_C3R
8u_AC4R    16u_AC4R    16s_AC4R    32f_AC4R

Case 3: In-place operation on one-channel data

IppStatus ippiThreshold_LT_<mod>(Ipp<datatype>* pSrcDst, int srcDstStep, IppiSize roiSize, Ipp<datatype> threshold);

Supported values for mod:

8u_C1IR    16u_C1IR    16s_C1IR    32f_C1IR

Case 4: In-place operation on multi-channel data

IppStatus ippiThreshold_LT_<mod>(Ipp<datatype>* pSrcDst, int srcDstStep, IppiSize roiSize, const Ipp<datatype> threshold[3]);

Supported values for mod:

8u_C3IR    16u_C3IR    16s_C3IR    32f_C3IR
8u_AC4IR   16u_AC4IR   16s_AC4IR   32f_AC4IR

Include Files
ippi.h

Domain Dependencies

Headers: ippcore.h, ippvm.h, ipps.h
Libraries: ippcore.lib, ippvm.lib, ipps.lib

Parameters

pSrc     Pointer to the source image ROI.
srcStep  Distance in bytes between starts of consecutive lines in the source image.
pDst     Pointer to the destination image ROI.
dstStep  Distance in bytes between starts of consecutive lines in the destination image.
pSrcDst  Pointer to the source and destination image ROI (for the in-place operation).
srcDstStep Distance in bytes between starts of consecutive lines in the source and destination image buffer (for the in-place operation).
roiSize  Size of the source and destination ROI in pixels.
**threshold**

The threshold level value to use for each pixel. In case of multi-channel data, an array of threshold values for each color channel is used.

**Description**

This function operates with ROI (see Regions of Interest in Intel IPP).

This function performs thresholding of pixels in the source image `pSrc` using the specified level `threshold`. Pixel values in the source image are compared to the `threshold` value for “less than”. If the result of the compare is true, the corresponding output pixel is set to the `threshold` value. Otherwise, it is set to the source pixel value.

**Return Values**

- **ippStsNoErr**
  - Indicates no error. Any other value indicates an error or a warning.
- **ippStsNullPtrErr**
  - Indicates an error condition if one of the specified pointer is NULL.
- **ippStsSizeErr**
  - Indicates an error condition if `roiSize` has a field with zero or negative value.
- **ippStsStepErr**
  - Indicates an error condition if `srcStep`, `dstStep`, or `srcDstStep` has a zero or negative value.

**Threshold_Val**

Performs thresholding of pixel values in an image buffer. Pixels that satisfy the compare condition are set to a specified value.

**Syntax**

**Case 1: Not-in-place operation on one-channel data**

```c
IppStatus ippiThreshold_Val_<mod>(const Ipp<datatype>* pSrc, int srcStep,
Ipp<datatype>* pDst, int dstStep, IppiSize roiSize, Ipp<datatype> threshold,
Ipp<datatype> value, IppCmpOp ippCmpOp);
```

Supported values for `mod`:

- 8u_C1R
- 16u_C1R
- 16s_C1R
- 32f_C1R

**Case 2: Not-in-place operation on multi-channel data**

```c
IppStatus ippiThreshold_Val_<mod>(const Ipp<datatype>* pSrc, int srcStep,
Ipp<datatype>* pDst, int dstStep, IppiSize roiSize, const Ipp<datatype> threshold[3],
const Ipp<datatype> value[3], IppCmpOp ippCmpOp);
```

Supported values for `mod`:

- 8u_C3R
- 16u_C3R
- 16s_C3R
- 32f_C3R
- 8u_AC4R
- 16u_AC4R
- 16s_AC4R
- 32f_AC4R
Case 3: In-place operation on one-channel data
IppStatus ippiThreshold_Val_<mod>(Ipp<datatype>* pSrcDst, int srcDstStep, IppiSize roiSize, Ipp<datatype> threshold, Ipp<datatype> value, IppCmpOp ippCmpOp);

Supported values for mod:
8u_C1IR 16u_C1IR 16s_C1IR 32f_C1IR

Case 4: In-place operation on multi-channel data
IppStatus ippiThreshold_Val_<mod>(Ipp<datatype>* pSrcDst, int srcDstStep, IppiSize roiSize, const Ipp<datatype> threshold[3], const Ipp<datatype> value[3], IppCmpOp ippCmpOp);

Supported values for mod:
8u_C3IR 16u_C3IR 16s_C3IR 32f_C3IR
8u_AC4IR 16u_AC4IR 16s_AC4IR 32f_AC4IR

Include Files
ippi.h

Domain Dependencies
Headers: ippicore.h, ippvm.h, ipps.h
Libraries: ippcore.lib, ippvm.lib, ipps.lib

Parameters
pSrc
Pointer to the source image ROI.
srcStep
Distance in bytes between starts of consecutive lines in the source image.
pDst
Pointer to the destination image ROI.
dstStep
Distance in bytes between starts of consecutive lines in the destination image.
pSrcDst
Pointer to the source and destination image ROI (for the in-place operation).
srcDstStep
Distance in bytes between starts of consecutive lines in the source and destination image buffer (for the in-place operation).
roiSize
Size of the source and destination ROI in pixels.
threshold
The threshold value to use for each pixel. In case of multi-channel data, an array of 3 threshold values (one for each color channel) is used.
value
The output value to be set for each pixel that satisfies the compare condition. In case of multi-channel data, an array of 3 output values (one for each color channel) is used.
ippCmpOp
The operation to use for comparing pixel values and the threshold. Comparison for either "less than" or "greater than" can be used.

Description
This function operates with ROI (see Regions of Interest in Intel IPP).
This function thresholds pixels in the source image \( pSrc \) using the specified level \( \text{threshold} \). Pixel values in the source image are compared to the \( \text{threshold} \) value using the \( \text{ippCmpOp} \) comparison operation. If the result of the compare is true, the corresponding output pixel is set to the specified \( \text{value} \). Otherwise, it is set to the source pixel value.

**Return Values**

- \( \text{ippStsNoErr} \): Indicates no error. Any other value indicates an error or a warning.
- \( \text{ippStsNullPtrErr} \): Indicates an error condition if one of the specified pointer is NULL.
- \( \text{ippStsSizeErr} \): Indicates an error condition if \( \text{roiSize} \) has a field with zero or negative value.
- \( \text{ippStsStepErr} \): Indicates an error condition if \( \text{srcStep}, \text{dstStep}, \) or \( \text{srcDstStep} \) has a zero or negative value.

**Threshold_GTVal**

*Performs thresholding of pixel values in an image.*

*Pixels that are greater than threshold, are set to a specified value.*

**Syntax**

**Case 1: Not-in-place operation on one-channel data**

\[
\text{IppStatus ippiThreshold GTVal <mod>}(\text{const Ipp<datatype>* pSrc, int srcStep, Ipp<datatype>* pDst, int dstStep, IppiSize roiSize, Ipp<datatype> threshold, Ipp<datatype> value});
\]

Supported values for \( \text{mod} \):

- 8u_C1R
- 16u_C1R
- 16s_C1R
- 32f_C1R

**Case 2: Not-in-place operation on multi-channel data**

\[
\text{IppStatus ippiThreshold GTVal <mod>}(\text{const Ipp<datatype>* pSrc, int srcStep, Ipp<datatype>* pDst, int dstStep, IppiSize roiSize, const Ipp<datatype> threshold[3], const Ipp<datatype> value[3]});
\]

Supported values for \( \text{mod} \):

- 8u_C3R
- 16u_C3R
- 16s_C3R
- 32f_C3R

- 8u_AC4R
- 16u_AC4R
- 16s_AC4R
- 32f_AC4R

\[
\text{IppStatus ippiThreshold GTVal <mod>}(\text{const Ipp<datatype>* pSrc, int srcStep, Ipp<datatype>* pDst, int dstStep, IppiSize roiSize, const Ipp<datatype> threshold[4], const Ipp<datatype> value[4]});
\]

Supported values for \( \text{mod} \):

- 8u_C4R
- 16u_C4R
- 16s_C4R
- 32f_C4R
Case 3: In-place operation on one-channel data

IppStatus ippiThreshold_GTVal_<mod>(Ipp<datatype>* pSrcDst, int srcDstStep, IppiSize roiSize, Ipp<datatype> threshold, Ipp<datatype> value);

Supported values for mod:

8u_C1IR   16u_C1IR   16s_C1IR   32f_C1IR

Case 4: In-place operation on multi-channel data

IppStatus ippiThreshold_GTVal_<mod>(Ipp<datatype>* pSrcDst, int srcDstStep, IppiSize roiSize, const Ipp<datatype> threshold[3], const Ipp<datatype> value[3]);

Supported values for mod:

8u_C3IR   16u_C3IR   16s_C3IR   32f_C3IR
8u_AC4IR   16u_AC4IR   16s_AC4IR   32f_AC4IR

IppStatus ippiThreshold_GTVal_<mod>(Ipp<datatype>* pSrcDst, int srcDstStep, IppiSize roiSize, const Ipp<datatype> threshold[4], const Ipp<datatype> value[4]);

Supported values for mod:

8u_C4IR   16u_C4IR   16s_C4IR   32f_C4IR

Include Files

ippi.h

Domain Dependencies

Headers: ippcore.h, ippvm.h, ipps.h
Libraries: ippcore.lib, ippvm.lib, ipps.lib

Parameters

- **pSrc**: Pointer to the source image ROI.
- **srcStep**: Distance in bytes between starts of consecutive lines in the source image.
- **pDst**: Pointer to the destination image ROI.
- **dstStep**: Distance in bytes between starts of consecutive lines in the destination image.
- **pSrcDst**: Pointer to the source and destination image ROI (for the in-place operation).
- **srcDstStep**: Distance in bytes between starts of consecutive lines in the source and destination image buffer (for the in-place operation).
- **roiSize**: Size of the source and destination ROI in pixels.
- **threshold**: The threshold value to use for each pixel. In case of multi-channel data, an array of threshold values (one for each channel) is used.
- **value**: The output value to be set for each pixel that satisfies the compare condition. In case of multi-channel data, an array of output values (one for each channel) is used.
Description
This function operates with ROI (see Regions of Interest in Intel IPP).

This function thresholds pixels in the source image \texttt{pSrc} using the specified level \textit{threshold}. Pixel values in the source image are compared to the \textit{threshold} value for "greater than".

If the result of the compare is true, the corresponding output pixel is set to the specified \textit{value}. Otherwise, it is set to the source pixel value.

Return Values

\begin{description}
\item[ippStsNoErr] Indicates no error. Any other value indicates an error or a warning.
\item[ippStsNullPtrErr] Indicates an error condition if one of the specified pointer is NULL.
\item[ippStsSizeErr] Indicates an error condition if \texttt{roiSize} has a field with zero or negative value.
\item[ippStsStepErr] Indicates an error condition if \texttt{srcStep}, \texttt{dstStep}, or \texttt{srcDstStep} has a zero or negative value.
\end{description}

Threshold\_LTVal

Perform thresholding of pixel values in an image. Pixels that are less than threshold, are set to a specified value.

Syntax

Case 1: Not-in-place operation on one-channel data

\begin{verbatim}
IppStatus ippiThreshold_LTVal_<mod>(const Ipp<datatype>* pSrc, int srcStep, Ipp<datatype>* pDst, int dstStep, IppiSize roiSize, Ipp<datatype> threshold, Ipp<datatype> value);
\end{verbatim}

Supported values for \texttt{mod}:

- 8u\_C1R
- 16u\_C1R
- 16s\_C1R
- 32f\_C1R

Case 2: Not-in-place operation on multi-channel data

\begin{verbatim}
IppStatus ippiThreshold_LTVal_<mod>(const Ipp<datatype>* pSrc, int srcStep, Ipp<datatype>* pDst, int dstStep, IppiSize roiSize, const Ipp<datatype> threshold[3], const Ipp<datatype> value[3]);
\end{verbatim}

Supported values for \texttt{mod}:

- 8u\_C3R
- 16u\_C3R
- 16s\_C3R
- 32f\_C3R
- 8u\_AC4R
- 16u\_AC4R
- 16s\_AC4R
- 32f\_AC4R
IppStatusippiThreshold_LTVal_<mod>(const Ipp<datatype>* pSrc, int srcStep, Ipp<datatype>* pDst, int dstStep, IppiSize roiSize, const Ipp<datatype> threshold[4], const Ipp<datatype> value[4]);

Supported values for mod:

8u_C4R   16u_C4R   16s_C4R   32f_C4R

Case 3: In-place operation on one-channel data

IppStatusippiThreshold_LTVal_<mod>(Ipp<datatype>* pSrcDst, int srcDstStep, IppiSize roiSize, Ipp<datatype> threshold, Ipp<datatype> value);

Supported values for mod:

8u_C1IR   16u_C1IR   16s_C1IR   32f_C1IR

Case 4: In-place operation on multi-channel data

IppStatusippiThreshold_LTVal_<mod>(Ipp<datatype>* pSrcDst, int srcDstStep, IppiSize roiSize, const Ipp<datatype> threshold[3], const Ipp<datatype> value[3]);

Supported values for mod:

8u_C3IR   16u_C3IR   16s_C3IR   32f_C3IR
8u_AC4IR  16u_AC4IR  16s_AC4IR  32f_AC4IR

IppStatusippiThreshold_LTVal_<mod>(Ipp<datatype>* pSrcDst, int srcDstStep, IppiSize roiSize, const Ipp<datatype> threshold[4], const Ipp<datatype> value[4]);

Supported values for mod:

8u_C4IR  16u_C4IR  16s_C4IR  32f_C4IR

Include Files

ippi.h

Domain Dependencies

Headers: ippcore.h, ippvm.h, ipps.h
Libraries: ippcore.lib, ippvm.lib, ipps.lib

Parameters

pSrc

Pointer to the source image ROI.

srcStep

Distance in bytes between starts of consecutive lines in the source image.

pDst

Pointer to the destination image ROI.

dstStep

Distance in bytes between starts of consecutive lines in the destination image.

pSrcDst

Pointer to the source and destination image ROI (for the in-place operation).
Threshold and Compare Operations

srcDstStep
Distance in bytes between starts of consecutive lines in the source and destination image buffer (for the in-place operation).

roiSize
Size of the source and destination ROI in pixels.

threshold
The threshold value to use for each pixel. In case of multi-channel data, an array of threshold values (one for each channel) is used.

value
The output value to be set for each pixel that satisfies the compare condition. In case of multi-channel data, an array of output values (one for each channel) is used.

Description
This function operates with ROI (see Regions of Interest in Intel IPP).

This function thresholds pixels in the source image pSrc using the specified level threshold. Pixel values in the source image are compared to the threshold value for "less than". If the result of the compare is true, the corresponding output pixel is set to the specified value. Otherwise, it is set to the source pixel value.

Return Values
ippStsNoErr
Indicates no error. Any other value indicates an error or a warning.

ippStsNullPtrErr
Indicates an error condition if one of the specified pointer is NULL.

ippStsSizeErr
Indicates an error condition if roiSize has a field with zero or negative value.

ippStsStepErr
Indicates an error condition if srcStep, dstStep, or srcDstStep has a zero or negative value.

Threshold_LTValGTVal
Performs double thresholding of pixel values in an image buffer.

Syntax
Case 1: Not-in-place operation on one-channel data

IppStatus ippiThreshold_LTValGTVal_<mod>(const Ipp<datatype>* pSrc, int srcStep, Ipp<datatype>* pDst, int dstStep, IppiSize roiSize, Ipp<datatype> thresholdLT, Ipp<datatype> valueLT, Ipp<datatype> thresholdGT, Ipp<datatype> valueGT);

Supported values for mod:

8u_C1R  16u_C1R  16s_C1R  32f_C1R

Case 2: Not-in-place operation on multi-channel data

IppStatus ippiThreshold_LTValGTVal_<mod>(const Ipp<datatype>* pSrc, int srcStep, Ipp<datatype>* pDst, int dstStep, IppiSize roiSize, const Ipp<datatype> thresholdLT[3], const Ipp<datatype> valueLT[3], const Ipp<datatype> thresholdGT[3], const Ipp<datatype> valueGT[3]);

Supported values for mod:

8u_C3R  16u_C3R  16s_C3R  32f_C3R
Case 3: In-place operation on one-channel data

IppStatus ippiThreshold_LTValGTVal_<mod>(Ipp<datatype>* pSrcDst, int srcDstStep, IppSize roiSize, Ipp<datatype> thresholdLT, Ipp<datatype> valueLT, Ipp<datatype> thresholdGT, Ipp<datatype> valueGT);

Supported values for mod:

8u_C1IR  16u_C1IR  16s_C1IR  32f_C1IR

Case 4: In-place operation on multi-channel data

IppStatus ippiThreshold_LTValGTVal_<mod>(Ipp<datatype>* pSrcDst, int srcDstStep, IppSize roiSize, const Ipp<datatype> thresholdLT[3], const Ipp<datatype> valueLT[3], const Ipp<datatype> thresholdGT[3], const Ipp<datatype> valueGT[3]);

Supported values for mod

8u_C3IR  16u_C3IR  16s_C3IR  32f_C3IR
8u_AC4IR  16u_AC4IR  16s_AC4IR  32f_AC4IR

Include Files

ippi.h

Domain Dependencies

Headers: ippcore.h, ippvm.h, ipps.h
Libraries: ippcore.lib, ippvm.lib, ipps.lib

Parameters

pSrc Pointer to the source image ROI.
srcStep Distance in bytes between starts of consecutive lines in the source image.
pDst Pointer to the destination image ROI.
dstStep Distance in bytes between starts of consecutive lines in the destination image.
pSrcDst Pointer to the source and destination image ROI (for the in-place operation).
srcDstStep Distance in bytes between starts of consecutive lines in the source and destination image buffer (for the in-place operation).
roiSize Size of the source and destination ROI in pixels.
thresholdLT The lower threshold value to use for each pixel. In case of multi-channel data, an array of three lower threshold values (one for each color channel) is used.
valueLT The lower output value to be set for each pixel that is less than thresholdLT. In case of multi-channel data, an array of 3 lower output values (one for each color channel) is used.
thresholdGT

The upper threshold value to use for each pixel. In case of multi-channel data, an array of three upper threshold values (one for each color channel) is used.

valueGT

The upper output value to be set for each pixel that exceeds thresholdGT. In case of multi-channel data, an array of three upper output values (one for each color channel) is used.

Description
This function operates with ROI (see Regions of Interest in Intel IPP).

This function thresholds pixels in the source image pSrc using two specified levels thresholdLT and thresholdGT. Pixel values in the source image are compared to these levels. If the pixel value is less than thresholdLT, the corresponding output pixel is set to valueLT. If the pixel value is greater than thresholdGT, the output pixel is set to valueGT. Otherwise, it is set to the source pixel value. The value of thresholdLT should be less than or equal to thresholdGT.

Return Values
ippStsNoErr
Indicates no error. Any other value indicates an error or a warning.

ippStsNullPtrErr
Indicates an error condition if one of the specified pointer is NULL.

ippStsSizeErr
Indicates an error condition if roiSize has a field with zero or negative value.

ippStsThresholdErr
Indicates an error when thresholdLT is greater than thresholdGT.

ippStsStepErr
Indicates an error condition if srcStep, dstStep, or srcDstStep has a zero or negative value.

Example
The code example below illustrates thresholding with two levels.

```c
IppStatus threshold( void ) {
    Ipp8u x[5*4];
    IppiSize roi = {5,4};
    int i;

    for( i=0; i<5*4; ++i ) x[i] = (Ipp8u)i;

    return ippiThreshold_LTValGTVal_8u_C1IR( x, 5, roi, 2,1,6,7 );
}
```

The destination image x contains:

```
 01 01 02 03 04
 05 06 07 07 07
 07 07 07 07 07
 07 07 07 07 07
```

ComputeThreshold_Otsu

*Computes the value of the Otsu threshold.*

Syntax

IppStatus ippiComputeThreshold_Otsu_8u_C1R(const Ipp8u* pSrc, int srcStep, IppiSize roiSize, Ipp8u* pThreshold);

Include Files

ippi.h

Domain Dependencies

Headers: ippcore.h, ippvm.h, ipps.h
Libraries: ippcore.lib, ippvm.lib, ipps.lib

Parameters

pSrc
Pointer to the source image ROI.

srcStep
Distance in bytes between starts of consecutive lines in the source image.

roiSize
Size of the image ROI in pixels.

pThreshold
Pointer to the Otsu threshold value.

Description

This function operates with ROI (see Regions of Interest in Intel IPP).

This function calculates the Otsu threshold ([Otsu79]) for the source image pSrc in accordance with the following formula:

\[
\frac{\min_{T} \left( \sigma \left( p_{\text{Src}}(x,y) \leq T \right) + \sigma \left( p_{\text{Src}}(x,y) > T \right) \right)}{T}
\]

where \(0 \leq x < \text{roiSize}.\text{width}, 0 \leq y < \text{roiSize}.\text{height},\) and \(T\) is the Otsu threshold.

\[
\sigma(p_{\text{Src}}(x,y)) = \frac{\sum_{x,y=0}^{x<\text{roiSize}.\text{width},\ y<\text{roiSize}.\text{height}} (p_{\text{Src}}(x,y) - \text{mean})^2}{\text{roiSize}.\text{height} \cdot \text{roiSize}.\text{width}}
\]

where

\[
\text{mean} = \frac{\sum_{x,y=0}^{x<\text{roiSize}.\text{width},\ y<\text{roiSize}.\text{height}} p_{\text{Src}}(x,y)}{\text{roiSize}.\text{height} \cdot \text{roiSize}.\text{width}}
\]

The computed Otsu threshold can be used in the thresholding functions described above.

For example, the following figures shows how the Otsu threshold can be used for background/foreground selection. The figure a) below shows the initial image, the figure b) shows the histogram of the initial image with the red line indicating the computed Otsu threshold. The figure c) demonstrates the resulting image obtained by applying the function ippiThreshold_8u_C1R with a computed Otsu threshold value to the source image.
Using Otsu Threshold for Background/Foreground Selection

Return Values

- ippStsNoErr: Indicates no error. Any other value indicates an error.
- ippStsNullPtrErr: Indicates an error condition if one of the specified pointer is NULL.
- ippStsSizeErr: Indicates an error condition if roiSize has a field with zero or negative value.
- ippStsStepErr: Indicates an error condition if srcStep is less than or equal to 0.

Compare Operations

This section describes functions that compare images. Each compare function writes its results to a one-channel Ipp8u output image. The output pixel is set to a non-zero value if the corresponding input pixel(s) satisfy the compare condition; otherwise, the output pixel is set to 0. You can compare either two images, or an image and a constant value, using the following compare conditions: “greater”, “greater or equal”, “less”, “less or equal”, “equal”. Compare condition is specified as a function argument of IppCmpOp type (see Structures and Enumerators in Chapter 2 for more information). Images containing floating-point data can also be compared for being equal within a given tolerance eps.

For images with multi-channel data, the compare condition for a given pixel is true only when each color channel value of that pixel satisfies this condition.
Compare

Compares pixel values of two images using a specified compare operation.

Syntax

IppStatus ippiCompare_<mod>(const Ipp<datatype>* pSrc1, int src1Step, const Ipp<datatype>* pSrc2, int src2Step, Ipp8u* pDst, int dstStep, IppiSize roiSize, IppCmpOp ippCmpOp);

Supported values for mod:

- 8u_C1R
- 16u_C1R
- 16s_C1R
- 32f_C1R
- 8u_C3R
- 16u_C3R
- 16s_C3R
- 32f_C3R
- 8u_C4R
- 16u_C4R
- 16s_C4R
- 32f_C4R

Include Files

ippi.h

Domain Dependencies

Headers: ippcore.h, ippvm.h, ipps.h
Libraries: ippcore.lib, ippvm.lib, ipps.lib

Parameters

- pSrc1, pSrc2: Pointers to the source image ROIs.
- src1Step, src2Step: Distances in bytes between starts of consecutive lines in the source images.
- pDst: Pointer to the destination image ROI.
- dstStep: Distance in bytes between starts of consecutive lines in the destination image.
- roiSize: Size of the source and destination ROI in pixels.
- ippCmpOp: Compare operation to be used for comparing the pixel values.

Description

This function operates with ROI (see Regions of Interest in Intel IPP).

This function compares the corresponding pixels of ROI in two source images pSrc1, pSrc2 using the ippCmpOp compare operation, and writes the results to a one-channel Ipp8u image pDst. If the result of the compare is true, the corresponding output pixel is set to an IPP_MAX_8U value; otherwise, it is set to 0.
Return Values

ippStsNoErr  Indicates no error. Any other value indicates an error or a warning.

ippStsNullPtrErr  Indicates an error condition if one of the specified pointer is NULL.

ippStsSizeErr  Indicates an error condition if roiSize has a field with zero or negative value.

ippStsStepErr  Indicates an error condition if src1Step, src2Step, or dstStep has a zero or negative value.

CompareC

Compares pixel values of a source image to a given value using a specified compare operation.

Syntax

Case 1: Operation on one-channel data

IppStatus ippiCompareC_<mod>(const Ipp<datatype>* pSrc, int srcStep, Ipp<datatype> value, Ipp8u* pDst, int dstStep, IppiSize roiSize, IppCmpOp ippCmpOp);

Supported values for mod:

8u_C1R  16u_C1R  16s_C1R  32f_C1R

Case 2: Operation on multi-channel data

IppStatus ippiCompareC_<mod>(const Ipp<datatype>* pSrc, int srcStep, const Ipp<datatype> value[3], Ipp8u* pDst, int dstStep, IppiSize roiSize, IppCmpOp ippCmpOp);

Supported values for mod:

8u_C3R  16u_C3R  16s_C3R  32f_C3R

IppStatus ippiCompareC_<mod>(const Ipp<datatype>* pSrc, int srcStep, const Ipp<datatype> value[4], Ipp8u* pDst, int dstStep, IppiSize roiSize, IppCmpOp ippCmpOp);

Supported values for mod:

8u_C4R  16u_C4R  16s_C4R  32f_C4R

IppStatus ippiCompareC_<mod>(const Ipp<datatype>* pSrc, int srcStep, const Ipp<datatype> value[3], Ipp8u* pDst, int dstStep, IppiSize roiSize, IppCmpOp ippCmpOp);

Supported values for mod:

8u_AC4R  16u_AC4R  16s_AC4R  32f_AC4R

Include Files

ippi.h
Domain Dependencies

Headers: ippcore.h, ippvm.h, ipps.h
Libraries: ippcore.lib, ippvm.lib, ipps.lib

Parameters

- **pSrc**: Pointer to the source image ROI.
- **srcStep**: Distance in bytes between starts of consecutive lines in the source image.
- **value**: The value to compare each pixel to. In case of multi-channel data, an array of separate values (one for each channel).
- **pDst**: Pointer to the destination image ROI.
- **dstStep**: Distance in bytes between starts of consecutive lines in the destination image.
- **roiSize**: Size of the source and destination ROI in pixels.
- **ippCmpOp**: Compare operation to be used for comparing the pixel values.

Description

This function operates with ROI (Regions of Interest in Intel IPP).

This function compares pixels of the each channel of the source image ROI `pSrc` to a given `value` specified for each channel using the `ippCmpOp` compare operation, and writes the results to a one-channel Ipp8u image `pDst`. If the result of the compare is true, that is, all pixels of all channels meet the specified condition, then the corresponding output pixel is set to an IPP_MAX_8U value; otherwise, it is set to 0.

Return Values

- **ippStsNoErr**: Indicates no error. Any other value indicates an error or a warning.
- **ippStsNullPtrErr**: Indicates an error condition if one of the specified pointer is NULL.
- **ippStsSizeErr**: Indicates an error condition if `roiSize` has a field with zero or negative value.
- **ippStsStepErr**: Indicates an error condition if `srcStep` or `dstStep` has a zero or negative value.

Example

The code example below shows how to use the comparison function and create a mask image:

```c
IppStatus compare ( void ) {
    Ipp8u x[5*4], y[5*4];
    IppiSize roi = {5,4};
    int i;
    for( i=0; i<5*4; ++i ) x[i] = (Ipp8u)i;
    return ippiCompareC_8u_C1R ( x, 5, 7, y, 5, roi, ippCmpGreater );
}
```
The mask image \( y \) contains:

\[
\begin{array}{cccc}
00 & 00 & 00 & 00 \\
00 & 00 & FF & FF \\
FF & FF & FF & FF \\
FF & FF & FF & FF \\
\end{array}
\]

**CompareEqualEps**

*Compares two images with floating-point data, testing whether pixel values are equal within a certain tolerance \( \text{eps} \).*

**Syntax**

\[
\text{IppStatus ippiCompareEqualEps}_\langle \text{mod} \rangle(\text{const Ipp32f* pSrc1, int src1Step, const Ipp32f* pSrc2, int src2Step, Ipp8u* pDst, int dstStep, IppSize roiSize, Ipp32f eps});
\]

Supported values for \( \text{mod} \):

- \( 32f\_C1R \)
- \( 32f\_C3R \)
- \( 32f\_C4R \)

\[
\text{IppStatus ippiCompareEqualEps}_\langle 32f\_AC4R \rangle(\text{const Ipp32f* pSrc1, int src1Step, const Ipp32f* pSrc2, int src2Step, Ipp8u* pDst, int dstStep, IppSize roiSize, Ipp32f eps});
\]

**Include Files**

ippi.h

**Domain Dependencies**

**Headers:** ippcore.h, ippvm.h, ipps.h

**Libraries:** ippcore.lib, ippvm.lib, ipps.lib

**Parameters**

- \( pSrc1, pSrc2 \)  
  Pointers to the source image ROIs.
- \( src1Step, src2Step \)  
  Distances in bytes between starts of consecutive lines in the source images.
- \( pDst \)  
  Pointer to the destination image ROI.
- \( dstStep \)  
  Distance in bytes between starts of consecutive lines in the destination image.
- \( roiSize \)  
  Size of the source and destination ROI in pixels.
- \( eps \)  
  The tolerance value.

**Description**

This function operates with ROI (see Regions of Interest in Intel IPP).

This function tests if the corresponding pixels of ROI in two source images \( pSrc1, pSrc2 \) are equal within the tolerance \( \text{eps} \), and writes the results to a one-channel Ipp8u image \( pDst \). If the absolute value of difference of the pixel values in \( pSrc1 \) and \( pSrc2 \) is less than or equal to \( \text{eps} \), then the corresponding pixel in \( pDst \) is
set to an IPP_MAX_8U value; otherwise the pixel in $pDst$ is set to 0. For multi-channel images, the differences for all color channel values of a pixel must be within the $eps$ tolerance for the compare condition to be true.

This function processes images with floating-point data only.

**Return Values**

- **ippStsNoErr**: Indicates no error. Any other value indicates an error or a warning.
- **ippStsNullPtrErr**: Indicates an error condition if one of the specified pointer is NULL.
- **ippStsSizeErr**: Indicates an error condition if $roiSize$ has a field with zero or negative value.
- **ippStsStepErr**: Indicates an error condition if $src1Step$, $src2Step$, or $dstStep$ has a zero or negative value.
- **ippStsEpsValErr**: Indicates an error condition if $eps$ has a negative value.

**CompareEqualEpsC**

Tests whether floating-point pixel values of an image are equal to a given value within a certain tolerance $eps$.

**Syntax**

**Case 1: Operation on one-channel data**

```c
IppStatus ippiCompareEqualEpsC_32f_C1R(const Ipp32f* pSrc, int srcStep, Ipp32f value, Ipp8u* pDst, int dstStep, IppiSize roiSize, Ipp32f eps);
```

**Case 2: Operation on multi-channel data**

```c
IppStatus ippiCompareEqualEpsC_32f_C3R(const Ipp32f* pSrc, int srcStep, const Ipp32f value[3], Ipp8u* pDst, int dstStep, IppiSize roiSize, Ipp32f eps);
IppStatus ippiCompareEqualEpsC_32f_C4R(const Ipp32f* pSrc, int srcStep, const Ipp32f value[4], Ipp8u* pDst, int dstStep, IppiSize roiSize, Ipp32f eps);
IppStatus ippiCompareEqualEpsC_32f_AC4R(const Ipp32f* pSrc, int srcStep, const Ipp32f value[3], Ipp8u* pDst, int dstStep, IppiSize roiSize, Ipp32f eps);
```

**Include Files**

ippi.h

**Domain Dependencies**

- **Headers**: ippcore.h, ippvm.h, ipps.h
- **Libraries**: ippcore.lib, ippvm.lib, ipps.lib

**Parameters**

- **pSrc**: Pointer to the source image ROI.
- **srcStep**: Distance in bytes between starts of consecutive lines in the source image.
value
The value to compare each pixel to. In case of multi-channel data, an array of separate values (one for each channel).

pDst
Pointer to the destination image ROI.

dstStep
Distance in bytes between starts of consecutive lines in the destination image.

roiSize
Size of the source and destination ROI in pixels.

eps
The tolerance value.

Description
This function operates with ROI (see Regions of Interest in Intel IPP).

This function tests if pixel values of the source image ROI pSrc are equal to a given constant value within the tolerance eps, and writes the results to a one-channel Ipp8u image pDst. If the absolute value of difference between the pixel value in pSrc and value is less than or equal to eps, then the corresponding pixel in pDst is set to an IPP_MAX_8U value; otherwise the pixel in pDst is set to 0. For multi-channel images, the differences between all color channel values of a pixel and the respective components of value must be within the tolerance eps for the compare condition to be true.

This function processes images with floating-point data only.

Return Values
ippStsNoErr
Indicates no error. Any other value indicates an error or a warning.

ippStsNullPtrErr
Indicates an error condition if one of the specified pointer is NULL.

ippStsSizeErr
Indicates an error condition if roiSize has a field with a zero or negative value.

ippStsStepErr
Indicates an error condition if srcStep or dstStep has a zero or negative value.

ippStsEpsValErr
Indicates an error condition if eps has a negative value.
This chapter describes the Intel® IPP image processing functions that perform morphological operations on images.

Generally, the erosion and dilation smooth the boundaries of objects without significantly changing their area. Opening and closing smooth thin projections or gaps. Morphological operations use a structuring element (SE) that is a user-defined rectangular mask, or for some functions - symmetric 3x3 mask.

In a more general sense, morphological operations involve an image $A$ called the object of interest and a kernel element $B$ called the structuring element. The image and structuring element could be in any number of dimensions, but the most common use is with a 2D binary image, or with a 3D gray scale image. The element $B$ is most often a square or a circle, but it could be of any shape. Just like in convolution, $B$ is a kernel or template with an anchor point. Figure "Dilation and Erosion of $A$ by $B$" shows dilation and erosion of object $A$ by $B$. In the figure, $B$ is rectangular with an anchor point at upper left shown as a dark square.

**Dilation and Erosion of $A$ by $B$**

Let $B_t$ is the SE with pixel $t$ in the anchor position, $\overline{B}$ is transpose of the SE.

Dilation of binary image $A \{A(t) = 1, t \in A; \ 0 - \text{otherwise}\}$ by binary SE $B$ is

$$A \oplus B = \{t : B_t \cap A \neq \emptyset\}$$

It means that every pixel is in the set, if the intersection is not null. That is, a pixel under the anchor point of $B$ is marked "on", if at least one pixel of $B$ is inside of $A$.

Erosion of the binary image $A$ by the binary SE $B$ is

$$A \odot B = \{t : B_t \subseteq A\}$$

That is, a pixel under the anchor of $B$ is marked "on", if $B$ is entirely within $A$.

Generalization of dilation and erosion for the gray-scale image $A$ and the binary SE $B$ is

$$A \oplus B = \left\{ \max_{u \in B_t \cap A} \{A(u)\} \right\}, \quad A \odot B = \left\{ \min_{u \in B_t \cap A} \{A(u)\} \right\}$$
Generalization of dilation and erosion for the gray-scale image $A$ and the gray-scale SE $B$ is

$$A \oplus B = \left\{ \max_{u \in B \cap A} (A(u) + B(u)) \right\}, \quad A \Theta B = \left\{ \min_{u \in B \cap A} (A(u) + B(u)) \right\}$$

Opening operation of $A$ by $B$ is $A \circ B = (A \Theta B) \ominus B$.

Closing operation of $A$ by $B$ is $A \bullet B = (A \ominus B) \Theta B$.

Top-hat operation of $A$ by $B$ is $A - A \circ B$.

Black-hat operation of $A$ by $B$ is $A \bullet B - A$.

Black-hat operation of $A$ by $B$ is $A \oplus B - A \Theta B$.

Morphological reconstruction $[Vincent93]_{\rho A}(C)$ of an image $A$ from the image $C$, $A(t) \geq C(t) \forall t$ by dilation with the mask $B$ is an image

$$C_k : k = \min_i C_i = C_{i-1}, \quad C_0 = C, \quad C_{i+1}(t) = \min\{ (C_i \oplus B)(t) , A(t) \}$$

Morphological reconstruction $[\rho A](C)$ of an image $A$ from the image $C$, $A(t) \leq C(t) \forall t$ by erosion with the mask $B$ is an image

$$C_k : k = \max_i C_i = C_{i-1}, \quad C_0 = C, \quad C_{i+1}(t) = \max\{ (C_i \Theta B)(t) , A(t) \}$$

Figure "Morphological Operations Performed by Intel IPP" presents the results of different morphological operations applied to the initial image. In these operations, the SE is a matrix of 3x3 size with the following values:

$$\begin{bmatrix}
-8 & 0 & -8 \\
0 & 8 & 0 \\
-8 & 0 & -8 \\
\end{bmatrix}$$

for common and advanced morphology, and

$$\begin{bmatrix}
-5 & 0 & -5 \\
0 & 5 & 0 \\
-5 & 0 & -5 \\
\end{bmatrix}$$

for gray morphology.
The anchor cell is in the center cell (1,1) of the matrix.

**Morphological Operations Performed by Intel IPP**

- Initial image
- Erosion
- Dilation
- Opening
- Closing
- Top-hat
- Black-hat
- Morphological gradient
- Gray-kernel erosion
- Gray-kernel dilation
Flat Structuring Elements for Grayscale Image

Erosion and dilation can be done in 3D space, that is, with gray levels. 3D structuring elements can be used, but the simplest and the best way is to use a flat structuring element $B$. Figure "1D Cross Section of Dilation and Erosion of $A$ by $B" is a 1D cross section of dilation and erosion of a grayscale image $A$ by a flat structuring element $B$. In the figure, $B$ has an anchor slightly to the right of the center as shown by the dark mark on $B$.

1D Cross Section of Dilation and Erosion of $A$ by $B$

In Figure "1D Cross Section of Dilation and Erosion of $A$ by $B" above, dilation is mathematically

$$\sup_{y \in B} A$$

and erosion is

$$\inf_{y \in B} A$$

Dilate3x3

Performs dilation of an image using a 3x3 mask.
Syntax

IppStatus ippiDilate3x3_64f_C1R(const Ipp64f * pSrc, int srcStep, Ipp<datatype>* pDst, int dstStep, IppiSize roiSize);

Include Files

ippi.h

Domain Dependencies

Headers: ippcore.h, ippvm.h, ipps.h
Libraries: ippcore.lib, ippvm.lib, ipps.lib

Parameters

pSrc Pointer to the source image ROI.
srcStep Distance in bytes between starts of consecutive lines in the source image.
pDst Pointer to the destination image ROI.
dstStep Distance in bytes between starts of consecutive lines in the destination image.
roiSize Size of the source and destination ROI in pixels.

Description

This function operates with ROI (see Regions of Interest in Intel IPP).
This function performs dilation of a rectangular ROI area inside a 2D image using a symmetric 3x3 mask.
Source and destination images can be of different sizes, but the ROI size is the same for both images. The output pixel is set to the maximum of the corresponding input pixel and its eight neighboring pixels.

Return Values

ippStsNoErr Indicates no error. Any other value indicates an error or a warning.
ippStsNullPtrErr Indicates an error condition if pSrc or pDst is NULL.
ippStsSizeErr Indicates an error condition if roiSize has a field with zero or negative value.
ippStsStepErr Indicates an error condition if srcStep or dstStep has a zero or negative value.

Dilate

Performs dilation of an image using a specified mask.

Syntax

IppStatus ippiDilate_8u_C1R_L(const Ipp8u* pSrc, IppSizeL srcStep, Ipp8u* pDst, IppSizeL dstStep, IppiSizeL roiSize, IppiBorderType borderType, const Ipp8u* borderValue[1], const IppiMorphStateL* pMorphSpec, Ipp8u* pBuffer);
IppStatus ippiDilate_8u_C3R_L(const Ipp8u* pSrc, IppSizeL srcStep, Ipp8u* pDst, IppSizeL dstStep, IppSizeL roiSize, IppiBorderType borderType, const Ipp8u* borderValue[3], const IppiMorphStateL* pMorphSpec, Ipp8u* pBuffer);

IppStatus ippiDilate_8u_C4R_L(const Ipp8u* pSrc, IppSizeL srcStep, Ipp8u* pDst, IppSizeL dstStep, IppSizeL roiSize, IppiBorderType borderType, const Ipp8u* borderValue[4], const IppiMorphStateL* pMorphSpec, Ipp8u* pBuffer);

IppStatus ippiDilate_32f_C3R_L(const Ipp32f* pSrc, IppSizeL srcStep, Ipp32f* pDst, IppSizeL dstStep, IppSizeL roiSize, IppiBorderType borderType, const Ipp32f* borderValue[3], const IppiMorphStateL* pMorphSpec, Ipp8u* pBuffer);

IppStatus ippiDilate_32f_C1R_L(const Ipp32f* pSrc, IppSizeL srcStep, Ipp32f* pDst, IppSizeL dstStep, IppSizeL roiSize, IppiBorderType borderType, const Ipp32f* borderValue[1], const IppiMorphStateL* pMorphSpec, Ipp8u* pBuffer);

IppStatus ippiDilate_32f_C4R_L(const Ipp32f* pSrc, IppSizeL srcStep, Ipp32f* pDst, IppSizeL dstStep, IppSizeL roiSize, IppiBorderType borderType, const Ipp32f* borderValue[4], const IppiMorphStateL* pMorphSpec, Ipp8u* pBuffer);

IppStatus ippiDilate_16u_C1R_L(const Ipp16u* pSrc, IppSizeL srcStep, Ipp16u* pDst, IppSizeL dstStep, IppSizeL roiSize, IppiBorderType borderType, const Ipp16u* borderValue[1], const IppiMorphStateL* pMorphSpec, Ipp8u* pBuffer);

IppStatus ippiDilate_16s_C1R_L(const Ipp16s* pSrc, IppSizeL srcStep, Ipp16s* pDst, IppSizeL dstStep, IppSizeL roiSize, IppiBorderType borderType, const Ipp16s* borderValue[1], const IppiMorphStateL* pMorphSpec, Ipp8u* pBuffer);

IppStatus ippiDilate_1u_C1R_L(const Ipp8u* pSrc, IppSizeL srcStep, int srcBitOffset, Ipp8u* pDst, IppSizeL dstStep, int dstBitOffset, IppSizeL roiSize, IppiBorderType borderType, const Ipp8u* borderValue[1], const IppiMorphStateL* pMorphSpec, Ipp8u* pBuffer);

Include Files

ippi.cv_l.h

Domain Dependencies

Headers: ippcore.h, ippvm.h, ipps.h
Libraries: ippcore.lib, ippvm.lib, ipps.lib

Parameters

pSrc
srcStep
pDst
dstStep
roiSize
borderType

Pointer to the source image ROI.

Distance, in bytes, between the starting points of consecutive lines in the source image.

Pointer to the destination image ROI.

Distance, in bytes, between the starting points of consecutive lines in the destination image.

Size of the source and destination ROI, in pixels.

Type of the border; supported values:

ippBorderDefault The border is set to ippBorderConst with
borderValue= MIN_VALUE, where
MIN_VALUE=IPP_MIN_8U/16U/16S/32F/1U
Border is replicated from the edge pixels.
Border pixels are mirrored from the source image boundary pixels.
Values of all border pixels are set to a constant.
Border is obtained from the source image pixels in memory.

Mixed borders are also supported. They can be obtained by the bitwise operation OR between any of the ippBorderRepl, ippBorderConst, or ippBorderMirror values and the ippBorderInMemTop, ippBorderInMemBottom, ippBorderInMemLeft, ippBorderInMemRight values.

Pointer to the vector of values for the constant border type.
Offset in the first byte of the source image row.
Offset in the first byte of the destination image row.
Pointer to the morphology specification structure.
Pointer to the external work buffer.

Description
This function operates with ROI (see Regions of Interest in Intel IPP).

This function performs dilation of a rectangular ROI area inside a one-, three-, or four-channel 2D image using a specified mask pMask of size maskSize and alignment anchor.

Source and destination images can be of different sizes, but the ROI size is the same for both images. The output pixel is set to the maximum of the corresponding input pixel and its neighboring pixels that are picked out by the nonzero mask values. In the four-channel image the alpha channel is not processed.

Return Values
Indicates no error. Any other value indicates an error or a warning.
Indicates an error condition if pSrc or pDst is NULL.
Indicates an error condition if roiSize has a field with a zero or negative value.
Indicates an error condition if srcStep or dstStep has a zero or negative value.
Indicates an error condition if srcStep or dstStep has a not pixel multiple value.
Indicates an error condition if borderType has an incorrect value.

DilateBorder
Performs dilation of an image.
Syntax

IppStatus ippiDilateBorder_<mod>(const Ipp<datatype>* pSrc, int srcStep, Ipp<datatype>* pDst, int dstStep, IppiSize roiSize, IppiBorderType borderType, Ipp<datatype> borderValue, const IppiMorphState* pSpec, Ipp8u* pBuffer);

Supported values for mod:

8u_C1R  16u_C1R  16s_C1R  32f_C1R

IppStatus ippiDilateBorder_<mod>(const Ipp<datatype>* pSrc, int srcStep, Ipp<datatype>* pDst, int dstStep, IppiSize roiSize, IppiBorderType borderType, const Ipp<datatype> borderValue[3], const IppiMorphState* pSpec, Ipp8u* pBuffer);

Supported values for mod:

8u_C3R  32f_C3R

IppStatus ippiDilateBorder_<mod>(const Ipp<datatype>* pSrc, int srcStep, Ipp<datatype>* pDst, int dstStep, IppiSize roiSize, IppiBorderType borderType, const Ipp<datatype> borderValue[4], const IppiMorphState* pSpec, Ipp8u* pBuffer);

Supported values for mod:

8u_C4R  32f_C4R

IppStatus ippiDilateBorder_1u_C1R(const Ipp8u* pSrc, int srcStep, int srcBitOffset, Ipp8u* pDst, int dstStep, int dstBitOffset, IppiSize roiSize, IppiBorderType borderType, Ipp8u borderValue, const IppiMorphState* pSpec, Ipp8u* pBuffer);

Include Files

ippcv.h

Domain Dependencies

Headers: ippcore.h, ippvm.h, ipps.h,ippi.h
Libraries: ippcore.lib, ippvm.lib, ipps.lib,ippi.lib

Parameters

pSrc

Pointer to the source image ROI.

srcStep

Distance in bytes between the starting points of consecutive lines in the source image.

crcBitOffset

Offset, in bits, from the first byte of the source image (for 1u_C1R flavor).

pDst

Pointer to the destination image ROI.

dstStep

Distance, in bytes, between the starting points of consecutive lines in the destination image.

dstBitOffset

Offset, in bits, from the first byte of the destination image (for 1u_C1R flavor).

roiSize

Size of the source and destination image ROI.

borderType

Type of border. Possible values are:

ippBorderRepl          Border is replicated from the edge pixels.
**ippiMorphologyBorderGetSize_16u_C1R**

ippiMorphologyBorderGetSize_16u_C1R(pSpec, &roiSize, &maskSize, &pBuffer);

ippiMorphologyBorderInit_16u_C1R(pSpec, ippBorderInMem);

ippiDilateBorder_16u_C1R(pSpec, pSrc, pBuffer, roiSize, maskSize);

The code example below demonstrates how to use the ippiMorphologyBorderGetSize_16u_C1R, ippiMorphologyBorderInit_16u_C1R, andippiDilateBorder_16u_C1R functions to perform dilation of the source image.

```c
IppStatus func_MorfDilateBorder()
{
    IppiMorphState* pSpec = NULL;
    Ipp8u* pBuffer = NULL;
    IppiSize roiSize = {5, 5};
    IppiSize maskSize = {3, 3};
    Ipp8u pMask[3*3] = {1, 1, 1, 1, 0, 1, 1, 1, 1};
    Ipp16u pSrc[5*5] = { 1, 2, 4, 1, 2, 5, 1, 2, 1, 2, 1, 2, 2, 1, 2, 1, 2, 1};
}```
Ipp16u pDst[5*5];
int srcStep = 5*sizeof(Ipp16u);
int dstStep = 5*sizeof(Ipp16u);
int dstSize = 5;
IppStatus status = ippStsNoErr;
int specSize = 0, bufferSize = 0;
Ipp1BorderType borderType= ippBorderRepl;
Ipp16u borderValue = 0;

status = ippiMorphologyBorderGetSize_16u_C1R( roiSize, maskSize, &specSize, &bufferSize );
if (status != ippStsNoErr) return status;
pSpec = (IppiMorphState*)ippsMalloc_8u(specSize);
pBuffer = (Ipp8u*)ippsMalloc_8u(bufferSize);

status = ippiMorphologyBorderInit_16u_C1R( roiSize, pMask, maskSize, pSpec, pBuffer );
if (status != ippStsNoErr) {
    ippFree(pBuffer);
    ippFree(pSpec);
    return status;
}

status = ippiDilateBorder_16u_C1R( pSrc, srcStep, pDst, dstStep, roiSize, borderType, borderValue, pSpec, pBuffer);
ippFree(pBuffer);
ippFree(pSpec);
return status;

The result is as follows:

pDst->
  5 5 4 4 2
  5 5 4 4 2
  5 5 2 2 2
  2 5 5 5 2
  2 5 5 5 2

See Also
Regions of Interest in Intel IPP
User-defined Border Types
MorphologyBorderInit Initializes the morphology specification structure for erosion or dilation operations.

DilateGetBufferSize

Computes the size of the working buffer for the Dilate function.

Syntax
IppStatus ippiDilateGetBufferSize_L(IippiSizeL roiSize, IippiSizeL maskSize, IppDataType datatype, int numChannels, IppSizeL* pBufferSize);
Include Files
ippcv_l.h

Domain Dependencies
Headers: ippcore.h, ippvm.h, ipps.h,ippi.h
Libraries: ippcore.lib, ippvm.lib, ipps.lib,ippi.lib

Parameters
roiSize
maskSize
dataType
numChannels
pBufferSize

Description
This function computes the size of the working buffer required for the ippiDilate functions with the _L suffix.

Return Values
ippStsNoErr
Indicates no error. Any other value indicates an error or a warning.
ippStsNullPtrErr
Indicates an error when one of the pointers is NULL.
ippStsSizeErr
Width of the image, or width or height of the structuring element is less than, or equal to zero.

DilateGetSpecSize
Computes the size of the internal state or specification structure for the Dilate function.

Syntax
IppStatus ippiDilateGetSpecSize_L(IppSizeL roiSize, IppSizeL maskSize, IppSizeL* pSpecSize);

Include Files
ippcv_l.h

Domain Dependencies
Headers: ippcore.h, ippvm.h, ipps.h,ippi.h
Libraries: ippcore.lib, ippvm.lib, ipps.lib,ippi.lib

Parameters
roiSize
Size of the source and destination image ROI, in pixels.
maskSize
Size of the structuring element.
pSpecSize

Pointer to the size of the specification structure.

**Description**

This function computes the size of the specification structure required for the *ippiDilate* functions with the _L suffix.

**Return Values**

- **ippStsNoErr**
  - Indicates no error. Any other value indicates an error or a warning.
- **ippStsNullPtrErr**
  - Indicates an error when one of the pointers is NULL.
- **ippStsSizeErr**
  - Width of the image, or width or height of the structuring element is less than, or equal to zero.

**Dilatelnit**

*Initializes the internal state or specification structure for the Dilate function.*

**Syntax**

```c
IppStatus ippiDilateInit_L(IppiSizeL roiSize, const Ipp8u* pMask, IppiSizeL maskSize, IppiMorphStateL* pMorphSpec);
```

**Include Files**

*ippcv_l.h*

**Domain Dependencies**

*Headers:* *ippcore.h, ippvm.h, ipps.h,ippi.h*

*Libraries:* *ippcore.lib, ippvm.lib, ipps.lib,ippi.lib*

**Parameters**

- **roiSize**
  - Size of the source and destination image ROI, in pixels.
- **pMask**
  - Pointer to the structuring element (mask).
- **maskSize**
  - Size of the structuring element.
- **pMorphSpec**
  - Pointer to the morphology specification structure.

**Description**

This function initializes the internal state or specification structure for the *ippiDilate* functions with the _L suffix.

**Return Values**

- **ippStsNoErr**
  - Indicates no error. Any other value indicates an error or a warning.
- **ippStsNullPtrErr**
  - Indicates an error when one of the pointers is NULL.
- **ippStsSizeErr**
  - Width of the image, or width or height of the structuring element is less than, or equal to zero.
- **ippStsAnchorErr**
  - Anchor point is outside the structuring element.
Erode3x3

Performs erosion of an image using a 3x3 mask.

Syntax

```c
IppStatus ippiErode3x3_64f_C1R(const Ipp64f* pSrc, int srcStep, Ipp64f* pDst, int dstStep, IppiSize roiSize);
```

Include Files

ippi.h

Domain Dependencies

Headers: ippcore.h, ippvm.h, ipps.h
Libraries: ippcore.lib, ippvm.lib, ipps.lib

Parameters

- `pSrc` Pointer to the source image ROI.
- `srcStep` Distance in bytes between starts of consecutive lines in the source image.
- `pDst` Pointer to the destination image ROI.
- `dstStep` Distance in bytes between starts of consecutive lines in the destination image.
- `roiSize` Size of the source and destination ROI in pixels.

Description

This function operates with ROI (see Regions of Interest in Intel IPP).

This function performs erosion of a rectangular ROI area inside a 2D image using a symmetric 3x3 mask.

Source and destination images can have different size, but the ROI size is the same for both images. The output pixel is set to the minimum of the corresponding input pixel and its 8 neighboring pixels.

Return Values

- `ippStsNoErr` Indicates no error. Any other value indicates an error or a warning.
- `ippStsNullPtrErr` Indicates an error condition if `pSrc` or `pDst` is NULL.
- `ippStsSizeErr` Indicates an error condition if `roiSize` has a field with a zero or negative value.
- `ippStsStepErr` Indicates an error condition if `srcStep` or `dstStep` has a zero or negative value.

Erode

Performs erosion of an image using a specified mask.
**Syntax**

IppStatus ippiErode_8u_C1R_L(const Ipp8u* pSrc, IppSizeL srcStep, Ipp8u* pDst, IppSizeL dstStep, IppSizeL roiSize, IppiBorderType borderType, const Ipp8u* borderValue[1], const IppiMorphStateL* pMorphSpec, Ipp8u* pBuffer);

IppStatus ippiErode_8u_C3R_L(const Ipp8u* pSrc, IppSizeL srcStep, Ipp8u* pDst, IppSizeL dstStep, IppSizeL roiSize, IppiBorderType borderType, const Ipp8u* borderValue[3], const IppiMorphStateL* pMorphSpec, Ipp8u* pBuffer);

IppStatus ippiErode_8u_C4R_L(const Ipp8u* pSrc, IppSizeL srcStep, Ipp8u* pDst, IppSizeL dstStep, IppSizeL roiSize, IppiBorderType borderType, const Ipp8u* borderValue[4], const IppiMorphStateL* pMorphSpec, Ipp8u* pBuffer);

IppStatus ippiErode_32f_C3R_L(const Ipp32f* pSrc, IppSizeL srcStep, Ipp32f* pDst, IppSizeL dstStep, IppSizeL roiSize, IppiBorderType borderType, const Ipp32f* borderValue[3], const IppiMorphStateL* pMorphSpec, Ipp8u* pBuffer);

IppStatus ippiErode_32f_C1R_L(const Ipp32f* pSrc, IppSizeL srcStep, Ipp32f* pDst, IppSizeL dstStep, IppSizeL roiSize, IppiBorderType borderType, const Ipp32f* borderValue[1], const IppiMorphStateL* pMorphSpec, Ipp8u* pBuffer);

IppStatus ippiErode_32f_C4R_L(const Ipp32f* pSrc, IppSizeL srcStep, Ipp32f* pDst, IppSizeL dstStep, IppSizeL roiSize, IppiBorderType borderType, const Ipp32f* borderValue[4], const IppiMorphStateL* pMorphSpec, Ipp8u* pBuffer);

IppStatus ippiErode_16u_C1R_L(const Ipp16u* pSrc, IppSizeL srcStep, Ipp16u* pDst, IppSizeL dstStep, IppSizeL roiSize, IppiBorderType borderType, const Ipp16u* borderValue[1], const IppiMorphStateL* pMorphSpec, Ipp8u* pBuffer);

IppStatus ippiErode_16u_C4R_L(const Ipp16u* pSrc, IppSizeL srcStep, Ipp16u* pDst, IppSizeL dstStep, IppSizeL roiSize, IppiBorderType borderType, const Ipp16u* borderValue[4], const IppiMorphStateL* pMorphSpec, Ipp8u* pBuffer);

IppStatus ippiErode_16s_C1R_L(const Ipp16s* pSrc, IppSizeL srcStep, Ipp16s* pDst, IppSizeL dstStep, IppSizeL roiSize, IppiBorderType borderType, const Ipp16s* borderValue[1], const IppiMorphStateL* pMorphSpec, Ipp8u* pBuffer);

IppStatus ippiErode_1u_C1R_L(const Ipp8u* pSrc, IppSizeL srcStep, int srcBitOffset, Ipp8u* pDst, IppSizeL dstStep, int dstBitOffset, IppSizeL roiSize, IppiBorderType borderType, const Ipp8u* borderValue[1], const IppiMorphStateL* pMorphSpec, Ipp8u* pBuffer);

Include Files

ippcv_l.h

Domain Dependencies

Headers: ippcore.h, ippvm.h, ipps.h

Libraries: ippcore.lib, ippvm.lib, ipps.lib

Parameters

- **pSrc**: Pointer to the source image ROI.
- **srcStep**: Distance, in bytes, between the starting points of consecutive lines in the source image.
- **pDst**: Pointer to the destination image ROI.
- **dstStep**: Distance, in bytes, between the starting points of consecutive lines in the destination image.
- **roiSize**: Size of the source and destination ROI, in pixels.
borderType

Type of the border; supported values:

ippBorderDefault  The border is set to ippBorderConst with
borderValue = MAX_VALUE, where
MAX_VALUE=IPP_MAX_8U/16U/16S/32F/1U

ippBorderRepl     Border is replicated from the edge pixels.

ippBorderMirror   Border pixels are mirrored from the source
image boundary pixels.

ippBorderConst    Values of all border pixels are set to a constant.

ippBorderInMem    Border is obtained from the source image pixels
in memory.

Mixed borders are also supported. They can be obtained by the bitwise
operation OR between any of the ippBorderRepl, ippBorderConst, or
ippBorderMirror values and the ippBorderInMemTop, ippBorderInMemBottom,
ippBorderInMemLeft, ippBorderInMemRight values.

borderValue       Pointer to the vector of values for the constant border type.

srcBitOffset      Offset in the first byte of the source image row.

dstBitOffset      Offset in the first byte of the destination image row.

pMorphSpec        Pointer to the morphology specification structure.

pBuffer           Pointer to the external work buffer.

Description

This function operates with ROI (see Regions of Interest in Intel IPP).
This function performs erosion of a rectangular ROI area inside a one-, three-, or four-channel 2D image
using a specified mask pMask of size maskSize and alignment anchor.

Source and destination images can be of different sizes, but the ROI size is the same for both images
destSize. The output pixel is set to the minimum of the corresponding input pixel and its neighboring
pixels that are picked out by the non-zero mask values. In the four-channel image the alpha channel is not
processed.

Return Values

ippStsNoErr        Indicates no error. Any other value indicates an error or a
warning.

ippStsNullPtrErr   Indicates an error condition if pSrc or pDst is NULL.

ippStsSizeErr      Indicates an error condition if roiSize has a field with a zero or
negative value.

ippStsStepErr      Indicates an error condition if srcStep or dstStep has a zero or
negative value.

ippStsNotEvenStepErr Indicates an error condition if srcStep or dstStep has a not
pixel multiple value.

ippStsBadArgErr    Indicates an error condition if borderType has an incorrect
value.
ErodeBorder

Performs erosion of an image.

Syntax

IppStatus ippiErodeBorder_<mod>(const Ipp<datatype>* pSrc, int srcStep, Ipp<datatype>* pDst, int dstStep, IppiSize roiSize, IppiBorderType borderType, Ipp<datatype> borderValue, const IppiMorphState* pSpec, Ipp8u* pBuffer);

Supported values for mod:

8u_C1R 16u_C1R 16s_C1R 32f_C1R

IppStatus ippiErodeBorder_<mod>(const Ipp<datatype>* pSrc, int srcStep, Ipp<datatype>* pDst, int dstStep, IppiSize roiSize, IppiBorderType borderType, const Ipp<datatype> borderValue[3], const IppiMorphState* pSpec, Ipp8u* pBuffer);

Supported values for mod:

8u_C3R 32f_C3R

IppStatus ippiErodeBorder_<mod>(const Ipp<datatype>* pSrc, int srcStep, Ipp<datatype>* pDst, int dstStep, IppiSize roiSize, IppiBorderType borderType, const Ipp<datatype> borderValue[4], const IppiMorphState* pSpec, Ipp8u* pBuffer);

Supported values for mod:

8u_C4R 32f_C4R

IppStatus ippiErodeBorder_1u_C1R(const Ipp8u* pSrc, int srcStep, int srcBitOffset, Ipp8u* pDst, int dstStep, int dstBitOffset, IppiSize roiSize, IppiBorderType borderType, Ipp8u borderValue, const IppiMorphState* pSpec, Ipp8u* pBuffer);

Include Files

ippcv.h

Domain Dependencies

Headers: ippcore.h, ippvm.h, ipps.h, ippi.h
Libraries: ippcore.lib, ippvm.lib, ipps.lib, ippi.lib

Parameters

pSrc
Pointer to the source image ROI.

srcStep
Distance, in bytes, between the starting points of consecutive lines in the source image.

srcBitOffset
Offset, in bits, from the first byte of the source image (for 1u_C1R flavor).

pDst
Pointer to the destination image ROI.

dstStep
Distance, in bytes, between the starting points of consecutive lines in the destination image.

dstBitOffset
Offset, in bits, from the first byte of the destination image (for 1u_C1R flavor).
Description

This function operates with ROI.

This function performs erosion of a rectangular ROI area inside a one-channel 2D image using a mask specified in the specification structure pSpec. Before using this function, you need to initialize the structure using the MorphologyBorderInit function.

The output pixel is set to the maximum of the corresponding input pixel and its neighboring pixels that are picked out by the nonzero mask values.

Return Values

<table>
<thead>
<tr>
<th>Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ippStsNoErr</td>
<td>Indicates no error.</td>
</tr>
<tr>
<td>ippStsNullPtrErr</td>
<td>Indicates an error when one of the specified pointers is NULL.</td>
</tr>
<tr>
<td>ippStsSizeErr</td>
<td>Indicates an error when:</td>
</tr>
<tr>
<td></td>
<td>• roiSize has a field with a zero or negative value</td>
</tr>
<tr>
<td></td>
<td>• roiSize.width is more than the maximum ROI roiWidth passed</td>
</tr>
<tr>
<td></td>
<td>to the initialization function</td>
</tr>
<tr>
<td></td>
<td>• srcBitOffset or dstBitOffset is less than zero</td>
</tr>
<tr>
<td>ippStsStepErr</td>
<td>Indicates an error when srcStep or dstStep is less than</td>
</tr>
<tr>
<td></td>
<td>roiSize.width* &lt; pixelSize&gt;.</td>
</tr>
<tr>
<td>ippStsNotEvenStepErr</td>
<td>Indicates an error when one of the step values for 16-bit</td>
</tr>
<tr>
<td></td>
<td>integer images is not divisible by 2.</td>
</tr>
<tr>
<td>ippStsBorderErr</td>
<td>Indicates an error when borderType has an illegal value.</td>
</tr>
<tr>
<td>ippStsInplaceModeNotSupportedErr</td>
<td>Indicates an error when the pSrc pointer is equal to the pDst pointer.</td>
</tr>
</tbody>
</table>

See Also

Regions of Interest in Intel IPP
User-defined Border Types
MorphologyBorderInit Initializes the morphology specification structure for erosion or dilation operations.

ErodeGetBufferSize

Computes the size of the working buffer for the Erode function.
Syntax

IppStatus ippiErodeGetBufferSize_L(IppSizeL roiSize, IppSizeL maskSize, IppDataType datatype, int numChannels, IppSizeL* pBufferSize);

Include Files

ippcv_l.h

Domain Dependencies

Headers: ippcore.h, ippvm.h, ipps.h,ippi.h
Libraries: ippcore.lib, ippvm.lib, ipps.lib,ippi.lib

Parameters

roiSize  
Size of the source and destination image ROI, in pixels.

maskSize  
Size of the structuring element.

dataType  
Data type for the morphological function.

numChannels  
Number of channels in the image.

pBufferSize  
Pointer to the buffer size value for the morphological initialization function.

Description

This function computes the size of the working buffer required for the ippiErode functions with the _L suffix.

Return Values

ippStsNoErr  
Indicates no error. Any other value indicates an error or a warning.

ippStsNullPtrErr  
Indicates an error when one of the pointers is NULL.

ippStsSizeErr  
Width of the image, or width or height of the structuring element is less than, or equal to zero.

ErodeGetSpecSize

*Computes the size of the internal state or specification structure for the Erode function.*

Syntax

IppStatus ippiErodeGetSpecSize_L(IppSizeL roiSize, IppSizeL maskSize, IppSizeL* pSpecSize);

Include Files

ippcv_l.h

Domain Dependencies

Headers: ippcore.h, ippvm.h, ipps.h,ippi.h
Libraries: ippcore.lib, ippvm.lib, ipps.lib,ippi.lib
Parameters

roiSize          Size of the source and destination image ROI, in pixels.
maskSize         Size of the structuring element.
pSpecSize        Pointer to the size of the specification structure.

Description

This function computes the size of the specification structure required for the `ippiErode` functions with the `_L` suffix.

Return Values

ippStsNoErr      Indicates no error. Any other value indicates an error or a warning.
ippStsNullPtrErr Indicates an error when one of the pointers is NULL.
ippStsSizeErr    Width of the image, or width or height of the structuring element is less than, or equal to zero.

ErodeInit

Initializes the internal state or specification structure for the Erode function.

Syntax

IppStatus ippiErodeInit_L(IppiSizeL roiSize, const Ipp8u* pMask, IppiSizeL maskSize, IppiMorphStateL* pMorphSpec);

Include Files

ippcv_l.h

Domain Dependencies

Headers: ippcore.h, ippvm.h, ipps.h,ippi.h
Libraries: ippcore.lib, ippvm.lib, ipps.lib,ippi.lib

Parameters

roiSize          Size of the source and destination image ROI, in pixels.
pMask            Pointer to the structuring element (mask).
maskSize         Size of the structuring element.
pMorphSpec       Pointer to the morphology specification structure.

Description

This function initializes the internal state or specification structure for the `ippiErode` functions with the `_L` suffix.

Return Values

ippStsNoErr      Indicates no error. Any other value indicates an error or a warning.
Indicates an error when one of the pointers is `NULL`.

Width of the image, or width or height of the structuring element is less than, or equal to zero.

Anchor point is outside the structuring element.

**GrayDilateBorder**

Performs gray-kernel dilation of an image.

**Syntax**

```c
IppStatus ippiGrayDilateBorder_8u_C1R(const Ipp8u* pSrc, int srcStep, Ipp8u* pDst, int dstStep, IppiSize roiSize, IppiBorderType border, IppiMorphGrayState_8u* pState);
```

```c
IppStatus ippiGrayDilateBorder_32f_C1R(const Ipp32f* pSrc, int srcStep, Ipp32f* pDst, int dstStep, IppiSize roiSize, IppiBorderType border, IppiMorphGrayState_32f* pState);
```

**Include Files**

ippcv.h

**Domain Dependencies**

Headers: ippcore.h, ippvm.h, ipps.h,ippi.h

Libraries: ippcore.lib, ippvm.lib, ipps.lib,ippi.lib

**Parameters**

- `pSrc` Pointer to the source image ROI.
- `srcStep` Distance in bytes between starts of consecutive lines in the source image.
- `pDst` Pointer to the destination image ROI.
- `dstStep` Distance in bytes between starts of consecutive lines in the destination image.
- `roiSize` Size of the source and destination image ROI.
- `border` Type of border; the possible value is `ippBorderRepl`, which means that a replicated border is used.
- `pState` Pointer to the morphology state structure.

**Description**

This function operates with ROI (see Regions of Interest in Intel IPP).

This function performs gray-kernel dilation of a rectangular ROI area inside a one-channel 2D image using a specified in the gray-kernel morphology state structure `pState` mask and the anchor cell. This structure must be initialized by `MorphGrayInit` beforehand.

**NOTE**

The structure can be used to process images with ROI that does not exceed the specified maximum width and height `roiSize`.  

411
Return Values

ippStsNoErr Indicates no error. Any other value indicates an error or a warning.
ippStsNullPtrErr Indicates an error condition if one of the specified pointers is NULL.
ippStsSizeErr Indicates an error condition if roiSize has a field with a zero or negative value, or if one of ROI width or height is greater than corresponding size of ROI passed to the initialization functions.
ippStsStepErr Indicates an error condition if srcStep or dstStep is less than roiSize.width * <pixelSize>.
ippStsNotEvenStepErr Indicates an error condition if one of the step values is not divisible by 4 for floating-point images.
ippStsBorderErr Indicates an error condition if border has an illegal value.

**GrayErodeBorder**

*Performs gray-kernel erosion of an image.*

**Syntax**

IppStatus ippiGrayErodeBorder_8u_C1R(const Ipp8u* pSrc, int srcStep, Ipp8u* pDst, int dstStep, IppiSize roiSize, IppiBorderType border, IppiMorphGrayState_8u* pState);
IppStatus ippiGrayErodeBorder_32f_C1R(const Ipp32f* pSrc, int srcStep, Ipp32f* pDst, int dstStep, IppiSize roiSize, IppiBorderType border, IppiMorphGrayState_32f* pState);

**Include Files**

ippcv.h

**Domain Dependencies**

Headers: ippcore.h, ippvm.h, ipps.h,ippi.h
Libraries: ippcore.lib, ippvm.lib, ipps.lib,ippi.lib

**Parameters**

- **pSrc**: Pointer to the source image ROI.
- **srcStep**: Distance in bytes between starts of consecutive lines in the source image.
- **pDst**: Pointer to the destination image ROI.
- **dstStep**: Distance in bytes between starts of consecutive lines in the destination image.
- **roiSize**: Size of the source and destination image ROI.
- **border**: Type of border; the possible value is ippBorderRepl, which means that a replicated border is used.
- **pState**: Pointer to the morphology state structure.

**Description**

This function operates with ROI (see Regions of Interest in Intel IPP).
This function performs gray-kernel erosion of a rectangular ROI area inside a one-channel 2D image using a specified in the gray-kernel morphology state structure `pState` mask and the anchor cell. This structure must be initialized by `MorphGrayInit` beforehand.

**NOTE**
The structure can be used to process images with ROI that does not exceed the specified maximum width and height `roiSize`.

**Return Values**
- `ippStsNoErr`: Indicates no error. Any other value indicates an error or a warning.
- `ippStsNullPtrErr`: Indicates an error condition if one of the specified pointers is NULL.
- `ippStsSizeErr`: Indicates an error condition if `roiSize` has a field with a zero or negative value, or if one of ROI width or height is greater than corresponding size of ROI passed to the initialization functions.
- `ippStsStepErr`: Indicates an error condition if `srcStep` or `dstStep` is less than `roiSize.width * <pixelSize>`.
- `ippStsNotEvenStepErr`: Indicates an error condition if one of the step values is not divisible by 4 for floating-point images.
- `ippStsBorderErr`: Indicates an error condition if `border` has an illegal value.

**MorphAdvInit**

*Initializes the specification structure for advanced morphological operations.*

**Syntax**

```c
IppStatus ippiMorphAdvInit_<mod>(IppiSize roiSize, const Ipp8u* pMask, IppiSize maskSize, IppiMorphAdvState* pSpec, Ipp8u* pBuffer);
```

*Supported values for mod:*

- `1u_C1R`
- `8u_C1R`
- `16u_C1R`
- `16s_C1R`
- `32f_C1R`
- `8u_C3R`
- `32f_C3R`
- `8u_C4R`
- `32f_C4R`

**Include Files**

`ippcv.h`

**Domain Dependencies**

*Headers: ippcore.h, ippvm.h, ipps.h,ippi.h*

*Libraries: ippcore.lib, ippvm.lib, ipps.lib,ippi.lib*

**Parameters**

- `pSpec`: Pointer to the specification structure.
**roiSize**
Maximal size of the image ROI (in pixels) that can be processed using the allocated structure.

**pMask**
Pointer to the mask.

**maskSize**
Size of the mask in pixels.

**pBuffer**
Pointer to the external buffer for advanced morphological operations.

**Description**
This function operates with ROI.

This function initializes the specification structure pSpec in the external buffer. This structure is used by the MorphOpenBorder and MorphCloseBorder functions that perform open and close morphological operations.

All advanced morphological operations are performed on the source image pixels corresponding to non-zero values of the structuring element (mask) pMask.

**NOTE**
This function required that the image ROI does not exceed the maximum width and height roiSize specified by the initialization functions.

For an example on how to use this function, see the code example provided with the MorphCloseBorder function description.

**Return Values**
- ippStsNoErr: Indicates no error.
- ippStsNullPtrErr: Indicates an error when one of the specified pointers is NULL.
- ippStsSizeErr: Indicates an error when maskSize has a field with a zero or negative value, or if roiWidth is less than 1.
- ippStsAnchorErr: Indicates an error when anchor is outside the mask.

**See Also**
- Regions of Interest in Intel IPP
- MorphAdvGetSize Computes the size of the specification structure for advanced morphological operations.
- MorphOpenBorder Opens an image.
- MorphCloseBorder Closes an image.
- MorphTophatBorder Performs top-hat operation on an image.
- MorphBlackhatBorder Performs black-hat operation on an image.
- MorphGradientBorder Calculates morphological gradient of an image.

**MorphAdvGetSize**
Computes the size of the specification structure for advanced morphological operations.
Syntax

Case 2: Computing the size of morphology specification structure

IppStatus ippiMorphAdvGetSize_<mod>(IppiSize roiSize, IppiSize maskSize, int* pSpecSize, int* pBufferSize);

Supported values for mod:

1u_C1R 8u_C1R 16u_C1R 16s_C1R 32f_C1R
8u_C3R 32f_C3R
8u_C4R 32f_C4R

Include Files

ippcv.h

Domain Dependencies

Headers: ippcore.h, ippvm.h, ipps.h,ippi.h
Libraries: ippcore.lib, ippvm.lib, ipps.lib,ippi.lib

Parameters

roiSize
Maximal size of the image ROI (in pixels) that can be processed using the allocated structure.

maskSize
Size of the mask, in pixels.

pSpecSize
Pointer to the size of the morphology specification structure.

pBufferSize
Pointer to the size of the external buffer required for advanced morphological operations.

Description

This function operates with ROI.

This function computes the size of the specification structure and the size of the buffer required for advanced morphological operations. Call this function before using the ippiMorphAdvInit function.

For an example on how to use this function, see the code example provided with the ippiMorphCloseBorder function description.

Return Values

ippStsNoErr Indicates no error.
ippStsNullPtrErr Indicates an error when one of the specified pointers is NULL.
ippStsSizeErr Indicates an error when maskSize or roiSize has a field with a zero or negative value.

See Also

Regions of Interest in Intel IPP
MorphAdvInit Initializes the specification structure for advanced morphological operations.
MorphCloseBorder Closes an image.
MorphGetBufferSize

Computes the size of the working buffer for advanced morphological operations.

Syntax

IppStatus ippiMorphGetBufferSize_L (IppiSizeL roiSize, IppiSizeL maskSize, IppDataType dataType, int numChannels, IppSizeL* pBufferSize);

Include Files

ippcv_l.h

Domain Dependencies

Headers: ippcore.h, ippvm.h, ipps.h, ippi.h
Libraries: ippcore.lib, ippvm.lib, ipps.lib, ippi.lib

Parameters

roiSize Maximum size of the image ROI, in pixels.
maskSize Size of the structuring element.
dataType Data type for the processing function.
numChannels Number of channels in the image.
pBufferSize Pointer to the buffer size value for the initialization function.

Description

This function computes the size of the working buffer required for advanced morphological operations.

Return Values

ippStsNoErr Indicates no error. Any other value indicates an error or a warning.
ippStsNullPtrErr Indicates an error when one of the pointers is NULL.
ippStsSizeErr Width of the image, or width or height of the structuring element is less than, or equal to zero.

MorphGetSpecSize

Computes the size of the internal state or specification structure for advanced morphological operations.

Syntax

ippiMorphGetSpecSize_L(IppiSizeL roiSize, IppiSizeL maskSize, IppDataType dataType, int numChannels, IppSizeL* pSpecSize);

Include Files

ippcv_l.h
Domain Dependencies

Headers: ippcore.h, ippvm.h, ipps.h,ippi.h
Libraries: ippcore.lib, ippvm.lib, ipps.lib,ippi.lib

Parameters

roiSize       Maximum size of the image ROI, in pixels.
maskSize      Size of the structuring element.
dataType      Data type for the processing function.
numChannels   Number of channels in the image.
pSpecSize     Pointer to the size of the specification structure.

Description

This function computes the size of the specification structure for advanced morphological operations.

Return Values

ippStsNoErr   Indicates no error. Any other value indicates an error or a warning.
ippStsNullPtrErr   Indicates an error when one of the pointers is NULL.
ippStsSizeErr   Width of the image, or width or height of the structuring element is less than, or equal to zero.

MorphInit

Initializes the internal state or specification structure for advanced morphological operations.

Syntax

IppStatus ippiMorphInit_L( IppiSizeL roiSize, const Ipp8u* pMask, IppiSizeL maskSize, IppDataType dataType, int numChannels, IppiMorphAdvStateL* pMorphSpec);

Include Files

ippcv_l.h

Domain Dependencies

Headers: ippcore.h, ippvm.h, ipps.h,ippi.h
Libraries: ippcore.lib, ippvm.lib, ipps.lib,ippi.lib

Parameters

roiSize       Maximum size of the image ROI, in pixels.
pMask        Pointer to the structuring element (mask).
maskSize      Size of the structuring element.
dataType      Data type for the processing function.
umChannels   Number of channels in the image.
**Description**

This function initializes the internal state or specification structure for advanced morphological operations.

**Return Values**

- `ippStsNoErr` Indicates no error. Any other value indicates an error or a warning.
- `ippStsNullPtrErr` Indicates an error when one of the pointers is `NULL`.
- `ippStsSizeErr` Width of the image, or width or height of the structuring element is less than, or equal to zero.
- `ippStsAnchorErr` Anchor point is outside the structuring element.

**MorphologyBorderGetSize**

**Description**

This function operates with ROI.

This function computes the size of the morphology specification structure `pMorphSpec` and the size of the buffer required for dilation and erosion operations. Call this function before using the `ippiMorphologyBorderInit` function.

**Syntax**

```c
IppStatus ippiMorphologyBorderGetSize_<mod>(IppiSize roiSize, IppiSize maskSize, int* pSpecSize, int* pBufferSize);
```

**Include Files**

`ippcv.h`

**Domain Dependencies**

**Headers:** `ippcore.h`, `ippvm.h`, `ipps.h`, `ippi.h`

**Libraries:** `ippcore.lib`, `ippvm.lib`, `ipps.lib`, `ippi.lib`

**Parameters**

- `roiSize` Size of the image ROI, in pixels.
- `maskSize` Size of the mask, in pixels.
- `pSpecSize` Pointer to the size of the morphology specification structure.
- `pBufferSize` Pointer to the size of the buffer required for dilation or erosion operations.

**Description**

This function operates with ROI.
Return Values

ippStsNoErr  Indicates no error.
ippStsNullPtrErr  Indicates an error when one of the specified pointers is NULL.
ippStsSizeErr  Indicates an error when maskSize has a field with a zero or negative value, or if width or height of roiSize is less than 1.

See Also
Regions of Interest in Intel IPP
MorphologyBorderInit Initializes the morphology specification structure for erosion or dilation operations.

MorphologyBorderInit
Initializes the morphology specification structure for erosion or dilation operations.

Syntax
IppStatus ippiMorphologyBorderInit_<mod>(IppSize roiSize, const Ipp8u* pMask, IppiSize maskSize, IppiMorphState* pSpec, Ipp8u* pBuffer);

Supported values for mod:

1u_C1R  8u_C1R  16u_C1R  16s_C1R  32f_C1R
  8u_C3R       32f_C3R
  8u_C4R       32f_C4R

Include Files
ippcv.h

Domain Dependencies
Headers:  ippcore.h, ippvm.h, ipps.h,ippi.h
Libraries:  ippcore.lib, ippvm.lib, ipps.lib,ippi.lib

Parameters
roiSize  Size of the image ROI, in pixels.
pMask  Pointer to the mask.
maskSize  Size of the mask, in pixels.
pSpec  Pointer to the specification structure.
pBuffer  Pointer to the external buffer required for dilation or erosion operations.

Description
This function operates with ROI.
This function initializes the specification structure \( pSpec \) in the external buffer. Before using this function, you need to compute the size of the specification structure using the `ippiMorphologyBorderGetSize` function. This structure is used by the `ippiDilateBorder` and `ippiErodeBorder` functions that perform morphological operations on the source image pixels corresponding to non-zero values of the structuring element (mask) \( pMask \).

**Return Values**

- `ippStsNoErr`: Indicates no error.
- `ippStsNullPtrErr`: Indicates an error when one of the specified pointers is `NULL`.
- `ippStsSizeErr`: Indicates an error when `maskSize` has a field with a zero or negative value, or if width or height of `roiSize` is less than 1.

**See Also**

- **Regions of Interest in Intel IPP**
- `ippiMorphologyBorderGetSize`: Computes the size of the morphology specification structure.
- `ippiDilateBorder`: Performs dilation of an image.
- `ippiErodeBorder`: Performs erosion of an image.

**MorphBlackhat**

*Performs top-hat operation on an image.*

**Syntax**

```c
IppStatus ippiMorphBlackhat_16u_C1R_L(const Ipp16u* pSrc, IppSizeL srcStep, Ipp16u* pDst, IppSizeL dstStep, IppSizeL roiSize, IppiBorderType borderType, const Ipp16u borderValue[1], const IppiMorphAdvStateL* pMorthSpec, Ipp8u* pBuffer);
```

```c
IppStatus ippiMorphBlackhat_16s_C1R_L(const Ipp16s* pSrc, IppSizeL srcStep, Ipp16s* pDst, IppSizeL dstStep, IppSizeL roiSize, IppiBorderType borderType, const Ipp16s borderValue[1], const IppiMorphAdvStateL* pMorthSpec, Ipp8u* pBuffer);
```

```c
IppStatus ippiMorphBlackhat_1u_C1R_L(const Ipp8u* pSrc, IppSizeL srcStep, int srcBitOffset, Ipp8u* pDst, IppSizeL dstStep, int dstBitOffset, IppSizeL roiSize, IppiBorderType borderType, const Ipp8u borderValue[1], const IppiMorphAdvStateL* pMorthSpec, Ipp8u* pBuffer);
```

```c
IppStatus ippiMorphBlackhat_8u_C3R_L(const Ipp8u* pSrc, IppSizeL srcStep, int srcBitOffset, Ipp8u* pDst, IppSizeL dstStep, IppSizeL roiSize, IppiBorderType borderType, const Ipp8u borderValue[3], const IppiMorphAdvStateL* pMorthSpec, Ipp8u* pBuffer);
```

```c
IppStatus ippiMorphBlackhat_8u_C4R_L(const Ipp8u* pSrc, IppSizeL srcStep, int srcBitOffset, Ipp8u* pDst, IppSizeL dstStep, IppSizeL roiSize, IppiBorderType borderType, const Ipp8u borderValue[4], const IppiMorphAdvStateL* pMorthSpec, Ipp8u* pBuffer);
```

```c
IppStatus ippiMorphBlackhat_32f_C1R_L(const Ipp32f* pSrc, IppSizeL srcStep, Ipp32f* pDst, IppSizeL dstStep, IppSizeL roiSize, IppiBorderType borderType, const Ipp32f borderValue[1], const IppiMorphAdvStateL* pMorthSpec, Ipp8u* pBuffer);
```
Include Files
ippcv_l.h

Domain Dependencies
Headers: ippcore.h, ippvm.h, ipps.h,ippi.h
Libraries: ippcore.lib, ippvm.lib, ipps.lib,ippi.lib

Parameters
pSrc  Pointer to the source image ROI.
srcStep Distance, in bytes, between the starting points of consecutive lines in the source image.
srcBitOffset Offset, in bits, from the first byte of the source image row.
pDst  Pointer to the destination image ROI.
dstStep Distance, in bytes, between the starting points of consecutive lines in the destination image.
dstBitOffset Offset, in bits, from the first byte of the destination image row.
roiSize Size of the source and destination image ROI.
borderType Type of border. Possible values are:
ippBorderDefault The border is set to ippBorderConst with borderValue= MAX_VALUE, where
MAX_VALUE=IPP_MAX_8U/16U/16S/32F/1U
ippBorderRepl Border is replicated from the edge pixels.
ippBorderMirror Border pixels are mirrored from the source image boundary pixels.
ippBorderConst Values of all border pixels are set to a constant.
ippBorderFirstStageInMem You can use this border type together with the ippBorderRepl, ippBorderMirror, ippBorderConst, ippBorderDefault types using the | operation. For the first stage, border pixels are obtained from the source image pixels in memory. For the second stage, the function uses the border type specified with the | operation.
Mixed borders are also supported. They can be obtained by the bitwise operation \texttt{OR} between any of the \texttt{ippBorderRepl}, \texttt{ippBorderConst}, \texttt{ippBorderDefault}, or \texttt{ippBorderMirror} values and the \texttt{ippBorderInMemTop}, \texttt{ippBorderInMemBottom}, \texttt{ippBorderInMemLeft}, \texttt{ippBorderInMemRight} modifiers.

\begin{itemize}
  \item \texttt{borderValue}, \texttt{borderValue[3]}, \texttt{borderValue[4]}
  \item \texttt{pMorphSpec}
  \item \texttt{pBuffer}
\end{itemize}

\textbf{Description}

Before using this function, you need to initialize the morphology specification structure using the \texttt{ippiMorphInit} function.

This function performs a black-hat operation on a rectangular ROI area inside a one-, three-, or four-channel 2D image using a specified in the advanced morphology state or specification structure mask and the anchor cell.

The result is equivalent to the subtraction of the initial source image from the closed source image.

\textbf{NOTE}

The function can only process a ROI that does not exceed the maximum width and height \texttt{roiSize} specified by the initialization functions.

\textbf{Return Values}

- \texttt{ippStsNoErr} Indicates no error.
- \texttt{ippStsNullPtrErr} Indicates an error when one of the specified pointers is \texttt{NULL}.
- \texttt{ippStsSizeErr} Indicates an error condition when:
  \begin{itemize}
    \item \texttt{roiSize} has a field with a zero or negative value
    \item ROI width is more than ROI width passed to the initialization function
  \end{itemize}
- \texttt{ippStsStepErr} Indicates an error condition when \texttt{srcStep} or \texttt{dstStep} is less than \texttt{roiSize.width * <pixelSize>}
- \texttt{ippStsNotEvenStepErr} Indicates an error when one of the step values is not a multiple of an element size.
- \texttt{ippStsBadArgErr} Indicates an error when \texttt{borderType} has an illegal value.

\textbf{See Also}

- Regions of Interest in Intel IPP
- User-defined Border Types
- \texttt{MorphInit} Initializes the internal state or specification structure for advanced morphological operations.
MorphBlackhatBorder

Performs black-hat operation on an image.

Syntax

IppStatus ippiMorphBlackhatBorder_<mod>(const Ipp<datatype>* pSrc, int srcStep, Ipp<datatype>* pDst, int dstStep, IppiSize roiSize, IppiBorderType borderType, Ipp<datatype> borderValue, const IppiMorphAdvState* pSpec, Ipp8u* pBuffer);

Supported values for mod:

- 8u_C1R 16u_C1R 16s_C1R 32f_C1R

IppStatus ippiMorphBlackhatBorder_<mod>(const Ipp<datatype>* pSrc, int srcStep, Ipp<datatype>* pDst, int dstStep, IppiSize roiSize, IppiBorderType borderType, const Ipp<datatype> borderValue[3], const IppiMorphAdvState* pSpec, Ipp8u* pBuffer);

Supported values for mod:

- 8u_C3R 32f_C3R

IppStatus ippiMorphBlackhatBorder_<mod>(const Ipp<datatype>* pSrc, int srcStep, Ipp<datatype>* pDst, int dstStep, IppiSize roiSize, IppiBorderType borderType, const Ipp<datatype> borderValue[4], const IppiMorphAdvState* pSpec, Ipp8u* pBuffer);

Supported values for mod:

- 8u_C4R 32f_C4R

IppStatus ippiMorphBlackhatBorder_1u_C1R(const Ipp8u* pSrc, int srcStep, int srcBitOffset, Ipp8u* pDst, int dstStep, int dstBitOffset, IppiSize roiSize, IppiBorderType borderType, Ipp8u borderValue, const IppiMorphAdvState* pSpec, Ipp8u* pBuffer);

Include Files

ippcv.h

Domain Dependencies

Headers: ippcore.h, ippvm.h, ipps.h,ippi.h

Libraries: ippcore.lib, ippvm.lib, ipps.lib,ippi.lib

Parameters

- pSrc: Pointer to the source image ROI.
- srcStep: Distance, in bytes, between the starting points of consecutive lines in the source image.
- srcBitOffset: Offset, in bits, from the first byte of the source image (for the 1u_C1R flavor).
- pDst: Pointer to the destination image ROI.
- dstStep: Distance, in bytes, between the starting points of consecutive lines in the destination image.
**dstBitOffset**
Offset, in bits, from the first byte of the destination image (for the 1u_CLR flavor).

**roiSize**
Size of the source and destination image ROI.

**borderType**
Type of border. Possible values are:
- ippBorderRepl: Border is replicated from the edge pixels.
- ippBorderInMem: Border is obtained from the source image pixels in memory.

**borderValue, borderValue[3], borderValue[4]**
Constant value to assign to pixels of the constant border. This parameter is applicable only to the ippBorderConst border type.

**pSpec**
Pointer to the specification structure.

**pBuffer**
Pointer to the external buffer.

**Description**
This function operates with ROI (see Regions of Interest in Intel IPP).

This function performs black-hat operation on a rectangular ROI area inside a one-, three-, or four-channel 2D image using a specified in the advanced morphology specification structure mask and anchor cell. The structure must be initialized by **MorphAdvInit** beforehand.

The result is equivalent to the subtraction of the initial source image from the closed source image.

**NOTE**
The function can process only images with ROI that does not exceed the maximum width and height **roiSize** specified by the initialization functions.

**Return Values**

**ippStsNoErr**
Indicates no error. Any other value indicates an error or a warning.

**ippStsNullPtrErr**
Indicates an error condition if one of the specified pointers is NULL.

**ippStsSizeErr**
Indicates an error condition if **roiSize** has a field with a zero or negative value, or if one of ROI width or height is greater than corresponding size of ROI passed to the initialization functions.

**ippStsStepErr**
Indicates an error condition if **srcStep** or **dstStep** is less than **roiSize.width * <pixelSize>**.

**ippStsNotEvenStepErr**
Indicates an error condition if one of the step values is not divisible by 4 for floating-point images.

**ippStsBorderErr**
Indicates an error condition if **borderType** has an illegal value.

**See Also**
**MorphAdvInit** Initializes the specification structure for advanced morphological operations.

**MorphClose**
*Closes an image.*
Morphological Operations

### Syntax

```c
IppStatusippiMorphClose_8u_C1R_L(constIpp8u*pSrc,IppSizeLsrcStep,Ipp8u*pDst,
IppSizeLdstStep,IppiSizeLroiSize,IppiBorderTypeborderType,constIpp8uborderValue[1],constIppiMorphAdvStateL*pMorphSpec,Ipp8u*pBuffer);
```

```c
IppStatusippiMorphClose_8u_C3R_L(constIpp8u*pSrc,IppSizeLsrcStep,Ipp8u*pDst,
IppSizeLdstStep,IppiSizeLroiSize,IppiBorderTypeborderType,constIpp8uborderValue[3],constIppiMorphAdvStateL*pMorphSpec,Ipp8u*pBuffer);
```

```c
IppStatusippiMorphClose_8u_C4R_L(constIpp8u*pSrc,IppSizeLsrcStep,Ipp8u*pDst,
IppSizeLdstStep,IppiSizeLroiSize,IppiBorderTypeborderType,constIpp8uborderValue[4],constIppiMorphAdvStateL*pMorphSpec,Ipp8u*pBuffer);
```

```c
IppStatusippiMorphClose_16u_C1R_L(constIpp16u*pSrc,IppSizeLsrcStep,Ipp16u*pDst,
IppSizeLdstStep,IppiSizeLroiSize,IppiBorderTypeborderType,constIpp16ubarborderValue[1],constIppiMorphAdvStateL*pMorphSpec,Ipp8u*pBuffer);
```

```c
IppStatusippiMorphClose_16u_C3R_L(constIpp16u*pSrc,IppSizeLsrcStep,Ipp16u*pDst,
IppSizeLdstStep,IppiSizeLroiSize,IppiBorderTypeborderType,constIpp16ubarborderValue[3],constIppiMorphAdvStateL*pMorphSpec,Ipp8u*pBuffer);
```

```c
IppStatusippiMorphClose_16u_C4R_L(constIpp16u*pSrc,IppSizeLsrcStep,Ipp16u*pDst,
IppSizeLdstStep,IppiSizeLroiSize,IppiBorderTypeborderType,constIpp16ubarborderValue[4],constIppiMorphAdvStateL*pMorphSpec,Ipp8u*pBuffer);
```

```c
IppStatusippiMorphClose_1u_C1R_L(constIpp8u*pSrc,IppSizeLsrcStep, intptr,pDst,
IppSizeLdstStep, intptr dstBitOffset, IppSizeL roiSize, IppiBorderTypeborderType, constIpp8ubarborderValue[1], constIppiMorphAdvStateL*pMorphSpec,Ipp8u*pBuffer);
```

```c
IppStatusippiMorphClose_32f_C1R_L(constIpp32f*pSrc,IppSizeLsrcStep,Ipp32f*pDst,
IppSizeLdstStep,IppiSizeLroiSize,IppiBorderTypeborderType,constIpp32fborderValue[1],constIppiMorphAdvStateL*pMorphSpec,Ipp8u*pBuffer);
```

```c
IppStatusippiMorphClose_32f_C3R_L(constIpp32f*pSrc,IppSizeLsrcStep,Ipp32f*pDst,
IppSizeLdstStep,IppiSizeLroiSize,IppiBorderTypeborderType,constIpp32fborderValue[3],constIppiMorphAdvStateL*pMorphSpec,Ipp8u*pBuffer);
```

```c
IppStatusippiMorphClose_32f_C4R_L(constIpp32f*pSrc,IppSizeLsrcStep,Ipp32f*pDst,
IppSizeLdstStep,IppiSizeLroiSize,IppiBorderTypeborderType,constIpp32fborderValue[4],constIppiMorphAdvStateL*pMorphSpec,Ipp8u*pBuffer);
```

### Include Files

```c
ippcv_l.h
```

### Domain Dependencies

**Headers:** ippcore.h, ippvm.h, ipps.h,ippi.h

**Libraries:** ippcore.lib, ippvm.lib, ipps.lib,ippi.lib

### Parameters

- **pSrc**
  Pointer to the source image ROI.

- **srcStep**
  Distance, in bytes, between the starting points of consecutive lines in the source image.

- **srcBitOffset**
  Offset, in bits, from the first byte of the source image row.

- **pDst**
  Pointer to the destination image ROI.

- **dstStep**
  Distance, in bytes, between the starting points of consecutive lines in the destination image.
Offset, in bits, from the first byte of the destination image row.

Size of the source and destination image ROI.

Type of border. Possible values are:

- **ippBorderDefault**: The border is set to ippBorderConst with borderValue= MAX_VALUE, where MAX_VALUE=IPP_MAX_8U/16U/16S/32F/1U
- **ippBorderRepl**: Border is replicated from the edge pixels.
- **ippBorderMirror**: Border pixels are mirrored from the source image boundary pixels.
- **ippBorderConst**: Values of all border pixels are set to a constant.
- **ippBorderFirstStageInMem**: You can use this border type together with the ippBorderRepl, ippBorderMirror, ippBorderConst, ippBorderDefault types using the | operation. For the first stage, border pixels are obtained from the source image pixels in memory. For the second stage, the function uses the border type specified with the | operation.

Mixed borders are also supported. They can be obtained by the bitwise operation OR between any of the ippBorderRepl, ippBorderConst, ippBorderDefault, or ippBorderMirror values and the ippBorderInMemTop, ippBorderInMemBottom, ippBorderInMemLeft, ippBorderInMemRight modifiers.

- **borderValue**: Constant value to assign to pixels of the constant border. This parameter is applicable only to the ippBorderConst border type.
- **pMorphSpec**: Pointer to the morphology specification structure.
- ** pBuffer**: Pointer to the external buffer.

**Description**

Before using this function, you need to initialize the morphology specification structure using the ippiMorphInit function.

This function operates with ROI.

This function performs closing of a rectangular ROI area inside a one-, three-, or four-channel 2D image using the mask specified in the pMorphSpec structure.

The result is equivalent to successive dilation of the source image by the structured element (mask) and erosion by the reverted structured element.

**NOTE**

The function can only process a ROI that does not exceed the maximum width and height roiSize specified by the initialization functions.
Return Values

- **ippStsNoErr**: Indicates no error.
- **ippStsNullPtrErr**: Indicates an error when one of the specified pointers is NULL.
- **ippStsSizeErr**: Indicates an error condition when:
  - `roiSize` has a field with a zero or negative value
  - ROI width is more than ROI width passed to the initialization function
- **ippStsStepErr**: Indicates an error condition when `srcStep` or `dstStep` is less than `roiSize.width * <pixelSize>`.
- **ippStsNotEvenStepErr**: Indicates an error when one of the step values is not a multiple of an element size.
- **ippStsBadArgErr**: Indicates an error when `borderType` has an illegal value.

See Also
- Regions of Interest in Intel IPP
- User-defined Border Types
- **MorphInit** initializes the internal state or specification structure for advanced morphological operations.

**MorphCloseBorder**

Closes an image.

Syntax

```c
IppStatus ippiMorphCloseBorder_<mod>(const Ipp<datatype>* pSrc, int srcStep, Ipp<datatype>* pDst, int dstStep, IppiSize roiSize, IppiBorderType borderType, Ipp<datatype> borderValue, const IppiMorphAdvState* pSpec, Ipp8u* pBuffer);
```

Supported values for `mod`:

- `8u_C1R`  `16u_C1R`  `16s_C1R`  `32f_C1R`

```c
IppStatus ippiMorphCloseBorder_<mod>(const Ipp<datatype>* pSrc, int srcStep, Ipp<datatype>* pDst, int dstStep, IppiSize roiSize, IppiBorderType borderType, const Ipp<datatype> borderValue[3], const IppiMorphAdvState* pSpec, Ipp8u* pBuffer);
```

Supported values for `mod`:

- `8u_C3R`  `32f_C3R`

```c
IppStatus ippiMorphCloseBorder_<mod>(const Ipp<datatype>* pSrc, int srcStep, Ipp<datatype>* pDst, int dstStep, IppiSize roiSize, IppiBorderType borderType, const Ipp<datatype> borderValue[4], const IppiMorphAdvState* pSpec, Ipp8u* pBuffer);
```

Supported values for `mod`:

- `8u_C4R`  `32f_C4R`

```c
IppStatus ippiMorphCloseBorder_1u_C1R(const Ipp8u* pSrc, int srcStep, int srcBitOffset, Ipp8u* pDst, int dstStep, int dstBitOffset, IppiSize roiSize, IppiBorderType borderType, Ipp8u borderValue, const IppiMorphAdvState* pSpec, Ipp8u* pBuffer);
```
Include Files
ippcv.h

Domain Dependencies
Headers: ippcore.h, ippvm.h, ipps.h, ippi.h
Libraries: ippcore.lib, ippvm.lib, ipps.lib,ippi.lib

Parameters

pSrc  
Pointer to the source image ROI.
srcStep  
Distance, in bytes, between the starting points of consecutive lines in the source image.
srcBitOffset  
Offset, in bits, from the first byte of the source image (for 1u_C1R flavor).
pDst  
Pointer to the destination image ROI.
dstStep  
Distance, in bytes, between the starting points of consecutive lines in the destination image.
dstBitOffset  
Offset, in bits, from the first byte of the destination image (for 1u_C1R flavor).
roiSize  
Size of the source and destination image ROI.
borderType  
Type of border. Possible values are:
ippBorderRepl  
Border is replicated from the edge pixels.
ippBorderInMem  
Border is obtained from the source image pixels in memory.
borderValue, borderValue[3], borderValue[4]  
Constant value to assign to pixels of the constant border. This parameter is applicable only to the ippBorderConst border type.
pSpec  
Pointer to the specification structure.
pBuffer  
Pointer to the external buffer required for dilation operations.

Description

Before using this function, you need to initialize the morphology specification structure using the MorphAdvInit function.

This function operates with ROI.

This function performs closing of a rectangular ROI area inside a one-, three-, or four-channel 2D image using the mask specified in the pSpec structure.

The result is equivalent to successive dilation of the source image by the structured element (mask) and erosion by the reverted structured element.
NOTE
This function requires that the image ROI does not exceed the maximum width and height
\textit{roiSize} specified by the initialization functions.

Usage example of this function is similar to the example provided with the \texttt{MorphOpenBorder} function description.

Return Values

\begin{itemize}
\item \texttt{ippStsNoErr} \hspace{1cm} Indicates no error.
\item \texttt{ippStsNullPtrErr} \hspace{1cm} Indicates an error when one of the specified pointers is \texttt{NULL}.
\item \texttt{ippStsSizeErr} \hspace{1cm} Indicates an error condition when:
  \begin{itemize}
  \item \texttt{roiSize} has a field with a zero or negative value
  \item one of the ROI width or height is more than the corresponding size
        of ROI passed to the initialization functions
  \item \texttt{srcBitOffset} or \texttt{dstBitOffset} is less than zero
  \end{itemize}
\item \texttt{ippStsStepErr} \hspace{1cm} Indicates an error condition if \texttt{srcStep} or \texttt{dstStep} is less than
  \texttt{roiSize.width * pixelSize}.
\item \texttt{ippStsNotEvenStepErr} \hspace{1cm} Indicates an error condition if one of the step values for 16-bit
  integer images is not divisible by 2.
\item \texttt{ippStsBorderErr} \hspace{1cm} Indicates an error condition if \texttt{borderType} has an illegal value.
\end{itemize}

Example

The code example below demonstrates how to use the \texttt{ippiMorphAdvGetSize\_16u\_C1R}, \texttt{ippiMorphAdvInit\_16u\_C1R}, and \texttt{ippiMorphCloseBorder\_16u\_C1R} functions.

\begin{verbatim}
IppStatus func_MorfCloseBorder()
{
    IppiMorphAdvState* pSpec = NULL;
    Ipp8u* pBuffer = NULL;
    IppiSize roiSize = {5, 5};
    Ipp8u pMask[3*3] = {1, 1, 1,
                       1, 0, 1,
                       1, 1, 1};
    IppiSize maskSize = {3, 3};
    Ipp16u pSrc[5*5] = { 1, 2, 4, 1, 2,
                        5, 1, 2, 1, 2,
                        1, 2, 1, 2, 1,
                        2, 1, 5, 1, 2};
    Ipp16u pDst[5*5];
    int srcStep = 5*\texttt{sizeof(Ipp16u)};
    int dstStep = 5*\texttt{sizeof(Ipp16u)};
    int dstSize = 5;
    IppStatus status = \texttt{ippStsNoErr};
    int specSize = 0, bufferSize = 0;
    IppiBorderType borderType = \texttt{ippBorderRepl};
    Ipp16u borderValue = 0;

    status = \texttt{ippiMorphAdvGetSize\_16u\_C1R( roiSize, maskSize, \&specSize, \&bufferSize )};
    if (status != \texttt{ippStsNoErr}) return status;

    pSpec = \texttt{ippiMorphAdvInit\_16u\_C1R( roiSize, maskSize, \&specSize, \&bufferSize, \&pSpec, \&pbuffer, \&specSize, \&bufferSize, \&specSize, \&bufferSize, borderType, \&borderValue )};
    if (status != \texttt{ippStsNoErr}) return status;

\end{verbatim}

\end{verbatim}
pSpec = (IppiMorphAdvState*)ippsMalloc_8u(specSize);
pBuffer = (Ipp8u*)ippsMalloc_8u(bufferSize);

status = ippiMorphAdvInit_16u_C1R( roiSize, pMask, maskSize, pSpec, pBuffer );
if (status != ippStsNoErr) {
    ippsFree(pBuffer);
    ippsFree(pSpec);
    return status;
}

status = ippiMorphCloseBorder_16u_C1R (pSrc, srcStep, pDst, dstStep, roiSize, borderType,
borderValue, pSpec, pBuffer);

ippsFree(pBuffer);
ippsFree(pSpec);
return status;

The result is as follows:

pDst->
5 4 4 2 2
5 4 4 2 2
2 2 2 2 2
2 2 5 2 2
2 2 2 2 2

See Also
Regions of Interest in Intel IPP
User-defined Border Types
MorphAdvInit Initializes the specification structure for advanced morphological operations.

MorphGradient

Calculates morphological gradient of an image.

Syntax

IppStatus ippiMorphGradient_16u_C1R_L(const Ipp16u* pSrc, IppSizeL srcStep, Ipp16u* pDst, IppSizeL dstStep, IppiSizeL roiSize, IppiBorderType borderType, const Ipp16u* borderValue[1], const IppiMorphAdvStateL* pMorphSpec, Ipp8u* pBuffer);

IppStatus ippiMorphGradient_16s_C1R_L(const Ipp16s* pSrc, IppSizeL srcStep, Ipp16s* pDst, IppSizeL dstStep, IppiSizeL roiSize, IppiBorderType borderType, const Ipp16s* borderValue[1], const IppiMorphAdvStateL* pMorphSpec, Ipp8u* pBuffer);

IppStatus ippiMorphGradient_1u_C1R_L(const Ipp8u* pSrc, IppSizeL srcStep, int srcBitOffset, Ipp8u* pDst, IppSizeL dstStep, int dstBitOffset, IppiSizeL roiSize, IppiBorderType borderType, const Ipp8u* borderValue[1], const IppiMorphAdvStateL* pMorphSpec, Ipp8u* pBuffer);

IppStatus ippiMorphGradient_8u_C1R_L(const Ipp8u* pSrc, IppSizeL srcStep, Ipp8u* pDst, IppSizeL dstStep, IppiSizeL roiSize, IppiBorderType borderType, const Ipp8u* borderValue[1], const IppiMorphAdvStateL* pMorphSpec, Ipp8u* pBuffer);

IppStatus ippiMorphGradient_8u_C3R_L(const Ipp8u* pSrc, IppSizeL srcStep, Ipp8u* pDst, IppSizeL dstStep, IppiSizeL roiSize, IppiBorderType borderType, const Ipp8u* borderValue[3], const IppiMorphAdvStateL* pMorphSpec, Ipp8u* pBuffer);
IppStatus ippiMorphGradient_8u_C4R_L(const Ipp8u* pSrc, IppSizeL srcStep, Ipp8u* pDst, IppSizeL dstStep, IppSizeL roiSize, IppiBorderType borderType, const Ipp8u borderValue[4], const IppiMorphAdvStateL* pMorphSpec, Ipp8u* pBuffer);

IppStatus ippiMorphGradient_32f_C1R_L(const Ipp32f* pSrc, IppSizeL srcStep, Ipp32f* pDst, IppSizeL dstStep, IppSizeL roiSize, IppiBorderType borderType, const Ipp32f borderValue[1], const IppiMorphAdvStateL* pMorphSpec, Ipp8u* pBuffer);

IppStatus ippiMorphGradient_32f_C3R_L(const Ipp32f* pSrc, IppSizeL srcStep, Ipp32f* pDst, IppSizeL dstStep, IppSizeL roiSize, IppiBorderType borderType, const Ipp32f borderValue[3], const IppiMorphAdvStateL* pMorphSpec, Ipp8u* pBuffer);

IppStatus ippiMorphGradient_32f_C4R_L(const Ipp32f* pSrc, IppSizeL srcStep, Ipp32f* pDst, IppSizeL dstStep, IppSizeL roiSize, IppiBorderType borderType, const Ipp32f borderValue[4], const IppiMorphAdvStateL* pMorphSpec, Ipp8u* pBuffer);

**Include Files**
ippcv_l.h

**Domain Dependencies**

**Headers:** ippcore.h, ippvm.h, ipps.h,ippi.h

**Libraries:** ippcore.lib, ippvm.lib, ipps.lib,ippi.lib

**Parameters**

- **pSrc**
  Pointer to the source image ROI.

- **srcStep**
  Distance, in bytes, between the starting points of consecutive lines in the source image.

- **srcBitOffset**
  Offset, in bits, from the first byte of the source image row.

- **pDst**
  Pointer to the destination image ROI.

- **dstStep**
  Distance, in bytes, between the starting points of consecutive lines in the destination image.

- **dstBitOffset**
  Offset, in bits, from the first byte of the destination image row.

- **roiSize**
  Size of the source and destination image ROI.

- **borderType**
  Type of border. Possible values are:
  - ippBorderDefault: The border is set to ippBorderConst with borderValue= MAX_VALUE, where MAX_VALUE=IPP_MAX_8U/16U/16S/32F/1U
  - ippBorderRepl: Border is replicated from the edge pixels.
  - ippBorderMirror: Border pixels are mirrored from the source image boundary pixels.
  - ippBorderConst: Values of all border pixels are set to a constant.
  - ippBorderFirstStageInMem: You can use this border type together with the ippBorderRepl, ippBorderMirror, ippBorderConst, ippBorderDefault types using the | operation. For the first stage, border pixels are obtained from the source image pixels.
in memory. For the second stage, the function uses the border type specified with the | operation.

Mixed borders are also supported. They can be obtained by the bitwise operation OR between any of the ippBorderRepl, ippBorderConst, ippBorderDefault, or ippBorderMirror values and the ippBorderInMemTop, ippBorderInMemBottom, ippBorderInMemLeft, ippBorderInMemRight modifiers.

borderValue, borderValue[3], borderValue[4]

Constant value to assign to pixels of the constant border. This parameter is applicable only to the ippBorderConst border type.

pMorphSpec

Pointer to the morphology specification structure.

pBuffer

Pointer to the external buffer.

Description

Before using this function, you need to initialize the morphology specification structure using the ippiMorphInit function.

This function calculates a morphological gradient of a rectangular ROI area inside a one-, three-, or four-channel 2D image using a specified in the advanced morphology specification structure mask and anchor cell. The result is equivalent to the subtraction of an opened source image from a closed source image.

NOTE

The function can only process a ROI that does not exceed the maximum width and height roiSize specified by the initialization functions.

Return Values

ippiStsNoErr

Indicates no error.

ippiStsNullPtrErr

Indicates an error when one of the specified pointers is NULL.

ippiStsSizeErr

Indicates an error condition when:

- roiSize has a field with a zero or negative value
- ROI width is more than ROI width passed to the initialization function

ippiStsStepErr

Indicates an error condition when srcStep or dstStep is less than roiSize.width * <pixelSize>.

ippiStsNotEvenStepErr

Indicates an error when one of the step values is not a multiple of an element size.

ippiStsBadArgErr

Indicates an error when borderType has an illegal value.

See Also

Regions of Interest in Intel IPP
User-defined Border Types
MorphInitInitializes the internal state or specification structure for advanced morphological operations.
**MorphGradientBorder**

Calculates morphological gradient of an image.

```
IppStatus ippiMorphGradientBorder_<mod>(const Ipp<datatype>* pSrc, int srcStep, Ipp<datatype>* pDst, int dstStep, IppiSize roiSize, IppiBorderType borderType, Ipp<datatype> borderValue, const IppiMorphAdvState* pSpec, Ipp8u* pBuffer);
```

Supported values for `mod`:

- 8u_C1R  16u_C1R  16s_C1R  32f_C1R

```
IppStatus ippiMorphGradientBorder_<mod>(const Ipp<datatype>* pSrc, int srcStep, Ipp<datatype>* pDst, int dstStep, IppiSize roiSize, IppiBorderType borderType, const Ipp<datatype> borderValue[3], const IppiMorphAdvState* pSpec, Ipp8u* pBuffer);
```

Supported values for `mod`:

- 8u_C3R  32f_C3R

```
IppStatus ippiMorphGradientBorder_<mod>(const Ipp<datatype>* pSrc, int srcStep, Ipp<datatype>* pDst, int dstStep, IppiSize roiSize, IppiBorderType borderType, const Ipp<datatype> borderValue[4], const IppiMorphAdvState* pSpec, Ipp8u* pBuffer);
```

Supported values for `mod`:

- 8u_C4R  32f_C4R

```
IppStatus ippiMorphGradientBorder_1u_C1R(const Ipp8u* pSrc, int srcStep, int srcBitOffset, Ipp8u* pDst, int dstStep, int dstBitOffset, IppiSize roiSize, IppiBorderType borderType, Ipp8u borderValue, const IppiMorphAdvState* pSpec, Ipp8u* pBuffer);
```

Include Files

ippcv.h

Domain Dependencies

Headers: ippcore.h, ippvm.h, ipps.h,ippi.h

Libraries: ippcore.lib, ippvm.lib, ipps.lib,ippi.lib

Parameters

- **pSrc**: Pointer to the source image ROI.
- **srcStep**: Distance in bytes between starts of consecutive lines in the source image.
- **pDst**: Pointer to the destination image ROI.
- **dstStep**: Distance in bytes between starts of consecutive lines in the destination image.
- **roiSize**: Size of the source and destination image ROI.
- **borderType**: Type of border; the possible value is ippBorderRepl, which means that a replicated border is used.
- **srcBitOffset**: Offset, in bits, from the first byte of the source image.
dstBitOffset
  Offset, in bits, from the first byte of the destination image.

pSpec
  Pointer to the specification structure.

pBuffer
  Pointer to the external work buffer.

borderValue, borderValue[3], borderValue[4]
  Constant value to assign to pixels of the constant border. This parameter is applicable only to the ippBorderConst border type.

Description
This function operates with ROI (see Regions of Interest in Intel IPP).

This function calculates a morphological gradient of a rectangular ROI area inside a one-, three-, or four-channel 2D image using a specified in the advanced morphology specification structure mask and anchor cell. The structure must be initialized by MorphAdvInit beforehand.

The result is equivalent to the subtraction of an opened source image from a closed source image.

NOTE
The function can process only images with ROI that does not exceed the maximum width and height roiSize specified by the initialization functions.

Return Values

ippStsNoErr
  Indicates no error. Any other value indicates an error or a warning.

ippStsNullPtrErr
  Indicates an error condition if one of the specified pointers is NULL.

ippStsSizeErr
  Indicates an error condition if roiSize has a field with a zero or negative value, or if one of ROI width or height is greater than corresponding size of ROI passed to the initialization functions.

ippStsStepErr
  Indicates an error condition if srcStep or dstStep is less than roiSize.width * <pixelSize>.

ippStsNotEvenStepErr
  Indicates an error condition if one of the step values is not divisible by 4 for floating-point images.

ippStsBorderErr
  Indicates an error condition if borderType has an illegal value.

See Also
MorphAdvInit Initializes the specification structure for advanced morphological operations.

MorphOpen

Opens an image.

Syntax

IppStatus ippiMorphOpen_16u_C1R_L(const Ipp16u* pSrc, IppSizeL srcStep, Ipp16u* pDst, IppSizeL dstStep, IppSizeL roiSize, IppiBorderType borderType, const Ipp16u* borderValue[1], const IppiMorphAdvStateL* pMorphSpec, Ipp8u* pBuffer );
IppStatus ippiMorphOpen_16s_C1R_L(const Ipp16s* pSrc, IppSizeL srcStep, Ipp16s* pDst, IppSizeL dstStep, IppSizeL roiSize, IppiBorderType borderType, const Ipp16s borderValue[1], const IppiMorphAdvStateL* pMorphSpec, Ipp8u* pBuffer);

IppStatus ippiMorphOpen_1u_C1R_L(const Ipp8u* pSrc, IppSizeL srcStep, int srcBitOffset, Ipp8u* pDst, IppSizeL dstStep, int dstBitOffset, IppSizeL roiSize, IppiBorderType borderType, const Ipp8u borderValue[1], const IppiMorphAdvStateL* pMorphSpec, Ipp8u* pBuffer);

IppStatus ippiMorphOpen_8u_C1R_L(const Ipp8u* pSrc, IppSizeL srcStep, Ipp8u* pDst, IppSizeL dstStep, IppSizeL roiSize, IppiBorderType borderType, const Ipp8u borderValue[1], const IppiMorphAdvStateL* pMorphSpec, Ipp8u* pBuffer);

IppStatus ippiMorphOpen_8u_C3R_L(const Ipp8u* pSrc, IppSizeL srcStep, Ipp8u* pDst, IppSizeL dstStep, IppSizeL roiSize, IppiBorderType borderType, const Ipp8u borderValue[3], const IppiMorphAdvStateL* pMorphSpec, Ipp8u* pBuffer);

IppStatus ippiMorphOpen_8u_C4R_L(const Ipp8u* pSrc, IppSizeL srcStep, Ipp8u* pDst, IppSizeL dstStep, IppSizeL roiSize, IppiBorderType borderType, const Ipp8u borderValue[4], const IppiMorphAdvStateL* pMorphSpec, Ipp8u* pBuffer);

IppStatus ippiMorphOpen_32f_C1R_L(const Ipp32f* pSrc, IppSizeL srcStep, Ipp32f* pDst, IppSizeL dstStep, IppSizeL roiSize, IppiBorderType borderType, const Ipp32f borderValue[1], const IppiMorphAdvStateL* pMorphSpec, Ipp8u* pBuffer);

IppStatus ippiMorphOpen_32f_C3R_L(const Ipp32f* pSrc, IppSizeL srcStep, Ipp32f* pDst, IppSizeL dstStep, IppSizeL roiSize, IppiBorderType borderType, const Ipp32f borderValue[3], const IppiMorphAdvStateL* pMorphSpec, Ipp8u* pBuffer);

IppStatus ippiMorphOpen_32f_C4R_L(const Ipp32f* pSrc, IppSizeL srcStep, Ipp32f* pDst, IppSizeL dstStep, IppSizeL roiSize, IppiBorderType borderType, const Ipp32f borderValue[4], const IppiMorphAdvStateL* pMorphSpec, Ipp8u* pBuffer);

**Include Files**

ippcv_l.h

**Domain Dependencies**

**Headers:** ippcore.h, ippvm.h, ipps.h,ippi.h

**Libraries:** ippcore.lib, ippvm.lib, ipps.lib,ippi.lib

**Parameters**

- **pSrc**
  - Pointer to the source image ROI.

- **srcStep**
  - Distance, in bytes, between the starting points of consecutive lines in the source image.

- **srcBitOffset**
  - Offset, in bits, from the first byte of the source image row.

- **pDst**
  - Pointer to the destination image ROI.

- **dstStep**
  - Distance, in bytes, between the starting points of consecutive lines in the destination image.

- **dstBitOffset**
  - Offset, in bits, from the first byte of the destination image row.

- **roiSize**
  - Size of the source and destination image ROI.

- **borderType**
  - Type of border. Possible values are:
ippBorderDefault

The border is set to ippBorderConst with borderValue = MAX_VALUE, where
MAX_VALUE = IPP_MAX_8U/16U/16S/32F/1U

ippBorderRepl

Border is replicated from the edge pixels.

ippBorderMirror

Border pixels are mirrored from the source image boundary pixels.

ippBorderConst

Values of all border pixels are set to a constant.

ippBorderFirstStageInMem

You can use this border type together with the ippBorderRepl, ippBorderMirror, ippBorderConst, ippBorderDefault types using the | operation. For the first stage, border pixels are obtained from the source image pixels in memory. For the second stage, the function uses the border type specified with the | operation.

Mixed borders are also supported. They can be obtained by the bitwise operation OR between any of the ippBorderRepl, ippBorderConst, ippBorderDefault, or ippBorderMirror values and the ippBorderInMemTop, ippBorderInMemBottom, ippBorderInMemLeft, ippBorderInMemRight modifiers.

borderValue, borderValue[3], borderValue[4]

Constant value to assign to pixels of the constant border. This parameter is applicable only to the ippBorderConst border type.

pMorphSpec

Pointer to the morphology specification structure.

pBuffer

Pointer to the external buffer.

Description

Before using this function, you need to initialize the morphology specification structure using the ippiMorphInit function.

This function operates with ROI.

This function performs opening of a rectangular ROI area inside a one-, three-, or four-channel 2D image using the mask specified in the pMorphSpec structure.

The result is equivalent to successive dilation of the source image by the structured element (mask) and erosion by the reverted structured element.

NOTE

The function can only process a ROI that does not exceed the maximum width and height roiSize specified by the initialization functions.

Return Values

ippStsNoErr

Indicates no error.

ippStsNullPtrErr

Indicates an error when one of the specified pointers is NULL.
ippStsSizeErr Indicates an error condition when:
• roiSize has a field with a zero or negative value
• ROI width is more than ROI width passed to the initialization function

ippStsStepErr Indicates an error condition when srcStep or dstStep is less than roiSize.width * <pixelSize>.

ippStsNotEvenStepErr Indicates an error when one of the step values is not a multiple of an element size.

ippStsBadArgErr Indicates an error when borderType has an illegal value.

See Also
Regions of Interest in Intel IPP
User-defined Border Types
MorphInitInitializes the internal state or specification structure for advanced morphological operations.

MorphOpenBorder
Opens an image.

Syntax
IppStatus ippiMorphOpenBorder_<mod>(const Ipp<datatype>* pSrc, int srcStep, Ipp<datatype>* pDst, int dstStep, IppiSize roiSize, IppiBorderType borderType, const Ipp<datatype> borderValue, const IppiMorphAdvState* pSpec, Ipp8u* pBuffer);

Supported values for mod:
8u_C1R 16u_C1R 16s_C1R 32f_C1R

IppStatus ippiMorphOpenBorder_<mod>(const Ipp<datatype>* pSrc, int srcStep, Ipp<datatype>* pDst, int dstStep, IppiSize roiSize, IppiBorderType borderType, const Ipp<datatype> borderValue[3], const IppiMorphAdvState* pSpec, Ipp8u* pBuffer);

Supported values for mod:
8u_C3R 32f_C3R

IppStatus ippiMorphOpenBorder_<mod>(const Ipp<datatype>* pSrc, int srcStep, Ipp<datatype>* pDst, int dstStep, IppiSize roiSize, IppiBorderType borderType, const Ipp<datatype> borderValue[4], const IppiMorphAdvState* pSpec, Ipp8u* pBuffer);

Supported values for mod:
8u_C4R 32f_C4R

IppStatus ippiMorphOpenBorder_1u_C1R(const Ipp8u* pSrc, int srcStep, int srcBitOffset, Ipp8u* pDst, int dstStep, int dstBitOffset, IppiSize roiSize, IppiBorderType borderType, Ipp8u borderValue, const IppiMorphAdvState* pSpec, Ipp8u* pBuffer);

Include Files
ippcv.h
Domain Dependencies

Headers: ippcore.h, ippvm.h, ipps.h,ippi.h
Libraries: ippcore.lib, ippvm.lib, ipps.lib,ippi.lib

Parameters

- **pSrc**: Pointer to the source image ROI.
- **srcStep**: Distance, in bytes, between the starting points of consecutive lines in the source image.
- **srcBitOffset**: Offset, in bits, from the first byte of the source image (for the 1u_C1R flavor).
- **pDst**: Pointer to the destination image ROI.
- **dstStep**: Distance, in bytes, between the starting points of consecutive lines in the destination image.
- **dstBitOffset**: Offset, in bits, from the first byte of the destination image (for the 1u_C1R flavor).
- **roiSize**: Size of the source and destination image ROI.
- **borderType**: Type of border. Possible values are:
  - ippBorderRepl: Border is replicated from the edge pixels.
  - ippBorderInMem: Border is obtained from the source image pixels in memory.
- **borderValue, borderValue[3], borderValue[4]**: Constant value to assign to pixels of the constant border. This parameter is applicable only to the ippBorderConst border type.
- **pSpec**: Pointer to the specification structure.
- **pBuffer**: Pointer to the external buffer required for dilation operations.

Description

Before using this function, you need to initialize the morphology specification structure by using MorphAdvInit function.

This function operates with ROI.

This function performs opening of a rectangular ROI area inside a one-, three-, or four-channel 2D image using the mask specified in the pSpec structure.

The result is equivalent to successive dilation of the source image by the structured element (mask) and erosion by the reverted structured element.

**NOTE**

The function can only process a ROI that does not exceed the maximum width and height roiSize specified by the initialization functions.

Return Values

- **ippStsNoErr**: Indicates no error.
Indicates an error when one of the specified pointers is NULL.

Indicates an error condition when:

- `roiSize` has a field with a zero or negative value
- one of the ROI width or height is more than the corresponding size of ROI passed to the initialization functions
- `srcBitOffset` or `dstBitOffset` is less than zero

Indicates an error condition if `srcStep` or `dstStep` is less than `roiSize.width * <pixelSize>`.

Indicates an error condition if one of the step values for 16-bit integer images is not divisible by 2.

Indicates an error condition if `borderType` has an illegal value.

**See Also**
- Regions of Interest in Intel IPP
- User-defined Border Types
- MorphAdvInit Initializes the specification structure for advanced morphological operations.

**MorphTophat**

*Performs top-hat operation on an image.*

**Syntax**

```c
IppStatus ippiMorphTophat_16u_C1R_L(const Ipp16u* pSrc, IppSizeL srcStep, Ipp16u* pDst, IppSizeL dstStep, Ipp16u* roSize, IppiBorderType borderType, const Ipp16u borderValue[1], const IppiMorphAdvStateL* pMorthSpec, Ipp8u* pBuffer);
IppStatus ippiMorphTophat_16s_C1R_L(const Ipp16s* pSrc, IppSizeL srcStep, Ipp16s* pDst, IppSizeL dstStep, Ipp16s* roSize, IppiBorderType borderType, const Ipp16s borderValue[1], const IppiMorphAdvStateL* pMorthSpec, Ipp8u* pBuffer);
IppStatus ippiMorphTophat_1u_C1R_L(const Ipp8u* pSrc, IppSizeL srcStep, int srcBitOffset, Ipp8u* pDst, IppSizeL dstStep, int dstBitOffset, Ipp8u* roSize, IppiBorderType borderType, const Ipp8u borderValue[1], const IppiMorphAdvStateL* pMorthSpec, Ipp8u* pBuffer);
IppStatus ippiMorphTophat_8u_C1R_L(const Ipp8u* pSrc, IppSizeL srcStep, Ipp8u* pDst, IppSizeL dstStep, Ipp8u* roSize, IppiBorderType borderType, const Ipp8u borderValue[1], const IppiMorphAdvStateL* pMorthSpec, Ipp8u* pBuffer);
IppStatus ippiMorphTophat_8u_C3R_L(const Ipp8u* pSrc, IppSizeL srcStep, Ipp8u* pDst, IppSizeL dstStep, IppSizeL roSize, IppiBorderType borderType, const Ipp8u borderValue[3], const IppiMorphAdvStateL* pMorthSpec, Ipp8u* pBuffer);
IppStatus ippiMorphTophat_8u_C4R_L(const Ipp8u* pSrc, IppSizeL srcStep, Ipp8u* pDst, IppSizeL dstStep, IppSizeL roSize, IppiBorderType borderType, const Ipp8u borderValue[4], const IppiMorphAdvStateL* pMorthSpec, Ipp8u* pBuffer);
IppStatus ippiMorphTophat_32f_C1R_L(const Ipp32f* pSrc, IppSizeL srcStep, Ipp32f* pDst, IppSizeL dstStep, IppSizeL roSize, IppiBorderType borderType, const Ipp32f borderValue[1], const IppiMorphAdvStateL* pMorthSpec, Ipp8u* pBuffer);
IppStatus ippiMorphTophat_32f_C3R_L(const Ipp32f* pSrc, IppSizeL srcStep, Ipp32f* pDst, IppSizeL dstStep, IppSizeL roSize, IppiBorderType borderType, const Ipp32f borderValue[3], const IppiMorphAdvStateL* pMorthSpec, Ipp8u* pBuffer);
```
IppStatus ippiMorphTophat_32f_C4R_L(const Ipp32f* pSrc, IppSizeL srcStep, Ipp32f* pDst, IppSizeL dstStep, IppSizeL roiSize, IppiBorderType borderType, const Ipp32f borderValue[4], const IppiMorphAdvStateL* pMorthSpec, Ipp8u* pBuffer);

Include Files
ippcv_l.h

Domain Dependencies
Headers: ippcore.h, ippvm.h, ipps.h, ippi.h
Libraries: ippcore.lib, ippvm.lib, ipps.lib, ippi.lib

Parameters
pSrc
Pointer to the source image ROI.
srcStep
Distance, in bytes, between the starting points of consecutive lines in the source image.
srcBitOffset
Offset, in bits, from the first byte of the source image row.
pDst
Pointer to the destination image ROI.
dstStep
Distance, in bytes, between the starting points of consecutive lines in the destination image.
dstBitOffset
Offset, in bits, from the first byte of the destination image row.
roiSize
Size of the source and destination image ROI.
borderType
Type of border. Possible values are:
ippBorderDefault The border is set to ippBorderConst with borderValue= MAX_VALUE, where MAX_VALUE=IPP_MAX_8U/16U/16S/32F/1U
ippBorderRepl Border is replicated from the edge pixels.
ippBorderMirror Border pixels are mirrored from the source image boundary pixels.
ippBorderConst Values of all border pixels are set to a constant.
ippBorderFirstStageInMem You can use this border type together with the ippBorderRepl, ippBorderMirror, ippBorderConst, ippBorderDefault types using the | operation. For the first stage, border pixels are obtained from the source image pixels in memory. For the second stage, the function uses the border type specified with the | operation.

Mixed borders are also supported. They can be obtained by the bitwise OR operation OR between any of the ippBorderRepl, ippBorderConst, ippBorderDefault, or ippBorderMirror values and the ippBorderInMemTop, ippBorderInMemBottom, ippBorderInMemLeft, ippBorderInMemRight modifiers.
**borderValue, borderValue[3], borderValue[4]**

Constant value to assign to pixels of the constant border. This parameter is applicable only to the ippBorderConst border type.

**pMorphSpec**

Pointer to the morphology specification structure.

**pBuffer**

Pointer to the external buffer.

**Description**

Before using this function, you need to initialize the morphology specification structure using theippiMorphInit function.

This function performs a top-hat operation on a rectangular ROI area inside a one-, three-, or four-channel 2D image using a specified in the advanced morphology state or specification structure mask and the anchor cell.

The result is equivalent to the opening the source image and following subtraction from the initial source image.

**NOTE**

The function can only process a ROI that does not exceed the maximum width and height roiSize specified by the initialization functions.

**Return Values**

- **ippStsNoErr** Indicates no error.
- **ippStsNullPtrErr** Indicates an error when one of the specified pointers is NULL.
- **ippStsSizeErr** Indicates an error condition when:
  - roiSize has a field with a zero or negative value
  - ROI width is more than ROI width passed to the initialization function
- **ippStsStepErr** Indicates an error condition when srcStep or dstStep is less than roiSize.width * <pixelSize>.
- **ippStsNotEvenStepErr** Indicates an error when one of the step values is not a multiple of an element size.
- **ippStsBadArgErr** Indicates an error when borderType has an illegal value.

**See Also**

- Regions of Interest in Intel IPP
- User-defined Border Types
- MorphInitInitializes the internal state or specification structure for advanced morphological operations.

**MorphTophatBorder**

Performs top-hat operation on an image.
Syntax

Case 1: Operating with morphology state structure

IppStatus ippiMorphTophatBorder_<mod>(const Ipp<datatype>* pSrc, int srcStep,
Ipp<datatype>* pDst, int dstStep, IppiSize roiSize, IppiBorderType borderType,
Ipp<datatype> borderValue, const IppiMorphAdvState* pSpec, Ipp8u* pBuffer);

Supported values for mod:

8u_C1R    32f_C1R

IppStatus ippiMorphTophatBorder_<mod>(const Ipp<datatype>* pSrc, int srcStep,
Ipp<datatype>* pDst, int dstStep, IppiSize roiSize, IppiBorderType borderType, const
Ipp<datatype> borderValue[3], const IppiMorphAdvState* pSpec, Ipp8u* pBuffer);

Supported values for mod:

8u_C3R    32f_C3R

IppStatus ippiMorphTophatBorder_<mod>(const Ipp<datatype>* pSrc, int srcStep,
Ipp<datatype>* pDst, int dstStep, IppiSize roiSize, IppiBorderType borderType, const
Ipp<datatype> borderValue[4], const IppiMorphAdvState* pSpec, Ipp8u* pBuffer);

Supported values for mod:

8u_C4R    32f_C4R

Case 2: Operating with morphology specification structure

IppStatus ippiMorphTophatBorder_<mod>(const Ipp<datatype>* pSrc, int srcStep,
Ipp<datatype>* pDst, int dstStep, IppiSize roiSize, IppiBorderType borderType,
Ipp<datatype> borderValue, const IppiMorphAdvState* pSpec, Ipp8u* pBuffer);

Supported values for mod:

16u_C1R   16s_C1R

IppStatus ippiMorphTophatBorder_1u_C1R(const Ipp8u* pSrc, int srcStep, int
srcBitOffset, Ipp8u* pDst, int dstStep, int dstBitOffset, IppiSize roiSize,
IppiBorderType borderType, Ipp8u borderValue, const IppiMorphAdvState* pSpec, Ipp8u* pBuffer);

Include Files

ippcv.h

Domain Dependencies

Headers: ippcore.h, ippvm.h, ipps.h,ippi.h
Libraries: ippcore.lib, ippvm.lib, ipps.lib,ippi.lib

Parameters

pSrc

Pointer to the source image ROI.

srcStep

Distance, in bytes, between the starting points of consecutive lines
in the source image.

srcBitOffset

Offset, in bits, from the first byte of the source image (for the
1u_C1R flavor).
**pDst**

Pointer to the destination image ROI.

**dstStep**

Distance, in bytes, between the starting points of consecutive lines in the destination image.

**dstBitOffset**

Offset, in bits, from the first byte of the destination image (for the \texttt{1uCLR} flavor).

**roiSize**

Size of the source and destination image ROI.

**borderType**

Type of border. Possible values are:

\begin{itemize}
  \item \texttt{ippBorderRepl} \quad \text{Border is replicated from the edge pixels.}
  \item \texttt{ippBorderInMem} \quad \text{Border is obtained from the source image pixels in memory.}
\end{itemize}

**borderValue, borderValue[3], borderValue[4]**

Constant value to assign to pixels of the constant border. This parameter is applicable only to the \texttt{ippBorderConst} border type.

**pSpec**

Pointer to the specification structure.

**pBuffer**

Pointer to the external buffer.

**Description**

This function operates with ROI (see \textit{Regions of Interest in Intel IPP}).

This function performs a top-hat operation on a rectangular ROI area inside a one-, three-, or four-channel 2D image using a specified in the advanced morphology state or specification structure mask and the anchor cell. The structure must be initialized by \texttt{MorphAdvInit} beforehand.

The result is equivalent to the opening the source image and following subtraction from the initial source image.

**NOTE**

The function can process only images with ROI that does not exceed the maximum width and height \texttt{roiSize} specified by the initialization functions.

**Return Values**

\begin{itemize}
  \item \texttt{ippStsNoErr} \quad \text{Indicates no error. Any other value indicates an error or a warning.}
  \item \texttt{ippStsNullPtrErr} \quad \text{Indicates an error condition if one of the specified pointers is NULL.}
  \item \texttt{ippStsSizeErr} \quad \text{Indicates an error condition if \texttt{roiSize} has a field with a zero or negative value, or if one of ROI width or height is greater than corresponding size of ROI passed to the initialization functions.}
  \item \texttt{ippStsStepErr} \quad \text{Indicates an error condition if \texttt{srcStep} or \texttt{dstStep} is less than \texttt{roiSize.width*<pixelSize>}.}
  \item \texttt{ippStsNotEvenStepErr} \quad \text{Indicates an error condition if one of the step values is not divisible by 4 for floating-point images.}
  \item \texttt{ippStsBorderErr} \quad \text{Indicates an error condition if \texttt{borderType} has an illegal value.}
\end{itemize}

**See Also**

\texttt{MorphAdvInit} \quad \text{Initializes the specification structure for advanced morphological operations.}
**MorphGrayInit**

Initializes morphology state structure for gray-kernel morphology operations.

**Syntax**

IppStatus ippiMorphGrayInit_8u_C1R(IppiMorphGrayState_8u* pState, IppiSize roiSize, const Ipp32s* pMask, IppiSize maskSize, IppiPoint anchor);

IppStatus ippiMorphGrayInit_32f_C1R(IppiMorphGrayState_32f* pState, IppiSize roiSize, const Ipp32f* pMask, IppiSize maskSize, IppiPoint anchor);

**Include Files**

ippcv.h

**Domain Dependencies**

Headers: ippcore.h, ippvm.h, ipps.h,ippi.h

Libraries: ippcore.lib, ippvm.lib, ipps.lib,ippi.lib

**Parameters**

- **pState**: Pointer to the gray-kernel morphology state structure.
- **roiSize**: Maximal size of the image ROI in pixels, that can be processed using the allocated structure.
- **pMask**: Pointer to the mask.
- **maskSize**: Size of the mask in pixels.
- **anchor**: Coordinates of the anchor cell.

**Description**

This function operates with ROI (see Regions of Interest in Intel IPP).

This function initializes the gray-kernel morphology state structure pState in the external buffer. Its size should be computed by the function MorphGrayGetSize. It is used by the functions GrayDilateBorder and GrayErodeBorder that perform gray-kernel dilation and erosion of the source image pixels corresponding to the specified pMask of size maskSize. The anchor cell anchor is positioned in the arbitrary point in the mask and is used for positioning the mask.

**WARNING**

The structure can be used to process images with ROI that does not exceed the specified maximum width and height roiSize.

**Return Values**

- **ippStsNoErr**: Indicates no error. Any other value indicates an error or a warning.
- **ippStsNullPtrErr**: Indicates an error when one of the specified pointers is NULL.
- **ippStsSizeErr**: Indicates an error condition if maskSize or if roiSize has a field with a zero or negative value.
MorphGrayGetSize

Computes the size of the gray-kernel morphology state structure.

Syntax

IppStatus ippiMorphGrayGetSize_8u_C1R(IppSize roiSize, const Ipp32s* pMask, IppiSize maskSize, int* pSize);
IppStatus ippiMorphGrayGetSize_32f_C1R(IppSize roiSize, const Ipp32f* pMask, IppiSize maskSize, int* pSize);

Include Files

ippcv.h

Domain Dependencies

Headers: ippcore.h, ippvm.h, ipps.h, ippi.h
Libraries: ippcore.lib, ippvm.lib, ipps.lib,ippi.lib

Parameters

roiSize  
Maximal size of the image ROI in pixels, that can be processed using the allocated structure.

pMask  
Pointer to the mask.

maskSize  
Size of the mask in pixels.

pSize  
Pointer to the size of the advanced morphology state structure.

Description

This function operates with ROI (see Regions of Interest in Intel IPP).
This function computes the size of the morphology state structure pState for gray-kernel dilation and erosion. This function should be run prior to the function MorphGrayInit.

Return Values

ippStsNoErr  
Indicates no error. Any other value indicates an error or a warning.

ippStsNullPtrErr  
Indicates an error when one of the specified pointers is NULL.

ippStsSizeErr  
Indicates an error condition if maskSize or if roiSize has a field with a zero or negative value.

MorphReconstructGetBufferSize

Computes the size of the buffer for morphological reconstruction operation.

IppStatus ippiMorphReconstructGetBufferSize(IppSize roiSize, IppDataType dataType, int numChannels, int* pBufSize);
Include Files
ippcv.h

Domain Dependencies
Headers: ippcore.h, ipps.h,ippi.h
Libraries: ippcore.lib, ipps.lib,ippi.lib

Parameters

roiSize     Maximal size of the image ROI in pixels, that can be processed using the buffer.
dataType    Data type of the image.
numChannels Number of channels in the image.
pBufSize    Pointer to the size of the work buffer (in bytes), returned by theippiMorphReconstructGetBufferSize function.

Description
This function operates with ROI (see Regions of Interest in Intel IPP).
This function computes the size of the buffer for the morphological reconstruction of the source image. This buffer can be used by the functions MorphReconstructDilate and MorphReconstructErode.

Return Values

ippStsNoErr Indicates no error. Any other value indicates an error or a warning.
ippStsNullPtrErr Indicates an error condition if pSize is NULL.
ippStsSizeErr Indicates an error condition if roiSize has a field with a zero or negative value.

MorphReconstructDilate
Reconstructs an image by dilation.

Syntax
IppStatus ippiMorphReconstructDilate_<mod>(const Ipp<datatype>* pSrc, int srcStep,
Ipp<datatype>* pSrcDst, int srcDstStep, IppiSize roiSize, Ipp8u* pBuffer, IppiNorm
norm);
Supported values for mod:
  8u_C1IR 16u_C1IR 64f_C1IR

IppStatus ippiMorphReconstructDilate_32f_C1IR(const Ipp32f* pSrc, int srcStep, Ipp32f*
pSrcDst, int srcDstStep, IppiSize roiSize, Ipp32f* pBuffer, IppiNorm norm);

Include Files
ippcv.h
Domain Dependencies

Headers: ippcore.h, ippvm.h, ipps.h,ippi.h
Libraries: ippcore.lib, ippvm.lib, ipps.lib,ippi.lib

Parameters

pSrc  
Pointer to the source image ROI.

srcStep  
Distance in bytes between starts of consecutive lines in the source image.

pSrcDst  
Pointer to the decreased and reconstructed image ROI.

srcDstStep  
Distance in bytes between starts of consecutive lines in the decreased and reconstructed image.

roiSize  
Size of the source and destination image ROI.

norm  
Type of norm to form the mask for dilation; the following values are possible:

ippiNormInf  
Infinity norm (8-connectivity, 3x3 rectangular mask).

ippiNormL1  
L1 norm (4-connectivity, 3x3 cross mask).

pBuffer  
Pointer to the buffer.

Description

This function operates with ROI (see Regions of Interest in Intel IPP).

This function performs morphological reconstruction of the decreased source image by dilation [Vincent93]. The operation is performed in the working buffer whose size should be computed using the function MorphReconstructGetBufferSize beforehand.

This operation enables detection of the regional maximums that can be used as markers for successive watershed segmentation.
Example below shows how the morphological reconstruction can be used to build markers of objects with different brightness. Some value (cap size) is subtracted from the initial image and then the subtracted image is reconstructed to the initial one. Thresholding and opening complete the building of markers. The figure below shows the results of these operations.

**Building Markers for Segmentation by the Morphological Reconstruction**

<table>
<thead>
<tr>
<th>a) Initial Image</th>
<th>b) Initial Image Decreased by the Cap Value</th>
<th>c) Reconstructed Image</th>
</tr>
</thead>
<tbody>
<tr>
<td>d) Cap Image</td>
<td>e) Thresholded Caps</td>
<td>f) Markers</td>
</tr>
</tbody>
</table>

**Return Values**

- ippStsNoErr: Indicates no error. Any other value indicates an error or a warning.
- ippStsNullPtrErr: Indicates an error condition if one of the specified pointers is NULL.
- ippStsSizeErr: Indicates an error condition if roiSize has a field with a zero or negative value.
- ippStsStepErr: Indicates an error condition if srcStep or srcDstStep is less than roiSize.width * pixelSize.
- ippStsNotEvenStepErr: Indicates an error condition if one of the step values is not divisible by 4 for floating-point images.
- ippStsBadArgErr: Indicates an error condition if norm has an illegal value.

**Example**

To better understand usage of this function, refer to the following example in the examples archive available for download from https://software.intel.com/en-us/ipp-manual-examples:

MorphReconstructDilate.c
MorphReconstructErode

Reconstructs an image by erosion.

Syntax

IppStatus ippiMorphReconstructErode_<mod>(const Ipp<datatype>* pSrc, int srcStep, Ipp<datatype>* pSrcDst, int srcDstStep, IppiSize roiSize, Ipp8u* pBuf, IppiNorm norm);

Supported values for mod:

8u_C1IR 16u_C1IR 64f_C1IR

IppStatus ippiMorphReconstructErode_32f_C1IR(const Ipp32f* pSrc, int srcStep, Ipp32f* pSrcDst, int srcDstStep, IppiSize roiSize, Ipp32f* pBuf, IppiNorm norm);

Include Files

ippcv.h.

Parameters

pSrc

Pointer to the source image ROI.

srcStep

Distance in bytes between starts of consecutive lines in the source image.

pSrcDst

Pointer to the decreased and reconstructed image ROI.

srcDstStep

Distance in bytes between starts of consecutive lines in the decreased and reconstructed image.

roiSize

Size of the source and destination image ROI.

norm

Type of norm to form the mask for dilation; the following values are possible:

ippiNormInf

Infinity norm (8-connectivity, 3x3 rectangular mask).

ippiNormL1

L1 norm (4-connectivity, 3x3 cross mask).

pBuf

Pointer to the buffer.

Description

This function operates with ROI (see Regions of Interest in Intel IPP).

This function performs morphological reconstruction of the increased source image by erosion [Vincent93]. The operation is performed in the working buffer whose size should be computed using the function MorphReconstructGetBufferSize beforehand.

This operation enables detection of the regional minimums that can be used as markers for successive watershed segmentation.

Return Values

ippStsNoErr

Indicates no error. Any other value indicates an error or a warning.

ippiStsNullPtrErr

Indicates an error condition if one of the specified pointers is NULL.
MorphSetMode

Sets the mask processing mode for advanced morphological operations.

Syntax
IppStatus ippiMorphSetMode_L(int mode, IppiMorphAdvStateL* pMorphSpec);

Include Files
ippcv_l.h

Domain Dependencies
Headers: ippcore.h, ippvm.h, ipps.h,ippi.h
Libraries: ippcore.lib, ippvm.lib, ipps.lib,ippi.lib

Parameters
mode
Mask processing mode; supported values:
IPP_MORPH_DEFAULT Invert the mask
IPP_MORPH_MASK_NO_FLIP Do not invert the mask; use the same mask for the first and second stage.

pMorphSpec
Pointer to the specification structure for advanced morphological operations.

Description
This function sets the mask processing mode for the second stage of an advanced morphological operation. Before using this function, initialize the specification structure using the ippiMorphInit_L function.

Return Values
ippStsNoErr Indicates no error.
ippStsNotSupportedModeErr Indicates an error when mode has an invalid value.
ippStsNullPtrErr Indicates an error when pMorphSpec is NULL.
Filtering Functions

This section describes the Intel® IPP image processing functions that perform linear and non-linear filtering operations on an image.

You can use filtering in image processing operations like edge detection, blurring, noise removal, and feature detection.

Most filtering functions operate with regions of interest (ROI). The size of the source image ROI is equal to the destination image ROI size $\text{dstRoiSize}$. Most functions use different source and destination image buffers. These functions are not in-place.

See Also
Regions of Interest in Intel IPP

Borders in Neighborhood Operations

Filtering functions described in this section perform neighborhood operations. They operate on the assumption that for each pixel to be processed, all neighborhood pixels required for the operation are also available.

The neighborhood for each given pixel is defined by the filter kernel (or mask) size and anchor cell position. For more information about anchors and how to define the anchor cell position refer to Neighborhood Operations.

As the following figure illustrates, if the input pixel is near the horizontal or vertical edge of the image, the overlaid kernel may refer to neighborhood pixels that do not exist within the source image and are located outside the image area.

![Diagram of source image with area extension](image.png)

The set of all boundary source image pixels that require such non-existent pixels to complete the neighborhood operation for the given kernel and anchor is shaded yellow, while the collection of all scanned external pixels (called border pixels) is shaded gray.

If you want to apply some filtering operation to the whole source image, you must figure out what additional border pixels are required for the operation, and define these non-existent pixels. To do this, you can use the Intel IPP functions `ippiCopyConstBorder`, `ippiCopyReplicateBorder`, or `ippiCopyWrapBorder`, which fill the border of the extended image with the pixel values that you define, or you can apply your own extension method.

**Note**
If the required border pixels are not defined prior to the filtering function call, you may get memory violation errors.
If you want to apply the filtering operation to the part of the source image, or ROI, then the necessity of extending the image area with border pixels depends on the ROI size and position within the image. The figure below shows that if ROI does not cover yellow (internal boundary) pixels, then no external pixels are scanned, and border extension is not required.

If boundary pixels are part of ROI, you still need to extend some area of the source image. To provide valid results of filtering operations, the application must check the following:

- ROI parameters passed to the filtering function have such values
- All required neighborhood pixels actually exist in the image and define the missing pixels when necessary.

**See Also**
- Neighborhood Operations
- Regions of Interest in Intel IPP
- User-defined Border Types
  - CopyConstBorder
  - CopyReplicateBorder
  - CopyWrapBorder

**User-defined Border Types**

Some of the Intel® IPP image processing functions operate on user-defined border types. It means that the values of border pixels are assigned in accordance with the `borderType (or border)` and `borderValue` parameters.

Intel® IPP supports the following border types:

- Constant border
- Replicated border
- Mirrored border
- Mirrored border with replication
- Border in memory
- Mixed borders

**Constant Border**

This type of border corresponds to the `ippBorderConst` value in the `IppiBorderType` enumerator. When using a constant border, values for all border pixels are set to the constant value that you specify in the `borderValue` parameter. In the figure below, this constant value is marked as V. Squares marked in red correspond to pixels copied from the source image ROI.
Replicated Border
This type of border corresponds to the `ippBorderRepl` value in the `IppiBorderType` enumerator. When using a replicated border, values for border pixels are obtained from the source image boundary pixels, as shown in the figure below. Squares marked in red correspond to pixels copied from the source image ROI. Squares with green values correspond to border pixels, which are replicated from the boundary pixels of the source image.

Mirrored Border
This type of border corresponds to the `ippBorderMirror` value in the `IppiBorderType` enumerator. When using a mirrored border, values for border pixels are obtained from the source image boundary pixels, as shown in the figure below. Squares marked in red correspond to pixels copied from the source image ROI. Squares with green values correspond to border pixels, which are mirrored from the source image pixels.
Mirrored Border with Replication
This type of border corresponds to the ippBorderMirrorR value in the IppiBorderType enumerator. When using a mirrored border with replication, values for border pixels are obtained from the source image boundary pixels, as shown in the figure below. Squares marked in red correspond to pixels copied from the source image ROI. Squares with green values correspond to border pixels, which are mirrored from the source image pixels. The difference of this border type from the mirrored border is that the anchor cell value is replicated to the border pixels.

Border in Memory
This type of border corresponds to the ippBorderInMem value and its flags combinations in the IppiBorderType enumerator. Use this border type if the ROI does not cover internal border pixels of the source image. In this case, values for border pixels are obtained from the source image pixels in memory. In the figure below, squares marked in red correspond to pixels copied from the source image ROI. Squares with black values correspond to source image pixels in memory.
Several Intel IPP filters operate in two or more stages. For example, the `ippiMorphOpenBorder` function performs filtering by applying the Erode and Dilate filters sequentially. You should note the following when setting borders for multistage filters:

- If you set the `ippBorderInMem` value or its flags combinations, the function tries to access pixels outside of image borders to get border pixels for each filtering stage. For example, the `ippiMorphOpenBorder` function uses two stages and with 5x5 mask will access \( \text{floor}(5/2)\times2=4 \) pixels in each direction across the current ROI.
- If you set `ippBorderFirstStageInMem`, the function tries to access \( \text{floor}(5/2)=2 \) pixels outside of the image borders to get pixels for the first stage of filtering. The second filter will use one of the following border types to reconstruct image borders: `ippBorderRepl`, `ippBorderConst`, `ippBorderMirror`, or `ippBorderMirrorR`. To specify the border type for the second and next stages, use the bitwise OR operation between one of the listed above border types and `ippBorderFirstStageInMem`.

**Mixed Borders**

You can use mixed borders by using a bitwise OR operation between one of the `ippBorderRepl`, `ippBorderConst`, `ippBorderMirror`, or `ippBorderMirrorR` types and any of the following border types: `ippBorderInMemTop`, `ippBorderInMemBottom`, `ippBorderInMemLeft`, `ippBorderInMemRight`, or `ippBorderFirstStageInMem`. In this case, values for border pixels are obtained from the source image pixels in memory in the direction specified by the flag.

The figure below demonstrates the use of the `ippBorderConst` with the `ippBorderInMemTop` and `ippBorderInMemRight` borders. Squares marked in red correspond to pixels copied from the source image, that is the source image ROI. As you can see from the figure, top and right border pixels are obtained from the source image pixels in memory, while the rest of the border pixels are set to the constant value \( V \).
NOTE
The combination of ippBorderInMem and its flags always has priority over any other border flags or types. For example, if you specify ippBorderFirstStageInMem|ippBorderRepl|ippBorderInMemLeft, the left border will use InMem mode for each stage and other borders will use InMem for the first stage and replication for remaining stages.

The figure below demonstrates the use of the ippBorderConst with the ippBorderInMemTop, ippBorderInMemRight, and ippBorderFirstStageInMem flags for two-stage filtering with one pixel border for both stages.

- **First stage:** squares marked in red correspond to pixels copied from the source image, which is the source image ROI, and squares marked in blue correspond to ROI assigned to the first stage filter. As you can see from the figure, the first stage enlarges ROI for top and right sides to consume more memory and provide valid pixels for the second stage memory border.
- **Second stage:** red squares and blue pixels correspond to resulting pixels from the first stage filter. Blue pixels lie outside of the ROI providing border values for the second stage in top and right directions. Left and bottom border pixels use the constant value \( V \) in accordance with the border flags combination.

### First stage:

<table>
<thead>
<tr>
<th></th>
<th>42</th>
<th>43</th>
<th>44</th>
<th>45</th>
<th>46</th>
<th>47</th>
</tr>
</thead>
<tbody>
<tr>
<td>54</td>
<td>53</td>
<td>52</td>
<td>51</td>
<td>50</td>
<td>49</td>
<td>48</td>
</tr>
<tr>
<td>61</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>58</td>
<td>59</td>
</tr>
<tr>
<td>61</td>
<td>5</td>
<td>6</td>
<td>7</td>
<td>8</td>
<td>61</td>
<td>60</td>
</tr>
<tr>
<td>66</td>
<td>9</td>
<td>10</td>
<td>11</td>
<td>12</td>
<td>66</td>
<td>67</td>
</tr>
<tr>
<td>69</td>
<td>13</td>
<td>14</td>
<td>15</td>
<td>16</td>
<td>69</td>
<td>68</td>
</tr>
<tr>
<td>74</td>
<td>18</td>
<td>19</td>
<td>20</td>
<td>21</td>
<td>74</td>
<td>75</td>
</tr>
<tr>
<td>54</td>
<td>53</td>
<td>52</td>
<td>51</td>
<td>50</td>
<td>49</td>
<td>74</td>
</tr>
</tbody>
</table>

### Second stage:

<table>
<thead>
<tr>
<th>( V )</th>
<th>37</th>
<th>25</th>
<th>70</th>
<th>21</th>
<th>22</th>
</tr>
</thead>
<tbody>
<tr>
<td>( V )</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>16</td>
</tr>
<tr>
<td>( V )</td>
<td>5</td>
<td>6</td>
<td>7</td>
<td>8</td>
<td>73</td>
</tr>
<tr>
<td>( V )</td>
<td>9</td>
<td>10</td>
<td>11</td>
<td>12</td>
<td>58</td>
</tr>
<tr>
<td>( V )</td>
<td>13</td>
<td>14</td>
<td>15</td>
<td>16</td>
<td>36</td>
</tr>
<tr>
<td>( V )</td>
<td>18</td>
<td>19</td>
<td>20</td>
<td>21</td>
<td>14</td>
</tr>
</tbody>
</table>

\( V \) pixels in memory from the first stage.

\( V \) constant value.
Filters with Borders

This section describes Intel® IPP filtering functions that automatically create a required border and define appropriate pixel values.

**FilterBilateral**

*Performs bilateral filtering of an image.*

**Syntax**

**Case 1: Operation on pixel-order data**

IppStatus ippiFilterBilateral_<mod>(const Ipp<srcdatatype>* pSrc, int srcStep,
Ipp<dstdatatype>* pDst, int dstStep, IppiSize dstRoiSize, IppiBorderType borderType,
const Ipp<datatype> pBorderValue[1], const IppFilterBilateralSpec* pSpec, Ipp8u* pBuffer);

Supported values for $mod$:

- 8u_C1R
- 32f_C1R
- 64f_C1R

IppStatus ippiFilterBilateral_<mod>(const Ipp<srcdatatype>* pSrc, int srcStep,
Ipp<dstdatatype>* pDst, int dstStep, IppiSize dstRoiSize, IppiBorderType borderType,
const Ipp<datatype> pBorderValue[3], const IppFilterBilateralSpec* pSpec, Ipp8u* pBuffer);

Supported values for $mod$:

- 8u_C3R
- 32f_C3R
- 64f_C3R

**Case 2: Operation on planar data**

IppStatus ippiFilterBilateral_<mod>(const Ipp<srcdatatype>* pSrc[3], int srcStep[3],
Ipp<dstdatatype>* pDst[3], int dstStep[3], IppiSize dstRoiSize, IppiBorderType borderType,
const Ipp<datatype> pBorderValue[3], const IppFilterBilateralSpec* pSpec, Ipp8u* pBuffer);

Supported values for $mod$:

- 8u_P3R
- 32f_P3R
- 64f_P3R
Case 3: Operation on pixel-order data with platform-aware functions

IppStatus ippiFilterBilateral_<mod>(const Ipp<srcdatatype>* pSrc, IppSizeL srcStep, Ipp<datatype>* pDst, IppSizeL dstStep, IppSizeL dstRoiSize, IppiBorderType borderType, const Ipp<datatype> pBorderValue[1], const IppiFilterBilateralSpec* pSpec, Ipp8u* pBuffer);

Supported values for mod:

8u_C1R_L 32f_C1R_L 64f_C1R_L

IppStatus ippiFilterBilateral_<mod>(const Ipp<srcdatatype>* pSrc, IppSizeL srcStep, Ipp<datatype>* pDst, IppSizeL dstStep, IppSizeL dstRoiSize, IppiBorderType borderType, const Ipp<datatype> pBorderValue[3], const IppiFilterBilateralSpec* pSpec, Ipp8u* pBuffer);

Supported values for mod:

8u_C3R_L 32f_C3R_L 64f_C3R_L

Case 4: Operation on planar data with platform-aware functions

IppStatus ippiFilterBilateral_<mod>(const Ipp<srcdatatype>* pSrc[3], IppSizeL srcStep[3], Ipp<datatype>* pDst[3], IppSizeL dstStep[3], IppSizeL dstRoiSize, IppiBorderType borderType, const Ipp<datatype> pBorderValue[3], const IppiFilterBilateralSpec* pSpec, Ipp8u* pBuffer);

Supported values for mod:

8u_P3R_L 32f_P3R_L 64f_P3R_L

Case 5: Operation on pixel-order data with Threading Layer (TL) functions based on the Platform Aware API

IppStatus ippiFilterBilateral_<mod>(const Ipp<srcdatatype>* pSrc, IppSizeL srcStep, Ipp<datatype>* pDst, IppSizeL dstStep, IppSizeL dstRoiSize, IppiBorderType borderType, const Ipp<datatype> pBorderValue[1], const IppiFilterBilateralSpec_LT* pSpec, Ipp8u* pBuffer);

Supported values for mod:

8u_C1R_LT 32f_C1R_LT 64f_C1R_LT

IppStatus ippiFilterBilateral_<mod>(const Ipp<srcdatatype>* pSrc, IppSizeL srcStep, Ipp<datatype>* pDst, IppSizeL dstStep, IppSizeL dstRoiSize, IppiBorderType borderType, const Ipp<datatype> pBorderValue[3], const IppiFilterBilateralSpec_LT* pSpec, Ipp8u* pBuffer);

Supported values for mod:

8u_C3R_LT 32f_C3R_LT 64f_C3R_LT
Case 6: Operation on planar data with Threading Layer (TL) functions based on the Platform Aware API

IppStatus ippiFilterBilateral_<mod>(const Ipp<srcdatatype>* pSrc[3], IppSizeL srcStep[3], Ipp<dstdatatype>* pDst[3], IppSizeL dstStep[3], IppiSizeL dstRoiSize, IppiBorderType borderType, const Ipp<datatype>* pBorderValue[3], const IppiFilterBilateralSpec_LT* pSpec, Ipp8u* pBuffer);

Supported values for mod:
- 8u_P3R_LT
- 32f_P3R_LT
- 64f_P3R_LT

Case 7: Operation on pixel-order data with Threading Layer (TL) functions based on the Classic API

IppStatus ippiFilterBilateral_<mod>(const Ipp<srcdatatype>* pSrc, int srcStep, Ipp<dstdatatype>* pDst, int dstStep, IppiSize dstRoiSize, IppiBorderType borderType, const Ipp<datatype>* pBorderValue[1], const IppiFilterBilateralSpec_T* pSpec, Ipp8u* pBuffer);

Supported values for mod:
- 8u_C1R_T
- 32f_C1R_T
- 64f_C1R_T

IppStatus ippiFilterBilateral_<mod>(const Ipp<srcdatatype>* pSrc, int srcStep, Ipp<dstdatatype>* pDst, int dstStep, IppiSize dstRoiSize, IppiBorderType borderType, const Ipp<datatype>* pBorderValue[3], const IppiFilterBilateralSpec_T* pSpec, Ipp8u* pBuffer);

Supported values for mod:
- 8u_C3R_T
- 32f_C3R_T
- 64f_C3R_T

Case 8: Operation on planar data with Threading Layer (TL) functions based on the Classic API

IppStatus ippiFilterBilateral_<mod>(const Ipp<srcdatatype>* pSrc[3], int srcStep[3], Ipp<dstdatatype>* pDst[3], int dstStep[3], IppiSize dstRoiSize, IppiBorderType borderType, const Ipp<datatype>* pBorderValue[3], const IppiFilterBilateralSpec_T* pSpec, Ipp8u* pBuffer);

Supported values for mod:
- 8u_P3R_T
- 32f_P3R_T
- 64f_P3R_T

Include Files

ippi.h
ippi_l.h
ippi_tl.h

Domain Dependencies

Flavors declared in ippi.h:

Headers: ippcore.h, ippvm.h, ipps.h
Libraries: ippcore.lib, ippvm.lib, ipps.lib
**Parameters**

- **pSrc**: Pointer to the source image ROI.
- **srcStep**: Distance, in bytes, between the starting points of consecutive lines in the source image.
- **pDst**: Pointer to the destination image ROI.
- **dstStep**: Distance, in bytes, between the starting points of consecutive lines in the destination image.
- **dstRoiSize**: Size of the source and destination ROI in pixels.
- **borderType**: Type of border. Possible values are:
  - ippBorderConst: Values of all border pixels are set to constant.
  - ippBorderRepl: Border is replicated from the edge pixels.
  - ippBorderInMem: Border is obtained from the source image pixels in memory.
  
  Mixed borders are also supported. They can be obtained by the bitwise operation OR between ippBorderRepl and ippBorderInMemTop, ippBorderInMemBottom, ippBorderInMemLeft, ippBorderInMemRight.
- **pBorderValue**: Constant value to assign to pixels of the constant border. This parameter is applicable only to the ippBorderConst border type.
- **pSpec**: Pointer to the bilateral context structure.
- **pBuffer**: Pointer to the work buffer.

**Description**

This function applies the bilateral filter with a square kernel to the source image. The linear dimension of the kernel is defined in the initialization function `FilterBilateralInit`. The bilateral context structure contains the parameters of filtering.

Before using the `ippiFilterBilateral` function, compute the size of the bilateral context structure and the external buffer using the `FilterBilateralGetBufferSize` function and initialize the structure using the `FilterBilateralInit` function.

**Return Values**

- **ippStsNoErr**: Indicates no error.
- **ippStsNullPtrErr**: Indicates an error when `pSrc`, `pDst`, `pSpec`, or `pBuffer` is NULL.
- **ippStsSizeErr**: Indicates an error when `dstRoiSize` has a field with a zero or negative value.
- **ippStsContextMatchErr**: Indicates an error when the `pSpec` structure does not match the function.
- **ippStsNotEvenStepErr**: Indicates an error when one of the step values is not divisible by 4 for floating-point images.
- **ippStsBorderErr**: Indicates an error when `borderType` has an illegal value.
See Also
Regions of Interest in Intel IPP
User-defined Border Types
FilterBilateralGetBufferSize Computes the size of the bilateral context structure and the size of the
work buffer for bilateral filtering with user-defined borders.
FilterBilateralInit Initializes the bilateral context structure.

FilterBilateralGetBufferSize
Computes the size of the bilateral context structure
and the size of the work buffer for bilateral filtering
with user-defined borders.

Syntax
IppStatus ippiFilterBilateralGetBufferSize(IppiFilterBilateralType filter, IppiSize dstRoiSize, int kernelWidthHeight, IppDataType dataType, int numChannels, IppiDistanceMethodType distMethodType, int* pSpecSize, int* pBufferSize);

Platform-aware function
IppStatus ippiFilterBilateralGetBufferSize_L(IppiFilterBilateralType filter, IppiSizeL dstRoiSize, IppSizeL kernelWidthHeight, IppDataType dataType, int numChannels, IppiDistanceMethodType distMethodType, IppSizeL* pSpecSize, IppSizeL* pBufferSize);

Threading Layer (TL) function based on the Platform Aware API
IppStatus ippiFilterBilateralGetBufferSize_LT(IppiFilterBilateralType filter, IppiSizeL dstRoiSize, IppSizeL kernelWidthHeight, IppDataType dataType, int numChannels, IppiDistanceMethodType distMethodType, IppSizeL* pSpecSize, IppSizeL* pBufferSize);

Threading Layer (TL) function based on the Classic API
IppStatus ippiFilterBilateralGetBufferSize_T(IppiFilterBilateralType filter, IppiSize dstRoiSize, int kernelWidthHeight, IppDataType dataType, int numChannels, IppiDistanceMethodType distMethodType, int* pSpecSize, int* pBufferSize);

Include Files
ippi.h
ippi_l.h
ippi_tl.h

Domain Dependencies
Headers: ippcore.h, ippvm.h, ipps.h
Libraries: ippcore.lib, ippvm.lib, ipps.lib

Parameters
filter
Type of the bilateral filter. Possible value is
ippiFilterBilateralGauss - Gaussian bilateral filter.
dstRoiSize
Size of the destination ROI in pixels.
kernelWidthHeight
Linear dimension of the square kernel. The value 1 corresponds to
the distance between two adjacent pixels.
dataType
Data type of the source and destination images. Possible values are
ipp8u, ipp32f, and ipp64f.
**numChannels**

Number of channels in the images.

**distMethodType**

Method of defining the differences in intensity between pixels. Depending on the number of channels in the image, possible values are:

<table>
<thead>
<tr>
<th>numChannels value</th>
<th>Possible distMethodType values</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>ippDistNormL1</td>
</tr>
<tr>
<td>3</td>
<td>ippDistNormL1, ippDistNormL2</td>
</tr>
</tbody>
</table>

**pSpecSize**

Pointer to the computed size of the specification structure.

**pBufferSize**

Pointer to the computed size of the external buffer.

**Description**

This function computes the size of the bilateral context structure and external work buffer for the `FilterBilateral` function. The results are stored in `pSpecSize` and `pBufferSize`.

Use the computed `pBufferSize` and `pSpecSize` values to allocate the memory using the `ippMalloc` or `ippsMalloc` functions. The allocated memory can be freed only by the `ippFree` or `ippsFree` functions, respectively. For more information about the memory allocation functions, refer to the "Support Functions" section of the *Intel IPP Developer Reference, vol. 1*.

**Return Values**

- **ippStsNoErr** Indicates no error. Any other value indicates an error.
- **ippStsNullPtrErr** Indicates an error when `pSpecSize` or `pBufferSize` is NULL.
- **ippStsSizeErr** Indicates an error when `dstRoiSize` has a field with a zero or negative value.
- **ippStsMaskSizeErr** Indicates an error when `kernelWidthHeight` is less than, or equal to zero.
- **ippStsNotSupportedModeErr** Indicates an error when the `filter` or `distMethodType` value is not supported.
- **ippStsDataTypeErr** Indicates an error when `dataType` has an illegal value.
- **ippStsNumChannelsErr** Indicates an error when `numChannels` has an illegal value.

**See Also**

- `FilterBilateral` Performs bilateral filtering of an image.
- `FilterBilateralInit` Initializes the bilateral context structure.

**FilterBilateralInit**

*Initializes the bilateral context structure.*

**Syntax**

```c
IppStatus ippiFilterBilateralInit(IppiFilterBilateralType filter, IppiSize dstRoiSize, int kernelWidthHeight, IppDataType dataType, int numChannels, IppiDistanceMethodType distMethod, Ipp64f valSquareSigma, Ipp64f posSquareSigma, IppiFilterBilateralSpec* pSpec);
```
Platform-aware function

IppStatus ippiFilterBilateralInit_L(IppFilterBilateralType filter, IppiSizeL dstRoiSize, IppiSizeL kernelWidthHeight, IppDataType dataType, int numChannels, IppiDistanceMethodType distMethod, Ipp64f valSquareSigma, Ipp64f posSquareSigma, IppiFilterBilateralSpec* pSpec);

Threading Layer (TL) function based on the Platform Aware API

IppStatus ippiFilterBilateralInit_LT(IppFilterBilateralType filter, IppiSizeL dstRoiSize, IppSizeL kernelWidthHeight, IppDataType dataType, int numChannels, IppiDistanceMethodType distMethod, Ipp64f valSquareSigma, Ipp64f posSquareSigma, IppiFilterBilateralSpec_LT* pSpec);

Threading Layer (TL) function based on the Classic API

IppStatus ippiFilterBilateralInit_T(IppFilterBilateralType filter, IppiSize dstRoiSize, int kernelWidthHeight, IppDataType dataType, int numChannels, IppiDistanceMethodType distMethod, Ipp64f valSquareSigma, Ipp64f posSquareSigma, IppiFilterBilateralSpec_T* pSpec);

Include Files

ippi.h
ippi_l.h
ippi_tl.h

Domain Dependencies

Flavors declared in ippi.h:

Headers: ippcore.h, ippvm.h, ipps.h
Libraries: ippcore.lib, ippvm.lib, ipps.lib

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>filter</td>
<td>Type of the bilateral filter. Possible value is ippiFilterBilateralGauss − Gaussian bilateral filter.</td>
</tr>
<tr>
<td>dstRoiSize</td>
<td>Size of the destination ROI in pixels.</td>
</tr>
<tr>
<td>kernelWidthHeight</td>
<td>Linear dimension of the square kernel. The value 1 corresponds to the distance between two adjacent pixels.</td>
</tr>
<tr>
<td>dataType</td>
<td>Data type of the source and destination images. Possible values are ipp8u, ipp32f, and ipp64f.</td>
</tr>
<tr>
<td>numChannels</td>
<td>Number of channels in the images.</td>
</tr>
<tr>
<td>distMethod</td>
<td>Method of defining the differences in intensity between pixels. Depending on the number of channels in the image, possible value are:</td>
</tr>
<tr>
<td></td>
<td><strong>numChannels</strong> value</td>
</tr>
<tr>
<td>1</td>
<td>ippDistNormL1</td>
</tr>
<tr>
<td>3</td>
<td>ippDistNormL3</td>
</tr>
<tr>
<td>valSquareSigma</td>
<td>Square of the range parameter, which controls smoothing based on the differences in intensity between pixels.</td>
</tr>
</tbody>
</table>
posSquareSigma

Square of the spatial parameter, which controls smoothing based on the geometric distance between pixels.

pSpec

Pointer to the bilateral context structure.

Description

This function initializes the bilateral context structure pSpecSize for bilateral filtering. Before using this function, compute the size of the context structure using the FilterBilateralGetBufferSize function.

The kernelWidthHeight parameter specifies the linear dimension of the square filter kernel. The value 1 corresponds to the distance between the centers of two adjacent pixels.

Coefficients of the bilateral filter kernel depend on their positions in the kernel and on the intensity value of the source image pixels lying in the kernel.

The value of the output pixel \( d \) is computed by the following formula:

\[
\begin{align*}
    d &= \frac{\sum_{i,j} w_{1,ij} * w_{2,ij} * v_{ij}}{\sum_{i,j} w_{1,ij} * w_{2,ij}}
\end{align*}
\]

For all indices \( i \) and \( j \) that fit within the square kernel

where

- \( v_{ij} \) is the value (or channel values) of a pixel in the kernel with coordinates \( i \) and \( j \)
- \( w_{1,ij} = \text{Fun}(\text{valSquareSigma}, \text{Intensity Distance}(v_{ij}, v_{00})) \)
- \( w_{2,ij} = \text{Fun}(\text{posSquareSigma}, \text{Geometric Distance}(v_{ij}, v_{00})) = \sqrt{i^2 + j^2} \)

\( \text{Fun}(S, I) = \exp(-I^2 S) \)

where

- \( S \) is valSquareSigma or posSquareSigma
- \( I \) is the difference between pixel values or position

Return Values

- ippStsNoErr
  Indicates no error. Any other value indicates an error.
- ippStsNullPtrErr
  Indicates an error when pSpec is NULL.
- ippStsSizeErr
  Indicates an error when dstRoiSize has a field with a zero or negative value.
- ippStsMaskSizeErr
  Indicates an error when kernelWidthHeight is less than, or equal to zero.
- ippStsNotSupportedModeErr
  Indicates an error when the filter or distMethod value is not supported.
Indicates an error when `valSquareSigma` or `posSquareSigma` is less than, or equal to zero.

Indicates an error when `numChannels` has an illegal value.

See Also
Structures and Enumerators for Platform-Aware Functions
FilterBilateralGetBufferSize Computes the size of the bilateral context structure and the size of the work buffer for bilateral filtering with user-defined borders.
FilterBilateral Performs bilateral filtering of an image.

FilterBilateralBorderGetBufferSize

*Computes the size of the bilateral context structure and the size of the work buffer for bilateral filtering with user-defined borders.*

**Syntax**

Processing images of 32-bit sizes

IppStatus ippiFilterBilateralBorderGetBufferSize(IppiFilterBilateralType filter, IppiSize dstRoiSize, int radius, IppDataType dataType, int numChannels, IppiDistanceMethodType distMethodType, int* pSpecSize, int* pBufferSize);

Platform-aware function

IppStatus ippiFilterBilateralBorderGetBufferSize_L(IppiFilterBilateralType filter, IppiSizeL dstRoiSize, int radius, IppDataType dataType, int numChannels, IppiDistanceMethodType distMethodType, IppSizeL* pSpecSize, IppSizeL* pBufferSize);

Threading layer function

IppStatus ippiFilterBilateralBorderGetBufferSize_LT(IppiFilterBilateralType filter, IppiSizeL dstRoiSize, int radius, IppDataType dataType, int numChannels, IppiDistanceMethodType distMethodType, IppSizeL* pSpecSize, IppSizeL* pBufferSize);

**Include Files**

ippi.h

Flavors with the `_LT` suffix: ippi_tl.h

Flavors with the `_L` suffix: ippi_l.h

**Domain Dependencies**

Flavors declared in ippi.h:

Headers: ippcore.h, ippvm.h, ipps.h
Libraries: ippcore.lib, ippvm.lib, ipps.lib

Flavors declared in ippi_tl.h:


**Parameters**

*filter* Type of the bilateral filter. Possible value is ippiFilterBilateralGauss - Gaussian bilateral filter.
dstRoiSize

Size of the destination ROI in pixels.

radius

Radius of the round kernel. The radius value equal to 1 corresponds to distance between the closest pixels.

dataType

Data type of the source and destination images. Possible values are ipp8u and ipp32f.

numChannels

Number of channels in the images.

distMethodType

Method of defining intensive distance between pixels. Possible value is ippDistNormL1.

pSpecSize

Pointer to the computed size of the specification structure.

pBufferSize

Pointer to the computed size of the external buffer.

Description

This function computes the size of the bilateral context structure and external work buffer for the FilterBilateralBorder function. The results are stored in pSpecSize and pBufferSize.

Use the computed pBufferSize and pSpecSize values to allocate the memory using the ippMalloc or ippsMalloc functions. The allocated memory can be freed only by the ippFree or ippsFree functions, respectively. For more information about the memory allocation functions, refer to the "Support Functions" section of the Intel IPP Developer Reference, vol. 1.

For an example on how to use this function, refer to the example provided with the FilterBilateralBorder function description.

Return Values

ippStsNoErr

Indicates no error. Any other value indicates an error.

ippStsNullPtrErr

Indicates an error when pSpecSize or pBufferSize is NULL.

ippStsSizeErr

Indicates an error when dstRoiSize has a field with a zero or negative value.

ippStsMaskSizeErr

Indicates an error when radius is less than, or equal to zero.

ippStsNotSupportedModeErr

Indicates an error when the filter or distMethodType value is not supported.

ippStsDataTypeErr

Indicates an error when dataType has an illegal value.

ippStsNumChannelsErr

Indicates an error when numChannels has an illegal value.

See Also
FilterBilateralBorder Performs bilateral filtering of an image.

FilterBilateralBorderInit

Initializes the bilateral context structure.

Syntax

IppStatus ippiFilterBilateralBorderInit(IppiFilterBilateralType filter, IppSize dstRoiSize, int radius, IppDataType dataType, int numChannels, IppiDistanceMethodType distMethod, Ipp32f valSquareSigma, Ipp32f posSquareSigma, IppiFilterBilateralSpec* pSpec);
Platform-aware function

IppStatus ippiFilterBilateralBorderInit_L(IppiFilterBilateralType filter, IppSizeL dstRoiSize, int radius, IppDataType dataType, int numChannels, IppiDistanceMethodType distMethod, Ipp32f valSquareSigma, Ipp32f posSquareSigma, IppiFilterBilateralSpec* pSpec);

Threading layer function

IppStatus ippiFilterBilateralBorderInit_LT(IppiFilterBilateralType filter, IppiSizeL dstRoiSize, int radius, IppDataType dataType, int numChannels, IppiDistanceMethodType distMethod, Ipp32f valSquareSigma, Ipp32f posSquareSigma, IppiFilterBilateralSpec_LT* pSpec);

Include Files

ippi.h

Flavors with the _LT suffix: ippi_tl.h
Flavors with the _L suffix: ippi_l.h

Domain Dependencies

Flavors declared in ippi.h:

Headers: ippcore.h, ippvm.h, ipps.h
Libraries: ippcore.lib, ippvm.lib, ipps.lib

Parameters

filter
Type of the bilateral filter. Possible value is ippiFilterBilateralGauss - Gaussian bilateral filter.

dstRoiSize
Size of the destination ROI in pixels.

radius
Radius of the round kernel. The radius value equal to 1 corresponds to distance between the closest pixels.

dataType
Data type of the source and destination images. Possible values are ipp8u and ipp32f.

numChannels
Number of channels in the images.

distMethod
Method of defining intensive distance between pixels. Possible value is ippDistNormL1.

valSquareSigma
Square of the sigma for intensive distance between pixels.

posSquareSigma
Square of the sigma for geometric distance between pixels.

pSpec
Pointer to the bilateral context structure.

Description

This function initializes the bilateral context structure pSpecSize for bilateral filtering. Before using this function, compute the size of the context structure using the FilterBilateralBorderGetBufferSize function.

The radius parameter specifies the radius of the round filter kernel. The radius value equal to 1 corresponds to the distance between centers of the closest pixels.
Coefficients of the bilateral filter kernel depend on their positions in the kernel and on the intensity value of the source image pixels lying in the kernel.

The value of the output pixel \( d \) is computed by the following formula:

\[
d = \frac{\sum_{i,j} w_{1ij} * w_{2ij} * v_{ij}}{\sum_{i,j} w_{1ij} * w_{2ij}}
\]

For all \( i \) and \( j \) that \( i^2+j^2 \leq \text{radius}^2 \times \text{radius}^2 \) (for central pixel of the kernel \( i=0, j=0 \))

where

- \( v_{ij} \) is the value (or channel values) of a pixel in the kernel with coordinates \( i \) and \( j \)
- \( w_{1ij} = \text{Fun}(\text{valSquareSigma}, \text{Intensity Distance}(v_{ij}, v_{00})) \)
  The \text{distMethodType} parameter specifies the method of defining the intensive distance between pixels. Currently the only supported method is \text{ippDistNormL1}, which defines the intensive distance as norma L1.
- \( w_{2ij} = \text{Fun}(\text{posSquareSigma}, \text{Geometric Distance}(v_{ij}, v_{00}) = \sqrt{i^2+j^2}) \)
  \text{Fun}(S, I) = \exp(-I^2/2*S)

where

- \( S \) is \text{valSquareSigma} or \text{posSquareSigma}
- \( I \) is the difference between pixel values or position

For an example on how to use this function, refer to the example provided with the \text{FilterBilateralBorder} function description.

**Return Values**

- ippStsNoErr Indicates no error. Any other value indicates an error.
- ippStsNullPtrErr Indicates an error when \( pSpec \) is NULL.
- ippStsSizeErr Indicates an error when \( dstRoiSize \) has a field with a zero or negative value.
- ippStsMaskSizeErr Indicates an error when \( radius \) is less than, or equal to zero.
- ippStsNotSupportedModeErr Indicates an error when the \text{filter} or \text{distMethod} value is not supported.
- ippStsBadArgErr Indicates an error when \text{valSquareSigma} or \text{posSquareSigma} is less than, or equal to zero.
- ippStsNumChannelsErr Indicates an error when \text{numChannels} has an illegal value.

**See Also**

\text{FilterBilateralBorderGetBufferSize} Computes the size of the bilateral context structure and the size of the work buffer for bilateral filtering with user-defined borders.
\text{FilterBilateralBorder} Performs bilateral filtering of an image.
**FilterBilateralBorder**

*Performs bilateral filtering of an image.*

**Syntax**

**Processing images of 32-bit sizes**

\[
\text{IppStatus ippiFilterBilateralBorder}_<\text{mod}>(\text{const Ipp\langle srcdatatype\rangle* pSrc}, \text{int srcStep}, \text{Ipp\langle dstdatatype\rangle* pDst}, \text{int dstStep}, \text{IppiSize dstRoiSize}, \text{IppiBorderType borderType}, \text{Ipp\langle datatype\rangle* pBorderValue}, \text{IppFilterBilateralSpec* pSpec}, \text{Ipp8u* pBuffer});
\]

Supported values for \text{mod}:

- 8u\_C1R
- 8u\_C3R
- 32f\_C1R
- 32f\_C3R

**Platform-aware functions**

\[
\text{IppStatus ippiFilterBilateralBorder}_<\text{mod}>(\text{const Ipp\langle srcdatatype\rangle* pSrc}, \text{IppSizeL srcStep}, \text{Ipp\langle dstdatatype\rangle* pDst}, \text{IppSizeL dstStep}, \text{IppSizeL dstRoiSize}, \text{IppiBorderType borderType}, \text{Ipp\langle datatype\rangle* pBorderValue}, \text{const IppFilterBilateralSpec* pSpec}, \text{Ipp8u* pBuffer});
\]

Supported values for \text{mod}:

- 8u\_C1R\_L
- 8u\_C3R\_L

**Threading layer functions**

\[
\text{IppStatus ippiFilterBilateralBorder}_<\text{mod}>(\text{const Ipp\langle srcdatatype\rangle* pSrc}, \text{IppSizeL srcStep}, \text{Ipp\langle dstdatatype\rangle* pDst}, \text{IppSizeL dstStep}, \text{IppSizeL dstRoiSize}, \text{IppiBorderType borderType}, \text{Ipp\langle datatype\rangle* pBorderValue}, \text{IppFilterBilateralSpec\_LT* pSpec}, \text{Ipp8u* pBuffer});
\]

Supported values for \text{mod}:

- 8u\_C1R\_L
- 8u\_C3R\_L

**Include Files**

ippi.h

Flavors with the \_LT suffix: ippi\_tl.h

Flavors with the \_L suffix: ippi\_l.h

**Domain Dependencies**

Flavors declared in ippi.h:

Headers: ippcore.h, ippvm.h, ipps.h

Libraries: ippcore.lib, ippvm.lib, ipps.lib

Flavors declared in ippi\_tl.h:

Libraries: ippcore.lib, ippvm.lib, ipps.lib, ipp.lib, ippcore\_tl.lib, ippi\_tl.lib

**Parameters**

\text{pSrc} \hspace{1cm} \text{Pointer to the source image ROI}.
**srcStep**  
Distance, in bytes, between the starting points of consecutive lines in the source image.

**pDst**  
Pointer to the destination image ROI.

**dstStep**  
Distance, in bytes, between the starting points of consecutive lines in the destination image.

**dstRoiSize**  
Size of the source and destination ROI in pixels.

**borderType**  
Type of border. Possible values are:
- **ippBorderConst**  
Values of all border pixels are set to constant.
- **ippBorderRepl**  
Border is replicated from the edge pixels.
- **ippBorderInMem**  
Border is obtained from the source image pixels in memory.

Mixed borders are also supported. They can be obtained by the bitwise operation OR between **ippBorderRepl** and **ippBorderInMemTop**, **ippBorderInMemBottom**, **ippBorderInMemLeft**, and **ippBorderInMemRight**.

**pBorderValue**  
Constant value to assign to pixels of the constant border. This parameter is applicable only to the **ippBorderConst** border type.

**pSpec**  
Pointer to the bilateral context structure.

**pBuffer**  
Pointer to the work buffer.

**Description**

This function applies the bilateral filter with the round kernel to the source image. The radius of the kernel is defined in the corresponding initialization function `FilterBilateralBorderInit`. The bilateral context structure contains the parameters of filtering.

Before using the `ippiFilterBilateralBorder` function, compute the size of the bilateral context structure and the external buffer using the `FilterBilateralBorderGetBufferSize` function and initialize the structure using the `FilterBilateralBorderInit` function.

**Return Values**

- **ippStsNoErr**  
Indicates no error.
- **ippStsNullPtrErr**  
Indicates an error when `pSrc`, `pDst`, `pSpec`, or `pBuffer` is NULL.
- **ippStsSizeErr**  
Indicates an error when `dstRoiSize` has a field with a zero or negative value.
- **ippStsContextMatchErr**  
Indicates an error when the `pSpec` structure does not match the function.
- **ippStsNotEvenStepErr**  
Indicates an error when one of the step values is not divisible by 4 for floating-point images.
- **ippStsBorderErr**  
Indicates an error when `borderType` has an illegal value.

**Example**

To better understand usage of this function, refer to the following example in the examples archive available for download from https://software.intel.com/en-us/ipp-manual-examples:

`FilterBilateralBorder.c`
See Also
Regions of Interest in Intel IPP
User-defined Border Types
FilterBilateralBorderGetBufferSize Computes the size of the bilateral context structure and the size of the work buffer for bilateral filtering with user-defined borders.
FilterBilateralBorderInit Initializes the bilateral context structure.

FilterBoxBorderGetBufferSize
Computes the size of the external buffer for the FilterBoxBorder function.

Syntax
IppStatus ippiFilterBoxBorderGetBufferSize (IppiSize roiSize, IppiSize maskSize, IppDataType dataType, int numChannels, int* pBufferSize);

Include Files
ippi.h

Domain Dependencies
Headers: ippcore.h, ippvm.h, ipps.h
Libraries: ippcore.lib, ippvm.lib, ipps.lib

Parameters
roiSize
Maximum size of the destination image ROI.

maskSize
Size of the filter mask, in pixels.

dataType
Data type of the image. Possible values are ipp8u, ipp16u, ipp16s, or ipp32f.

numChannels
Number of channels in the image. Possible values are 1, 3, or 4.

pBufferSize
Pointer to the size of the external work buffer.

Description
The ippiFilterBoxBorderGetBufferSize function computes the size, in bytes, of the external work buffer needed for the ippiFilterBoxBorder function. The result is stored in the pBufferSize parameter.

For an example on how to use this function, see the code example provided with the ippiFilterBoxBorder function description.

Return Values
ippStsNoErr Indicates no error.
ippStsSizeErr Indicates an error when roiSize is negative, or equal to zero.
ippStsMaskSizeErr Indicates an error when mask has an illegal value.
ippStsDataTypeErr Indicates an error when dataType has an illegal value.
ippStsNumChannelsError Indicates an error when numChannels has an illegal value.
See Also

FilterBoxBorder Blurs an image using a simple box filter.

FilterBoxBorder
Blurs an image using a simple box filter.

Syntax

IppStatusippiFilterBoxBorder_<mod>(const Ipp<datatype>* pSrc, int srcStep, Ipp<datatype>* pDst, int dstStep, IppiSize roiSize, IppiSize maskSize, IppiBorderType border, const Ipp<datatype>* borderValue, Ipp8u* pBuffer);

Supported values for mod:

- 8u_C1R
- 16u_C1R
- 16s_C1R
- 32f_C1R

IppStatusippiFilterBoxBorder_<mod>(const Ipp<datatype>* pSrc, int srcStep, Ipp<datatype>* pDst, int dstStep, IppiSize roiSize, IppiSize maskSize, IppiBorderType border, const Ipp<datatype> borderValue[3], Ipp8u* pBuffer);

Supported values for mod:

- 8u_C3R
- 16u_C3R
- 16s_C3R
- 32f_C3R
- 16u_C4R
- 16s_C4R
- 16u_AC4R
- 16s_AC4R

IppStatusippiFilterBoxBorder_<mod>(const Ipp<datatype>* pSrc, int srcStep, Ipp<datatype>* pDst, int dstStep, IppiSize roiSize, IppiSize maskSize, IppiBorderType border, const Ipp<datatype> borderValue[4], Ipp8u* pBuffer);

Supported values for mod:

- 8u_C4R
- 32f_C4R
- 8u_AC4R
- 32f_AC4R

Include Files

ippi.h

Domain Dependencies

Headers: ippcore.h, ippvm.h, ipps.h
Libraries: ippcore.lib, ippvm.lib, ipps.lib

Parameters

- **pSrc**
  Pointer to the source image.

- **srcStep**
  Distance, in bytes, between the starting points of consecutive lines in the source image.

- **pDst**
  Pointer to the destination image.

- **dstStep**
  Distance, in bytes, between the starting points of consecutive lines in the destination image.
roiSize
Size of the destination ROI in pixels.

maskSize
Size of the mask in pixels.

border
Type of border. Possible values are:

- ippBorderConst: Values of all border pixels are set to constant.
- ippBorderRepl: Border is replicated from the edge pixels.
- ippBorderInMem: Border is obtained from the source image pixels in memory.
- ippBorderMirror: Border pixels are mirrored from the source image boundary pixels.

Mixed borders are also supported. They can be obtained by the bitwise operation OR between ippBorderRepl and ippBorderInMemTop, ippBorderInMemBottom, ippBorderInMemLeft, ippBorderInMemRight.

borderValue
Constant value to assign to pixels of the constant border. This parameter is applicable only to the ippBorderConst border type.

pBuffer
Pointer to the work buffer.

Description
Before using this function, you need to compute the size of the work buffer pBuffer using the ippiFilterBoxBorderGetBufferSize function.

This function operates with ROI.

This function sets each pixel in the destination image as the average of all pixels of the source image in the rectangular neighborhood of size maskSize with the anchor cell at that pixel. This has the effect of smoothing or blurring the input image. To ensure valid operation when image boundary pixels are processed, the application must correctly define additional border pixels. If pSrc is equal to pDst, ippiFilterBoxBorder operates as an in-place function.

Return Values

- ippStsNoErr: Indicates no error.
- ippStsNullPtrErr: Indicates an error when pSrc or pDst is NULL.
- ippStsSizeErr: Indicates an error if roiSize has a field with zero or negative value.
- ippStsMaskSizeErr: Indicates an error if mask has an illegal value.
- ippStsBorderErr: Indicates an error when border has an illegal value.

Example
To better understand usage of this function, refer to the following example in the examples archive available for download from https://software.intel.com/en-us/ipp-manual-examples:

FilterBoxBorder.c
Blurs an image using a simple box filter.
Syntax

IppStatus ippiFilterBox_64f_C1R(const Ipp<datatype>* pSrc, int srcStep, Ipp<datatype>* pDst, int dstStep, IppSize dstRoiSize, IppSize maskSize, IppiPoint anchor);

Include Files

ippi.h

Domain Dependencies

Headers: ippcore.h, ippvm.h, ipps.h
Libraries: ippcore.lib, ippvm.lib, ipps.lib

Parameters

pSrc
Pointer to the source image ROI.

srcStep
Distance in bytes between starts of consecutive lines in the source image.

pDst
Pointer to the destination image ROI.

dstStep
Distance in bytes between starts of consecutive lines in the destination image.

dstRoiSize
Size of the destination ROI in pixels.

maskSize
Size of the mask in pixels.

anchor
Anchor cell specifying the mask alignment with respect to the position of the input pixel.

Description

This function operates with ROI (see Regions of Interest in Intel IPP).

This function sets each pixel in the destination image as the average of all the input image pixels in the rectangular neighborhood of size maskSize with the anchor cell at that pixel. This has the effect of smoothing or blurring on the input image. To ensure valid operation when image boundary pixels are processed, the application must correctly define additional border pixels (see Borders in Neighborhood Operations).

Return Values

ippStsNoErr
Indicates no error. Any other value indicates an error or a warning.

ippStsNullPtrErr
Indicates an error if pSrc, pDst, or pSrcDst is NULL.

ippStsSizeErr
Indicates an error if dstRoiSize has a field with zero or negative value.

ippStsStepErr
Indicates an error if srcStep or dstStep has a zero or negative value.

ippStsMaskSizeErr
Indicates an error if maskSize has a field with zero or negative value.

ippStsAnchorErr
Indicates an error if anchor is outside the mask size.

ippStsMemAllocErr
Indicates a memory allocation error.
FilterGaussianBorder
Performs Gaussian filtering of an image with user-defined borders.

Syntax
IppStatus ippiFilterGaussianBorder_<mod>(const Ippdatatype* pSrc, int srcStep, Ippdatatype* pDst, int dstStep, IppSize roiSize, Ippdatatype borderValue, IppFilterGaussianSpec* pSpec, Ipp8u* pBuffer);

Supported values for mod:
8u_C1R  16u_C1R  16s_C1R  32f_C1R

IppStatus ippiFilterGaussianBorder_<mod>(const Ippdatatype* pSrc, int srcStep, Ippdatatype* pDst, int dstStep, IppSize roiSize, Ippdatatype borderValue[3], IppFilterGaussianSpec* pSpec, Ipp8u* pBuffer);

Supported values for mod:
8u_C3R  16u_C3R  16s_C3R  32f_C3R

Include Files
ippcv.h

Domain Dependencies
Headers: ippcore.h, ippvm.h, ipps.h,ippi.h
Libraries: ippcore.lib, ippvm.lib, ipps.lib,ippi.lib

Parameters

\begin{itemize}
\item \textbf{pSrc} \hspace{1cm} Pointer to the source image ROI.
\item \textbf{srcStep} \hspace{1cm} Distance, in bytes, between the starting points of consecutive lines in the source image.
\item \textbf{pDst} \hspace{1cm} Pointer to the destination image ROI.
\item \textbf{dstStep} \hspace{1cm} Distance, in bytes, between the starting points of consecutive lines in the destination image.
\item \textbf{roiSize} \hspace{1cm} Size of the source and destination image ROI in pixels.
\item \textbf{borderValue} \hspace{1cm} Constant value to assign to pixels of the constant border. This parameter is applicable only to the ippBorderConst border type.
\item \textbf{pSpec} \hspace{1cm} Pointer to the Gaussian specification structure.
\item \textbf{pBuffer} \hspace{1cm} Pointer to the work buffer.
\end{itemize}

Description
This function operates with ROI (see Regions of Interest in Intel IPP).
This function applies the Gaussian filter to the source image ROI \texttt{pSrc}. The kernel of the Gaussian filter is the matrix of size \texttt{kernelSize x kernelSize} with the standard deviation \texttt{sigma}. The values of the Gaussian kernel elements are computed by the FilterGaussianInit function. Elements of the kernel are normalized. The anchor cell is the center of the kernel.
Before using the ippFilterGaussianBorder function, compute the size of the Gaussian specification structure and the external buffer using the FilterGaussianGetBufferSize function and initialize the structure using the FilterGaussianInit function.

Return Values

- ippStsNoErr: Indicates no error.
- ippStsNullPtrErr: Indicates an error when one of the specified pointers is NULL.
- ippStsSizeErr: Indicates an error when roiSize has a field with a zero or negative value.
- ippStsStepErr: Indicates an error when srcStep or dstStep is less than roiSize.width*<pixelSize>.
- ippStsNotEvenStepErr: Indicates an error when one of the step values is not divisible by sizeof(Ipp<dataType>).
- ippStsBorderErr: Indicates an error when borderType has an illegal value.
- ippStsBadArgErr: Indicates an error when kernelSize is even, or less than 3.

Example

To better understand usage of this function, refer to the following example in the examples archive available for download from https://software.intel.com/en-us/ipp-manual-examples:

FilterGaussianBorder.c

See Also

Regions of Interest in Intel IPP
User-defined Border Types
FilterGaussianGetBufferSize Computes the size of the Gaussian specification structure and the size of the external work buffer for Gaussian filtering with user-defined borders.
FilterGaussianInit Initializes the Gaussian context structure.

SumWindow

Sums pixel values in a rectangular area applied to an image.

Syntax

IppStatus ippiSumWindow_8u32s_C1R(const Ipp8u* pSrc, int srcStep, Ipp32s* pDst, int dstStep, IppSize roiSize, IppSize maskSize, IppiBorderType BorderType, const Ipp8u* borderValue, Ipp8u* pBuffer);
IppStatus ippiSumWindow_8u32s_C3R(const Ipp8u* pSrc, int srcStep, Ipp32s* pDst, int dstStep, IppSize roiSize, IppSize maskSize, IppiBorderType BorderType, const Ipp8u* borderValue, Ipp8u* pBuffer);
IppStatus ippiSumWindow_8u32s_C4R(const Ipp8u* pSrc, int srcStep, Ipp32s* pDst, int dstStep, IppSize roiSize, IppSize maskSize, IppiBorderType BorderType, const Ipp8u* borderValue, Ipp8u* pBuffer);
IppStatus ippiSumWindow_8u32s_AC4R(const Ipp8u* pSrc, int srcStep, Ipp32s* pDst, int dstStep, IppSize roiSize, IppSize maskSize, IppiBorderType BorderType, const Ipp8u* borderValue, Ipp8u* pBuffer);
IppStatus ippiSumWindow_16s32f_C1R(const Ipp16s* pSrc, int srcStep, Ipp32f* pDst, int dstStep, IppiSize roiSize, IppiSize maskSize, IppiBorderType BorderType, const Ipp16s* borderValue, Ipp8u* pBuffer);
IppStatus ippiSumWindow_16s32f_C3R(const Ipp16s* pSrc, int srcStep, Ipp32f* pDst, int dstStep, IppiSize roiSize, IppiSize maskSize, IppiBorderType BorderType, const Ipp16s* borderValue, Ipp8u* pBuffer);
IppStatus ippiSumWindow_16s32f_C4R(const Ipp16s* pSrc, int srcStep, Ipp32f* pDst, int dstStep, IppiSize roiSize, IppiSize maskSize, IppiBorderType BorderType, const Ipp16s* borderValue, Ipp8u* pBuffer);
IppStatus ippiSumWindow_16s32f_AC4R(const Ipp16s* pSrc, int srcStep, Ipp32f* pDst, int dstStep, IppiSize roiSize, IppiSize maskSize, IppiBorderType BorderType, const Ipp16s* borderValue, Ipp8u* pBuffer);
IppStatus ippiSumWindow_16u32f_C1R(const Ipp16u* pSrc, int srcStep, Ipp32f* pDst, int dstStep, IppiSize roiSize, IppiSize maskSize, IppiBorderType BorderType, const Ipp16u* borderValue, Ipp8u* pBuffer);
IppStatus ippiSumWindow_16u32f_C3R(const Ipp16u* pSrc, int srcStep, Ipp32f* pDst, int dstStep, IppiSize roiSize, IppiSize maskSize, IppiBorderType BorderType, const Ipp16u* borderValue, Ipp8u* pBuffer);
IppStatus ippiSumWindow_16u32f_C4R(const Ipp16u* pSrc, int srcStep, Ipp32f* pDst, int dstStep, IppiSize roiSize, IppiSize maskSize, IppiBorderType BorderType, const Ipp16u* borderValue, Ipp8u* pBuffer);
IppStatus ippiSumWindow_16u32f_AC4R(const Ipp16u* pSrc, int srcStep, Ipp32f* pDst, int dstStep, IppiSize roiSize, IppiSize maskSize, IppiBorderType BorderType, const Ipp16u* borderValue, Ipp8u* pBuffer);
IppStatus ippiSumWindow_32f_C1R(const Ipp32f* pSrc, int srcStep, Ipp32f* pDst, int dstStep, IppiSize roiSize, IppiSize maskSize, IppiBorderType BorderType, const Ipp32f* borderValue, Ipp8u* pBuffer);
IppStatus ippiSumWindow_32f_C3R(const Ipp32f* pSrc, int srcStep, Ipp32f* pDst, int dstStep, IppiSize roiSize, IppiSize maskSize, IppiBorderType BorderType, const Ipp32f* borderValue, Ipp8u* pBuffer);
IppStatus ippiSumWindow_32f_C4R(const Ipp32f* pSrc, int srcStep, Ipp32f* pDst, int dstStep, IppiSize roiSize, IppiSize maskSize, IppiBorderType BorderType, const Ipp32f* borderValue, Ipp8u* pBuffer);
IppStatus ippiSumWindow_32f_AC4R(const Ipp32f* pSrc, int srcStep, Ipp32f* pDst, int dstStep, IppiSize roiSize, IppiSize maskSize, IppiBorderType BorderType, const Ipp32f* borderValue, Ipp8u* pBuffer);

Include Files
ippi.h

Domain Dependencies

Headers: ippcore.h, ippvm.h, ipps.h
Libraries: ippcore.lib, ippvm.lib, ipps.lib

Parameters

pSrc  Pointer to the source image ROI.
**srcStep**
Distance in bytes between the starting points of consecutive lines in the source image.

**pDst**
Pointer to the destination image ROI.

**dstStep**
Distance in bytes between the starting points of consecutive lines in the destination image.

**roiSize**
Size of the destination ROI in pixels.

**maskSize**
Size of the mask in pixels.

**BorderType**
Type of border. Possible values are:

- **ippBorderConst**
  Values of all border pixels are set to constant.

- **ippBorderRepl**
  Border is replicated from the edge pixels.

- **ippBorderInMem**
  Border is obtained from the source image pixels in memory.

- **ippBorderMirror**
  Border pixels are mirrored from the source image boundary pixels.

Mixed borders are also supported. They can be obtained by the bitwise operation OR between any of the **ippBorderRepl**, **ippBorderConst**, **ippBorderMirror**, and the **ippBorderInMemTop**, **ippBorderInMemBottom**, **ippBorderInMemLeft**, **ippBorderInMemRight** values.

**borderValue**
Constant value to assign to border pixels. This parameter is applicable only to the **ippBorderConst** border type.

**pBuffer**
Pointer to the work buffer.

**Description**
Before using this function, you need to compute the size of the work buffer **pBuffer** using the **SumWindowGetBufferSize** function.

This function operates with ROI (see Regions of Interest in Intel IPP).

This function sets each pixel in the destination image ROI **pDst** to the sum of all the source image pixels in the rectangular neighborhood of size **maskSize** with the anchor cell at the corresponding pixel in the source image ROI **pSrc**. To ensure valid operation while processing the image boundary pixels, the application must correctly define additional border pixels (see Borders in Neighborhood Operations).

**Return Values**

- **ippStsNoErr**
  Indicates no error. Any other value indicates an error or a warning.

- **ippStsNullPtrErr**
  Indicates an error if **pSrc** or **pDst** is NULL.

- **ippStsSizeErr**
  Indicates an error if **roiSize** has a field with a zero or negative value.

- **ippStsMaskSizeErr**
  Indicates an error if **maskSize** has a field with a zero or negative value.

- **ippStsBorderErr**
  Indicates an error if **BorderType** has an illegal value.
SumWindowGetBufferSize

Computes the size of the external buffer for the SumWindow function.

Syntax

IppStatus ippiSumWindowGetBufferSize(IppiSize roiSize, IppiSize maskSize, IppDataType dataType, int numChannels, int* pBufferSize);

Include Files

ippi.h

Domain Dependencies

Headers: ippcore.h, ippvm.h, ipps.h
Libraries: ippcore.lib, ippvm.lib, ipps.lib

Parameters

roiSize
Maximum size of the destination ROI in pixels.

maskSize
Size of the filter mask in pixels.

dataType
Data type of the image. Possible values are: ipp8u, ipp16s, ipp16u, and ipp32f

numChannels
Number of channels in the image. Possible values are 1, 3, and 4.

pBufferSize
Pointer to the size, in bytes, of the external work buffer.

Description

This function operates with ROI (see Regions of Interest in Intel IPP).
This function computes the size, in bytes, of the external work buffer for the ippiSumWindow function. The result is stored in the pBufferSize parameter.

Return Values

ippStsNoErr
Indicates no error. Any other value indicates an error or a warning.

ippStsSizeErr
Indicates an error if roiSize has a field with a zero or negative value.

ippStsMaskSizeErr
Indicates an error if maskSize has an illegal value.

ippStsDataTypeErr
Indicates an error if dataType has an illegal value.

ippStsNumChannelsError
Indicates an error if numChannels has an illegal value.

SumWindowRow

Sums pixel values in the row mask applied to the image.
**Syntax**

```c
IppStatus ippiSumWindowRow_<mod>(const Ipp<srcDatatype>* pSrc, int srcStep, Ipp32f* pDst, int dstStep, IppiSize dstRoiSize, int maskSize, int anchor);
```

**Supported values for mod:**

- 8u32f_C1R
- 16u32f_C1R
- 16s32f_C1R
- 8u32f_C3R
- 16u32f_C3R
- 16s32f_C3R
- 8u32f_C4R
- 16u32f_C4R
- 16s32f_C4R

**Include Files**

ippi.h

**Domain Dependencies**

Headers: ippcore.h, ippvm.h, ipps.h

Libraries: ippcore.lib, ippvm.lib, ipps.lib

**Parameters**

- **pSrc**
  Pointer to the source image ROI.
- **srcStep**
  Distance in bytes between starts of consecutive lines in the source image.
- **pDst**
  Pointer to the destination image ROI.
- **dstStep**
  Distance in bytes between starts of consecutive lines in the destination image.
- **dstRoiSize**
  Size of the destination ROI in pixels.
- **maskSize**
  Size of the horizontal row mask in pixels.
- **anchor**
  Anchor cell specifying the row mask alignment with respect to the position of the input pixel.

**Description**

This function operates with ROI (see Regions of Interest in Intel IPP).

This function sets each pixel in the destination image ROI `pDst` as the sum of all the source image pixels in the horizontal row mask of size `maskSize` with the anchor cell `anchor` at the corresponding pixel in the source image ROI `pSrc`. To ensure valid operation when image boundary pixels are processed, the application must correctly define additional border pixels (see Borders in Neighborhood Operations).

**Return Values**

- **ippStsNoErr**
  Indicates no error. Any other value indicates an error or a warning.
  Indicates an error if `pSrc`, `pDst` is NULL.
- **ippStsNullPtrErr**
  Indicates an error if `dstRoiSize` has a field with a zero or negative value.
- **ippStsSizeErr**
  Indicates an error if `maskSize` has a field with a zero or negative value.
- **ippStsAnchorErr**
  Indicates an error if `anchor` is outside the mask size.
ippStsMemAllocErr  Indicates a memory allocation error.

**SumWindowColumn**
*Sums pixel values in the column mask applied to the image.*

**Syntax**

\[
\text{IppStatus ippiSumWindowColumn_<mod>(const Ipp<srcDatatype>* pSrc, int srcStep, Ipp32f* pDst, int dstStep, IppiSize dstRoiSize, int maskSize, int anchor);}\
\]

**Include Files**

ippi.h

**Domain Dependencies**

*Headers*: ippcore.h, ippvm.h, ipps.h

*Libraries*: ippcore.lib, ippvm.lib, ipps.lib

**Parameters**

- **pSrc**
  Pointer to the source image ROI.

- **srcStep**
  Distance in bytes between starts of consecutive lines in the source image.

- **pDst**
  Pointer to the destination image ROI.

- **dstStep**
  Distance in bytes between starts of consecutive lines in the destination image.

- **dstRoiSize**
  Size of the destination ROI in pixels.

- **maskSize**
  Size of the vertical column mask in pixels.

- **anchor**
  Anchor cell specifying the column mask alignment with respect to the position of the input pixel.

**Description**

This function operates with ROI (see Regions of Interest in Intel IPP).

This function sets each pixel in the destination image ROI \( pDst \) as the sum of all the source image pixels in the vertical column mask of size \( maskSize \) with the anchor cell \( anchor \) at the corresponding pixel in the source image ROI \( pSrc \). To ensure valid operation when image boundary pixels are processed, the application must correctly define additional border pixels (see Borders in Neighborhood Operations).

**Return Values**

- **ippStsNoErr**  Indicates no error. Any other value indicates an error or a warning.
Indicates an error if \( pSrc, pDst \) is NULL.

Indicates an error if \( dstRoiSize \) has a field with a zero or negative value.

Indicates an error if \( maskSize \) has a field with a zero or negative value.

Indicates an error if \( anchor \) is outside the mask size.

Indicates a memory allocation error.

\[ \text{FilterMaxBorderGetBufferSize, FilterMinBorderGetBufferSize} \]

*Compute the size of the work buffer for the maximum/minimum filter.*

**Syntax**

\[
\text{IppStatus ippiFilterMaxBorderGetBufferSize(IppiSize dstRoiSize, IppiSize maskSize, IppDataType dataType, int numChannels, int* pBufferSize);}\
\]

\[
\text{IppStatus ippiFilterMinBorderGetBufferSize(IppiSize dstRoiSize, IppiSize maskSize, IppDataType dataType, int numChannels, int* pBufferSize);}\
\]

**Include Files**

ippi.h

**Domain Dependencies**

Headers: ippcore.h, ippvm.h, ipps.h

Libraries: ippcore.lib, ippvm.lib, ipps.lib

**Parameters**

- \( dstRoiSize \): Size of the destination ROI, in pixels.
- \( maskSize \): Size of the filter kernel.
- \( dataType \): Data type of the source and destination images.
- \( numChannels \): Number of channels in the image. Possible values are 1, 3, or 4.
- \( pBufferSize \): Pointer to the size, in bytes, of the external buffer.

**Description**

The \( \text{ippiFilterMaxBorderGetBufferSize} \) and \( \text{ippiFilterMinBorderGetBufferSize} \) functions compute the size, in bytes, of the external work buffer for the \( \text{ippiFilterMaxBorder} \) and \( \text{ippiFilterMinBorder} \) functions, respectively. The result is stored in the \( pBufferSize \) parameter.

**Return Values**

- \( \text{ippStsNoErr} \): Indicates no error. Any other value indicates an error.
- \( \text{ippStsNullPtrErr} \): Indicates an error when \( pBufferSize \) is NULL.
- \( \text{ippStsSizeErr} \): Indicates an error when \( dstRoiSize \) has a field with a zero or negative value.
FilterMaxBorder, FilterMinBorder  
Filter an image using the maximum/minimum filter.

**Syntax**

**Case 1: Operating on one-channel data**

IppStatus ippiFilterMaxBorder_8u_C1R(const Ipp8u* pSrc, int srcStep, Ipp8u* pDst, int dstStep, IppiSize dstRoiSize, IppiSize maskSize, IppiBorderType borderType, Ipp8u borderValue, Ipp8u* pBuffer);

IppStatus ippiFilterMaxBorder_16s_C1R(const Ipp16s* pSrc, int srcStep, Ipp16s* pDst, int dstStep, IppiSize dstRoiSize, IppiSize maskSize, IppiBorderType borderType, Ipp16s borderValue, Ipp8u* pBuffer);

IppStatus ippiFilterMaxBorder_16u_C1R(const Ipp16u* pSrc, int srcStep, Ipp16u* pDst, int dstStep, IppiSize dstRoiSize, IppiSize maskSize, IppiBorderType borderType, Ipp16u borderValue, Ipp8u* pBuffer);

IppStatus ippiFilterMaxBorder_32f_C1R(const Ipp32f* pSrc, int srcStep, Ipp32f* pDst, int dstStep, IppiSize dstRoiSize, IppiSize maskSize, IppiBorderType borderType, Ipp32f borderValue, Ipp8u* pBuffer);

**Case 2: Operating on multi-channel data**

IppStatus ippiFilterMaxBorder_8u_C3R(const Ipp8u* pSrc, int srcStep, Ipp8u* pDst, int dstStep, IppiSize dstRoiSize, IppiSize maskSize, IppiBorderType borderType, const Ipp8u pBorderValue[3], Ipp8u* pBuffer);

IppStatus ippiFilterMaxBorder_16s_C3R(const Ipp16s* pSrc, int srcStep, Ipp16s* pDst, int dstStep, IppiSize dstRoiSize, IppiSize maskSize, IppiBorderType borderType, const Ipp16s pBorderValue[3], Ipp8u* pBuffer);

See Also
FilterMaxBorder, FilterMinBorder  
Filter an image using the maximum/minimum filter.
IppStatus ippiFilterMaxBorder_16u_C3R(const Ipp16u* pSrc, int srcStep, Ipp16u* pDst, int dstStep, IppSize dstRoiSize, IppSize maskSize, IppBorderType borderType, const Ipp16u pBorderValue[3], Ipp8u* pBuffer);
IppStatus ippiFilterMaxBorder_32f_C3R(const Ipp32f* pSrc, int srcStep, Ipp32f* pDst, int dstStep, IppSize dstRoiSize, IppSize maskSize, IppBorderType borderType, const Ipp32f pBorderValue[3], Ipp8u* pBuffer);
IppStatus ippiFilterMaxBorder_8u_AC4R(const Ipp8u* pSrc, int srcStep, Ipp8u* pDst, int dstStep, IppSize dstRoiSize, IppSize maskSize, IppBorderType borderType, const Ipp8u pBorderValue[3], Ipp8u* pBuffer);
IppStatus ippiFilterMaxBorder_16s_C3R(const Ipp16s* pSrc, int srcStep, Ipp16s* pDst, int dstStep, IppSize dstRoiSize, IppSize maskSize, IppBorderType borderType, const Ipp16s pBorderValue[3], Ipp8u* pBuffer);
IppStatus ippiFilterMaxBorder_16u_AC4R(const Ipp16u* pSrc, int srcStep, Ipp16u* pDst, int dstStep, IppSize dstRoiSize, IppSize maskSize, IppBorderType borderType, const Ipp16u pBorderValue[3], Ipp8u* pBuffer);
IppStatus ippiFilterMaxBorder_32f_AC4R(const Ipp32f* pSrc, int srcStep, Ipp32f* pDst, int dstStep, IppSize dstRoiSize, IppSize maskSize, IppBorderType borderType, const Ipp32f pBorderValue[3], Ipp8u* pBuffer);
IppStatus ippiFilterMinBorder_8u_C3R(const Ipp8u* pSrc, int srcStep, Ipp8u* pDst, int dstStep, IppSize dstRoiSize, IppSize maskSize, IppBorderType borderType, const Ipp8u pBorderValue[3], Ipp8u* pBuffer);
IppStatus ippiFilterMinBorder_16s_C3R(const Ipp16s* pSrc, int srcStep, Ipp16s* pDst, int dstStep, IppSize dstRoiSize, IppSize maskSize, IppBorderType borderType, const Ipp16s pBorderValue[3], Ipp8u* pBuffer);
IppStatus ippiFilterMinBorder_16u_C3R(const Ipp16u* pSrc, int srcStep, Ipp16u* pDst, int dstStep, IppSize dstRoiSize, IppSize maskSize, IppBorderType borderType, const Ipp16u pBorderValue[3], Ipp8u* pBuffer);
IppStatus ippiFilterMinBorder_32f_C3R(const Ipp32f* pSrc, int srcStep, Ipp32f* pDst, int dstStep, IppSize dstRoiSize, IppSize maskSize, IppBorderType borderType, const Ipp32f pBorderValue[3], Ipp8u* pBuffer);
IppStatus ippiFilterMinBorder_8u_AC4R(const Ipp8u* pSrc, int srcStep, Ipp8u* pDst, int dstStep, IppSize dstRoiSize, IppSize maskSize, IppBorderType borderType, const Ipp8u pBorderValue[3], Ipp8u* pBuffer);
IppStatusippiFilterMinBorder_16s_AC4R(const Ipp16s* pSrc, int srcStep, Ipp16s* pDst, int dstStep, IppSize dstRoiSize, IppiSize maskSize, IppiBorderType borderType, const Ipp16s pBorderValue[3], Ipp8u* pBuffer);

IppStatusippiFilterMinBorder_16u_AC4R(const Ipp16u* pSrc, int srcStep, Ipp16u* pDst, int dstStep, IppiSize dstRoiSize, IppiSize maskSize, IppiBorderType borderType, const Ipp16u pBorderValue[3], Ipp8u* pBuffer);

IppStatusippiFilterMinBorder_32f_AC4R(const Ipp32f* pSrc, int srcStep, Ipp32f* pDst, int dstStep, IppiSize dstRoiSize, IppiSize maskSize, IppiBorderType borderType, const Ipp32f pBorderValue[3], Ipp8u* pBuffer);

IppStatusippiFilterMinBorder_8u_C4R(const Ipp8u* pSrc, int srcStep, Ipp8u* pDst, int dstStep, IppiSize dstRoiSize, IppiSize maskSize, IppiBorderType borderType, const Ipp8u pBorderValue[4], Ipp8u* pBuffer);

IppStatusippiFilterMinBorder_16s_C4R(const Ipp16s* pSrc, int srcStep, Ipp16s* pDst, int dstStep, IppiSize dstRoiSize, IppiSize maskSize, IppiBorderType borderType, const Ipp16s pBorderValue[4], Ipp8u* pBuffer);

IppStatusippiFilterMinBorder_16u_C4R(const Ipp16u* pSrc, int srcStep, Ipp16u* pDst, int dstStep, IppiSize dstRoiSize, IppiSize maskSize, IppiBorderType borderType, const Ipp16u pBorderValue[4], Ipp8u* pBuffer);

IppStatusippiFilterMinBorder_32f_C4R(const Ipp32f* pSrc, int srcStep, Ipp32f* pDst, int dstStep, IppiSize dstRoiSize, IppiSize maskSize, IppiBorderType borderType, const Ipp32f pBorderValue[4], Ipp8u* pBuffer);

Include Files

ippi.h

Domain Dependencies

Headers: ippicore.h, ippvm.h, ipps.h
Libraries: ippcore.lib, ippvm.lib, ipps.lib

Parameters

\textbf{pSrc}  
Pointer to the source image ROI.

\textbf{srcStep}  
Distance, in bytes, between the starting points of consecutive lines in the source image.

\textbf{pDst}  
Pointer to the destination image ROI.

\textbf{dstStep}  
Distance, in bytes, between the starting points of consecutive lines in the destination image.

\textbf{dstRoiSize}  
Size of the source and destination ROI, in pixels.

\textbf{maskSize}  
Size of the filter kernel.

\textbf{borderType}  
Type of border. Possible values are:

- ippBorderConst  
Values of all border pixels are set to a constant.

- ippBorderRepl  
Border is replicated from the edge pixels.

- ippBorderInMem  
Border is obtained from the source image pixels in memory.
Mixed borders are also supported. They can be obtained by the bitwise operation OR between any of the ippBorderRepl, ippBorderConst, or ippBorderMirror values and the ippBorderInMemTop, ippBorderInMemBottom, ippBorderInMemLeft, ippBorderInMemRight values.

`borderValue`  
Constant value to assign to pixels of the constant border. This parameter is applicable only to the ippBorderConst border type.

`pBorderValue[3], pBorderValue[4]`  
Pointer to constant values to assign to pixels of the constant border. This parameter is applicable only to the ippBorderConst border type.

`pBuffer`  
Pointer to the work buffer.

**Description**

Before using the `ippiFilterMaxBorder` and `ippiFilterMinBorder` functions, you need to compute the size of the work buffer `pBuffer` using the `ippiFilterMaxBorderGetBufferSize` or `ippiFilterMinBorderGetBufferSize` functions, respectively.

These functions operate with ROI (see Regions of Interest in Intel IPP).

The `ippiFilterMaxBorder` and `ippiFilterMinBorder` functions apply the maximum/minimum filters, respectively, to the source image ROI. The size of the source image ROI is equal to the destination image ROI size `dstRoiSize`. The values of border pixels are assigned in accordance with the `borderType` and `borderValue` parameters. The kernel size of the filter is arbitrary and depends on the `mask` value.

The anchor cell is the center cell of the kernel, highlighted in red. The anchor cell is defined as:

\[
x = \frac{\text{maskSize.width} - 1}{2}
\]
\[
y = \frac{\text{maskSize.height} - 1}{2}
\]

where

\((x, y)\) are cell coordinates.

**Return Values**

- **ippiStsNoErr** Indicates no error.
- **ippiStsNullPtrErr** Indicates an error when one of the specified pointers is NULL.
- **ippiStsSizeErr** Indicates an error when `dstRoiSize` is negative, or equal to zero.
- **ippiStsBorderErr** Indicates an error when `mask` is less than, or equal to zero.
- **ippiStsBorderErr** Indicates an error when `borderType` has an illegal value.

**See Also**

Borders in Neighborhood Operations  
Regions of Interest in Intel IPP  
User-defined Border Types  
FilterMaxBorderGetBufferSize, FilterMinBorderGetBufferSize Compute the size of the work buffer for the maximum/minimum filter.

**DecimateFilterRow, DecimateFilterColumn**

Decimates an image by rows or by columns.
Syntax

IppStatus ippiDecimateFilterRow_8u_C1R(const Ipp8u* pSrc, int srcStep, IppSize srcRoiSize, Ipp8u* pDst, int dstStep, IppiFraction fraction);

IppStatus ippiDecimateFilterColumn_8u_C1R(const Ipp8u* pSrc, int srcStep, IppSize srcRoiSize, Ipp8u* pDst, int dstStep, IppiFraction fraction);

Include Files

ippi.h

Domain Dependencies

Headers: ippcore.h, ippvm.h, ipps.h
Libraries: ippcore.lib, ippvm.lib, ipps.lib

Parameters

pSrc
Distance in bytes between starts of consecutive lines in the source image.

srcStep
Size of the source image ROI in pixels.

srcRoiSize
Pointer to the source image ROI.

pDst
Distance in bytes between starts of consecutive lines in the destination image.

dstStep
Pointer to the destination image.

fraction
Specifies how the decimating is performed. Possible values:

ippPolyphase_1_2,
ippPolyphase_3_5,
ippPolyphase_2_3,
ippPolyphase_7_10,
ippPolyphase_3_4.

Description

These functions operate with ROI (see Regions of Interest in Intel IPP).

Functions DecimateFilterRow and DecimateFilterColumn perform decimating of the source image by rows or by columns respectively. These functions use the set of special internal polyphase filters. The parameter fraction specifies how the decimating is performed, for example, if the parameter is set to ippPolyphase_3_5, then each 5 pixels in the row (or column) of the source image give 3 pixels to the destination image, if the parameter is set to ippPolyphase_1_2, then each two pixels in the row (or column) of the source image give 1 pixel to the destination image, an so on.

To ensure valid operation, the application must correctly define additional border pixels (see Borders in Neighborhood Operations). For all fraction values the width of the border is four columns/rows all around the source image ROI.

Return Values

ippStsNoErr
Indicates no error. Any other value indicates an error or a warning.

ippStsNullPtrErr
Indicates an error if one of the specified pointers is NULL.
Median Filters

The median filter functions perform non-linear filtering of a source image data. These functions use either an arbitrary rectangular mask, or the following predefined masks of the IppiMaskSize type to filter an image:

- ippMskSize3x1: Horizontal mask of length 3
- ippMskSize5x1: Horizontal mask of length 5
- ippMskSize1x3: Vertical mask of length 3
- ippMskSize3x3: Square mask of size 3
- ippMskSize1x5: Vertical mask of length 5
- ippMskSize5x5: Square mask of size 5

The size of the neighborhood and coordinates of the anchor cell in the neighborhood depend on the mask mean value. Table “Median Filter Mask, Neighborhood, and Anchor Cell” lists the mask types with the corresponding neighborhood sizes and anchor cell coordinates. Mask size in mask names is indicated in (XY) order. The anchor cell is specified by its coordinates anchor.x and anchor.y in the coordinate system associated with the upper left corner of the mask.

<table>
<thead>
<tr>
<th>Mask</th>
<th>Neighborhood Size</th>
<th>Anchor Cell</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Columns</td>
<td>Rows</td>
</tr>
<tr>
<td>ippMskSize3x1</td>
<td>3</td>
<td>1</td>
</tr>
<tr>
<td>ippMskSize5x1</td>
<td>5</td>
<td>1</td>
</tr>
<tr>
<td>ippMskSize1x3</td>
<td>1</td>
<td>3</td>
</tr>
<tr>
<td>ippMskSize3x3</td>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>ippMskSize1x5</td>
<td>1</td>
<td>5</td>
</tr>
<tr>
<td>ippMskSize5x5</td>
<td>5</td>
<td>5</td>
</tr>
</tbody>
</table>

Median filters have the effect of removing the isolated intensity spikes and can be used to reduce noise in an image.

For details on algorithms used in Intel IPP for median filtering, see [APMF].
Syntax

IppStatus ippiFilterMedianBorderGetBufferSize (IppiSize dstRoiSize, IppiSize maskSize, IppDataType dataType, int numChannels, int* pBufferSize);

Include Files

ippi.h

Parameters

dstRoiSize
Size of the destination ROI, in pixels.

maskSize
Size of the filter mask, in pixels.

dataType
Data type of the source and destination images. Possible values are ipp8u, ipp16u, ipp16s, or ipp32f.

numChannels
Number of channels in the image. Possible values are 1, 3, or 4.

pBufferSize
Pointer to the computed size of the external work buffer, in bytes.

Description

The ippiFilterMedianBorderGetBufferSize function computes the size, in bytes, of the external work buffer needed for the ippiFilterMedianBorder function. The result is stored in the pBufferSize parameter.

For an example on how to use this function, see the code example provided with the ippiFilterMedianBorder function description.

Return Values

ippStsNoErr
Indicates no error.

ippStsNullPtrErr
Indicates an error when pBufferSize is NULL.

ippStsSizeErr
Indicates an error when one of the dstRoiSize fields has a negative or zero value.

ippStsMaskSizeErr
Indicates an error when mask has a field with a negative, zero, or even value.

ippStsDataTypeErr
Indicates an error when dataType has an illegal value.

ippStsNumChannelsError
Indicates an error when numChannels has an illegal value.

See Also

FilterMedianBorder Performs median filtering of an image.

FilterMedianBorder
Performs median filtering of an image.
**Syntax**

**Case 1: Operating on one-channel data**

```c
IppStatus ippiFilterMedianBorder_<mod>(const Ipp<datatype>* pSrc, int srcStep, Ipp<datatype>* pDst, int dstStep, IppiSize dstRoiSize, IppiSize maskSize, IppiBorderType borderType, Ipp<datatype> borderValue, Ipp8u* pBuffer);
```

Supported values for `mod`:

- 8u_C1R
- 16u_C1R
- 16s_C1R
- 32f_C1R

**Case 2: Operating on multi-channel data**

```c
IppStatus ippiFilterMedianBorder_<mod>(const Ipp<datatype>* pSrc, int srcStep, Ipp<datatype>* pDst, int dstStep, IppiSize dstRoiSize, IppiSize maskSize, IppiBorderType borderType, const Ipp<datatype> pBorderValue[3], Ipp8u* pBuffer);
```

Supported values for `mod`:

- 8u_C3R
- 16u_C3R
- 16s_C3R

- 8u_AC4R
- 16u_AC4R
- 16s_AC4R

```c
IppStatus ippiFilterMedianBorder_<mod>(const Ipp<datatype>* pSrc, int srcStep, Ipp<datatype>* pDst, int dstStep, IppiSize dstRoiSize, IppiSize maskSize, IppiBorderType borderType, const Ipp<datatype> pBorderValue[4], Ipp8u* pBuffer);
```

Supported values for `mod`:

- 8u_C4R
- 16u_C4R
- 16s_C4R

**Include Files**

ippi.h

**Domain Dependencies**

**Headers:** ippcore.h, ippvm.h, ipps.h

**Libraries:** ippcore.lib, ippvm.lib, ipps.lib

**Parameters**

- `pSrc`  
  Pointer to the source image.

- `srcStep`  
  Distance, in bytes, between the starting points of consecutive lines in the source image.

- `pDst`  
  Pointer to the destination image.

- `dstStep`  
  Distance, in bytes, between the starting points of consecutive lines in the destination image.

- `dstRoiSize`  
  Size of the destination ROI, in pixels.

- `maskSize`  
  Size of the filter mask, in pixels.

- `borderType`  
  Type of border. Possible values are:

  - ippBorderConst  
    Values of all border pixels are set to constant.
ippBorderRepl
ippBorderInMem

Border is replicated from the edge pixels.
Border is obtained from the source image pixels in memory.

borderValue

Constant value to assign to pixels of the constant border. This parameter is applicable only to the ippBorderConst border type.

pBorderValue[3], pBorderValue[4]

Pointers to the constant value to assign to pixels of the constant border. This parameter is applicable only to the ippBorderConst border type.

pBuffer

Pointer to the work buffer.

Description

This function operates with ROI.

Before using this function, you need to compute the size of the work buffer pBuffer using the ippiFilterMedianBorderGetBufferSize function.

The ippiFilterMedianBorder function applies a median filter to an image ROI. The anchor cell is the center of the filter kernel. The size of the source image ROI is equal to the size of the destination image ROI dstRoiSize.

This function sets each pixel in the destination buffer as the median value of all source pixels values from the neighborhood of the processed pixel.

This function removes noise and does not cut out signal brightness drops, as an averaging filter does.

Return Values

ippStsNoErr

Indicates no error.
ippStsNullPtrErr

Indicates an error when pSrc, pDst, pBuffer is NULL.
ippStsSizeErr

Indicates an error if dstRoiSize has a field with a zero or negative value.
ippStsMaskSizeErr

Indicates an error if maskSize has a field with a zero, negative, or even value.
ippStsNotEvenStepErr

Indicates an error if one of the step values is not divisible by 4 for floating-point images, or by 2 for short-integer images.
ippStsBorderErr

Indicates an error when borderType has an illegal value.

Example

To better understand usage of this function, refer to the following example in the examples archive available for download from https://software.intel.com/en-us/ipp-manual-examples:
FilterMedianBorder.c

See Also

Regions of Interest in Intel IPP
User-defined Border Types
FilterMedianBorderGetBufferSize Computes the size of the work buffer for the FilterMedianBorder function.
**FilterMedianGetBufferSize**

*Computes the size of the external buffer for ippiFilterMedian function.*

**Syntax**

IppStatus ippiFilterMedianGetBufferSize_32f(IppiSize dstRoiSize, IppiSize maskSize, Ipp32u nChannels, Ipp32u* pBufferSize);

IppStatus ippiFilterMedianGetBufferSize_64f(IppiSize dstRoiSize, IppiSize maskSize, Ipp32u nChannels, Ipp32u* pBufferSize);

**Include Files**

ippi.h

**Domain Dependencies**

**Headers:** ippcore.h, ippvm.h, ipps.h

**Libraries:** ippcore.lib, ippvm.lib, ipps.lib

**Parameters**

- **dstRoiSize**
  - Size of the destination ROI in pixels.

- **maskSize**
  - Size of the mask in pixels.

- **nChannels**
  - Number of channels in the image.

- **pBufferSize**
  - Pointer to the computed value of the external buffer size.

**Description**

This function computes the size in bytes of an external memory buffer that is required for the ippiFilterMedian function, and stores the result in the pBufferSize.

**Return Values**

- **ippStsNoErr**
  - Indicates no error. Any other value indicates an error.

- **ippStsNullPtrErr**
  - Indicates an error if the pBufferSize pointer is NULL.

- **ippStsSizeErr**
  - Indicates an error if one of the fields of dstRoiSize has a zero or negative value.

- **ippStsMaskSizeErr**
  - Indicates an error if one of the fields of maskSize has a value less than or equal to 1.

- **ippStsNumChannelsErr**
  - Indicates an error if nChannels is not equal to 1, 3, or 4.

**FilterMedian**

*Filters an image using a median filter.*
Syntax

IppStatus ippiFilterMedian_<mod>(const Ipp<datatype> * pSrc, int srcStep, Ipp<datatype> * pDst, int dstStep, IppSize dstRoiSize, IppSize maskSize, IppPoint anchor, Ipp8u* pBuffer);

Supported values for mod:

32f_C3R
32f_C4R
64f_C1R

Include Files
ippi.h

Domain Dependencies
Headers: ippcore.h, ippvm.h, ipps.h
Libraries: ippcore.lib, ippvm.lib, ipps.lib

Parameters

pSrc
Pointer to the source image ROI.

srcStep
Distance in bytes between starts of consecutive lines in the source image.

pDst
Pointer to the destination image ROI.

dstStep
Distance in bytes between starts of consecutive lines in the destination image.

dstRoiSize
Size of the source and destination ROI in pixels.

maskSize
Size of the mask in pixels.

anchor
Anchor cell specifying the mask alignment with respect to the position of the input pixel.

pBuffer
Pointer to the external work buffer.

Description

This function operates with ROI (see Regions of Interest in Intel IPP).

This function sets each pixel in the output buffer as the median value of all the input pixel values taken in the neighborhood of the processed pixel. To ensure valid operation when image boundary pixels are processed, the application should correctly define additional border pixels (see Borders in Neighborhood Operations). The anchor cell is specified by its coordinates anchor.x and anchor.y in the coordinate system associated with the top left corner of the kernel. The size of the source image ROI is equal to the size of the destination image ROI dstRoiSize.

Some flavors of the function require the external buffer pBuffer. Prior to using this functions, compute the size of the external buffer by using the function FilterMedianGetBufferSize.

Return Values

ippStsNoErr
Indicates no error. Any other value indicates an error or a warning.
FilterMedianCross
Filters an image using a cross median filter.

Syntax
IppStatus ippiFilterMedianCross_<mod>(const Ipp<datatype>* pSrc, int srcStep, Ipp<datatype>* pDst, int dstStep, IppiSize dstRoiSize, IppiMaskSize mask);

Supported values for mod:
8u_C1R  16u_C1R  16s_C1R
8u_C3R  16u_C3R  16s_C3R
8u_AC4R 16u_AC4R  16s_AC4R

Include Files
ippi.h

Domain Dependencies
Headers: ippcore.h, ippvm.h, ipps.h
Libraries: ippcore.lib, ippvm.lib, ipps.lib

Parameters
pSrc
Pointer to the source image ROI.

srcStep
Distance in bytes between starts of consecutive lines in the source image.

pDst
Pointer to the destination image ROI.

dstStep
Distance in bytes between starts of consecutive lines in the destination image.

dstRoiSize
Size of the source and destination ROI in pixels.

mask
Predefined mask of the IppiMaskSize type.

Description
This function operates with ROI (see Regions of Interest in Intel IPP).
This function sets each pixel in the output buffer as the median value of all the input pixel values taken in the neighborhood of the processed pixel. The neighborhood is determined by the square mask of the predefined size, which can be either ippMaskSize3x3 or ippMaskSize5x5 (see Table "Median Filter Mask, Neighborhood, and Anchor Cell"). The function operates on the assumption that the pixels outside the source image ROI exist along the distance equal to half of the mask size. It means that the application program should provide...
appropriate values for the pSrc and dstRoiSize arguments, or define additional border pixels (see Borders in Neighborhood Operations). The size of the source image ROI is equal to the size of the destination image ROI dstRoiSize.

**Return Values**

- **ippStsNoErr**
  Indicates no error. Any other value indicates an error or a warning.
- **ippStsNullPtrErr**
  Indicates an error if pSrc or pDst is NULL.
- **ippStsSizeErr**
  Indicates an error if dstRoiSize has a field with a zero or negative value.
- **ippStsStepErr**
  Indicates an error if srcStep or dstStep has a zero or negative value.
- **ippStsMaskSizeErr**
  Indicates an error if mask has an illegal value.

**FilterMedianWeightedCenter3x3**

*Filters an image using a median filter with a weighted center pixel.*

**Syntax**

```c
IppStatus ippiFilterMedianWeightedCenter3x3_8u_C1R(const Ipp8u* pSrc, int srcStep, Ipp8u* pDst, int dstStep, IppiSize dstRoiSize, int weight);
```

**Include Files**

ippi.h

**Domain Dependencies**

**Headers:** ippcore.h, ippvm.h, ipps.h

**Libraries:** ippcore.lib, ippvm.lib, ipps.lib

**Parameters**

- **pSrc**
  Pointer to the source image ROI.
- **srcStep**
  Distance in bytes between starts of consecutive lines in the source image.
- **pDst**
  Pointer to the destination image ROI.
- **dstStep**
  Distance in bytes between starts of consecutive lines in the destination image.
- **dstRoiSize**
  Size of the source and destination ROI in pixels.
- **weight**
  Weight of the pixel, must be an odd number.

**Description**

This function operates with ROI (see Regions of Interest in Intel IPP).

This function sets each pixel in the destination image as the median value of all the input pixel values taken in the neighborhood of the processed pixel. The neighborhood is determined by the fixed square mask of the 3x3 size with the anchor cell as the center cell of the mask. The parameter weight specifies the weight of
the processed pixel, that is how many times its value is included into calculations. The value of this parameter should be odd. If it is even, the function changes its value to the nearest less odd number and returns the warning message.

The function operates on the assumption that the pixels outside of the source image ROI exist along the distance equal to half of the mask size. It means that the application program should provide appropriate values for the pSrc and dstRoiSize arguments, or define additional border pixels (see Borders in Neighborhood Operations). The size of the source image ROI is equal to the size of the destination image ROI dstRoiSize.

**Return Values**

- **ippStsNoErr**: Indicates no error. Any other value indicates an error or a warning.
- **ippStsNullPtrErr**: Indicates an error if pSrc or pDst is NULL.
- **ippStsSizeErr**: Indicates an error if dstRoiSize has a field with a zero or negative value.
- **ippStsStepErr**: Indicates an error if srcStep or dstStep has a zero or negative value.
- **ippStsWeightErr**: Indicates an error if weight is less than or equal to 0.
- **ippStsEvenMedianWeight**: Indicates a warning if weight has an even value.

**FilterMedianColor**

*Filters an image using a color median filter.*

**Syntax**

```
IppStatus ippiFilterMedianColor_<mod>(const Ipp<datatype>* pSrc, int srcStep, Ipp<datatype>* pDst, int dstStep, IppiSize dstRoiSize, IppiMaskSize mask);
```

Supported values for mod:

- 8u_C3R
- 16s_C3R
- 32f_C3R
- 8u_AC4R
- 16s_AC4R
- 32f_AC4R

**Include Files**

`ippi.h`

**Domain Dependencies**

**Headers**: ippcore.h, ippvm.h, ipps.h

**Libraries**: ippcore.lib, ippvm.lib, ipps.lib

**Parameters**

- **pSrc**: Pointer to the source image ROI.
- **srcStep**: Distance in bytes between starts of consecutive lines in the source image.
- **pDst**: Pointer to the destination image ROI.
- **dstStep**: Distance in bytes between starts of consecutive lines in the destination image.
dstRoiSize  
Size of the source and destination ROI in pixels.

mask  
Predefined mask of the IppiMaskSize type.

**Description**

This function operates with ROI (see Regions of Interest in Intel IPP).

When applied to a color image, the previously described median filtering functions process color planes of an image separately, and as a result any correlation between color components is lost. If you want to preserve this information, use the `ippiFilterMedianColor` function instead. For each input pixel, this function computes differences between red (R), green (G), and blue (B) color components of pixels in the mask neighborhood and the input pixel. The distance between the input pixel \( i \) and the neighborhood pixel \( j \) is formed as the sum of absolute values:

\[
\text{abs}(R(i)-R(j)) + \text{abs}(G(i)-G(j)) + \text{abs}(B(i)-B(j))
\]

After scanning the entire neighborhood, the function sets the output value for pixel \( i \) as the value of the neighborhood pixel with the smallest distance to \( i \).

The function `ippiFilterMedianColor` supports square masks of size either `ippMskSize3x3` or `ippMskSize5x5` and processes color images only. To ensure valid operation when image boundary pixels are processed, the application should correctly define additional border pixels (see Borders in Neighborhood Operations).

**Return Values**

- `ippStsNoErr`  
  Indicates no error. Any other value indicates an error or a warning.

- `ippStsNullPtrErr`  
  Indicates an error if `pSrc` or `pDst` is NULL.

- `ippStsSizeErr`  
  Indicates an error if `dstRoiSize` has a field with a zero or negative value.

- `ippStsStepErr`  
  Indicates an error if `srcStep` or `dstStep` has a zero or negative value.

- `ippStsMaskSizeErr`  
  Indicates an error if `mask` has an illegal value.

**General Linear Filters**

These functions use a general rectangular kernel to filter an image. The kernel is a matrix of signed integers or single-precision real values. For each input pixel, the kernel is placed on the image in such a way that the fixed anchor cell within the kernel coincides with the input pixel. The anchor cell is usually a geometric center of the kernel, but can be skewed with respect to the geometric center.

A pointer to an array of kernel values is passed to filtering functions. These values are read in row-major order starting from the top left corner. This array must exactly have `kernelSize.width * kernelSize.height` entries. The anchor cell is specified by its coordinates `anchor.x` and `anchor.y` in the coordinate system associated with the lower right corner of the kernel.

The output value is computed as a sum of neighbor pixels values, with kernel matrix elements used as weight factors. Summation formulas implement a convolution operation, which means that kernel coefficients are used in direct order.

**NOTE**

In Intel IPP 8.2 and lower versions, kernel coefficients are used in inverse order.
Optionally, the output pixel values may be scaled. To ensure valid operation when image boundary pixels are processed, the application should correctly define additional border pixels (see Borders in Neighborhood Operations).

**FilterBorderGetSize**

*Computes the size of the filter specification structure and the size of the work buffer.*

**Syntax**

```c
IppStatus ippiFilterBorderGetSize (IppiSize kernelSize, IppiSize dstRoiSize, IppDataType dataType, IppDataType kernelType, int numChannels, int* pSpecSize, int* pBufferSize);
```

**Include Files**

ippi.h

**Domain Dependencies**

Headers: ippcore.h, ippvm.h, ipps.h

Libraries: ippcore.lib, ippvm.lib, ipps.lib

**Parameters**

- `kernelSize`: Size of the rectangular kernel in pixels.
- `dstRoiSize`: Maximal size of the destination image ROI (in pixels).
- `dataType`: Data type of the source image. Possible values are ipp8u, ipp16u, ipp16s, ipp32f, or ipp64f.
- `kernelType`: Data type of the filter kernel. Possible values are ipp16s, ipp32f, or ipp64f.
- `numChannels`: Number of channels in the image. Possible values are 1, 3, or 4.
- `pSpecSize`: Pointer to the size of the filter specification structure.
- `pBufferSize`: Pointer to the size of the work buffer required for filtering.

**Description**

This function operates with ROI.

This function computes the size of the filter specification structure `pSpec` and the size of the buffer required for filtering operations. Call this function before using the `ippiFilterBorderInit` function.

**Return Values**

- `ippStsNoErr`: Indicates no error.
- `ippStsNullPtrErr`: Indicates an error when one of the specified pointers is NULL.
- `ippStsSizeErr`: Indicates an error when `kernelSize` has a field with a zero or negative value, or if `dstRoiSize` is less than 1.
- `ippStsDataTypeErr`: Indicates an error when `dataType` or `kernelType` has an illegal value.
Indicates an error when numChannels has an illegal value.

See Also
Regions of Interest in Intel IPP
FilterBorderInit Initializes the filter specification structure.
FilterBorder Filters an image using a rectangular filter.

FilterBorderInit

Initializes the filter specification structure.

Syntax

\[
\text{IppStatus ippiFilterBorderInit}_{16s}(\text{const Ipp16s* } pKernel, \text{ IppSize kernelSize}, \text{ int divisor, IppDataType dataType, int numChannels, IppRoundMode roundMode, IppiFilterBorderSpec* pSpec});
\]

\[
\text{IppStatus ippiFilterBorderInit}_{32f}(\text{const Ipp32f* } pKernel, \text{ IppSize kernelSize}, \text{ IppDataType dataType, int numChannels, IppRoundMode roundMode, IppiFilterBorderSpec* pSpec});
\]

\[
\text{IppStatus ippiFilterBorderInit}_{64f}(\text{const Ipp64f* } pKernel, \text{ IppSize kernelSize}, \text{ IppDataType dataType, int numChannels, IppRoundMode roundMode, IppiFilterBorderSpec* pSpec});
\]

Include Files

ippi.h

Domain Dependencies

Headers: ippcore.h, ippvm.h, ipps.h
Libraries: ippcore.lib, ippvm.lib, ipps.lib

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>pKernel</td>
<td>Pointer to the kernel values.</td>
</tr>
<tr>
<td>kernelSize</td>
<td>Size of the rectangular kernel in pixels.</td>
</tr>
<tr>
<td>divisor</td>
<td>Integer value by which the computed result is divided.</td>
</tr>
<tr>
<td>dataType</td>
<td>Data type of the source image. Possible values are ipp8u, ipp16u, ipp16s, ipp32f, or ipp64f.</td>
</tr>
<tr>
<td>numChannels</td>
<td>Number of channels in the image. Possible values are 1, 3, or 4.</td>
</tr>
<tr>
<td>roundMode</td>
<td>Rounding mode, possible values: ippRndZero, ippRndNear</td>
</tr>
</tbody>
</table>

Floating-point values are truncated to zero.

Floating-point values are rounded to the nearest even integer when the fractional part equals to 0.5; otherwise they are rounded to the nearest integer.
Floating-point values are rounded down to the nearest integer when the fractional part is less than 0.5, or rounded up to the nearest integer if the fractional part is equal or greater than 0.5.

The result of calculations is accurate. This mode is supported only when:

- `dataType` is equal to `8u` and `numChannels` is equal to 1, 3, or 4
- `dataType` is equal to `16s` and `numChannels` is equal to 1

`pSpec` Pointer to the filter specification structure.

**Description**

This function initializes the filter specification structure `pSpec` in the external buffer. Before using this function, you need to compute the size of the specification structure using the `FilterBorderGetSize` function. This structure is used by the `FilterBorder` function that performs filtering operations on the source image pixels.

**Return Values**

- `ippStsNoErr` Indicates no error.
- `ippStsNullPtrErr` Indicates an error when one of the specified pointers is `NULL`.
- `ippStsSizeErr` Indicates an error when `kernelSize` has a field with a zero or negative value.
- `ippStsDataTypeErr` Indicates an error when `dataType` has an illegal value.
- `ippStsRoundModeNotSupportedErr` Indicates an error when the specified rounding mode is not supported.
- `ippStsAccurateModeNotSupported` Indicates a warning when the `ippRndHintAccurate` mode is not supported.

**See Also**

- `FilterBorderGetSize` Computes the size of the filter specification structure and the size of the work buffer.
- `FilterBorder` Filters an image using a rectangular filter.

**FilterBorder**

*Filters an image using a rectangular filter.*

**Syntax**

**Case 1: Operation on one-channel data**

```c
IppStatus ippiFilterBorder_<mod>(const Ipp<datatype>* pSrc, int srcStep, Ipp<datatype>* pDst, int dstStep, IppiSize dstRoiSize, IppiBorderType border, const Ipp<datatype> borderValue[1], const IppiFilterBorderSpec* pSpec, Ipp8u* pBuffer);
```

Supported values for `mod`:

- `8u_C1R`
Case 2: Operation on multi-channel data

IppStatus ippiFilterBorder_<mod>(const Ipp<datatype>* pSrc, int srcStep, Ipp<datatype>* pDst, int dstStep, IppSize dstRoiSize, IppiBorderType border, const Ipp<datatype> borderValue[3], const IppiFilterBorderSpec* pSpec, Ipp8u* pBuffer);

Supported values for mod:

8u_C3R
16u_C3R
16s_C3R
32f_C3R

IppStatus ippiFilterBorder_<mod>(const Ipp<datatype>* pSrc, int srcStep, Ipp<datatype>* pDst, int dstStep, IppSize dstRoiSize, IppiBorderType border, const Ipp<datatype> borderValue[4], const IppiFilterBorderSpec* pSpec, Ipp8u* pBuffer);

Supported values for mod:

8u_C4R
16u_C4R
16s_C4R
32f_C4R

Include Files

ippi.h

Domain Dependencies

Headers: ippcore.h, ippvm.h, ipps.h
Libraries: ippcore.lib, ippvm.lib, ipps.lib

Parameters

\begin{itemize}
\item **pSrc** \hspace{1cm} Pointer to the source image ROI.
\item **srcStep** \hspace{1cm} Distance, in bytes, between the starting points of consecutive lines in the source image.
\item **pDst** \hspace{1cm} Pointer to the destination image ROI.
\item **dstStep** \hspace{1cm} Distance, in bytes, between starting points of consecutive lines in the destination image.
\item **dstRoiSize** \hspace{1cm} Size of the source and destination image ROI in pixels.
\item **border** \hspace{1cm} Type of border. Possible values are:
\begin{itemize}
\item **ippBorderConst** \hspace{1cm} Values of all border pixels are set to constant.
\item **ippBorderRepl** \hspace{1cm} Border is replicated from the edge pixels.
\end{itemize}
\end{itemize}
**ippBorderInMem**  
Border is obtained from the source image pixels in memory.

Mixed borders are also supported. They can be obtained by the bitwise operation OR between the ippBorderRepl or ippBorderConst values and ippBorderInMemTop, ippBorderInMemBottom, ippBorderInMemLeft, ippBorderInMemRight.

**borderValue**  
Constant value to assign to pixels of the constant border. This parameter is applicable only to the ippBorderConst border type.

**pSpec**  
Pointer to the filter specification structure.

**pBuffer**  
Pointer to the work buffer for filtering operations.

**Description**
Before using this function, you need to initialize the filter specification structure using the ippiFilterBorderInit function.

This function operates with ROI.

This function performs filtering of a rectangular ROI inside a two-dimensional image using a specified structure pSpec. Type of the image border is defined by the value of the border parameter.

To change the function behavior (add offset to the result or set the rounding mode), use ippiFilterBorderSetMode after the ippiFilterBorderInit function.

**Return Values**

- ippStsNoErr: Indicates no error.
- ippStsNullPtrErr: Indicates an error when one of the specified pointers is NULL.
- ippStsSizeErr: Indicates an error when dstRoiSize has a field with a zero or negative value, or if dstRoiSize.width is more than the maximum ROI roiWidth passed to the initialization function.
- ippStsStepErr: Indicates an error when the srcStep value is less than, or equal to zero.
- ippStsBorderErr: Indicates an error when border has an illegal value.

**Example**
To better understand usage of this function, refer to the following example in the examples archive available for download from https://software.intel.com/en-us/ipp-manual-examples:

FilterBorder.c

**See Also**
- Regions of Interest in Intel IPP
- User-defined Border Types
- FilterBorderInit: Initializes the filter specification structure.
- FilterBorderSetMode: Adds the offset value after filtering operation for ipp8u and ipp16u data types, and sets the rounding mode.
FilterBorderSetMode

Adds the offset value after filtering operation for ipp8u and ipp16u data types, and sets the rounding mode.

Syntax

IppStatus ippiFilterBorderSetMode(IppHintAlgorithm hint, int offset, IppiFilterBorderSpec* pSpec);

Include Files
ippi.h

Domain Dependencies
Headers: ippcore.h, ippvm.h, ipps.h
Libraries: ippcore.lib, ippvm.lib, ipps.lib

Parameters

hint
Suggests using specific code for rounding. Supported values:
ippAlgHintNone, ippAlgHintFast

Default modes. The function performs rounding in accordance with the roundMode parameter passed to the Init function, but function performance takes precedence over accuracy and some output pixels can differ by +-1 from the exact result.

ippAlgHintAccurate
All output pixels are exact; accuracy takes precedence over performance.

offset
Constant that is added to the final signed result before converting it to unsigned for ipp8u and ipp16u data types.

pSpec
Pointer to the initialized filter specification structure.

Description

This function adds the offset value after filtering operation for ipp8u and ipp16u data types with the ippiFilterBorder function:

pDst=(summ(src[i]*kern[i]))+offset

You can also use this function to set the rounding mode for the filtering result.

The 8u_C1R, 8u_C3R, 8u_C4R, and 16s_C1RFilterBorder function flavors initialized with the ipp16s coefficients support ippAlgHintNone and ippAlgHintAccurate rounding modes.

Use this function after the ippiFilterBorderInit function and before calling ippiFilterBorder.

Return Values

ippStsNoErr
Indicates no error.

ippStsNullPtrErr
Indicates an error when one of the specified pointers is NULL.
The offset value is not supported (for ipp16s and ipp32f data types).
The accurate mode is not supported for some data types. The result of rounding may be not exact.

Example
To better understand usage of this function, refer to the following example in the examples archive available for download from https://software.intel.com/en-us/ipp-manual-examples:
FilterBorderSetMode.c

See Also
FilterBorder Filters an image using a rectangular filter.
FilterBorderInit Initializes the filter specification structure.

FilterGetBufSize
Computes the size of the work buffer.

Syntax
IppStatus ippiFilterGetBufSize_64f_C1R(IppiSize kernelSize, int roiWidth, int* pSize);

Include Files
ippi.h

Domain Dependencies
Headers: ippcore.h, ippvm.h, ipps.h
Libraries: ippcore.lib, ippvm.lib, ipps.lib

Parameters
kernelSize  Size of the rectangular kernel in pixels.
roiWidth  Width of the image ROI in pixels.
pSize  Pointer to the size of the work buffer.

Description
This function operates with ROI (see Regions of Interest in Intel IPP).
This function computes the size of the work buffer pSize that is required for the function ippiFilter (flavor that operates on data of the Ipp64f type).

Return Values
ippStsNoErr  Indicates no error. Any other value indicates an error.
ippStsNullPtrErr  Indicates an error if pSize is NULL.
ippStsSizeErr  Indicates an error if kernelSize has a field with a zero or negative value, or roiWidth is less than or equal to zero.
Filter
Filters an image using a general rectangular kernel.

Syntax

IppStatus ippiFilter_64f_C1R(const Ipp64f* pSrc, int srcStep, Ipp64f* pDst, int dstStep, IppSize dstRoiSize, const Ipp64f* pKernel, IppSize kernelSize, IppiPoint anchor, Ipp8u* pBuffer);

Include Files
ippi.h

Domain Dependencies
Headers: ippcore.h, ippvm.h, ipps.h
Libraries: ippcore.lib, ippvm.lib, ipps.lib

Parameters

pSrc
Pointer to the source image ROI.

srcStep
Distance, in bytes, between the starting points of consecutive lines in the source image.

pDst
Pointer to the destination image ROI.

dstStep
Distance, in bytes, between the starting points of consecutive lines in the destination image.

dstRoiSize
Size of the source and destination ROI in pixels.

pKernel
Pointer to the kernel values.

kernelSize
Size of the rectangular kernel in pixels.

anchor
Anchor cell specifying the rectangular kernel alignment with respect to the position of the input pixel.

pBuffer
Pointer to the work buffer.

Description

This function operates with ROI.

The *ippiFilter* function uses the general rectangular kernel of *kernelSize* size to filter an image ROI. This function sums the products of the kernel coefficients *pKernel* and pixel values taken over the source pixel neighborhood defined by *kernelSize* and *anchor*. The anchor cell is specified by its coordinates *anchor.x* and *anchor.y* in the coordinate system associated with the bottom right corner of the kernel.

Kernel coefficients are used in inverse order. The sum is written to the destination pixel.

To ensure valid operation when image boundary pixels are processed, the application should correctly define additional border pixels (see *Borders in Neighborhood Operations*).

This function requires a temporary work buffer. Before using this function flavor, you need to compute the buffer size using the *ippiFilterGetBufSize* function.

Return Values

ippStsNoErr
Indicates no error. Any other value indicates an error.
Indicates an error when \( pSrc, pDst, \) or \( pKernel \) is NULL.

Indicates an error when \( \text{dstRoiSize} \) or \( \text{kernelSize} \) has a field with a zero or negative value.

Indicates an error when the \( \text{divisor} \) has a zero value.

Indicates an error when:
- \( \text{srcStep} \) is less than \( (\text{roiSize}.\text{width} + \text{kernelSize}.\text{width}) \times \text{sizeof(Ipp64f)} \)
- \( \text{dstStep} \) is less than \( \text{roiSize}.\text{width} \times \text{sizeof(Ipp64f)} \)

See Also
Regions of Interest in Intel IPP
Borders in Neighborhood Operations
FilterGetBufSize Computes the size of the work buffer.

Separable Filters
Separable filters use a spatial kernel consisting of a single column (as in the FilterColumn function) or a single row (as in the FilterRow function) to filter the source image.

FilterRowBorderPipelineGetBufferSize, FilterRowBorderPipelineGetBufferSize_Low
Compute the size of working buffer for the strow filter.

Syntax
\[
\text{IppStatus } \text{ippiFilterRowBorderPipelineGetBufferSize}_{<\text{mod}>}(\text{IppSize } \text{roiSize}, \text{int } \text{kernelSize}, \text{int* } \text{pBufferSize});
\]

Supported values for \( \text{mod} \):
- \( 8u16s\_C1R \quad 16s\_C1R \quad 16u\_C1R \quad 32f\_C1R \)
- \( 8u16s\_C3R \quad 16s\_C3R \quad 16u\_C3R \quad 32f\_C3R \)

\[
\text{IppStatus } \text{ippiFilterRowBorderPipelineGetBufferSize\_Low}_{<\text{mod}>}(\text{IppSize } \text{roiSize}, \text{int } \text{kernelSize}, \text{int* } \text{pBufferSize});
\]

Supported values for \( \text{mod} \):
- \( 16s\_C1R \)
- \( 16s\_C3R \)

Include Files
ippcv.h

Domain Dependencies
Headers: ippcore.h, ippvm.h, ipp.h, ippi.h
Libraries: ippcore.lib, ippvm.lib, ipp.lib, ippi.lib
Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>roiSize</td>
<td>Maximum size of the source and destination image ROI.</td>
</tr>
<tr>
<td>kernelSize</td>
<td>Size of the kernel in pixels.</td>
</tr>
<tr>
<td>pBufferSize</td>
<td>Pointer to the computed size of the buffer.</td>
</tr>
</tbody>
</table>

Description

These functions operate with ROI (see Regions of Interest in Intel IPP).

These functions compute the size of the working buffer required for the functions `ippiFilterRowBorderPipeline` and `ippiFilterRowBorderPipeline_Low` respectively. The buffer with the length `pBufferSize[0]` can be used to filter images with width equal to or less than `roiSize`.

Return Values

<table>
<thead>
<tr>
<th>Return Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ippStsNoErr</td>
<td>Indicates no error. Any other value indicates an error or a warning.</td>
</tr>
<tr>
<td>ippStsNullPtrErr</td>
<td>Indicates an error condition if the pointer <code>pBufferSize</code> is NULL.</td>
</tr>
<tr>
<td>ippStsSizeErr</td>
<td>Indicates an error condition if <code>maskSize</code> has a field with a zero or negative value, or if <code>roiWidth</code> is less than 1.</td>
</tr>
</tbody>
</table>

FilterRowBorderPipeline, FilterRowBorderPipeline_Low

Apply the filter with border to image rows.

Syntax

Case 1: Operation on one-channel integer data

```c
IppStatus ippiFilterRowBorderPipeline_<mod>(const Ipp<srcDatatype>* pSrc, int srcStep, Ipp<dstDatatype>** ppDst, IppiSize roiSize, const Ipp<srcDatatype>* pKernel, int kernelSize, int xAnchor, IppiBorderType borderType, Ipp<srcDatatype> borderValue, int divisor, Ipp8u* pBuffer);
```

Supported values for `mod`:

- 8u16s_C1R
- 16s_C1R
- 16u_C1R

```c
IppStatus ippiFilterRowBorderPipeline_Low_16s_C1R(const Ipp16s* pSrc, int srcStep, Ipp16s** ppDst, IppiSize roiSize, const Ipp16s* pKernel, int kernelSize, int xAnchor, IppiBorderType borderType, Ipp16s borderValue, int divisor, Ipp8u* pBuffer);
```

Case 2: Operation on one-channel floating point data

```c
IppStatus ippiFilterRowBorderPipeline_32f_C1R(const Ipp32f* pSrc, int srcStep, Ipp32f** ppDst, IppiSize roiSize, const Ipp32f* pKernel, int kernelSize, int xAnchor, IppiBorderType borderType, Ipp32f borderValue, Ipp8u* pBuffer);
```

```c
IppStatus ippiFilterRowBorderPipeline_Low_32f_C1R(const Ipp32f* pSrc, int srcStep, Ipp32f** ppDst, IppiSize roiSize, const Ipp32f* pKernel, int kernelSize, int xAnchor, IppiBorderType borderType, Ipp32f borderValue, Ipp8u* pBuffer);
```
Case 3: Operation on three-channel integer data

IppStatus ippiFilterRowBorderPipeline_<mod>(const Ipp<srcDatatype>* pSrc, int srcStep, Ipp<dstDatatype>** ppDst, IppiSize roiSize, const Ipp<dstDatatype>* pKernel, int kernelSize, int xAnchor, IppiBorderType borderType, Ipp<srcDatatype> borderValue[3], int divisor, Ipp8u* pBuffer);

Supported values for mod:

  8u16s_C3R 16s_C3R 16u_C3R

IppStatus ippiFilterRowBorderPipeline_Low_16s_C3R(const Ipp16s* pSrc, int srcStep, Ipp16s** ppDst, IppiSize roiSize, const Ipp16s* pKernel, int kernelSize, int xAnchor, IppiBorderType borderType, Ipp16s borderValue[3], int divisor, Ipp8u* pBuffer);

Case 4: Operation on three-channel floating point data

IppStatus ippiFilterRowBorderPipeline_32f_C3R(const Ipp32f* pSrc, int srcStep, Ipp32f** ppDst, IppiSize roiSize, const Ipp32f* pKernel, int kernelSize, int xAnchor, IppiBorderType borderType, Ipp32f borderValue[3], Ipp8u* pBuffer);

Include Files

ippcv.h

Domain Dependencies

Headers: ippcore.h, ippvm.h, ipps.h,ippi.h
Libraries: ippcore.lib, ippvm.lib, ipps.lib,ippi.lib

Parameters

\n\texttt{pSrc} \quad \text{Pointer to the source image ROI.}

\texttt{srcStep} \quad \text{Distance in bytes between starts of consecutive lines in the source image.}

\texttt{ppDst} \quad \text{Double pointer to the destination image ROI.}

\texttt{roiSize} \quad \text{Size of the source and destination ROI in pixels.}

\texttt{pKernel} \quad \text{Pointer to the row kernel values.}

\texttt{kernelSize} \quad \text{Size of the kernel in pixels.}

\texttt{xAnchor} \quad \text{Anchor value specifying the kernel row alignment with respect to the position of the input pixel.}

\texttt{borderType} \quad \text{Type of border (see Borders in Neighborhood Operations); following values are possible:}

\texttt{ippBorderZero} \quad \text{Values of all border pixel are set to zero.}

\texttt{ippBorderConst} \quad \text{Values of all border pixels are set to constant.}

\texttt{ippBorderRepl} \quad \text{Replicated border is used.}

\texttt{ippBorderWrap} \quad \text{Wrapped border is used}

\texttt{ippBorderMirror} \quad \text{Mirrored border is used}

\texttt{ippBorderMirrorR} \quad \text{Mirrored border with replication is used}
The constant value (constant vector in case of three-channel data) to assign to the pixels in the constant border (not applicable for other border's type).

Value by which the computed result is divided (for operations on integer data only).

Pointer to the working buffer.

Description
These functions operate with ROI (see Regions Regions of Interest in Intel IPP ).

The function ippiFilterRowBorderPipeline_Low performs calculation exclusively with the 16s-data, and the input data must be in the range ensuring that the overflow does not occur during calculation and the result can be represented by a 32-bit integer number.

These functions apply the horizontal row filter of the separable convolution kernel to the source image pSrc. The filter coefficients are placed in the reversed order. For integer data:

\[
ppDst[i][j] = \frac{1}{\text{divisor}} \sum_{k=0}^{\text{kernelSize}-1} pSrc[i,j+k-xAnchor] \cdot pKernel[k]
\]

and for floating point data:

\[
ppDst[i][j] = \sum_{k=0}^{\text{kernelSize}-1} pSrc[i,j+k-xAnchor] \cdot pKernel[k]
\]

Here \( j = 0, \ldots \text{roiSize.width} - 1, i = 0, \ldots \text{roiSize.height} - 1. \)

The values of pixels of the source image that lies outside of the image ROI (that is, if for pixel \( pSrc[i,1] \) \( l \notin [0, \text{roiSize.width}-1] \) are set in accordance with the specified parameters borderType and borderValue.

This function can be used to organize the separable convolution as a step of the image processing pipeline.

The functions requires the external buffer pBuffer, its size should be previously computed by the functions ippiFilterRowBorderPipelineGetBufferSize and ippiFilterRowBorderPipelineGetBufferSize_Low respectively.

Return Values
ippStsNoErr Indicates no error. Any other value indicates an error.
ippStsNullPtrErr Indicates an error condition if one of the specified pointers is NULL.
ippStsSizeErr Indicates an error condition if roiSize has a field with a zero or negative value.
ippStsStepErr Indicates an error condition if srcStep or dstStep is less than roiSize.width * <pixelSize>
ippStsNotEvenStepErr Indicates an error condition if one of the step values is not divisible by 4 for floating-point images, or by 2 for short-integer images.
ippStsAnchorErr Indicates an error condition if xAnchor has a wrong value.
ippStsBorderErr Indicates an error condition if borderType has a wrong value.
ippStsBadArgErr Indicates an error condition if divisor is equal to 0.
FilterColumnPipelineGetBufferSize, FilterColumnPipelineGetBufferSize_Low

Compute the size of working buffer for the column filter.

**Syntax**

IppStatus ippiFilterColumnPipelineGetBufferSize_<mod>(IppiSize roiSize, int kernelSize, int* pBufferSize);

Supported values for `mod`:

- 16s_C1R
- 16u_C1R
- 16s8u_C1R
- 16s8s_C1R
- 32f_C1R
- 16s_C3R
- 16u_C3R
- 16s8u_C3R
- 16s8s_C3R
- 32f_C3R

IppStatus ippiFilterColumnPipelineGetBufferSize_Low_<mod>(IppiSize roiSize, int kernelSize, int* pBufferSize);

Supported values for `mod`:

- 16s_C1R
- 16s_C3R

**Include Files**

ippcv.h

**Domain Dependencies**

Headers: ippcore.h, ippvm.h, ipps.h,ippi.h

Libraries: ippcore.lib, ippvm.lib, ipps.lib,ippi.lib

**Parameters**

- `roiSize` Maximum size of the source and destination image ROI.
- `kernelSize` Size of the kernel in pixels.
- `pBufferSize` Pointer to the computed size of the buffer.

**Description**

These functions operate with ROI (see Regions of Interest in Intel IPP).

These functions compute the size of the working buffer required for the functions `ippiFilterColumnPipeline` and `ippiFilterColumnPipeline_Low` respectively. The buffer with the length `pBufferSize[0]` can be used to filter images with width equal to or less than `roiSize`.

**Return Values**

- `ippStsNoErr` Indicates no error. Any other value indicates an error or a warning.
- `ippStsNullPtrErr` Indicates an error condition if the pointer `pBufferSize` is NULL.
- `ippStsSizeErr` Indicates an error condition if `maskSize` has a field with a zero or negative value, or if `roiWidth` is less than 1.
FilterColumnPipeline, FilterColumnPipeline_Low

Apply the filter to image columns.

Syntax

Case 1: Operation on integer data

IppStatus ippiFilterColumnPipeline_<mod>(const Ipp<srcDatatype>** ppSrc, Ipp<dstDatatype>* pDst, int dstStep, IppiSize roiSize, const Ipp<srcDatatype>* pKernel, int kernelSize, int divisor, Ipp8u* pBuffer);

Supported values for mod:

16s_C1R    16s8u_C1R    16s8s_C1R    16u_C1R
16s_C3R    16s8u_C3R    16s8s_C3R    16u_C3R

IppStatus ippiFilterColumnPipeline_Low_16s_C1R(const Ipp16s** ppSrc, Ipp16s* pDst, int dstStep, IppiSize roiSize, const Ipp16s* pKernel, int kernelSize, int divisor, Ipp8u* pBuffer);

IppStatus ippiFilterColumnPipeline_Low_16s_C3R(const Ipp16s** ppSrc, Ipp16s* pDst, int dstStep, IppiSize roiSize, const Ipp16s* pKernel, int kernelSize, int divisor, Ipp8u* pBuffer);

Case 2: Operation on floating-point data

IppStatus ippiFilterColumnPipeline_<mod>(const Ipp<datatype>** ppSrc, Ipp<datatype>* pDst, int dstStep, IppiSize roiSize, const Ipp<datatype>* pKernel, int kernelSize, Ipp8u* pBuffer);

Supported values for mod:

32f_C1R
32f_C3R

Include Files

ippcv.h

Domain Dependencies

Headers: ippcore.h, ippvm.h, ipps.h,ippi.h
Libraries: ippcore.lib, ippvm.lib, ipps.lib,ippi.lib

Parameters

ppSrc
   Double pointer to the source image ROI.

pDst
   Pointer to the destination image ROI.

dstStep
   Distance in bytes between starts of consecutive lines in the destination image.

roiSize
   Size of the destination ROI in pixels.

pKernel
   Pointer to the strow kernel values.

kernelSize
   Size of the kernel in pixels.
Divisor

Value by which the computed result is divided (for operations on integer data only).

pBuffer

Pointer to the working buffer.

Description

This function operates with ROI (see Regions of Interest in Intel IPP).

The function ippiFilterColumnPipeline_Low performs calculation exclusively with the 16s-data, and the input data must be in the range ensuring that the overflow does not occur during calculation and the result can be represented by a 32-bit integer number.

These functions apply the column filter of the separable convolution kernel to the source image pSrc. The filter coefficients are placed in the reversed order. For integer data:

\[
P_{Dst}[i,j] = \frac{1}{\text{divisor}} \sum_{k=0}^{\text{kernelSize}-1} p_{Src}[i+k,j] \cdot p_{Kernel}[k]
\]

and for floating point data:

\[
P_{Dst}[i,j] = \sum_{k=0}^{\text{kernelSize}-1} p_{Src}[i+k,j] \cdot p_{Kernel}[k]
\]

Here \( j = 0, \ldots, \text{roiSize}.width-1 \), \( i=0,\ldots, \text{roiSize}.height-1 \).

The size of the source image is

\[
(\text{roiSize}.height + \text{kernelSize} - 1) * \text{roiSize}.width.
\]

The functions requires the external buffer pBuffer, its size should be previously computed by the functions ippiFilterColumnPipelineGetBufferSize and ippiFilterColumnPipelineGetBufferSize_Low respectively.

Return Values

- ippStsNoErr: Indicates no error. Any other value indicates an error.
- ippStsNullPtrErr: Indicates an error condition if one of the specified pointers is NULL.
- ippStsSizeErr: Indicates an error condition if roiSize has a field with a zero or negative value.
- ippStsStepErr: Indicates an error condition if srcStep or dstStep is less than roiSize.width * <pixelSize>.
- ippStsNotEvenStepErr: Indicates an error condition if one of the step values is not divisible by 4 for floating-point images, or by 2 for short-integer images.
- ippStsBadArgErr: Indicates an error condition if divisor is equal to 0.

Example

To better understand usage of this function, refer to the following example in the examples archive available for download from https://software.intel.com/en-us/ipp-manual-examples:

FilterColumnPipeline_Low.c
FilterSeparable
Apply the filter to an image.

Syntax
IppStatus ippiFilterSeparable_<mod>(const Ipp<datatype>* pSrc, IppSizeL srcStep,
Ipp<datatype>* pDst, IppSizeL dstStep, IppSizeL roiSize, IppiBorderType borderType,
Ipp<datatype> borderValue, const IppiFilterSeparableSpec* pSpec, Ipp8u* pBuffer);

Supported values for mod:

8u_C1R_L 8u16s_C1R_L 16s_C1R_L 16u_C1R_L 32f_C1R_L
8u_C3R_L 8u16s_C3R_L 16s_C3R_L 16u_C3R_L 32f_C3R_L
8u_C4R_L 8u16s_C4R_L 16s_C4R_L 16u_C4R_L 32f_C4R_L

Include Files
ippi_cv_l.h

Domain Dependencies
Headers: ippcore.h, ippvm.h, ipps.h, ippi.h
Libraries: ippcore.lib, ippvm.lib, ipps.lib, ippi.lib

Parameters
pSrc
Pointer to the source image ROI.
srcStep
Distance in bytes between starts of consecutive lines in the source image.
pDst
Pointer to the destination image ROI.
dstStep
Distance in bytes between starts of consecutive lines in the destination image.
roiSize
Size of the source and destination ROI in pixels.
borderType
Type of border. Possible values are:

ippBorderConst
Values of all border pixels are set to constant.
ippBorderRept
Replicated border is used.
ippBorderWrap
Wrapped border is used.
ippBorderMirror
Mirrored border is used.
ippBorderMirrorR
Mirrored border with replication is used.
borderValue
The constant value (constant vector in case of three- or four-channel data) to assign to the pixels in the ippBorderConst border type (not applicable for other border types).
pSpec
Pointer to the filter specification structure.
pBuffer
Pointer to the working buffer.

Description
This function operates with ROI.
Before using this function, compute the size of the external buffer \( pBuffer \) using the \textbf{ippiFilterSeparableGetBufferSize} function.

This function applies the horizontal row filter of the separable convolution kernel to the source image \( pSrc \) and the column filter of the separable convolution kernel to the intermediate result.

For integer data:

\[
\text{intermediate}[i][j] = \frac{1}{\text{divisor}} \sum_{k=0}^{\text{rowKernelSize}-1} (pSrc[i, j + k - \text{xAnchor}] \times \text{pRowKernel}[k])
\]

\[
pDst[i, j] = \text{offset} + \frac{1}{\text{divisor}} \sum_{k=0}^{\text{columnKernelSize}-1} (\text{intermediate}[i + k, j] \times \text{pColumnKernel}[k])
\]

and for floating point data:

\[
\text{intermediate}[i][j] = \sum_{k=0}^{\text{rowKernelSize}-1} (pSrc[i, j + k - \text{xAnchor}] \times \text{pRowKernel}[k])
\]

\[
pDst[i, j] = \sum_{k=0}^{\text{columnKernelSize}-1} (\text{intermediate}[i + k, j] \times \text{pColumnKernel}[k])
\]

Here \( j = 0, \ldots \) \( \text{roiSize.width} - 1 \), \( i = 0, \ldots \) \( \text{roiSize.height} - 1 \). The values of pixels of the source image that lies outside of the image ROI (that is, if for pixel \( pSrc[i, l] \) \( l \notin [0, \text{roiSize.width}-1] \) are set in accordance with the specified parameters \text{borderType} \) and \text{borderValue}.

\( \text{xAnchor} = (\text{rowKernelSize} - 1) / 2 \).

**Return Values**

- \textbf{ippStsNoErr} Indicates no error. Any other value indicates an error condition.
- \textbf{ippStsNullPtrErr} Indicates an error condition if one of the specified pointers is \text{NULL}.
- \textbf{ippStsSizeErr} Indicates an error condition if \text{roiSize} has a field with a zero or negative value.
- \textbf{ippStsStepErr} Indicates an error condition if the \text{srcStep} or \text{dstStep} value is less than \( \text{roiSize.width} \times <\text{pixelSize}> \).
- \textbf{ippStsNotEvenStepErr} Indicates an error condition if one of the step values is not divisible by 4 for floating-point images, or by 2 for short-integer images.

**See Also**

- Regions of Interest in Intel IPP
- User-defined Border Types
- \textbf{FilterSeparableInit} Initializes the filter specification structure.
- \textbf{FilterSeparableGetBufferSize} Computes the size of the work buffer.

**FilterSeparableGetBufferSize**

*Computes the size of the work buffer.*
Syntax

IppStatusippiFilterSeparableGetBufferSize_L(IppiSizeL roiSize, IppiSize kernelSize, IppDataType dataType, IppDataType kernelType, int numChannels, IppSizeL* pBufferSize);

Include Files

ippcv_l.h

Domain Dependencies

Headers: ippcore.h, ippvm.h, ipps.h,ippi.h
Libraries: ippcore.lib, ippvm.lib, ipps.lib,ippi.lib

Parameters

type roiSize Size of the image ROI in pixels.
type kernelSize Size of the rectangular kernel in pixels.
type dataType Data type of the source image. Possible values are Ipp8u, Ipp16s, Ipp16u, Ipp32f.
type kernelType Data type of the filter kernel. Possible values are Ipp16s and Ipp32f.
type numChannels Number of channels in the image. Possible values are 1, 3, and 4.
type pBufferSize Pointer to the size of the work buffer required for filtering.

Description

This function computes the size of the buffer required for filtering operations. Call this function before using theippiFilterSeparable function.

Return Values

- ippStsNoErr Indicates no error. Any other value indicates an error.
- ippStsNullPtrErr Indicates an error condition if one of the specified pointers is NULL.
- ippStsSizeErr Indicates an error condition if kernelSize has a field with a zero or negative value.
- ippStsDataTypeErr Indicates an error condition if dataType has an illegal value.
- ippStsChannelErr Indicates an error condition if numChannels has an illegal value.

FilterSeparableGetSpecSize

Computes the size of the filter specification structure.

Syntax

IppStatusippiFilterSeparableGetSpecSize_L(IppiSize kernelSize, IppDataType dataType, int numChannels, int* pSpecSize);

Include Files

ippcv_l.h
Domain Dependencies

Headers: ippcore.h, ippvm.h, ipps.h,ippi.h
Libraries: ippcore.lib, ippvm.lib, ipps.lib,ippi.lib

Parameters

- **kernelSize**: Size of the rectangular kernel in pixels.
- **dataType**: Data type of the source image. Possible values are Ipp8u, Ipp16s, Ipp16u, Ipp32f.
- **numChannels**: Number of channels in the image. Possible values are 1, 3, and 4.
- **pSpecSize**: Pointer to the size of the filter specification structure.

Description

This function computes the size of the filter specification structure **pSpec**. Call this function before using the ippiFilterSeparableInit function.

Return Values

- **ippStsNoErr**: Indicates no error. Any other value indicates an error.
- **ippStsNullPtrErr**: Indicates an error condition if one of the specified pointers is NULL.
- **ippStsSizeErr**: Indicates an error condition if **kernelSize** has a field with a zero or negative value.
- **ippStsDataTypeErr**: Indicates an error condition if **dataType** has an illegal value.
- **ippStsChannelErr**: Indicates an error condition if **numChannels** has an illegal value.

See Also

FilterSeparableInit Initializes the filter specification structure.

FilterSeparableInit

*Initializes the filter specification structure.*

**Syntax**

```c
IppStatus ippiFilterSeparableInit_16s_L(const Ipp16s* pRowKernel, const Ipp16s* pColumnKernel, IppiSize kernelSize, int divisor, int offset, IppDataType dataType, int numChannels, IppiFilterSeparableSpec* pSpec);

IppStatus ippiFilterSeparableInit_32f_L(const Ipp32f* pRowKernel, const Ipp32f* pColumnKernel, IppiSize kernelSize, IppDataType dataType, int numChannels, IppiFilterSeparableSpec* pSpec);
```

Include Files

ippcv_l.h

Domain Dependencies

Headers: ippcore.h, ippvm.h, ipps.h,ippi.h
Libraries: ippcore.lib, ippvm.lib, ipps.lib,ippi.lib
Parameters

- **pRowKernel, pColumnKernel**: Pointer to the row and column kernel values.
- **kernelSize**: Size of the rectangular kernel in pixels.
- **divisor**: Integer value by which the computer result is divided.
- **offset**: Offset value for the destination image.
- **dataType**: Data type of the source image. Possible values are Ipp8u, Ipp16s, Ipp16u, Ipp32f.
- **numChannels**: Number of channels in the image. Possible values are 1, 3, and 4.
- **pSpec**: Pointer to the filter specification structure.

Description

This function initializes the filter specification structure *pSpec* in the external buffer. Before using this function, you need to compute the size of the specification structure using the `FilterSeparableGetSpecSize` function. This structure is used by the `FilterSeparable` function that performs filtering operations on the source image pixels.

Return Values

- **ippStsNoErr**: Indicates no error. Any other value indicates an error.
- **ippStsNullPtrErr**: Indicates an error condition if one of the specified pointers is NULL.
- **ippStsSizeErr**: Indicates an error condition if `kernelSize` has a field with a zero or negative value.
- **ippStsDataTypeErr**: Indicates an error condition if `dataType` has an illegal value.
- **ippStsChannelErr**: Indicates an error condition if `numChannels` has an illegal value.
- **ippStsDivisorErr**: Indicates an error condition if `divisor` has a zero value.

See Also

- **FilterSeparable** Apply the filter to an image.
- **FilterSeparableGetSpecSize** Computes the size of the filter specification structure.

Wiener Filters

Intel IPP functions described in this section perform adaptive noise-removal filtering of an image using Wiener filter [Lim90]. The adaptive filter is more selective than a comparable linear filter in preserving edges and other high frequency parts of an image. Wiener filters are commonly used in image processing applications to remove additive noise from degraded images, to restore a blurry image, and in similar operations.

These functions use a pixel-wise adaptive Wiener method based on statistics estimated from a local neighborhood (mask) of arbitrary size for each pixel.

**FilterWienerGetBufferSize**

*Computes the size of the external buffer for ippiFilterWiener function.*
Syntax

IppStatus ippiFilterWienerGetBufferSize(IppSize dstRoiSize, IppSize maskSize, int channels, int* pBufferSize);

Include Files

ippi.h

Domain Dependencies

Headers: ippcore.h, ippvm.h, ipps.h
Libraries: ippcore.lib, ippvm.lib, ipps.lib

Parameters

- `dstRoiSize`: Size of the destination ROI in pixels.
- `maskSize`: Size of the mask in pixels.
- `channels`: Number of channels in the image.
- `pBufferSize`: Pointer to the computed value of the external buffer size.

Description

This function computes the size in bytes of an external memory buffer that is required for the function `ippiFilterWiener`, and stores the result in the `pBufferSize`.

Return Values

- `ippStsNoErr`: Indicates no error. Any other value indicates an error.
- `ippStsNullPtrErr`: Indicates an error condition if the `pBufferSize` pointer is NULL.
- `ippStsSizeErr`: Indicates an error condition if one of the fields of `dstRoiSize` has a zero or negative value.
- `ippStsMaskSizeErr`: Indicates an error condition if one of the fields of `maskSize` has a value less than or equal to 1.
- `ippStsNumChannelsErr`: Indicates an error condition if `channels` is not 1, 3 or 4.

FilterWiener

Filters an image using the Wiener algorithm.

Syntax

Case 1: Operation on one-channel images

IppStatus ippiFilterWiener_<mod>(const Ipp<datatype>* pSrc, int srcStep, Ipp<datatype>* pDst, int dstStep, IppSize dstRoiSize, IppSize maskSize, IppiPoint anchor, Ipp32f noise[1], Ipp8u* pBuffer);

Supported values for `mod`:

- 8u_C1R
- 16s_C1R
- 32f_C1R
Case 2: Operation on multi-channel images

IppStatus ippiFilterWiener_<mod>(const Ipp<datatype>* pSrc, int srcStep, Ipp<datatype>* pDst, int dstStep, IppSize dstRoiSize, IppSize maskSize, IppiPoint anchor, Ipp32f noise[3], Ipp8u* pBuffer);

Supported values for mod:

8u_C3R  16s_C3R  32f_C3R
8u_AC4R  16s_AC4R  32f_AC4R

IppStatus ippiFilterWiener_<mod>(const Ipp<datatype>* pSrc, int srcStep, Ipp<datatype>* pDst, int dstStep, IppSize dstRoiSize, IppSize maskSize, IppiPoint anchor, Ipp32f noise[4], Ipp8u* pBuffer);

Supported values for mod:

8u_C4R  16s_C4R  32f_C4R

Include Files
ippi.h

Domain Dependencies
Headers: ippcore.h, ippvm.h, ipps.h
Libraries: ippcore.lib, ippvm.lib, ipps.lib

Parameters

pSrc  Pointer to the source image ROI.
srcStep  Distance in bytes between starts of consecutive lines in the source image.
pDst  Pointer to the destination image ROI.
dstStep  Distance in bytes between starts of consecutive lines in the destination image.
dstRoiSize  Size of the source and destination ROI in pixels.
maskSize  Size of the mask in pixels.
anchor  Anchor cell specifying the mask alignment with respect to the position of the input pixel.
noise  Noise level value or array of the noise level values in case of multi-channel image. This value must be in the range [0,1].
pBuffer  Pointer to the external work buffer.

Description
This function operates with ROI (see Regions of Interest in Intel IPP). This function performs adaptive filtering of the image degraded by constant power additive noise. For each pixel of the input image pSrc, the function estimates the local image mean $\mu$ and variance $\sigma$ in the rectangular neighborhood (mask) of size maskSize with the anchor cell anchor centered on the pixel. The anchor cell is specified by its coordinates anchor.x and anchor.y in the coordinate system associated with the bottom right corner of the mask.

The following formulas are used in computations:
Here $\mu_{i,j}$ and $\sigma_{i,j}$ stand for local mean and variance for pixel $X_{i,j}$, respectively, and $H$, $W$ are the vertical and horizontal sizes of the mask, respectively.

The corresponding value for the output pixel $Y_{i,j}$ is computed as:

$$Y_{i,j} = \mu_{i,j} + \frac{\sigma_{i,j}^2 - \nu^2}{\sigma^2} \cdot [X_{i,j} - \mu_{i,j}]$$

and stored in the $pDst$. Here $\nu^2$ is the noise variance, specified for each channel by the noise level parameter $\text{noise}$. If this parameter is not defined ($\text{noise} = 0$), then the function estimates the noise level by averaging through the image of all local variances $\sigma_{i,j}$, and stores the corresponding values in the $\text{noise}$ for further use.

The function `ippiFilterWiener` uses the external work buffer $pBuffer$, which must be allocated before the function call. To determine the required buffer size, the function `ippiFilterWienerGetBufferSize` can be used.

Figure "Applying the function ippiFilterWiener" illustrates the result of using `ippiFilterWiener_32f_C1R` function.

**Applying the function ippiFilterWiener**

![source image + noise (generated by the function ippiAddRandom)](image1) ![destination image](image2)

**Return Values**

*ippStsNoErr* Indicates no error. Any other value indicates an error.
Indicates an error condition if one of the specified pointers is NULL.

Indicates an error condition if one of the fields of `dstRoiSize` has a zero or negative value.

Indicates an error condition if one of the fields of `maskSize` has a value less than or equal to 1.

Indicates an error condition if one of the `noise` values is less than 0 or greater than 1.

Example

To better understand usage of this function, refer to the following example in the examples archive available for download from https://software.intel.com/en-us/ipp-manual-examples:

FilterWiener.c

Convolution

Intel IPP functions described in this section perform two-dimensional finite linear convolution operation between two source images and write the result into the destination image. Convolution is used to perform many common image processing operations including sharpening, blurring, noise reduction, embossing, and edge enhancement. For convenience, any digital image \( f \) is represented here as a matrix with \( M_f \) columns and \( N_f \) rows that contains pixel values \( f[i,j] \), \( 0 \leq i < M_f \), \( 0 \leq j < N_f \).

ConvGetBufferSize

Computes the size of the work buffer for the `ippiConv` function.

Syntax

```c
IppStatus ippiConvGetBufferSize (IppiSize src1Size, IppiSize src2Size, IppDataType dataType, int numChannels, IppEnum algType, int* pBufferSize);
```

Include Files

ippi.h

Parameters

- `src1Size, src2Size` Size, in pixels, of the source images.
- `dataType` Data type for convolution. Possible values are `ipp32f`, `ipp16s`, or `ipp8u`.
- `numChannels` Number of image channels. Possible values are 1, 3, or 4.
- `algType` Bit-field mask for the algorithm type definition. Possible values are the results of composition of the `IppAlgType` and `IppROIShape` values.
- `pBufferSize` Pointer to the size of the work buffer.

Description

The `ippiConvGetBufferSize` function computes the size, in bytes, of the external work buffer needed for the function that performs two-dimensional convolution. The result is stored in the `pBufferSize` parameter.
Return Values

ippStsNoErr
Indicates no error.

ippStsSizeErr
Indicates an error when the src1Size or src2Size is negative, or equal to zero.

ippStsNullChannelsErr
Indicates an error when the numChannels value differs from 1, 3, or 4.

ippStsDataTypeErr
Indicates an error when the dataType value differs from the ipp32f, ipp16s, or ipp8u.

ippStsAlgTypeErr
Indicates an error when:

- the result of the bitwise AND operation between algType and ippAlgMask differs from the ippAlgAuto, ippAlgDirect, or ippAlgFFT values;
- the result of the bitwise AND operation between algType andippiROI Mask differs from the ippiROIFull or ippiROIValid values.

ippStsNullPtrErr
Indicates an error when the pBufferSize is NULL.

See Also

Structures and Enumerators

Conv Performs two-dimensional convolution of two images.

Syntax

Case 1: Operating on integer data

IppStatus ippiConv_<mod>(const Ipp<datatype>* pSrc1, int src1Step, IppiSize src1Size, const Ipp<datatype>* pSrc2, int src2Step, IppiSize src2Size, Ipp<datatype>* pDst, int dstStep, int divisor, IppEnum algType, Ipp8u* pBuffer);

Supported values for mod

8u_C1R  16s_C1R
8u_C3R  16s_C3R
8u_C4R  16s_C4R

Case 2: Operating on floating-point data

IppStatus ippiConv_<mod>(const Ipp32f* pSrc1, int src1Step, IppiSize src1Size, const Ipp32f* pSrc2, int src2Step, IppiSize src2Size, Ipp32f* pDst, int dstStep, IppEnum algType, Ipp8u* pBuffer);

Supported values for mod

32f_C1R
32f_C3R
32f_C4R
Include Files
ippi.h

Domain Dependencies
Headers: ippicore.h, ippvm.h, ipps.h
Libraries: ippcore.lib, ippvm.lib, ipps.lib

Parameters

pSrc1, pSrc2  
Pointers to the source images ROI.

src1Step, src2Step  
Distance, in bytes, between the starting points of consecutive lines in the source images.

src1Size, src2Size  
Size in pixels of the source images.

pDst  
Pointer to the destination image ROI.

dstStep  
Distance, in bytes, between the starting points of consecutive lines in the destination image.

divisor  
The integer value by which the computed result is divided (for operations on integer data only).

algType  
Bit-field mask for the algorithm type definition. Possible values are the results of composition of the IppAlgType and IppiROIShape values.

pBuffer  
Pointer to the buffer for internal calculations.

Description
Before using this function, you need to compute the size of the work buffer using the ippiConvGetBufferSize function.

The ippiConv function operates with ROI. The type of convolution that function performs is defined by the value of the algType parameter:

1. If the ippiROIFull flag is set, the function performs full two-dimensional finite linear convolution between two source images pointed by the pSrc1 and pSrc2 parameters. The resulting destination image h[i, j] is computed by the following formula:

\[ h[i, j] = \frac{1}{\text{divisor}} \sum_{k=0}^{M_g-1} \sum_{l=0}^{M_f-1} f[k, l] \times g[i-k, j-l] , \]

where

- \( M_h = M_f + M_g - 1 \)

where

- \( M_f \) is the number of rows in the first source image matrix \( f \)
- \( M_g \) is the number of rows in the second source image matrix \( g \)
- \( N_h = N_f + N_g - 1 \)

where

- \( N_f \) is the number of columns in the first source image matrix \( f \)
- \( N_g \) is the number of columns in the second source image matrix \( g \)
2. If the ippiROIValid flag is set up, the function performs valid two-dimensional finite linear convolution between two source images pointed by the pSrc1 and pSrc2 parameters. The destination image $h[i, j]$ obtained as a result of the function operation is computed by the following formula:

$$h[i, j] = \frac{1}{\text{divisor}} \sum_{k=0}^{N_g-1} \sum_{j=0}^{M_g-1} f[i+k, j+l] \times g[M_g-k-1, N_g-l-1]$$

where

- $M_h = |M_f - M_g| + 1$
- $N_h = |N_f - N_g| + 1$

This case assumes that $M_f \geq M_g$ and $N_f \geq N_g$. In case when $M_f < M_g$ and $N_f < N_g$, the subscript index $g$ in this equation must be replaced with the index $f$. For any other combination of source image sizes, the function performs no operation.

**NOTE**

The above formula provides the same result as in the case with the ippiROIFull flag, but produces only the part of the convolution image that is computed without zero-padded values.

Function flavors that accept input data of the Ipp32f type use the same summation formula, but without scaling of the result ($\text{divisor} = 1$ is assumed).

The following examples illustrate the function operation. For the source images $f, g$ of size 3 x 5 represented as

\[
\begin{bmatrix}
  f[1, 1] & f[1, 2] & f[1, 3] \\
\end{bmatrix}
\]

\[
\begin{bmatrix}
  g[1, 1] & g[1, 2] \\
  g[2, 1] & g[2, 2] \\
  g[3, 1] & g[3, 2]
\end{bmatrix}
\]
\[
\begin{bmatrix}
1 & 1 & 1 \\
1 & 0 & 0 \\
1 & 1 & 1 \\
0 & 0 & 1 \\
1 & 1 & 1
\end{bmatrix}
\]

\[f = \begin{bmatrix}
1 & 1 & 1 \\
1 & 0 & 0 \\
1 & 1 & 1 \\
0 & 0 & 1 \\
1 & 1 & 1
\end{bmatrix}, \quad g = f\]

with \(g = f\):

- for the \textit{ippiROIFull} case, the resulting convolution image \(h\) is of size \(5 \times 9\) and contains the following data:

\[
\begin{bmatrix}
1 & 2 & 3 & 2 & 1 \\
2 & 2 & 2 & 0 & 0 \\
3 & 4 & 6 & 4 & 2 \\
2 & 2 & 4 & 2 & 2 \\
3 & 6 & 1 & 1 & 6 & 3 \\
2 & 2 & 4 & 2 & 2 \\
2 & 4 & 6 & 4 & 3 \\
0 & 0 & 2 & 2 & 2 \\
1 & 2 & 3 & 2 & 1
\end{bmatrix}
\]

- for the \textit{ippiROIValid} case, the resulting convolution image \(h\) is of size \(1 \times 1\) and contains the following data:

\[h = [11]\]

\textbf{Return Values}

- \texttt{ippStsNoErr} indicates no error. Any other value indicates an error.
- \texttt{ippStsNullPtrErr} indicates an error when any of the specified pointers is NULL.
- \texttt{ippStsSizeErr} indicates an error when \(\texttt{src1Size}\) or \(\texttt{src2Size}\) has a zero or negative value.
- \texttt{ippStsStepErr} indicates an error when \(\texttt{src1Step}, \texttt{src2Step},\) or \(\texttt{dstStep}\) has a zero or negative value.
- \texttt{ippStsDivisorErr} indicates an error when \(\texttt{divisor}\) has a zero value.
- \texttt{ippStsAlgTypeErr} indicates an error when:
  - the result of the bitwise AND operation between the \texttt{algType} and \texttt{ippAlgMask} differs from the \texttt{ippAlgAuto, ippAlgDirect,} or \texttt{ippAlgFFT} values;
  - the result of the bitwise AND operation between the \texttt{algType} and \texttt{ippiROIIMask} differs from the \texttt{ippiROIFull} or \texttt{ippiROIValid} values.
Example

To better understand usage of this function, refer to the following example in the examples archive available for download from https://software.intel.com/en-us/ipp-manual-examples:

Conv.c

See Also

Regions of Interest in Intel IPP
Structures and Enumerators
ConvGetBufferSize Computes the size of the work buffer for the ippiConv function.

Deconvolution

Functions described in this section perform image deconvolution. They can be used for restoring the degraded image, in particular image that was obtained by applying the convolution operation with known kernel. The Intel IPP functions implement two methods: the Fourier deconvolution (noniterative method) [see for example, [Puetter05]], and the Richardson-Lucy method (iterative method) [Ric72]. Border pixels of a source image are restored before deconvolution.

DeconvFFTGetSize

Computes the size of the state structure for deconvolution with the fast Fourier transform (FFT).

Syntax

IppStatus ippiDeconvFFTGetSize_32f(int nChannels, int kernelSize, int FFTorder, int* pSize);

Include Files

ippi.h

Domain Dependencies

Headers: ippcore.h, ippvm.h, ipps.h
Libraries: ippcore.lib, ippvm.lib, ipps.lib

Parameters

nChannels Number of channels in the image. Possible values are 1 or 3.
kernelSize Size of the kernel.
FFTorder Order of the created FFT state structure.
pSize Pointer to the size of the IppiDeconvFFTState_32f_C1R or IppiDeconvFFTState_32f_C3R structure, in bytes.

Description

This function computes the fast Fourier transform (FFT) deconvolution state structure size that is required to initialize the structure with the ippiDeconvFFTInit function. This structure is used by the ippiDeconvFFT function, which performs deconvolution of the source image using FFT.
Return Values

ippStsNoErr
Indicates no error. Any other value indicates an error or a warning.

ippStsNullPtrErr
Indicates an error when any of the specified pointers is NULL.

ippStsNumChannelsErr
Indicates an error when nChannels has an illegal value.

ippStsSizeErr
Indicates an error when kernelSize is less than, or equal to 0; or if kernelSize is greater than $2^{FFTorder}$.

See Also
DeconvFFTInit Initializes the FFT deconvolution state structure.
DeconvFFT Performs FFT deconvolution of an image.

DeconvFFTInit
Initializes the FFT deconvolution state structure.

Syntax
IppStatus ippiDeconvFFTInit_32f_C1R(IppiDeconvFFTState_32f_C1R* pDeconvFFTState, const Ipp32f* pKernel, int kernelSize, int FFTorder, Ipp32f threshold);
IppStatus ippiDeconvFFTInit_32f_C3R(IppiDeconvFFTState_32f_C3R* pDeconvFFTState, const Ipp32f* pKernel, int kernelSize, int FFTorder, Ipp32f threshold);

Include Files
ippi.h

Domain Dependencies
Headers: ippcore.h, ippvm.h, ipps.h
Libraries: ippcore.lib, ippvm.lib, ipps.lib

Parameters
pDeconvFFTState Pointer to the FFT deconvolution state structure.
pKernel Pointer to the kernel array.
kernalsize Size of the kernel.
FFTorder Order of the created FFT state structure.
threshold Value of the threshold level (to exclude dividing by zero).

Description
This function initializes the FFT deconvolution state structure that is used by the ippiDeconvFFT function to perform deconvolution of the source image using FFT. Before using the ippiDeconvFFTInit function, compute the size of the structure using the ippiDeconvFFTGetSize function.

Return Values
ippStsNoErr Indicates no error. Any other value indicates an error or a warning.
ippStsNullPtrErr Indicates an error when any of the specified pointers is NULL.
Indicates an error when `kernelSize` is less than, or equal to 0; or if `kernelSize` is greater than $2^{\text{FFTorder}}$.

Indicates an error when `threshold` is less than, or equal to 0.

See Also
DeconvFFTGetSize Computes the size of the state structure for deconvolution with the fast Fourier transform (FFT).

DeconvFFT Performs FFT deconvolution of an image.

DeconvFFT
Performs FFT deconvolution of an image.

Syntax

```c
IppStatus ippiDeconvFFT_32f_C1R(const Ipp32f* pSrc, int srcStep, Ipp32f* pDst, int dstStep, IppiSize roiSize, IppiDeconvFFTState_32f_C1R* pDeconvFFTState);
IppStatus ippiDeconvFFT_32f_C3R(const Ipp32f* pSrc, int srcStep, Ipp32f* pDst, int dstStep, IppiSize roiSize, IppiDeconvFFTState_32f_C3R* pDeconvFFTState);
```

Include Files
ippi.h

Domain Dependencies
Headers: ippcore.h, ippvm.h, ipps.h
Libraries: ippcore.lib, ippvm.lib, ipps.lib

Parameters

- `pSrc`
  Pointer to the source image ROI.

- `srcStep`
  Distance in bytes between starts of consecutive lines in the source image.

- `pDst`
  Pointer to the destination image ROI.

- `dstStep`
  Distance in bytes between starts of consecutive lines in the destination image.

- `roiSize`
  Size of the source and destination image ROI.

- `pDeconvFFTState`
  Pointer to the FFT deconvolution state structure.

Description

This function operates with ROI (see Regions of Interest in Intel IPP).

This function performs deconvolution of the source image `pSrc` using FFT with parameters specified in the FFT deconvolution state structure `pDeconvFFTState` and stores results to the destination image `pDst`. The FFT deconvolution state structure must be initialized by calling the function `DeconvFFTInit` beforehand.

Return Values

- `ippStsNoErr`
  Indicates no error. Any other value indicates an error or a warning.
DeconvLRGetSize

Computes the size of the state structure for Lucy-Richardson (LR) deconvolution.

Syntax

IppStatusippiDeconvLRGetSize_32f(int numChannels, int kernelSize, IppiSize maxRoi, int* pSize);

Include Files

ippi.h

Domain Dependencies

Headers: ippcore.h, ippvm.h, ipps.h
Libraries: ippcore.lib, ippvm.lib, ipps.lib

Parameters

numChannels

Number of channels in the image. Possible values are 1 or 3.

kernelSize

Size of the kernel.

maxRoi

Maximum size of the image ROI, in pixels.

pSize

Pointer to the size of the IppiDeconvLRState_32f_C1R or IppiDeconvLRState_32f_C3R structure, in bytes.

Description

This function computes the Lucy-Richardson (LR) deconvolution state structure size that is required to initialize the structure with theippiDeconvLRInit function. This structure is used by theippiDeconvLR function, which performs LR deconvolution of the source image.

Return Values

ippStsNoErr

Indicates no error. Any other value indicates an error or a warning.

ippStsNullPtrErr

Indicates an error when any of the specified pointers is NULL.

ippStsSizeErr

Indicates an error when:

- kernelSize is less than, or equal to 0
- kernelSize is greater than maxRoi.height or maxRoi.width
- maxRoi.height or maxRoi.width is less than, or equal to zero

See Also

DeconvLRInit Initializes the LR deconvolution state structure.
DeconvLR Performs LR deconvolution of an image.

DeconvLRInit

Initializes the LR deconvolution state structure.
Syntax

IppStatus ippiDeconvLRInit_32f_C1R(IppiDeconvLR_32f_C1R* pDeconvLR, const Ipp32f* pKernel, int kernelSize, IppiSize maxRoi, Ipp32f threshold);
IppStatus ippiDeconvLRInit_32f_C3R(IppiDeconvLR_32f_C3R* pDeconvLR, const Ipp32f* pKernel, int kernelSize, IppiSize maxRoi, Ipp32f threshold);

Include Files
ippi.h

Domain Dependencies

Headers: ippcore.h, ippvm.h, ipps.h
Libraries: ippcore.lib, ippvm.lib, ipps.lib

Parameters

  pDeconvLR  Pointer to the LR deconvolution state structure.
  pKernel  Pointer to the kernel array.
  kernelSize  Size of the kernel.
  maxRoi  Maximum size of the image ROI, in pixels.
  threshold  Value of the threshold level (to exclude dividing by zero).

Description

This function initializes the LR deconvolution state structure that is used by the ippiDeconvLR function to perform LR deconvolution of the source image. Before using the ippiDeconvLRInit function, compute the size of the structure using the ippiDeconvLRGetSize function.

Return Values

  ippStsNoErr  Indicates no error. Any other value indicates an error or a warning.
  ippStsNullPtrErr  Indicates an error when any of the specified pointers is NULL.
  ippStsSizeErr  Indicates an error when:
      - kernelSize is less than, or equal to 0
      - kernelSize is greater than maxRoi.height or maxRoi.width
      - maxRoi.height or maxRoi.width is less than, or equal to zero
  ippStsBadArgErr  Indicates an error when threshold is less than, or equal to 0.

See Also

DeconvLRGetSize Computes the size of the state structure for Lucy-Richardson (LR) deconvolution.
DeconvLR Performs LR deconvolution of an image.

DeconvLR
Performs LR deconvolution of an image.
Syntax

IppStatus ippiDeconvLR_32f_C1R(const Ipp32f* pSrc, int srcStep, Ipp32f* pDst, int dstStep, IppSize roiSize, int numIter, IppiDeconvLR_32f_C1R* pDeconvLR);
IppStatus ippiDeconvLR_32f_C3R(const Ipp32f* pSrc, int srcStep, Ipp32f* pDst, int dstStep, IppSize roiSize, int numIter, IppiDeconvLR_32f_C3R* pDeconvLR);

Include Files

ippi.h

Domain Dependencies

Headers: ippcore.h, ippvm.h, ipps.h
Libraries: ippcore.lib, ippvm.lib, ipps.lib

Parameters

pSrc            Pointer to the source image ROI.
srcStep         Distance in bytes between starts of consecutive lines in the source image.
pDst            Pointer to the destination image ROI.
dstStep         Distance in bytes between starts of consecutive lines in the destination image.
roiSize         Size of the source and destination image ROI.
numIter         Number of algorithm iterations.
pDeconvLR       Pointer to the LR deconvolution state structure.

Description

This function operates with ROI (see Regions of Interest in Intel IPP).
This function performs deconvolution of the source image pSrc using the Lucy-Richardson algorithm with parameters specified in the state structure pDeconvLR and stores results to the destination image pDst. The Lucy-Richardson deconvolution state structure must be initialized by calling the function DeconvLRInit beforehand.

Return Values

ippStsNoErr        Indicates no error. Any other value indicates an error or a warning.
ippStsNullPtrErr   Indicates an error condition if one of the specified pointers is NULL.
ippStsSizeErr      Indicates an error condition if roi.width or roi.height is less than or equal to 0, or if roi.width is greater than (maxRoi.width - kernelSize), or roi.height is greater than (maxRoi.height - kernelSize).
ippStsStepErr      Indicates an error condition if srcStep or dstStep is less than roiSize.width *<pixelSize>.
ippStsNotEvenStepErr Indicates an error condition if steps for floating-point images are not divisible by 4.
Indicates an error condition if numIter is less than or equal to 0.

## Fixed Filters

The fixed filter functions perform linear filtering of a source image using one of the predefined convolution kernels. The supported fixed filters and their respective kernel sizes are listed in the following table:

### Types of the Fixed Filter Functions

<table>
<thead>
<tr>
<th>Fixed Filter Type</th>
<th>Kernel Size</th>
</tr>
</thead>
<tbody>
<tr>
<td>Horizontal Prewitt operator</td>
<td>3x3</td>
</tr>
<tr>
<td>Vertical Prewitt operator</td>
<td>3x3</td>
</tr>
<tr>
<td>Horizontal Scharr operator</td>
<td>3x3</td>
</tr>
<tr>
<td>Vertical Scharr operator</td>
<td>3x3</td>
</tr>
<tr>
<td>Horizontal Sobel operator</td>
<td>3x3 or 5x5</td>
</tr>
<tr>
<td>Vertical Sobel operator</td>
<td>3x3 or 5x5</td>
</tr>
<tr>
<td>Second derivative horizontal Sobel operator</td>
<td>3x3 or 5x5</td>
</tr>
<tr>
<td>Second derivative vertical Sobel operator</td>
<td>3x3 or 5x5</td>
</tr>
<tr>
<td>Second cross derivative Sobel operator</td>
<td>3x3 or 5x5</td>
</tr>
<tr>
<td>Horizontal Roberts operator</td>
<td>3x3</td>
</tr>
<tr>
<td>Vertical Roberts operator</td>
<td>3x3</td>
</tr>
<tr>
<td>Laplacian highpass filter</td>
<td>3x3 or 5x5</td>
</tr>
<tr>
<td>Gaussian lowpass filter</td>
<td>3x3 or 5x5</td>
</tr>
<tr>
<td>Highpass filter</td>
<td>3x3 or 5x5</td>
</tr>
<tr>
<td>Lowpass filter</td>
<td>3x3 or 5x5</td>
</tr>
<tr>
<td>Sharpening filter</td>
<td>3x3</td>
</tr>
</tbody>
</table>

Using fixed filter functions with predefined kernels is more efficient as it eliminates the need to create the convolution kernel in your application program.

**NOTE**

The anchor cell is the center cell of the kernel for all fixed filters.

## FilterGaussianGetBufferSize

*Computes the size of the Gaussian specification structure and the size of the external work buffer for Gaussian filtering with user-defined borders.*

**Syntax**

```c
IppStatus ippiFilterGaussianGetBufferSize(IppiSize maxRoiSize, Ipp32u kernelSize, IppDataType dataType, int numChannels, int* pSpecSize, int* pBufferSize);
```

**Platform-aware functions**

```c
IppStatus ippiFilterGaussianGetBufferSize_L(IppSizeL maxRoiSize, int kernelSize, IppDataType dataType, IppBorderType borderType, int numChannels, IppSizeL* pSpecSize, int* pBufferSize);
```

**Include Files**

ippcv.h

Flavors with the **_L** suffix: ippcv_l.h
Domain Dependencies

Headers: ippcore.h, ippvm.h, ipps.h,ippi.h
Libraries: ippcore.lib, ippvm.lib, ipps.lib,ippi.lib

Parameters

- maxRoiSize: Maximal size of the image ROI in pixels.
- kernelSize: Size of the Gaussian filter kernel. The value must be odd and greater, or equal to 3.
- dataType: Data type of the source and destination images.
- borderType: One of border supported types.
- numChannels: Number of channels in the images. Possible values are 1 and 3.
- pSpecSize: Pointer to the computed size (in bytes) of the Gaussian specification structure.
- pBufferSize: Pointer to the computed size (in bytes) of the external buffer.

Description

This function computes the size of the Gaussian context structure and external work buffer for the FilterGaussianBorder function or for the platform-aware function FilterGaussian. The results are stored in pSpecSize and pBufferSize. The buffer with the length pBufferSize[0] can be used to filter an image with the width and height less than, or equal to the corresponding fields of maxRoiSize, and/or kernel size that is less than, or equal to kernelSize.

NOTE

The platform-aware function FilterGaussianGetBufferSize_L computes only the size of the external work buffer for Gaussian filtering with user-defined borders. To compute the size of the Gaussian specification structure, please use the platform-aware function FilterGaussianGetSpecSize.

Use the computed pBufferSize and pSpecSize values to allocate the memory using the ippMalloc or ippsMalloc functions. The allocated memory can be freed only by the ippFree or ippsFree functions, respectively. For more information about the memory allocation functions, refer to the "Support Functions" section of the Intel IPP Developer Reference, vol. 1.

For an example on how to use this function, refer to the example provided with the FilterGaussianBorder function description.

Return Values

- ippStsNoErr: Indicates no error. Any other value indicates an error.
- ippStsNullPtrErr: Indicates an error when one of the specified pointers is NULL.
- ippStsSizeErr: Indicates an error when maxRoiSize has a field with a zero or negative value.
- ippStsBadArgErr: Indicates an error when kernelSize is even, or less than 3.
- ippStsDataTypeErr: Indicates an error when dataType has an illegal value.
- ippStsNumChannelErr: Indicates an error when numChannels has an illegal value.
See Also
FilterGaussianBorder Performs Gaussian filtering of an image with user-defined borders.

FilterGaussianGetSpecSize
Computes the size of the Gaussian specification structure.

Syntax
Platform-aware function
IppStatus ippiFilterGaussianGetSpecSize_L(int kernelSize, IppDataType dataType, int numChannels, IppSizeL* pSpecSize, IppSizeL* pInitBufferSize);

Include Files
ippcv_l.h

Domain Dependencies
Headers: ippcore.h, ippvm.h, ipps.h,ippi.h
Libraries: ippcore.lib, ippvm.lib, ipps.lib,ippi.lib

Parameters
kernelSize  Size of the Gaussian filter kernel. The value must be odd and greater, or equal to 3.
dataType  Data type of the source and destination images.
numChannels  Number of channels in the images. Possible values are 1 and 3.
pSpecSize  Pointer to the computed size (in bytes) of the Gaussian specification structure.
pInitBufferSize  Pointer to the computed size (in bytes) of the external buffer.

Description
This function computes the size of the Gaussian context structure and the size of the initialization external work buffer for the FilterGaussianBorder function or for the platform-aware functions FilterGaussianInit and FilterGaussian. The results are stored in pSpecSize and pInitBufferSize. The buffer with the length pInitBufferSize[0] can be used to initialize the specification structure for the Gaussian filter.

Use the computed pInitBufferSize and pSpecSize values to allocate the memory using the ippMalloc or ippsMalloc functions. The allocated memory can be freed only by the ippFree or ippsFree functions, respectively. For more information about the memory allocation functions, refer to the "Support Functions" section of the Intel IPP Developer Reference, vol. 1.

Return Values
ippStsNoErr  Indicates no error. Any other value indicates an error.
ippStsNullPtrErr  Indicates an error when one of the specified pointers is NULL.
ippStsBadArgErr  Indicates an error when kernelSize is even, or less than 3.
ippStsDataTypeErr  Indicates an error when dataType has an illegal value.
ippStsNumChannelErr

Indicates an error when numChannels has an illegal value.

See Also
FilterGaussian Performs Gaussian filtering of an image with user-defined borders.

FilterGaussianInit

Initializes the Gaussian context structure.

Syntax

IppStatus ippiFilterGaussianInit(IppiSize roiSize, Ipp32u kernelSize, Ipp32f sigma, IppiBorderType borderType, IppDataType dataType, int numChannels, IppFilterGaussianSpec* pSpec, Ipp8u* pBuffer);

Platform-aware functions

IppStatus ippiFilterGaussianInit_L(IppiSizeL roiSize, int kernelSize, Ipp32f sigma, IppiBorderType borderType, IppDataType dataType, int numChannels, IppFilterGaussianSpec* pSpec, Ipp8u* pInitBuffer);

Include Files

ippcv.h

Flavors with the _L suffix: ippcv_l.h

Domain Dependencies

Headers: ippcore.h, ippvm.h, ipps.h, ippi.h

Libraries: ippcore.lib, ippvm.lib, ipps.lib, ippi.lib

Parameters

roiSize

Size of the image ROI in pixels.

kernelSize

Size of the Gaussian filter kernel. The value must be odd and greater, or equal to 3.

sigma

Standard deviation of the Gaussian kernel.

borderType

Type of border. Possible values are:

ippBorderConst

Values of all border pixels are set to constant.

ippBorderRepl

Border is replicated from the edge pixels.

ippBorderInMem

Border is obtained from the image pixels in memory.

ippBorderMirror

Border pixels are mirrored from the source image boundary pixels.

dataType

Data type of the source and destination images.

numChannels

Number of channels in the images. Possible values are 1 and 3.

pSpec

Pointer to the Gaussian specification structure.

pBuffer, pInitBuffer

Pointer to the external buffer.
Description
This function operates with ROI (see Regions of Interest in Intel IPP).

This function initializes the Gaussian specification structure \texttt{pSpec} for the \texttt{FilterGaussianBorder} function or for the platform-aware function \texttt{FilterGaussian}. Before using this function, compute the size of the specification structure and the external buffer using the \texttt{FilterGaussianGetBufferSize} function and \texttt{FilterGaussianGetSpecSize} for the platform-aware function.

The \texttt{roiSize} and \texttt{kernelSize} values must be less than, or equal to the corresponding values specified in the \texttt{FilterGaussianGetBufferSize} function.

The kernel of the Gaussian filter is the matrix of size \texttt{kernelSize}x\texttt{kernelSize} with the standard deviation \texttt{sigma}. The values of the Gaussian kernel elements are calculated by the formula below and then normalized:

\[
G(i, j) = \exp\left\{-\frac{(K / 2 - i)^2 + (K / 2 - j)^2}{2\sigma^2}\right\}, \quad i, j = 0, \ldots, K
\]

The anchor cell is the center of the kernel.

For an example on how to use this function, refer to the example provided with the \texttt{FilterGaussianBorder} function description.

Return Values
\begin{itemize}
\item \texttt{ippStsNoErr} Indicates no error. Any other value indicates an error.
\item \texttt{ippStsNullPtrErr} Indicates an error when one of the specified pointers is \texttt{NULL}.
\item \texttt{ippStsSizeErr} Indicates an error when \texttt{roiSize} has a field with a zero or negative value.
\item \texttt{ippStsBadArgErr} Indicates an error when \texttt{kernelSize} is even, or less than 3; or \texttt{sigma} is less than, or equal to zero.
\item \texttt{ippStsDataTypeErr} Indicates an error when \texttt{dataType} has an illegal value.
\item \texttt{ippStsChannelErr} Indicates an error when \texttt{numChannels} has an illegal value.
\item \texttt{ippStsBorderErr} Indicates an error when \texttt{borderType} has an illegal value.
\end{itemize}

See Also
\texttt{FilterGaussianGetBufferSize} Computes the size of the Gaussian specification structure and the size of the external work buffer for Gaussian filtering with user-defined borders.
\texttt{FilterGaussianBorder} Performs Gaussian filtering of an image with user-defined borders.

\textbf{FilterGaussian}

\textit{Performs Gaussian filtering of an image with user-defined borders.}
Syntax
IppStatusippiFilterGaussian_<mod>(const Ipp<datatype>* pSrc, IppSizeL srcStep,
Ipp<datatype>* pDst, IppSizeL dstStep, IppSizeL roiSize, IppiBorderType borderType,
const Ipp<datatype> borderValue[1], IppFilterGaussianSpec* pSpec, Ipp8u* pBuffer);

Supported values for mod:

<table>
<thead>
<tr>
<th>Format</th>
</tr>
</thead>
<tbody>
<tr>
<td>8u_C1R_L</td>
</tr>
<tr>
<td>16u_C1R_L</td>
</tr>
<tr>
<td>16s_C1R_L</td>
</tr>
<tr>
<td>32f_C1R_L</td>
</tr>
</tbody>
</table>

IppStatusippiFilterGaussian <mod>(const Ipp<datatype>* pSrc, IppSizeL srcStep,
Ipp<datatype>* pDst, IppSizeL dstStep, IppSizeL roiSize, IppiBorderType borderType,
const Ipp<datatype> borderValue[3], IppFilterGaussianSpec* pSpec, Ipp8u* pBuffer);

Supported values for mod:

<table>
<thead>
<tr>
<th>Format</th>
</tr>
</thead>
<tbody>
<tr>
<td>8u_C3R_L</td>
</tr>
<tr>
<td>16u_C3R_L</td>
</tr>
<tr>
<td>16s_C3R_L</td>
</tr>
<tr>
<td>32f_C3R_L</td>
</tr>
</tbody>
</table>

Include Files
ippcv_l.h

Domain Dependencies
Headers: ippcore.h, ippvm.h, ipps.h,ippi.h
Libraries: ippcore.lib, ippvm.lib, ipps.lib,ippi.lib

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>pSrc</td>
<td>Pointer to the source image ROI.</td>
</tr>
<tr>
<td>srcStep</td>
<td>Distance, in bytes, between the starting points of consecutive lines in the source image.</td>
</tr>
<tr>
<td>pDst</td>
<td>Pointer to the destination image ROI.</td>
</tr>
<tr>
<td>dstStep</td>
<td>Distance, in bytes, between the starting points of consecutive lines in the destination image.</td>
</tr>
<tr>
<td>roiSize</td>
<td>Size of the source and destination image ROI, in pixels.</td>
</tr>
<tr>
<td>borderType</td>
<td>One of the border supported types.</td>
</tr>
<tr>
<td>borderValue</td>
<td>Constant value to assign to pixels of the constant border. This parameter is applicable only to the ippBorderConst border type.</td>
</tr>
<tr>
<td>pSpec</td>
<td>Pointer to the Gaussian specification structure.</td>
</tr>
<tr>
<td>pBuffer</td>
<td>Pointer to the work buffer.</td>
</tr>
</tbody>
</table>

Description
This function operates with ROI (see Regions of Interest in Intel IPP).

These functions apply the Gaussian filter to the source image ROI pSrc. The kernel of the Gaussian filter is the matrix of size kernelSize x kernelSize with the standard deviation sigma. The values of the Gaussian kernel elements are computed by the FilterGaussianInit function. Elements of the kernel are normalized. The anchor cell is the center of the kernel.
Before using the `ippiFilterGaussian` function, compute the size of the Gaussian specification structure using the `FilterGaussianGetSpecSize` function and the external buffer using the `FilterGaussianGetBufferSize` function and initialize the structure using the `FilterGaussianInit` function.

**Return Values**

- `ippStsNoErr` Indicates no error.
- `ippStsNullPtrErr` Indicates an error when one of the specified pointers is `NULL`.
- `ippStsSizeErr` Indicates an error when `roiSize` has a field with a zero or negative value.
- `ippStsStepErr` Indicates an error when `srcStep` or `dstStep` is less than `roiSize.width * pixelSize`.
- `ippStsNotEvenStepErr` Indicates an error when one of the step values is not divisible by `sizeof(Ipp<dataType>)`.
- `ippStsBorderErr` Indicates an error when `borderType` has an illegal value.
- `ippStsBadArgErr` Indicates an error when `kernelSize` is even, or less than 3.

**Example**

To better understand usage of this function, refer to the following example in the examples archive available for download from https://software.intel.com/en-us/ipp-manual-examples:

`FilterGaussianBorder.c`

**See Also**

- Regions of Interest in Intel IPP
- User-defined Border Types
- `FilterGaussianGetSpecSize` Computes the size of the Gaussian specification structure.
- `FilterGaussianGetBufferSize` Computes the size of the Gaussian specification structure and the size of the external work buffer for Gaussian filtering with user-defined borders.
- `FilterGaussianInit` Initializes the Gaussian context structure.

**FilterHipassBorderGetBufferSize**

`Computes the size of the work buffer for high-pass filtering with the `ippiFilterHipassBorder` function.`

**Syntax**

```c
IppStatus ippiFilterHipassBorderGetBufferSize(IppiSize dstRoiSize, IppiMaskSize mask, IppDataType srcDataType, IppDataType dstDataType, int numChannels, int* pBufferSize);
```

**Include Files**

```
ippi.h
```

**Domain Dependencies**

- **Headers**: `ippcore.h`, `ippvm.h`, `ipps.h`
- **Libraries**: `ippcore.lib`, `ippvm.lib`, `ipps.lib`
Parameters

dstRoiSize
Size of the destination ROI, in pixels.

mask
Predefined mask of IppiMaskSize type. Possible values are: ippMskSize3x3 or ippMskSize5x5.

srcDataType
Data type of the source image.

dstDataType
Data type of the destination image.

numChannels
Number of channels in the image. Possible values are 1, 3, or 4.

pBufferSize
Pointer to the size, in bytes, of the external buffer.

Description
This function computes the size, in bytes, of the external work buffer for the ippiFilterHipassBorder function. The result is stored in the pBufferSize parameter.

Return Values

ippStsNoErr
Indicates no error. Any other value indicates an error.

ippStsNullPtrErr
Indicates an error when pBufferSize is NULL.

ippStsSizeErr
Indicates an error when dstRoiSize has a field with a zero or negative value.

ippStsMaskSizeErr
Indicates an error when mask has an illegal value.

ippStsDataTypeErr
Indicates an error when srcDataType or dstDataType has an illegal value.

ippStsNumChannelErr
Indicates an error when numChannels has an illegal value.

See Also
FilterHipassBorder Filters an image using a high-pass filter.

FilterHipassBorder
Filters an image using a high-pass filter.

Syntax

Case 1: Operating on one-channel data

IppStatus ippiFilterHipassBorder_8u_C1R(const Ipp8u* pSrc, int srcStep, Ipp8u* pDst, int dstStep, IppiSize dstRoiSize, IppiMaskSize mask, IppiBorderType borderType, Ipp8u borderValue, Ipp8u* pBuffer);

IppStatus ippiFilterHipassBorder_16s_C1R(const Ipp16s* pSrc, int srcStep, Ipp16s* pDst, int dstStep, IppiSize dstRoiSize, IppiMaskSize mask, IppiBorderType borderType, Ipp16s borderValue, Ipp8u* pBuffer);

IppStatus ippiFilterHipassBorder_32f_C1R(const Ipp32f* pSrc, int srcStep, Ipp32f* pDst, int dstStep, IppiSize dstRoiSize, IppiMaskSize mask, IppiBorderType borderType, Ipp32f borderValue, Ipp8u* pBuffer);
Case 2: Operating on multi-channel data

IppStatusippiFilterHipassBorder_8u_C3R(const Ipp8u* pSrc, int srcStep, Ipp8u* pDst, int dstStep, IppSize dstRoiSize, IppiMaskSize mask, IppiBorderType borderType, const Ipp8u pBorderValue[3], Ipp8u* pBuffer);

IppStatusippiFilterHipassBorder_16s_C3R(const Ipp16s* pSrc, int srcStep, Ipp16s* pDst, int dstStep, IppSize dstRoiSize, IppiMaskSize mask, IppiBorderType borderType, const Ipp16s pBorderValue[3], Ipp8u* pBuffer);

IppStatusippiFilterHipassBorder_32f_C3R(const Ipp32f* pSrc, int srcStep, Ipp32f* pDst, int dstStep, IppSize dstRoiSize, IppiMaskSize mask, IppiBorderType borderType, const Ipp32f pBorderValue[3], Ipp8u* pBuffer);

IppStatusippiFilterHipassBorder_8u_AC4R(const Ipp8u* pSrc, int srcStep, Ipp8u* pDst, int dstStep, IppSize dstRoiSize, IppiMaskSize mask, IppiBorderType borderType, const Ipp8u pBorderValue[3], Ipp8u* pBuffer);

IppStatusippiFilterHipassBorder_16s_AC4R(const Ipp16s* pSrc, int srcStep, Ipp16s* pDst, int dstStep, IppSize dstRoiSize, IppiMaskSize mask, IppiBorderType borderType, const Ipp16s pBorderValue[3], Ipp8u* pBuffer);

IppStatusippiFilterHipassBorder_32f_AC4R(const Ipp32f* pSrc, int srcStep, Ipp32f* pDst, int dstStep, IppSize dstRoiSize, IppiMaskSize mask, IppiBorderType borderType, const Ipp32f pBorderValue[3], Ipp8u* pBuffer);

IppStatusippiFilterHipassBorder_8u_C4R(const Ipp8u* pSrc, int srcStep, Ipp8u* pDst, int dstStep, IppSize dstRoiSize, IppiMaskSize mask, IppiBorderType borderType, const Ipp8u pBorderValue[4], Ipp8u* pBuffer);

IppStatusippiFilterHipassBorder_16s_C4R(const Ipp16s* pSrc, int srcStep, Ipp16s* pDst, int dstStep, IppSize dstRoiSize, IppiMaskSize mask, IppiBorderType borderType, const Ipp16s pBorderValue[4], Ipp8u* pBuffer);

IppStatusippiFilterHipassBorder_32f_C4R(const Ipp32f* pSrc, int srcStep, Ipp32f* pDst, int dstStep, IppSize dstRoiSize, IppiMaskSize mask, IppiBorderType borderType, const Ipp32f pBorderValue[4], Ipp8u* pBuffer);

Include Files

ippi.h

Domain Dependencies

Headers: ippcore.h, ippvm.h, ipps.h

Libraries: ippcore.lib, ippvm.lib, ipps.lib

Parameters

pSrc

Pointer to the source image ROI.

srcStep

Distance, in bytes, between the starting points of consecutive lines in the source image.

pDst

Pointer to the destination image ROI.

dstStep

Distance, in bytes, between the starting points of consecutive lines in the destination image.

dstRoiSize

Size of the source and destination ROI, in pixels.
**mask**

Predefined mask of IppiMaskSize. Possible values are ippMskSize3x3 or ippMskSize5x5.

**borderType**

Type of border. Possible values are:

- ippBorderConst: Values of all border pixels are set to constant.
- ippBorderRepl: Border is replicated from the edge pixels.
- ippBorderInMem: Border is obtained from the source image pixels in memory.

Mixed borders are also supported. They can be obtained by the bitwise operation OR between any of the ippBorderRepl, ippBorderConst, or ippBorderMirror values and the ippBorderInMemTop, ippBorderInMemBottom, ippBorderInMemLeft, ippBorderInMemRight values.

**borderValue**

Constant value to assign to pixels of the constant border. This parameter is applicable only to the ippBorderConst border type.

**pBorderValue[3]**, **pBorderValue[4]**

Pointer to constant values to assign to pixels of the constant border. This parameter is applicable only to the ippBorderConst border type.

**pBuffer**

Pointer to the work buffer.

**Description**

Before using this function, you need to compute the size of the work buffer pBuffer using theippiFilterHpassBorderGetBufferSize function.

This function operates with ROI (see Regions of Interest in Intel IPP).

This function applies a high-pass filter to the pSrc source image ROI. The size of the source image ROI is equal to the destination image ROI size dstRoiSize. The values of border pixels are assigned in accordance with the borderType and borderValue parameters. The kernel of the filter is a matrix of 3x3 or 5x5 size depending on the mask value. The kernels have the following values:

<table>
<thead>
<tr>
<th>-1 -1 -1</th>
</tr>
</thead>
<tbody>
<tr>
<td>-1 -1 -1</td>
</tr>
<tr>
<td>-1 8 -1</td>
</tr>
<tr>
<td>-1 -1 24</td>
</tr>
<tr>
<td>-1 -1 -1</td>
</tr>
</tbody>
</table>

The anchor cell is the center cell of the kernel, highlighted in red.

**Return Values**

- ippStsNoErr: Indicates no error.
- ippStsNullPtrErr: Indicates an error when one of the specified pointers is NULL.
- ippStsSizeErr: Indicates an error when dstRoiSize is negative, or equal to zero.
- ippStsBorderErr: Indicates an error when mask has an illegal value.
- ippStsBorderErr: Indicates an error when borderType has an illegal value.
See Also
Borders in Neighborhood Operations
Regions of Interest in Intel IPP
User-defined Border Types
FilterHipassBorderGetBufferSize Computes the size of the work buffer for high-pass filtering with
the ippiFilterHipassBorder function.

FilterLaplaceBorderGetBufferSize
*Computes the size of the work buffer for Laplace filtering.*

Syntax
IppStatus ippiFilterLaplaceBorderGetBufferSize(IppiSize dstRoiSize, IppiMaskSize mask,
IppDataType srcDataType, IppDataType dstDataType, int numChannels, int* pBufferSize);

Include Files
ippi.h

Domain Dependencies
Headers: ippcore.h, ippvm.h, ipps.h
Libraries: ippcore.lib, ippvm.lib, ipps.lib

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>dstRoiSize</td>
<td>Size of the destination ROI, in pixels.</td>
</tr>
<tr>
<td>mask</td>
<td>Predefined mask of IppiMaskSize type. Possible values are: ippMskSize3x3 or ippMskSize5x5.</td>
</tr>
<tr>
<td>srcDataType</td>
<td>Data type of the source image.</td>
</tr>
<tr>
<td>dstDataType</td>
<td>Data type of the destination image.</td>
</tr>
<tr>
<td>numChannels</td>
<td>Number of channels in the image. Possible values are 1, 3, or 4.</td>
</tr>
<tr>
<td>pBufferSize</td>
<td>Pointer to the size, in bytes, of the external buffer.</td>
</tr>
</tbody>
</table>

Description
This function computes the size, in bytes, of the external work buffer for the ippiFilterLaplaceBorder function. The result is stored in the pBufferSize parameter.

Return Values

<table>
<thead>
<tr>
<th>Return Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ippStsNoErr</td>
<td>Indicates no error. Any other value indicates an error.</td>
</tr>
<tr>
<td>ippStsNullPtrErr</td>
<td>Indicates an error when pBufferSize is NULL.</td>
</tr>
<tr>
<td>ippStsSizeErr</td>
<td>Indicates an error when dstRoiSize has a field with a zero or negative value.</td>
</tr>
<tr>
<td>ippStsMaskSizeErr</td>
<td>Indicates an error when mask has an illegal value.</td>
</tr>
<tr>
<td>ippStsDataTypeErr</td>
<td>Indicates an error when srcDataType or dstDataType has an illegal value.</td>
</tr>
</tbody>
</table>
See Also
FilterLaplaceBorder Filters an image using a Laplace filter.

FilterLaplaceBorder
Filters an image using a Laplace filter.

Syntax
Case 1: Operating on one-channel data

IppStatus ippiFilterLaplaceBorder_8u_C1R(const Ipp8u* pSrc, int srcStep, Ipp8u* pDst, int dstStep, IppiSize dstRoiSize, IppiMaskSize mask, IppiBorderType borderType, Ipp8u borderValue, Ipp8u* pBuffer);

IppStatus ippiFilterLaplaceBorder_16s_C1R(const Ipp16s* pSrc, int srcStep, Ipp16s* pDst, int dstStep, IppiSize dstRoiSize, IppiMaskSize mask, IppiBorderType borderType, Ipp16s borderValue, Ipp8u* pBuffer);

IppStatus ippiFilterLaplaceBorder_32f_C1R(const Ipp32f* pSrc, int srcStep, Ipp32f* pDst, int dstStep, IppiSize dstRoiSize, IppiMaskSize mask, IppiBorderType borderType, Ipp32f borderValue, Ipp8u* pBuffer);

Case 2: Operating on multi-channel data

IppStatus ippiFilterLaplaceBorder_8u_C3R(const Ipp8u* pSrc, int srcStep, Ipp8u* pDst, int dstStep, IppiSize dstRoiSize, IppiMaskSize mask, IppiBorderType borderType, const Ipp8u pBorderValue[3], Ipp8u* pBuffer);

IppStatus ippiFilterLaplaceBorder_16s_C3R(const Ipp16s* pSrc, int srcStep, Ipp16s* pDst, int dstStep, IppiSize dstRoiSize, IppiMaskSize mask, IppiBorderType borderType, const Ipp16s pBorderValue[3], Ipp8u* pBuffer);

IppStatus ippiFilterLaplaceBorder_32f_C3R(const Ipp32f* pSrc, int srcStep, Ipp32f* pDst, int dstStep, IppiSize dstRoiSize, IppiMaskSize mask, IppiBorderType borderType, const Ipp32f pBorderValue[3], Ipp8u* pBuffer);

IppStatus ippiFilterLaplaceBorder_8u_AC4R(const Ipp8u* pSrc, int srcStep, Ipp8u* pDst, int dstStep, IppiSize dstRoiSize, IppiMaskSize mask, IppiBorderType borderType, const Ipp8u pBorderValue[3], Ipp8u* pBuffer);

IppStatus ippiFilterLaplaceBorder_16s_AC4R(const Ipp16s* pSrc, int srcStep, Ipp16s* pDst, int dstStep, IppiSize dstRoiSize, IppiMaskSize mask, IppiBorderType borderType, const Ipp16s pBorderValue[3], Ipp8u* pBuffer);

IppStatus ippiFilterLaplaceBorder_32f_AC4R(const Ipp32f* pSrc, int srcStep, Ipp32f* pDst, int dstStep, IppiSize dstRoiSize, IppiMaskSize mask, IppiBorderType borderType, const Ipp32f pBorderValue[3], Ipp8u* pBuffer);
IppStatusippiFilterLaplaceBorder_32f_C4R(constIpp32f*pSrc, int srcStep, Ipp32f*pDst, int dstStep, IppiSize dstRoiSize, IppiMaskSize mask, IppiBorderTypeborderType, constIpp32fpBorderValue[4], Ipp8upBuffer);

Include Files
ippi.h

Domain Dependencies
Headers: ippcore.h, ippvm.h, ipps.h
Libraries: ippcore.lib, ippvm.lib, ipps.lib

Parameters
pSrc Pointer to the source image ROI.
srcStep Distance, in bytes, between the starting points of consecutive lines in the source image.
pDst Pointer to the destination image ROI.
dstStep Distance, in bytes, between the starting points of consecutive lines in the destination image.
dstRoiSize Size of the source and destination ROI, in pixels.
mask Predefined mask of IppiMaskSize. Possible values are ippMskSize3x3 or ippMskSize5x5.
borderType Type of border. Possible values are:
ippBorderConst Values of all border pixels are set to constant.
ippBorderRepl Border is replicated from the edge pixels.
ippBorderInMem Border is obtained from the source image pixels in memory.

Mixed borders are also supported. They can be obtained by the bitwise operation OR between any of the ippBorderRepl, ippBorderConst, or ippBorderMirror values and the ippBorderInMemTop, ippBorderInMemBottom, ippBorderInMemLeft, ippBorderInMemRight values.

borderValue Constant value to assign to pixels of the constant border. This parameter is applicable only to the ippBorderConst border type.
pBorderValue[3] Pointer to constant values to assign to pixels of the constant border. This parameter is applicable only to the ippBorderConst border type.
pBorderValue[4]
pBuffer Pointer to the work buffer.

Description
Before using this function, you need to compute the size of the work buffer pBuffer using theippiFilterLaplaceBorderGetBufferSize function.

This function operates with ROI (see Regions of Interest in Intel IPP).
This function applies a Laplace filter to the \textit{pSrc} source image ROI. The size of the source image ROI is equal to the destination image ROI size \textit{dstRoiSize}. The values of border pixels are assigned in accordance with the \textit{borderType} and \textit{borderValue} parameters. The kernel of the filter is a matrix of 3x3 or 5x5 size depending on the \textit{mask} value. The kernels have the following values:

\begin{verbatim}
-1 -3 -4 -3 -1
-1 -1 -1
-1 8 -1
-1 -1 -1
-1 -3 -4 -3 -1
\end{verbatim}

The anchor cell is the center cell of the kernel, highlighted in red.

**Return Values**

- \textbf{ippStsNoErr} Indicates no error.
- \textbf{ippStsNullPtrErr} Indicates an error when one of the specified pointers is \texttt{NULL}.
- \textbf{ippStsSizeErr} Indicates an error when \textit{dstRoiSize} is negative, or equal to zero.
- \textbf{ippStsBorderErr} Indicates an error when \textit{mask} has an illegal value.
- \textbf{ippStsBorderErr} Indicates an error when \textit{borderType} has an illegal value.

**See Also**

Borders in Neighborhood Operations
Regions of Interest in Intel IPP
User-defined Border Types
FilterLaplaceBorder Filters an image using a Laplace filter.

\textbf{FilterLaplacianGetBufferSize}

\textit{Computes the size of the external buffer for the Laplace filter with border.}

**Syntax**

```
IppStatus ippiFilterLaplacianGetBufferSize_<mod>(IppiSize roiSize, IppiMaskSize mask, int* pBufferSize);
```

**Supported values for \textit{mod}:**

- 8u16s_C1R
- 32f_C1R

**Include Files**

ippcv.h

**Domain Dependencies**

**Headers:** ippcore.h, ippvm.h, ipps.h,ippi.h

**Libraries:** ippcore.lib, ippvm.lib, ipps.lib,ippi.lib
Parameters

roiSize

Maximum size of the source and destination image ROI.

mask

Predefined mask of IppiMaskSize type.

pBufferSize

Pointer to the buffer size.

Description

This function computes the size of the external buffer that is required for the filter function ippiFilterLaplacianBorder. The kernel of the filter is the matrix of either 3x3 or 5x5 size that is specified by the parameter mask (see Table “Types of the Fixed Filter Functions”). This buffer pBufferSize[0] can be used to filter an image whose width and height are equal to or less than corresponding fields of roiSize.

Return Values

ippStsNoErr

Indicates no error. Any other value indicates an error or a warning.

ippStsNullPtrErr

Indicates an error condition if the pBufferSize pointer is NULL.

ippStsSizeErr

Indicates an error condition if roiSize has a field with a zero or negative value.

ippStsMaskSizeErr

Indicates an error condition if mask has a wrong value.

FilterLaplacianBorder

Applies Laplacian filter with border.

Syntax

IppStatus ippiFilterLaplacianBorder_<mod>(const Ipp<srcDatatype>* pSrc, int srcStep, Ipp<dstDatatype>* pDst, int dstStep, IppiSize roiSize, IppiMaskSize mask, IppiBorderType borderType, Ipp<srcDatatype> borderValue, Ipp8u* pBuffer);

Supported values for mod:

8u16s_C1R  32f_C1R

Include Files

ippcv.h

Domain Dependencies

Headers: ippcore.h, ippvm.h, ipps.h,ippi.h
Libraries: ippcore.lib, ippvm.lib, ipps.lib,ippi.lib

Parameters

pSrc

Pointer to the source image ROI.

srcStep

Distance in bytes between starts of consecutive lines in the source image.

pDst

Pointer to the destination image ROI.
**dstStep**
Distance in bytes between starts of consecutive lines in the destination image.

**roiSize**
Size of the source and destination image ROI.

**mask**
Type of the filter kernel.

**borderType**
Type of border (see Borders in Neighborhood Operations); following values are possible:
- `ippBorderConst`
  Values of all border pixels are set to constant.
- `ippBorderRepl`
  Replicated border is used.
- `ippBorderMirror`
  Border pixels are mirrored from the source image boundary pixels.

**borderValue**
The constant value to assign to the pixels in the constant border (not applicable for other border's type).

**pBuffer**
Pointer to the working buffer.

### Description
This function operates with ROI (see Regions of Interest in Intel IPP).

This function applies the laplacian filter to the source image `pSrc` and stores results to the destination image of the same size `pDst`. Source image can be used as the destination image if they both have the same data type. The values of border pixels are assigned in accordance with the `borderType` and `borderValue` parameters. The kernel of this filter is the matrix of either 3x3 or 5x5 size that is specified by the parameter `mask`. The kernels have the following values with the anchor in the center cell (red):

```
  2  4  4  4  2
  2  0  2        4  0 -8  0  4
 0 -8  0  or 4 -8 -24 -8  4
  2  0  2        4  0 -8  0  4
  2  4  4  4  2
```

The function requires the working buffer `pBuffer` whose size should be computed by the function `ippiFilterLaplacianGetBufferSize` beforehand.

### Return Values
- `ippStsNoErr` Indicates no error. Any other value indicates an error or a warning.
- `ippStsNullPtrErr` Indicates an error condition if one of the specified pointers is NULL.
- `ippStsSizeErr` Indicates an error condition if `roiSize` has a field with a zero or negative value.
- `ippStsStepErr` Indicates an error condition if `srcStep` or `dstStep` is less than `roiSize.width` * `<pixelSize>`.
- `ippStsNotEvenStepErr` Indicates an error condition if one of the step values is not divisible by 4 for floating-point images, or by 2 for short-integer images.
- `ippStsBadArgErr` Indicates an error if `borderType` or `divisor` has a wrong value.
Indicates an error condition if mask has a wrong value.

FilterLowpassGetBufferSize

Computes the size of the external buffer for the lowpass filter with border.

Syntax

IppStatus ippiFilterLowpassGetBufferSize_8u_C1R(IppiSize roiSize, IppiMaskSize mask, int* pBufferSize);
IppStatus ippiFilterLowpassGetBufferSize_32f_C1R(IppiSize roiSize, IppiMaskSize mask, int* pBufferSize);

Include Files

ippcv.h

Domain Dependencies

Headers: ippcore.h, ippvm.h, ipps.h, ippi.h
Libraries: ippcore.lib, ippvm.lib, ipps.lib, ippi.lib

Parameters

roiSize Maximum size of the source and destination image ROI.
mask Predefined mask of IppiMaskSize type.
pBufferSize Pointer to the buffer size.

Description

This function computes the size of the external buffer that is required for the filter function ippiFilterLowpassBorder. The kernel of the filter is the matrix of either 3x3 or 5x5 size that is specified by the parameter mask (see Table "Types of the Fixed Filter Functions"). This buffer pBufferSize[0] can be used to filter an image whose width and height are equal to or less than corresponding fields of roiSize.

Return Values

ippStsNoErr Indicates no error. Any other value indicates an error or a warning.
ippStsNullPtrErr Indicates an error condition if the pBufferSize pointer is NULL.
ippStsSizeErr Indicates an error condition if roiSize has a field with a zero or negative value.
ippStsMaskSizeErr Indicates an error condition if mask has a wrong value.

FilterLowpassBorder

Applies lowpass filter with border.

Syntax

IppStatus ippiFilterLowpassBorder_8u_C1R(const Ipp8u* pSrc, int srcStep, Ipp8u* pDst, int dstStep, IppiSize roiSize, IppiMaskSize mask, IppiBorderType borderType, Ipp8u borderValue, Ipp8u* pBuffer);
IppStatus ippiFilterLowpassBorder_32f_C1R(const Ipp32f* pSrc, int srcStep, Ipp32f* pDst, int dstStep, IppiSize roiSize, IppiMaskSize mask, IppiBorderType borderType, Ipp32f borderValue, Ipp8u* pBuffer);

Include Files
ippcv.h

Domain Dependencies
Headers: ippcore.h, ippvm.h, ipps.h,ippi.h
Libraries: ippcore.lib, ippvm.lib, ipps.lib,ippi.lib

Parameters

- `pSrc`: Pointer to the source image ROI.
- `srcStep`: Distance in bytes between starts of consecutive lines in the source image.
- `pDst`: Pointer to the destination image ROI.
- `dstStep`: Distance in bytes between starts of consecutive lines in the destination image.
- `roiSize`: Size of the source and destination image ROI.
- `mask`: Type of the filter kernel.
- `borderType`: Type of border (see Borders in Neighborhood Operations); the following values are possible:
  - ippBorderConst: Values of all border pixels are set to constant.
  - ippBorderRepl: Replicated border is used.
- `borderValue`: The constant value to assign to the pixels in the constant border (not applicable for other border's type).
- `pBuffer`: Pointer to the working buffer.

Description

This function operates with ROI (see Regions of Interest in Intel IPP).

This function applies the lowpass filter (blur operation) to the source image `pSrc` and stores results to the destination image of the same size `pDst`. Source image can be used as the destination image if they both have the same data type. The values of border pixels are assigned in accordance with the `borderType` and `borderValue` parameters. The kernel of this filter is the matrix of either 3x3 or 5x5 size that is specified by the parameter `mask`. The anchor cell is the center cell (red) of the kernel.

The 3x3 filter uses the kernel with the following values:

<table>
<thead>
<tr>
<th>1/9</th>
<th>1/9</th>
<th>1/9</th>
</tr>
</thead>
<tbody>
<tr>
<td>1/9</td>
<td>1/9</td>
<td>1/9</td>
</tr>
<tr>
<td>1/9</td>
<td>1/9</td>
<td>1/9</td>
</tr>
</tbody>
</table>
The 5x5 filter uses the kernel with the following values:

```
1/25  1/25  1/25  1/25  1/25
1/25  1/25  1/25  1/25  1/25
1/25  1/25  1/25  1/25  1/25
1/25  1/25  1/25  1/25  1/25
1/25  1/25  1/25  1/25  1/25
```

The function requires the working buffer `pBuffer` whose size should be computed by the function `ippiFilterLowpassGetBufferSize` beforehand.

**Return Values**

- **ippStsNoErr**: Indicates no error. Any other value indicates an error or a warning.
- **ippStsNullPtrErr**: Indicates an error condition if one of the specified pointers is NULL.
- **ippStsSizeErr**: Indicates an error condition if `roiSize` has a field with a zero or negative value.
- **ippStsStepErr**: Indicates an error condition if `srcStep` or `dstStep` is less than `roiSize.width * <pixelSize>.
- **ippStsNotEvenStepErr**: Indicates an error condition if one of the step values is not divisible by 4 for floating-point images.
- **ippStsBorderErr**: Indicates an error condition if `borderType` has a wrong value.
- **ippStsMaskErr**: Indicates an error condition if `mask` has a wrong value.

**FilterPrewittHorizBorderGetBufferSize**

*Computes the size of the work buffer for the Prewitt Horizontal filter.*

**Syntax**

```c
IppStatus ippiFilterPrewittHorizBorderGetBufferSize (IppiSize dstRoiSize, IppiMaskSize mask, IppDataType srcDataType, IppDataType dstDataType, int numChannels, int* pBufferSize);
```

**Include Files**

`ippi.h`

**Parameters**

- **dstRoiSize**: Size of the destination ROI in pixels.
- **mask**: Predefined mask of `IppiMaskSize` type. Possible value is `ippMskSize3x3`.
- **srcDataType**: Data type of the source image.
- **dstDataType**: Data type of the destination image.
- **numChannels**: Number of channels in the image. Possible value is 1.
pBufferSize  
Pointer to the size of the external work buffer.

Description
The ippiFilterPrewittHorizBorderGetBufferSize function computes the size, in bytes, of the external work buffer needed for the ippiFilterPrewittHorizBorder function. The result is stored in the pBufferSize parameter.

For an example on how to use this function, see the code example provided with the ippiFilterPrewittHorizBorder function description.

Return Values
ippStsNoErr Indicates no error.
ippStsNullPtrErr Indicates an error when pBufferSize is NULL.
ippStsSizeErr Indicates an error when dstRoiSize is negative, or equal to zero.
ippStsMaskSizeErr Indicates an error when mask has an illegal value.
ippStsDataTypeErr Indicates an error when srcDataType or dstDataType has an illegal value.
ippStsNumChannelsError Indicates an error when numChannels has an illegal value.

See Also
FilterPrewittHorizBorder Filters and image using a horizontal Prewitt filter.

FilterPrewittHorizBorder
Filters and image using a horizontal Prewitt filter.

Syntax
IppStatus ippiFilterPrewittHorizBorder_<mod>(const Ipp<srcDatatype>* pSrc, int srcStep, Ipp<dstDatatype>* pDst, int dstStep, IppiSize dstRoiSize, IppiMaskSize mask, IppiBorderType borderType, Ipp<srcDatatype> borderValue, Ipp8u* pBuffer);

Supported values for mod:
8u16s_C1R 16s_C1R 32f_C1R

Include Files
ippi.h

Domain Dependencies
Headers: ippcore.h, ippvm.h, ipps.h
Libraries: ippcore.lib, ippvm.lib, ipps.lib

Parameters

pSrc  
Pointer to the source image ROI.

srcStep  
Distance, in bytes, between the starting points of consecutive lines in the source image.
**pDst**
Pointer to the destination image ROI.

**dstStep**
Distance, in bytes, between the starting points of consecutive lines in the destination image.

**dstRoiSize**
Size of the source and destination ROI in pixels.

**mask**
Predefined mask of IppiMaskSize. Possible value is ippMskSize3x3.

**borderType**
Type of border. Possible values are:
- ippBorderConst: Values of all border pixels are set to constant.
- ippBorderRepl: Border is replicated from the edge pixels.
- ippBorderMirror: Mirrored border is used.
- ippBorderMirrorR: Mirrored border with replication is used.
- ippBorderInMem: Border is obtained from the source image pixels in memory.

Mixed borders are also supported. They can be obtained by the bitwise operation OR between any of the ippBorderRepl, ippBorderConst, ippBorderMirror, or ippBorderMirrorR values, and the ippBorderInMemTop, ippBorderInMemBottom, ippBorderInMemLeft, ippBorderInMemRight values.

**borderValue**
Constant value to assign to pixels of the constant border. This parameter is applicable only to the ippBorderConst border type.

**pBuffer**
Pointer to the work buffer.

**Description**
Before using this function, you need to compute the size of the work buffer pBuffer using the FilterPrewittHorizBorderGetBufferSize function.

This function operates with ROI.

This function applies a horizontal Prewitt filter to the pSrc source image ROI. The size of the source image ROI is equal to the destination image ROI size dstRoiSize. The values of border pixels are assigned in accordance with the borderType and borderValue parameters. The kernel of the filter is a matrix of 3x3 size with the following values:

<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>-1</td>
<td>-1</td>
<td>-1</td>
</tr>
</tbody>
</table>

The anchor cell is the center cell of the kernel, highlighted in red.

This filter enhances horizontal edges of an image.

**Return Values**
- ippStsNoErr: Indicates no error.
- ippStsNullPtrErr: Indicates an error when one of the specified pointers is NULL.
- ippStsSizeErr: Indicates an error when dstRoiSize is negative, or equal to zero.
ippStsStepErr

Indicates an error when `srcStep` or `dstStep` is negative, or equal to zero.

ippStsNotEvenStepErr

Indicates an error when one of the step values is not divisible by 4 for floating-point images, or by 2 for short-integer images.

ippStsBorderErr

Indicates an error when `borderType` has an illegal value.

Example

To better understand usage of this function, refer to the following examples in the examples archive available for download from https://software.intel.com/en-us/ipp-manual-examples:

FilteringConvolutionKernels.c
FilterPrewittHorizBorder.c

See Also

Borders in Neighborhood Operations
Regions of Interest in Intel IPP
User-defined Border Types

FilterPrewittHorizBorderGetBufferSize Computes the size of the work buffer for the Prewitt Horizontal filter.

FilterPrewittVertBorderGetBufferSize

Computes the size of the work buffer for the Prewitt Vertical filter.

Syntax

```c
IppStatus ippiFilterPrewittVertBorderGetBufferSize (IppiSize dstRoiSize, IppiMaskSize mask, IppDataType srcDataType, IppDataType dstDataType, int numChannels, int* pBufferSize);
```

Include Files

ippi.h

Parameters

- **dstRoiSize**: Size of the destination ROI in pixels.
- **mask**: Predefined mask of `IppiMaskSize` type. Possible value is `ippMskSize3x3`.
- **srcDataType**: Data type of the source image.
- **dstDataType**: Data type of the destination image.
- **numChannels**: Number of channels in the image. Possible value is 1.
- **pBufferSize**: Pointer to the size of the external work buffer.

Description

The `ippiFilterPrewittVertBorderGetBufferSize` function computes the size, in bytes, of the external work buffer needed for the `ippiFilterPrewittVertBorder` function. The result is stored in the `pBufferSize` parameter.
For an example on how to use this function, see the code example provided with the `ippiFilterPrewittVertBorder` function description.

**Return Values**

<table>
<thead>
<tr>
<th>Return Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>ippStsNoErr</code></td>
<td>Indicates no error.</td>
</tr>
<tr>
<td><code>ippStsNullPtrErr</code></td>
<td>Indicates an error when <code>pBufferSize</code> is NULL.</td>
</tr>
<tr>
<td><code>ippStsSizeErr</code></td>
<td>Indicates an error when <code>dstRoiSize</code> is negative, or equal to zero.</td>
</tr>
<tr>
<td><code>ippStsMaskSizeErr</code></td>
<td>Indicates an error when <code>mask</code> has an illegal value.</td>
</tr>
<tr>
<td><code>ippStsDataTypeErr</code></td>
<td>Indicates an error when <code>srcDataType</code> or <code>dstDataType</code> has an illegal value.</td>
</tr>
<tr>
<td><code>ippStsNumChannelsError</code></td>
<td>Indicates an error when <code>numChannels</code> has an illegal value.</td>
</tr>
</tbody>
</table>

**See Also**

`FilterPrewittVertBorder` Filters and image using a vertical Prewitt kernel.

**FilterPrewittVertBorder**

*Filters and image using a vertical Prewitt kernel.*

**Syntax**

```c
IppStatus ippiFilterPrewittVertBorder_<mod>(const Ipp<srcDatatype>* pSrc, int srcStep, Ipp<dstDatatype>* pDst, int dstStep, IppiSize dstRoiSize, IppiMaskSize mask, IppiBorderType borderType, Ipp<srcDatatype> borderValue, Ipp8u* pBuffer);
```

Supported values for `mod`:

- `8u16s_C1R`
- `16s_C1R`
- `32f_C1R`

**Include Files**

`ippi.h`

**Domain Dependencies**

Headers: `ippcore.h`, `ippvm.h`, `ipps.h`

Libraries: `ippcore.lib`, `ippvm.lib`, `ipps.lib`

**Parameters**

- `pSrc`
  - Pointer to the source image ROI.
- `srcStep`
  - Distance, in bytes, between the starting points of consecutive lines in the source image.
- `pDst`
  - Pointer to the destination image ROI.
- `dstStep`
  - Distance, in bytes, between the starting points of consecutive lines in the destination image.
- `dstRoiSize`
  - Size of the source and destination ROI in pixels.
- `mask`
  - Predefined mask of `IppiMaskSize`. Possible value is `ippMskSize3x3`.
- `borderType`
  - Type of border. Possible values are:
ippBorderConst

Values of all border pixels are set to constant.

ippBorderRepl

Border is replicated from the edge pixels.

ippBorderMirror

Mirrored border is used.

ippBorderMirrorR

Mirrored border with replication is used.

ippBorderInMem

Border is obtained from the source image pixels in memory.

Mixed borders are also supported. They can be obtained by the bitwise operation OR between any of the ippBorderRepl, ippBorderConst, ippBorderMirror, or ippBorderMirrorR values and the ippBorderInMemTop, ippBorderInMemBottom, ippBorderInMemLeft, ippBorderInMemRight values.

borderValue

Constant value to assign to pixels of the constant border. This parameter is applicable only to the ippBorderConst border type.

pBuffer

Pointer to the work buffer.

Description

Before using this function, you need to compute the size of the work buffer pBuffer using the FilterPrewittVertBorderGetBufferSize function.

This function operates with ROI.

This function applies a vertical Prewitt filter to the pSrc source image ROI. The size of the source image ROI is equal to the destination image ROI size dstRoiSize. The values of border pixels are assigned in accordance with the borderType and borderValue parameters. The kernel of the filter is a matrix of 3x3 size with the following values:

```
-1  0  1
-1  0  1
-1  0  1
```

The anchor cell is the center cell of the kernel, highlighted in red.

This filter enhances vertical edges of an image.

Return Values

ippStsNoErr

Indicates no error.

ippStsNullPtrErr

Indicates an error when one of the specified pointers is NULL.

ippStsSizeErr

Indicates an error when dstRoiSize is negative, or equal to zero.

ippStsStepErr

Indicates an error when srcStep or dstStep is negative, or equal to zero.

ippStsNotEvenStepErr

Indicates an error when one of the step values is not divisible by 4 for floating-point images, or by 2 for short-integer images.

ippStsBorderErr

Indicates an error when borderType has an illegal value.
Example
To better understand usage of this function, refer to the following example in the examples archive available for download from https://software.intel.com/en-us/ipp-manual-examples:
FilteringConvolutionKernels.c

See Also
Borders in Neighborhood Operations
Regions of Interest in Intel IPP
User-defined Border Types
FilterPrewittVertBorderGetBufferSize Computes the size of the work buffer for the Prewitt Vertical filter.

FilterRobertsUpBorderGetBufferSize
*Computes the size of the work buffer for the vertical Roberts edge filter.*

Syntax
IppStatus ippiFilterRobertsUpBorderGetBufferSize (IppiSize dstRoiSize, IppiMaskSize mask, IppDataType srcDataType, IppDataType dstDataType, int numChannels, int* pBufferSize);

Include Files
ippi.h

Parameters

dstRoiSize
Size of the destination ROI in pixels.

mask
Predefined mask of IppiMaskSize type. Possible value is ippMskSize3x3.

srcDataType
Data type of the source image.

dstDataType
Data type of the destination image.

numChannels
Number of channels in the image. Possible value is 1.

pBufferSize
Pointer to the size of the external work buffer.

Description
The ippiFilterRobertsUpBorderGetBufferSize function computes the size, in bytes, of the external work buffer needed for the ippiFilterRobertsUpBorder function. The result is stored in the pBufferSize parameter.
For an example on how to use this function, see the code example provided with the ippiFilterRobertsUpBorder function description.

Return Values

ippStsNoErr
Indicates no error.

ippStsNullPtrErr
Indicates an error when pBufferSize is NULL.

ippStsSizeErr
Indicates an error when dstRoiSize is negative, or equal to zero.
ippStsMaskSizeErr Indicates an error when mask has an illegal value.

ippStsDataTypeErr Indicates an error when srcDataType or dstDataType has an illegal value.

ippStsNumChannelsError Indicates an error when numChannels has an illegal value.

See Also
FilterRobertsUpBorder Filters an image using a vertical Roberts edge filter.

FilterRobertsUpBorder
Filters an image using a vertical Roberts edge filter.

Syntax
IppStatus ippiFilterRobertsUpBorder_<mod>(const Ipp<srcDatatype>* pSrc, int srcStep, Ipp<dstDatatype>* pDst, int dstStep, IppiSize dstRoiSize, IppiMaskSize mask, IppiBorderType borderType, Ipp<srcDatatype> borderValue, Ipp8u* pBuffer);

Supported values for mod:
8u16s_C1R 16s_C1R 32f_C1R

Include Files
ippi.h

Domain Dependencies
Headers: ippcore.h, ippvm.h, ipps.h
Libraries: ippcore.lib, ippvm.lib, ipps.lib

Parameters
pSrc Pointer to the source image ROI.
srcStep Distance in bytes between starting points of consecutive lines in the source image.
pDst Pointer to the destination image ROI.
dstStep Distance in bytes between starting points of consecutive lines in the destination image.
dstRoiSize Size of the source and destination ROI in pixels.
mask Predefined mask of IppiMaskSize. Possible value is ippMskSize3x3.
borderType Type of border. Possible values are:
    ippBorderConst Values of all border pixels are set to constant.
    ippBorderRepl Border is replicated from the edge pixels.
    ippBorderMirror Mirrored border is used.
    ippBorderMirrorR Mirrored border with replication is used.
    ippBorderInMem Border is obtained from the source image pixels in memory.
Mixed borders are also supported. They can be obtained by the bitwise operation \texttt{OR} between any of the \texttt{ippBorderRepl}, \texttt{ippBorderConst}, \texttt{ippBorderMirror}, or \texttt{ippBorderMirrorR} values and the \texttt{ippBorderInMemTop}, \texttt{ippBorderInMemBottom}, \texttt{ippBorderInMemLeft}, or \texttt{ippBorderInMemRight} values.

\textit{borderValue} 

Constant value to assign to pixels of the constant border. This parameter is applicable only to the \texttt{ippBorderConst} border type.

\textit{pBuffer} 

Pointer to the work buffer.

\textbf{Description} 

This function operates with ROI.

This function applies a vertical Roberts edge filter to the \textit{pSrc} source image ROI. The size of the source image ROI is equal to the destination image ROI size \textit{dstRoiSize}. The values of border pixels are assigned in accordance with the \textit{borderType} and \textit{borderValue} parameters. The kernel of the filter is a matrix of 3x3 size with the following values:

\begin{center}
\begin{tabular}{ccc}
0 & 0 & 0 \\
0 & 1 & 0 \\
-1 & 0 & 0 \\
\end{tabular}
\end{center}

The anchor cell is the center cell of the kernel, highlighted in red.

This filter provides the gross approximation of the pixel values gradient in the vertical direction.

Before using this function, you need to compute the size of the work buffer \textit{pBuffer} using the \texttt{FilterRobertsUpBorderGetBufferSize} function.

\textbf{Return Values} 

- \texttt{ippStsNoErr} Indicates no error.
- \texttt{ippStsNullPtrErr} Indicates an error when one of the specified pointers is NULL.
- \texttt{ippStsSizeErr} Indicates an error when \textit{dstRoiSize} is negative, or equal to zero.
- \texttt{ippStsStepErr} Indicates an error when \textit{srcStep} or \textit{dstStep} is negative, or equal to zero.
- \texttt{ippStsNotEvenStepErr} Indicates an error when one of the step values is not divisible by 4 for floating-point images, or by 2 for short-integer images.
- \texttt{ippStsBorderErr} Indicates an error when \textit{borderType} has an illegal value.

\textbf{Example} 

To better understand usage of this function, refer to the following example in the examples archive available for download from https://software.intel.com/en-us/ipp-manual-examples:

\texttt{FilteringConvolutionKernels.c}

\textbf{See Also} 

Borders in Neighborhood Operations 
Regions of Interest in Intel IPP 
User-defined Border Types
FilterRobertsUpBorderGetBufferSize Computes the size of the work buffer for the vertical Roberts edge filter.

FilterRobertsDownBorderGetBufferSize

*Computes the size of the work buffer for the horizontal Roberts edge filter.*

**Syntax**

```c
IppStatus ippiFilterRobertsDownBorderGetBufferSize (IppiSize dstRoiSize, IppiMaskSize mask, IppDataType srcDataType, IppDataType dstDataType, int numChannels, int* pBufferSize);
```

**Include Files**

ippi.h

**Parameters**

- **dstRoiSize**
  
  Size of the destination ROI in pixels.

- **mask**
  
  Predefined mask of IppiMaskSize type. Possible value is ippMskSize3x3.

- **srcDataType**
  
  Data type of the source image.

- **dstDataType**
  
  Data type of the destination image.

- **numChannels**
  
  Number of channels in the image. Possible value is 1.

- **pBufferSize**
  
  Pointer to the size of the external work buffer.

**Description**

The `ippiFilterRobertsDownBorderGetBufferSize` function computes the size, in bytes, of the external work buffer needed for the `ippiFilterRobertsDownBorder` function. The result is stored in the `pBufferSize` parameter.

For an example on how to use this function, see the code example provided with the `ippiFilterRobertsDownBorder` function description.

**Return Values**

- **ippStsNoErr**
  
  Indicates no error.

- **ippStsNullPtrErr**
  
  Indicates an error when `pBufferSize` is NULL.

- **ippStsSizeErr**
  
  Indicates an error when `dstRoiSize` is negative, or equal to zero.

- **ippStsMaskSizeErr**
  
  Indicates an error when `mask` has an illegal value.

- **ippStsDataTypeErr**
  
  Indicates an error when `srcDataType` or `dstDataType` has an illegal value.

- **ippStsNumChannelsError**
  
  Indicates an error when `numChannels` has an illegal value.

FilterRobertsDownBorder Filters an image using a horizontal Roberts edge filter.
FilterRobertsDownBorder
Filters an image using a horizontal Roberts edge filter.

Syntax
IppStatus ippiFilterRobertsDownBorder_<mod>(const Ipp<srcDatatype>* pSrc, int srcStep, Ipp<dstDatatype>* pDst, int dstStep, IppiSize dstRoiSize, IppiMaskSize mask, IppiBorderType borderType, Ipp<srcDatatype> borderValue, Ipp8u* pBuffer);

Supported values for mod:
8u16s_C1R  16s_C1R  32f_C1R

Include Files
ippi.h

Domain Dependencies
Headers:  ippcore.h, ippvm.h, ipps.h
Libraries:  ippcore.lib, ippvm.lib, ipps.lib

Parameters
pSrc
Pointer to the source image ROI.
srcStep
Distance, in bytes, between the starting points of consecutive lines in the source image.
pDst
Pointer to the destination image ROI.
dstStep
Distance, in bytes, between the starting points of consecutive lines in the destination image.
dstRoiSize
Size of the source and destination ROI in pixels.
mask
Predefined mask of IppiMaskSize. Possible value is ippMskSize3x3.
borderType
Type of border. Possible values are:
ippBorderConst  Values of all border pixels are set to constant.
ippBorderRepl  Border is replicated from the edge pixels.
ippBorderMirror  Mirrored border is used.
ippBorderMirrorR  Mirrored border with replication is used.
ippBorderInMem  Border is obtained from the source image pixels in memory.

Mixed borders are also supported. They can be obtained by the bitwise operation OR between any of the ippBorderRepl, ippBorderConst, ippBorderMirror, or ippBorderMirrorR values, and the ippBorderInMemTop, ippBorderInMemBottom, ippBorderInMemLeft, ippBorderInMemRight values.

borderValue
Constant value to assign to pixels of the constant border. This parameter is applicable only to the ippBorderConst border type.

pBuffer
Pointer to the work buffer.
Description
Before using this function, you need to compute the size of the work buffer \textit{pBuffer} using the \textit{FilterRobertsDownBorderGetBufferSize} function.

This function operates with ROI.

This function applies a horizontal Roberts edge filter to the \textit{pSrc} source image ROI. The size of the source image ROI is equal to the destination image ROI size \textit{dstRoiSize}. The values of border pixels are assigned in accordance with the \textit{borderType} and \textit{borderValue} parameters. The kernel of the filter is a matrix of 3x3 size with the following values:

\begin{verbatim}
0   0   0
0 1 0
0 0 -1
\end{verbatim}

The anchor cell is the center cell of the kernel, highlighted in red.

This filter provides the gross approximation of the pixel values gradient in the horizontal direction.

Return Values
\begin{itemize}
\item \textbf{ippStsNoErr} Indicates no error.
\item \textbf{ippStsNullPtrErr} Indicates an error when one of the specified pointers is \textit{NULL}.
\item \textbf{ippStsSizeErr} Indicates an error when \textit{dstRoiSize} is negative, or equal to zero.
\item \textbf{ippStsStepErr} Indicates an error when \textit{srcStep} or \textit{dstStep} is negative, or equal to zero.
\item \textbf{ippStsNotEvenStepErr} Indicates an error when one of the step values is not divisible by 4 for floating-point images, or by 2 for short-integer images.
\item \textbf{ippStsBorderErr} Indicates an error when \textit{borderType} has an illegal value.
\end{itemize}

Example
To better understand usage of this function, refer to the following example in the examples archive available for download from https://software.intel.com/en-us/ipp-manual-examples :

FilteringConvolutionKernels.c

See Also
Borders in Neighborhood Operations
Regions of Interest in Intel IPP
User-defined Border Types
\textbf{FilterRobertsDownBorderGetBufferSize} Computes the size of the work buffer for the horizontal Roberts edge filter.

\textbf{FilterScharrHorizMaskBorderGetBufferSize}
\textit{Computes the size of the work buffer for the Scharr Horizontal filter.}
Syntax

IppStatus ippiFilterScharrHorizMaskBorderGetBufferSize (IppiSize dstRoiSize, IppiMaskSize mask, IppDataType srcDataType, IppDataType dstDataType, int numChannels, int* pBufferSize);

Include Files
ippi.h

Parameters

dstRoiSize
Size of the destination ROI in pixels.

mask
Predefined mask of IppiMaskSize type. Possible value is ippMskSize3x3.

srcDataType
Data type of the source image.

dstDataType
Data type of the destination image.

numChannels
Number of channels in the image. Possible value is 1.

pBufferSize
Pointer to the size of the external work buffer.

Description

The ippiFilterScharrHorizMaskBorderGetBufferSize function computes the size, in bytes, of the external work buffer needed for the ippiFilterScharrHorizMaskBorder function. The result is stored in the pBufferSize parameter.

For an example on how to use this function, see the code example provided with the ippiFilterScharrHorizMaskBorder function description.

Return Values

ippiStsNoErr
Indicates no error.

ippiStsNullPtrErr
Indicates an error when pBufferSize is NULL.

ippiStsSizeErr
Indicates an error when dstRoiSize is negative, or equal to zero.

ippiStsMaskSizeErr
Indicates an error when mask has an illegal value.

ippiStsDataTypeErr
Indicates an error when srcDataType or dstDataType has an illegal value.

ippiStsNumChannelsError
Indicates an error when numChannels has an illegal value.

See Also
FilterScharrHorizMaskBorder Filters an image using a horizontal Scharr filter.

FilterScharrHorizMaskBorder
Filters an image using a horizontal Scharr filter.
Syntax

IppStatus ippiFilterScharrHorizMaskBorder_<mod>(const Ipp<srcDatatype>* pSrc, int srcStep, Ipp<dstDatatype>* pDst, int dstStep, IppiSize dstRoiSize, IppiMaskSize mask, IppiBorderType borderType, Ipp<srcDatatype> borderValue, Ipp8u* pBuffer);

Supported values for mod:

8u16s_C1R 16s_C1R 32f_C1R

Include Files

ippi.h

Domain Dependencies

Headers: ippcore.h, ippvm.h, ipps.h
Libraries: ippcore.lib, ippvm.lib, ipps.lib

Parameters

pSrc
Pointer to the source image ROI.
srcStep
Distance, in bytes, between the starting points of consecutive lines in the source image.
pDst
Pointer to the destination image ROI.
dstStep
Distance, in bytes, between the starting points of consecutive lines in the destination image.
dstRoiSize
Size of the source and destination ROI in pixels.
mask
Predefined mask of IppiMaskSize. Possible value is ippMskSize3x3.
borderType
Type of border. Possible values are:
ippBorderConst
Values of all border pixels are set to constant.
ippBorderRepl
Border is replicated from the edge pixels.
ippBorderMirror
Mirrored border is used.
ippBorderMirrorR
Mirrored border with replication is used.
ippBorderInMem
Border is obtained from the source image pixels in memory.

Mixed borders are also supported. They can be obtained by the bitwise operation OR between any of the ippBorderRepl, ippBorderConst, ippBorderMirror, or ippBorderMirrorR values and the ippBorderInMemTop, ippBorderInMemBottom, ippBorderInMemLeft, ippBorderInMemRight values.
borderValue
Constant value to assign to pixels of the constant border. This parameter is applicable only to the ippBorderConst border type.
pBuffer
Pointer to the work buffer.

Description

Before using this function, you need to compute the size of the work buffer pBuffer using the FilterScharrHorizMaskBorderGetBufferSize function.
This function operates with ROI.

This function applies a horizontal Scharr filter to the \( pSrc \) source image ROI. The size of the source image ROI is equal to the destination image ROI size \( dstRoiSize \). The values of border pixels are assigned in accordance with the \( \text{borderType} \) and \( \text{borderValue} \) parameters. The kernel of the filter is a matrix of 3x3 size with the following values:

\[
\begin{bmatrix}
3 & 10 & 3 \\
0 & 0 & 0 \\
-3 & -10 & -3
\end{bmatrix}
\]

The anchor cell is the center cell of the kernel, highlighted in red.

This filter enhances horizontal edges of an image.

**Return Values**

- **ippStsNoErr**: Indicates no error.
- **ippStsNullPtrErr**: Indicates an error when one of the specified pointers is NULL.
- **ippStsSizeErr**: Indicates an error when \( dstRoiSize \) is negative, or equal to zero.
- **ippStsStepErr**: Indicates an error when \( srcStep \) or \( dstStep \) is negative, or equal to zero.
- **ippStsNotEvenStepErr**: Indicates an error when one of the step values is not divisible by 4 for floating-point images, or by 2 for short-integer images.
- **ippStsBorderErr**: Indicates an error when \( \text{borderType} \) has an illegal value.

**Example**

To better understand usage of this function, refer to the following example in the examples archive available for download from https://software.intel.com/en-us/ipp-manual-examples:

FilteringConvolutionKernels.c

**See Also**

- Borders in Neighborhood Operations
- Regions of Interest in Intel IPP
- User-defined Border Types
- FilterPrewittHorizBorderGetBufferSize Computes the size of the work buffer for the Prewitt Horizontal filter.

**FilterScharrVertMaskBorderGetBufferSize**

*Computes the size of the work buffer for the Scharr Vertical filter.*

**Syntax**

\[
\text{IppStatus \ ippiFilterScharrVertMaskBorderGetBufferSize (IppiSize \ dstRoiSize,} \\
\text{IppiMaskSize \ mask, IppDataType \ srcDataType, IppDataType \ dstDataType, int \ numChannels,} \\
\text{int* \ pBufferSize)};
\]

**Include Files**

ippi.h
Parameters

- **dstRoiSize**: Size of the destination ROI in pixels.
- **mask**: Predefined mask of IppiMaskSize type. Possible value is ippMskSize3x3.
- **srcDataType**: Data type of the source image.
- **dstDataType**: Data type of the destination image.
- **numChannels**: Number of channels in the image. Possible value is 1.
- **pBufferSize**: Pointer to the size of the external work buffer.

Description

The *ippiFilterScharrVertMaskBorderGetBufferSize* function computes the size, in bytes, of the external work buffer needed for the *ippiFilterScharrVertMaskBorder* function. The result is stored in the *pBufferSize* parameter.

For an example on how to use this function, see the code example provided with the *ippiFilterScharrVertMaskBorder* function description.

Return Values

- **ippStsNoErr**: Indicates no error.
- **ippStsNullPtrErr**: Indicates an error when *pBufferSize* is NULL.
- **ippStsSizeErr**: Indicates an error when *dstRoiSize* is negative, or equal to zero.
- **ippStsMaskSizeErr**: Indicates an error when *mask* has an illegal value.
- **ippStsDataTypeErr**: Indicates an error when *srcDataType* or *dstDataType* has an illegal value.
- **ippStsNumChannelsError**: Indicates an error when *numChannels* has an illegal value.

See Also

*FilterScharrVertMaskBorder* Filters an image using a vertical Scharr kernel.

*FilterScharrVertMaskBorder*

*Filters an image using a vertical Scharr kernel.*

Syntax

```c
IppStatusippiFilterScharrVertMaskBorder_<mod>(constIpp<srcDatatype>*pSrc,intsrcStep,Ipp<dstDatatype>*pDst,intdstStep,IppiSizedstRoiSize,IppiMaskSizemask,IppiBorderTypeborderType,Ipp<srcDatatype>borderValue,Ipp8upBuffer);
```

Supported values for *mod*:

- 8u16s_C1R
- 16s_C1R
- 32f_C1R

Include Files

- ippi.h
Domain Dependencies
Headers: ippcore.h, ippvm.h, ipps.h
Libraries: ippcore.lib, ippvm.lib, ipps.lib

Parameters

- **pSrc**: Pointer to the source image ROI.
- **srcStep**: Distance, in bytes, between starting points of consecutive lines in the source image.
- **pDst**: Pointer to the destination image ROI.
- **dstStep**: Distance, in bytes, between starting points of consecutive lines in the destination image.
- **dstRoiSize**: Size of the source and destination ROI in pixels.
- **mask**: Predefined mask of IppiMaskSize. Possible value is ippMskSize3x3.
- **borderType**: Type of border. Possible values are:
  - ippBorderConst: Values of all border pixels are set to constant.
  - ippBorderRepl: Border is replicated from the edge pixels.
  - ippBorderMirror: Mirrored border is used.
  - ippBorderMirrorR: Mirrored border with replication is used.
  - ippBorderInMem: Border is obtained from the source image pixels in memory.
  Mixed borders are also supported. They can be obtained by the bitwise operation OR between any of the ippBorderRepl, ippBorderConst, ippBorderMirror, or ippBorderMirrorR values and the ippBorderInMemTop, ippBorderInMemBottom, ippBorderInMemLeft, ippBorderInMemRight values.
- **borderValue**: Constant value to assign to pixels of the constant border. This parameter is applicable only to the ippBorderConst border type.
- **pBuffer**: Pointer to the work buffer.

Description

This function operates with ROI.

This function applies a vertical Scharr filter to the pSrc source image ROI. The size of the source image ROI is equal to the destination image ROI size dstRoiSize. The values of border pixels are assigned in accordance with the borderType and borderValue parameters. The kernel of the filter is a matrix of 3x3 size with the following values:

```
-3   0   3
-10  0  10
-3   0   3
```

The anchor cell is the center cell of the kernel, highlighted in red.

This filter enhances horizontal edges of an image.
Before using this function, you need to compute the size of the work buffer \( pBuffer \) using the FilterScharrVertMaskBorderGetBufferSize function.

**Return Values**

- **ippStsNoErr**
  Indicates no error.

- **ippStsNullPtrErr**
  Indicates an error when one of the specified pointers is \( NULL \).

- **ippStsSizeErr**
  Indicates an error when \( dstRoiSize \) is negative, or equal to zero.

- **ippStsStepErr**
  Indicates an error when \( srcStep \) or \( dstStep \) is negative, or equal to zero.

- **ippStsNotEvenStepErr**
  Indicates an error when one of the step values is not divisible by 4 for floating-point images, or by 2 for short-integer images.

- **ippStsBorderErr**
  Indicates an error when \( borderType \) has an illegal value.

**Example**

The code example below demonstrates how to use the \( ippiFilterScharrVertMaskBorderGetBufferSize \) and \( ippiFilterScharrVertMaskBorder_8u16s_C1R \) functions.

```c
IppStatus fix_scharrvert_8u16( void ) {
    Ipp8u pSrc[9*8] = {
        0, 1, 2, 120, 121, 122, 50, 51, 52,
        1, 2, 3, 121, 122, 123, 52, 52, 53,
        3, 4, 5, 130, 131, 132, 63, 64, 65,
        4, 5, 6, 131, 132, 133, 64, 65, 66,
        5, 6, 7, 132, 133, 134, 65, 66, 67,
        8, 7, 6, 134, 133, 132, 67, 66, 65,
        7, 6, 5, 133, 132, 131, 66, 65, 64,
        6, 5, 4, 132, 131, 130, 65, 64, 63
    };
    Ipp16s  pDst[8*7];
    Ipp8u *pBuffer;
    IppiSize roiSize = {8, 7};
    IppiBorderType borderType = ippBorderRepl | ippBorderInMemTop | ippBorderInMemRight;
    int    srcStep = 9 * sizeof(Ipp8u);
    int    dstStep = 8 * sizeof(Ipp16s);
    int    bufferSize;
    IppStatus status;
    ippiFilterScharrVertMaskBorderGetBufferSize(roiSize, ippMskSize3x3, ipp8u, ipp16s, 1,
                                              &bufferSize);
    pBuffer = ippsMalloc_8u(bufferSize);
    status = ippiFilterScharrVertMaskBorder_8u16s_C1R(pSrc + srcStep, srcStep, pDst, dstStep,
                                                    roiSize, ippMskSize3x3,
                                                    borderType, 0, pBuffer);
    ippsFree(pBuffer);
    return status;
}
```

The result is as follows:

```
pDst -->
  16  32  1925 1925  32 -1117 -1127  22
  16  32  1995 1995  32 -1094 -1097  29
  16  32  2016 2016  32 -1088 -1088  32
```
See Also
Borders in Neighborhood Operations
Regions of Interest in Intel IPP
User-defined Border Types

FilterSharpenBorderGetBufferSize Computes the size of the work buffer for image sharpening.

Syntax
IppStatus ippiFilterSharpenBorderGetBufferSize(IppiSize dstRoiSize, IppiMaskSize mask, IppDataType srcDataType, IppDataType dstDataType, int numChannels, int* pBufferSize);

Include Files
ippi.h

Domain Dependencies
Headers: ippcore.h, ippvm.h, ipps.h
Libraries: ippcore.lib, ippvm.lib, ipps.lib

Parameters

dstRoiSize        Size of the destination ROI, in pixels.
mask             Predefined mask of IppiMaskSize type. Possible value is ippMskSize3x3.
srcDataType      Data type of the source image.
dstDataType      Data type of the destination image.
numChannels      Number of channels in the image. Possible values are 1, 3, or 4.
pBufferSize      Pointer to the size, in bytes, of the external buffer.

Description
This function computes the size, in bytes, of the external work buffer for the ippiFilterSharpenBorder function. The result is stored in the pBufferSize parameter.

Return Values
ippStsNoErr      Indicates no error. Any other value indicates an error.
ippStsNullPtrErr Indicates an error when pBufferSize is NULL.
ippStsSizeErr    Indicates an error when dstRoiSize has a field with a zero or negative value.
**:ippStsMaskSizeErr**

Indicates an error when *mask* has an illegal value.

**:ippStsDataTypeErr**

Indicates an error when *srcDataType* or *dstDataType* has an illegal value.

**:ippStsNumChannelErr**

Indicates an error when *numChannels* has an illegal value.

**See Also**

*FilterSharpenBorder* Performs image sharpening with a high-pass filter.

**FilterSharpenBorder**

*Performs image sharpening with a high-pass filter.*

**Syntax**

**Case 1: Operating on one-channel data**

```c
IppStatusippiFilterSharpenBorder_8u_C1R(const Ipp8u* pSrc, int srcStep, Ipp8u* pDst, int dstStep, IppSize dstRoiSize, IppiMaskSize mask, IppiBorderType borderType, Ipp8u borderValue, Ipp8u* pBuffer);
```

```c
IppStatusippiFilterSharpenBorder_16s_C1R(const Ipp16s* pSrc, int srcStep, Ipp16s* pDst, int dstStep, IppSize dstRoiSize, IppiMaskSize mask, IppiBorderType borderType, Ipp16s borderValue, Ipp8u* pBuffer);
```

```c
IppStatusippiFilterSharpenBorder_32f_C1R(const Ipp32f* pSrc, int srcStep, Ipp32f* pDst, int dstStep, IppSize dstRoiSize, IppiMaskSize mask, IppiBorderType borderType, Ipp32f borderValue, Ipp8u* pBuffer);
```

**Case 2: Operating on multi-channel data**

```c
IppStatusippiFilterSharpenBorder_8u_C3R(const Ipp8u* pSrc, int srcStep, Ipp8u* pDst, int dstStep, IppSize dstRoiSize, IppiMaskSize mask, IppiBorderType borderType, const Ipp8u pBorderValue[3], Ipp8u* pBuffer);
```

```c
IppStatusippiFilterSharpenBorder_16s_C3R(const Ipp16s* pSrc, int srcStep, Ipp16s* pDst, int dstStep, IppSize dstRoiSize, IppiMaskSize mask, IppiBorderType borderType, const Ipp16s pBorderValue[3], Ipp8u* pBuffer);
```

```c
IppStatusippiFilterSharpenBorder_32f_C3R(const Ipp32f* pSrc, int srcStep, Ipp32f* pDst, int dstStep, IppSize dstRoiSize, IppiMaskSize mask, IppiBorderType borderType, const Ipp32f pBorderValue[3], Ipp8u* pBuffer);
```

```c
IppStatusippiFilterSharpenBorder_8u_AC4R(const Ipp8u* pSrc, int srcStep, Ipp8u* pDst, int dstStep, IppSize dstRoiSize, IppiMaskSize mask, IppiBorderType borderType, const Ipp8u pBorderValue[3], Ipp8u* pBuffer);
```

```c
IppStatusippiFilterSharpenBorder_16s_AC4R(const Ipp16s* pSrc, int srcStep, Ipp16s* pDst, int dstStep, IppSize dstRoiSize, IppiMaskSize mask, IppiBorderType borderType, const Ipp16s pBorderValue[3], Ipp8u* pBuffer);
```

```c
IppStatusippiFilterSharpenBorder_32f_AC4R(const Ipp32f* pSrc, int srcStep, Ipp32f* pDst, int dstStep, IppSize dstRoiSize, IppiMaskSize mask, IppiBorderType borderType, const Ipp32f pBorderValue[3], Ipp8u* pBuffer);
```

```c
IppStatusippiFilterSharpenBorder_8u_C4R(const Ipp8u* pSrc, int srcStep, Ipp8u* pDst, int dstStep, IppSize dstRoiSize, IppiMaskSize mask, IppiBorderType borderType, const Ipp8u pBorderValue[4], Ipp8u* pBuffer);
```
IppStatus ippiFilterSharpenBorder_16s_C4R(const Ipp16s* pSrc, int srcStep, Ipp16s* pDst, int dstStep, IppiSize dstRoiSize, IppiMaskSize mask, IppiBorderType borderType, const Ipp16s pBorderValue[4], Ipp8u* pBuffer);

IppStatus ippiFilterSharpenBorder_32f_C4R(const Ipp32f* pSrc, int srcStep, Ipp32f* pDst, int dstStep, IppiSize dstRoiSize, IppiMaskSize mask, IppiBorderType borderType, const Ipp32f pBorderValue[4], Ipp8u* pBuffer);

Include Files
ippi.h

Domain Dependencies
Headers: ippcore.h, ippvm.h, ipps.h
Libraries: ippcore.lib, ippvm.lib, ipps.lib

Parameters
pSrc
Pointer to the source image ROI.
srcStep
Distance, in bytes, between the starting points of consecutive lines in the source image.
pDst
Pointer to the destination image ROI.
dstStep
Distance, in bytes, between the starting points of consecutive lines in the destination image.
dstRoiSize
Size of the source and destination ROI, in pixels.
mask
Predefined mask of IppiMaskSize. Possible value is ippMskSize3x3.
borderType
Type of border. Possible values are:
ippBorderConst
Values of all border pixels are set to constant.
ippBorderRepl
Border is replicated from the edge pixels.
ippBorderInMem
Border is obtained from the source image pixels in memory.

Mixed borders are also supported. They can be obtained by the bitwise operation OR between any of the ippBorderRepl, ippBorderConst, or ippBorderMirror values and the ippBorderInMemTop, ippBorderInMemBottom, ippBorderInMemLeft, ippBorderInMemRight values.

borderValue
Constant value to assign to pixels of the constant border. This parameter is applicable only to the ippBorderConst border type.
pBorderValue[3], pBorderValue[4]
Pointer to constant values to assign to pixels of the constant border. This parameter is applicable only to the ippBorderConst border type.
pBuffer
Pointer to the work buffer.

Description
Before using this function, you need to compute the size of the work buffer pBuffer using the ippiFilterSharpenBorderGetBufferSize function.

This function operates with ROI (see Regions of Interest in Intel IPP).
This function applies a high-pass filter to the \textit{pSrc} source image ROI. The size of the source image ROI is equal to the destination image ROI size \textit{dstRoiSize}. The values of border pixels are assigned in accordance with the \textit{borderType} and \textit{borderValue} parameters. The kernel of the filter is a matrix of 3x3 size. The kernel has the following value:

\[
\begin{bmatrix}
-1/8 & -1/8 & -1/8 \\
-1/8 & 16/8 & -1/8 \\
-1/8 & -1/8 & -1/8 \\
\end{bmatrix}
\]

The anchor cell is the center cell of the kernel, highlighted in red.

**Return Values**

- \texttt{ippStsNoErr} indicates no error.
- \texttt{ippStsNullPtrErr} indicates an error when one of the specified pointers is NULL.
- \texttt{ippStsSizeErr} indicates an error when \textit{dstRoiSize} is negative, or equal to zero.
- \texttt{ippStsBorderErr} indicates an error when \textit{mask} has an illegal value.
- \texttt{ippStsBorderErr} indicates an error when \textit{borderType} has an illegal value.

**See Also**

Borders in Neighborhood Operations
Regions of Interest in Intel IPP
User-defined Border Types
FilterSharpenBorder Performs image sharpening with a high-pass filter.

**FilterSobelGetBufferSize**

*Computes the size of the work buffer for the Sobel filter.*

**Syntax**

\[
\text{IppStatus ippiFilterSobelGetBufferSize (IppiSize dstRoiSize, IppiMaskSize mask, IppNormType normType, IppDataType srcDataType, IppDataType dstDataType, int numChannels, int* pBufferSize);}
\]

**Include Files**

\texttt{ippi.h}

**Domain Dependencies**

Headers: \texttt{ippcore.h, ippvm.h, ipps.h}

Libraries: \texttt{ippcore.lib, ippvm.lib, ipps.lib}

**Parameters**

- \textit{dstRoiSize}  
  Size of the destination ROI in pixels.
- \textit{mask}  
  Predefined mask of \texttt{IppiMaskSize} type. Possible values are \texttt{ippMskSize3x3} or \texttt{ippMskSize5x5}.
normType
srcDataType
dstDataType
numChannels
pBufferSize

Normalization mode of IppNormType type.
Data type of the source image.
Data type of the destination image.
Number of channels in the image. Possible value is 1.
Pointer to the size of the external work buffer.

Description
The ippiFilterSobelGetBufferSize function computes the size, in bytes, of the external work buffer needed for the ippiFilterSobel function. The result is stored in the pBufferSize parameter.

For an example on how to use this functions, see the code example provided with the ippiFilterSobel function description.

Return Values
ippStsNoErr Indicates no error.
ippStsNullPtrErr Indicates an error when pBufferSize is NULL.
ippStsSizeErr Indicates an error when dstRoiSize is less than, or equal to zero.
ippStsMaskSizeErr Indicates an error when mask has an illegal value.
ippStsBadArgErr Indicates an error when normType has an illegal value.
ippStsDataTypeErr Indicates an error when srcDataType or dstDataType has an illegal value.
ippStsNumChannelsError Indicates an error when numChannels has an illegal value.

See Also
FilterSobel Filters an image using a Sobel filter.

FilterSobel
Filters an image using a Sobel filter.

Syntax
IppStatus ippiFilterSobel_<mod>(const Ipp<srcdatatype>* pSrc, int srcStep, Ipp<dstdatatype>* pDst, int dstStep, IppSize dstRoiSize, IppiMaskSize maskSize, IppNormType normType, IppiBorderType borderType, Ipp<srcdatatype> borderValue, Ipp8u* pBuffer);

Supported values for mod:
8u16s_C1R 16s32f_C1R 16u32f_C1R 32f_C1R

Include Files
ippi.h

Domain Dependencies
Headers: ippcore.h, ippvm.h, ipps.h
Libraries: ippcore.lib, ippvm.lib, ipps.lib
Parameters

- pSrc: Pointer to the source image ROI.
- srcStep: Distance, in bytes, between the starting points of consecutive lines in the source image.
- pDst: Pointer to the destination image ROI.
- dstStep: Distance, in bytes, between the starting points of consecutive lines in the destination image.
- dstRoiSize: Size of the source and destination ROI in pixels.
- maskSize: Size of the predefined mask. Possible values are ippMskSize3x3 or ippMskSize5x5.
- normType: Normalization mode of IppNormType.
- borderType: Type of border. Possible values are:
  - ippBorderConst: Values of all border pixels are set to constant.
  - ippBorderRepl: Border is replicated from the edge pixels.
  - ippBorderInMem: Border is obtained from the source image pixels in memory.

- borderValue: Constant value to assign to pixels of the constant border. This parameter is applicable only to the ippBorderConst border type.
- pBuffer: Pointer to the work buffer.

Description

Before using this function, you need to compute the size of the work buffer pBuffer using the ippiFilterSobelBufferSize function.

This function applies a Sobel filter to the source image with the specified kernel size and normalization type. The values of border pixels are assigned in accordance with the borderType and borderValue parameters. The kernel of the filter is a matrix of 3x3 or 5x5 size depending on the maskSize value. The formulas below describe the algorithm for the 3x3 and 5x5 Sobel operators.

3x3 Sobel operator:

\[
G_x = \begin{bmatrix}
1 & 0 & -1 \\
2 & 0 & -2 \\
1 & 0 & -1 \\
\end{bmatrix} \quad \text{and} \quad G_y = \begin{bmatrix}
1 & 2 & 1 \\
0 & 0 & 0 \\
-1 & -2 & -1 \\
\end{bmatrix}
\]

5x5 Sobel operator:

\[
G_x = \begin{bmatrix}
1 & 0 & 0 & 0 & -1 \\
2 & 0 & 0 & -2 & -2 \\
0 & 0 & 0 & 0 & 0 \\
1 & 0 & 0 & 0 & -1 \\
\end{bmatrix} \quad \text{and} \quad G_y = \begin{bmatrix}
1 & 2 & 1 \\
0 & 0 & 0 \\
-1 & -2 & -1 \\
2 & 2 & 2 \\
\end{bmatrix}
\]
where

- $A$ is the source image
- $*$ is the 2D convolution operator
- $G_x$ and $G_y$ are horizontal and vertical magnitude of the source image, respectively

Sobel filter output $G$, as overall gradient magnitude, is generated through L1 and L2 normalization of $G_x$ and $G_y$.

L1 normalization:

$$G = |G_x| + |G_y|$$

L2 normalization:

$$G = \sqrt{G_x^2 + G_y^2}$$

**Return Values**

- ippStsNoErr: Indicates no error.
- ippStsNullPtrErr: Indicates an error when one of the specified pointers is NULL.
- ippStsSizeErr: Indicates an error when $dstRoiSize$ has a field with a zero or negative value.
- ippStsMaskSizeErr: Indicates an error when $maskSize$ has an illegal value.
- ippStsStepErr: Indicates an error when $srcStep$ or $dstStep$ is less than, or equal to zero.
- ippStsNotEvenStepErr: Indicates an error when one of the step values is not divisible by 4 for floating-point images, or by 2 for short-integer images.
- ippStsBorderErr: Indicates an error when $borderType$ has an illegal value.

**Example**

The code example below demonstrates how to use the `ippiFilterSobelGetBufferSize` and `ippiFilterSobel_8u16s_C1R` functions.

```c
IppStatus filter_sobel_8u16s_c1( void ) {
    Ipp8u pSrc[9*8] = {
        0, 1, 2, 120, 121, 122, 50, 51, 52,
        1, 2, 3, 121, 122, 123, 52, 52, 53,
        3, 4, 5, 130, 131, 132, 63, 64, 65,
    }
    ...}
```
/**
 * Filters a source image using the Sobel operator.
 *
 * @param pSrc     Image source.
 * @param srcStep  Stride of the source image.
 * @param pDst     Image destination.
 * @param dstStep  Stride of the destination image.
 * @param roiSize  Region of interest (width, height).
 * @param mask     Mask used for filtering.
 * @param normType Normalization type.
 * @param borderType Border type.
 * @param pBuffer  Pointer to the work buffer.
 * @returns    ippStatus
 *
 * The result is as follows:
 *
 * |   132   |  20   |  502  |  516  |  48   |  322  |  330  |  58   |
 * |   126   |  20   |  516  |  530  |  48   |  316  |  322  |  58   |
 * |   118   |  16   |  512  |  512  |  16   |  280  |  280  |  16   |
 * |   118   |  12   |  510  |  512  |  8    |  272  |  276  |   8   |
 * |   110   |   4   |  510  |  508  |  8    |  272  |  268  |   8   |
 * |   114   |  16   |  516  |  516  |  16   |  272  |  272  |  16   |
 * |   162   |  114  |  398  |  652  |  402  |  526  |  394  |  134  |
 *
 * See Also
 * Borders in Neighborhood Operations
 * Regions of Interest in Intel IPP
 * User-defined Border Types
 *
 * FilterSobelGetBufferSize Computes the size of the work buffer for the Sobel filter.
 *
 * FilterSobelHorizBorderGetBufferSize Computes the size of the work buffer for the Sobel Horizontal filter.
 */

*Ipp8u *pBuffer;

ippiFilterSobelGetBufferSize(roiSize, mask, normType, ipp8u, ipp16s, 1, &bufferSize);
pBuffer = ippsMalloc_8u(bufferSize);
status = ippiFilterSobel_8u16s_C1R(pSrc+srcStep, srcStep, pDst, dstStep, roiSize, mask, normType, borderType, 33, pBuffer);
ippiFree(pBuffer);
return status;
Domain Dependencies

Headers: ippcore.h, ippvm.h, ipps.h
Libraries: ippcore.lib, ippvm.lib, ipps.lib

Parameters

dstRoiSize  
Size of the destination ROI in pixels.

mask  
Predefined mask of IppiMaskSize type. Possible values are ippMskSize3x3 or ippMskSize5x5.

srcDataType  
Data type of the source image.

dstDataType  
Data type of the destination image.

numChannels  
Number of channels in the image. Possible value is 1.

pBufferSize  
Pointer to the size of the external work buffer.

Description

The ippiFilterSobelHorizBorderGetBufferSize function computes the size, in bytes, of the external work buffer needed for the ippiFilterSobelHorizBorder function. The result is stored in the pBufferSize parameter.

For an example on how to use this function, see the code example provided with the ippiFilterSobelHorizBorder function description.

Return Values

ippiStsNoErr  
Indicates no error.

ippiStsNullPtrErr  
Indicates an error when pBufferSize is NULL.

ippiStsSizeErr  
Indicates an error when dstRoiSize is negative, or equal to zero.

ippiStsMaskSizeErr  
Indicates an error when mask has an illegal value.

ippiStsDataTypeErr  
Indicates an error when srcDataType or dstDataType has an illegal value.

ippiStsNumChannelsError  
Indicates an error when numChannels has an illegal value.

See Also
FilterSobelHorizBorder Filters an image using a horizontal Sobel filter.

FilterSobelHorizBorder
Filters an image using a horizontal Sobel filter.
Syntax
IppStatus ippiFilterSobelHorizBorder_<mod>(const Ipp<srcDatatype>* pSrc, int srcStep, Ipp<dstDatatype>* pDst, int dstStep, IppiSize dstRoiSize, IppiMaskSize mask, IppiBorderType borderType, Ipp<srcDatatype> borderValue, Ipp8u* pBuffer);

Supported values for <mod>:
8u16s_C1R 16s_C1R 32f_C1R

Include Files
ippi.h

Domain Dependencies
Headers: ippcore.h, ippvm.h, ipps.h
Libraries: ippcore.lib, ippvm.lib, ipps.lib

Parameters
pSrc
Pointer to the source image ROI.
srcStep
Distance, in bytes, between the starting points of consecutive lines in the source image.
pDst
Pointer to the destination image ROI.
dstStep
Distance, in bytes, between the starting points of consecutive lines in the destination image.
dstRoiSize
Size of the source and destination ROI in pixels.
mask
Predefined mask of IppiMaskSize. Possible values are ippMskSize3x3 or ippMskSize5x5.
borderType
Type of border. Possible values are:
ippBorderConst
Values of all border pixels are set to constant.
ippBorderRepl
Border is replicated from the edge pixels.
ippBorderMirror
Mirrored border is used.
ippBorderMirrorR
Mirrored border with replication is used.
ippBorderInMem
Border is obtained from the source image pixels in memory.

Mixed borders are also supported. They can be obtained by the bitwise operation OR between any of the ippBorderRepl, ippBorderConst, ippBorderMirror, or ippBorderMirrorR values and the ippBorderInMemTop, ippBorderInMemBottom, ippBorderInMemLeft, ippBorderInMemRight values.

borderValue
Constant value to assign to pixels of the constant border. This parameter is applicable only to the ippBorderConst border type.
pBuffer
Pointer to the work buffer.

Description
Before using this function, you need to compute the size of the work buffer pBuffer using the FilterSobelHorizBorderGetBufferSize function.
This function operates with ROI.

This function applies a horizontal Sobel filter to the pSrc source image ROI. The size of the source image ROI is equal to the destination image ROI size dstRoiSize. The values of border pixels are assigned in accordance with the borderType and borderValue parameters. The kernel of the filter is a matrix of 3x3 or 5x5 size depending on the mask value. The kernels have the following values:

|   1   4   6   4   1   |
|  1   2   1   2   8  12  8  2   |
|  0   0   0   0   0   0   0   0   |
| -1  -2  -1  -2  -8  -12  -8  -2   |
| -1  -4  -1 -1  -4  -12  -8  -2   |

The anchor cell is the center cell of the kernel, highlighted in red.

This filter enhances horizontal edges of an image.

Return Values

ippStsNoErr Indicates no error.
ippStsNullPtrErr Indicates an error when one of the specified pointers is NULL.
ippStsSizeErr Indicates an error when dstRoiSize is negative, or equal to zero.
ippStsStepErr Indicates an error when srcStep or dstStep is negative, or equal to zero.
ippStsNotEvenStepErr Indicates an error when one of the step values is not divisible by 4 for floating-point images, or by 2 for short-integer images.
ippStsBorderErr Indicates an error when borderType has an illegal value.

Example

The code example below demonstrates how to use theippiFilterSobelHorizBorderGetBufferSize andippiFilterSobelHorizBorder_8u16s_C1R functions to filter an image with the Sobel horizontal kernel.

```c
IppStatus fix_sobelhoriz_8u16( void ) {
    Ipp8u pSrc[9*8] =
    { 0, 1, 2, 120, 121, 122, 50, 51, 52,
       1, 2, 3, 121, 122, 123, 52, 52, 53,
       3, 4, 5, 130, 131, 132, 63, 64, 65,
       4, 5, 6, 131, 132, 133, 64, 65, 66,
       5, 6, 7, 132, 133, 134, 65, 66, 67,
       8, 7, 6, 134, 133, 132, 67, 66, 65,
       7, 6, 5, 133, 132, 131, 66, 65, 64,
       6, 5, 4, 132, 131, 130, 65, 64, 63  
    };
    Ipp16s  pDst[8*7];
    Ipp8u  *pBuffer;
    IppiSize roiSize = {8, 7};
    IppiBorderType borderType = ippBorderRepl | ippBorderInMemTop | ippBorderInMemRight;
    int    srcStep = 9 * sizeof(Ipp8u);
    int    dstStep = 8 * sizeof(Ipp16s);
    int    bufferSize;
    ...
IppStatus status;
ippiFilterSobelHorizBorderGetBufferSize(roiSize, ippMskSize3x3, ipp8u, ipp16s, 1,
&bufferSize);
pBuffer = ippsMalloc_8u(bufferSize);
status = ippiFilterSobelHorizBorder_8u16s_C1R(pSrc + srcStep, srcStep, pDst, dstStep,
roiSize, ippMskSize3x3,
    borderType, 0, pBuffer);
ippsFree(pBuffer);
return status;
}

The result is as follows:

<table>
<thead>
<tr>
<th>P dst --&gt;</th>
</tr>
</thead>
<tbody>
<tr>
<td>12 12 19 33 40 43 49 52</td>
</tr>
<tr>
<td>12 12 19 33 40 42 47 51</td>
</tr>
<tr>
<td>8 8 8 8 8 8 8 8 8 8</td>
</tr>
<tr>
<td>14 8 5 7 4 2 6 4</td>
</tr>
<tr>
<td>6 0 -3 -1 -4 -6 -2 -4</td>
</tr>
<tr>
<td>-8 -8 -8 -8 -8 -8 -8</td>
</tr>
<tr>
<td>-4 -4 -4 -4 -4 -4 -4</td>
</tr>
</tbody>
</table>

See Also
Borders in Neighborhood Operations
Regions of Interest in Intel IPP
User-defined Border Types
FilterSobelHorizBorderGetBufferSize Computes the size of the work buffer for the Sobel Horizontal filter.

FilterSobelHorizSecondBorderGetBufferSize
Computes the size of the work buffer for the Sobel Horizontal (second derivative) filter.

Syntax
IppStatus ippiFilterSobelHorizSecondBorderGetBufferSize (IppiSize dstRoiSize, IppiMaskSize mask, IppDataType srcDataType, IppDataType dstDataType, int numChannels, int* pBufferSize);

Include Files
ippi.h

Domain Dependencies
Headers: ippcore.h, ippvm.h, ipps.h
Libraries: ippcore.lib, ippvm.lib, ipps.lib

Parameters

dstRoiSize  Size of the destination ROI in pixels.

mask  Predefined mask of IppiMaskSize type. Possible values are ippMskSize3x3 or ippMskSize5x5.

srcDataType  Data type of the source image.
The `ippiFilterSobelHorizSecondBorderGetBufferSize` function computes the size, in bytes, of the external work buffer needed for the `ippiFilterSobelHorizSecondBorder` function. The result is stored in the `pBufferSize` parameter.

### Return Values
- **ippStsNoErr**: Indicates no error.
- **ippStsNullPtrErr**: Indicates an error when `pBufferSize` is NULL.
- **ippStsSizeErr**: Indicates an error when `dstRoiSize` is negative, or equal to zero.
- **ippStsMaskSizeErr**: Indicates an error when `mask` has an illegal value.
- **ippStsDataTypeErr**: Indicates an error when `srcDataType` or `dstDataType` has an illegal value.
- **ippStsNumChannelsError**: Indicates an error when `numChannels` has an illegal value.

### See Also
- **FilterSobelHorizSecondBorder**: Applies horizontal (second derivative) Sobel filter with border.

### Syntax
```c
IppStatus ippiFilterSobelHorizSecondBorder_<mod>(const Ipp<srcDatatype> * pSrc, int srcStep, Ipp<dstDatatype> * pDst, int dstStep, IppiSize dstRoiSize, IppiMaskSize mask, IppiBorderType borderType, Ipp<srcDatatype> borderValue, Ipp8u* pBuffer);
```

**Supported values for mod:**
- `8u16s_C1R`
- `32f_C1R`

### Include Files
- `ippi.h`

### Domain Dependencies
- **Headers**: `ippcore.h`, `ippvm.h`, `ipps.h`
- **Libraries**: `ippcore.lib`, `ippvm.lib`, `ipps.lib`

### Parameters
- **pSrc**: Pointer to the source image ROI.
srcStep
Distance in bytes between starts of consecutive lines in the source image.

pDst
Pointer to the destination image ROI.

dstStep
Distance in bytes between starts of consecutive lines in the destination image.

dstRoiSize
Size of the source and destination image ROI.

mask
Type of the filter kernel.

borderType
Type of border (see Borders in Neighborhood Operations); following values are possible:

ippBorderConst
Values of all border pixels are set to constant.

ippBorderRepl
Replicated border is used.

ippBorderMirror
Mirrored border is used

ippBorderMirrorR
Mirrored border with replication is used

borderValue
The constant value to assign to the pixels in the constant border (not applicable for other border's type).

pBuffer
Pointer to the working buffer.

Description
This function operates with ROI (see Regions of Interest in Intel IPP). This function applies the second derivative horizontal Sobel filter (y-derivative) to the source image pSrc and stores results to the destination image of the same size pDst. Source image can be used as the destination image if they both have the same data type. The values of border pixels are assigned in accordance with the borderType and borderValue parameters. The kernel of this filter is the matrix of either 3x3 or 5x5 size that is specified by the parameter mask. The kernels have the following values with the anchor in the center cell (red):

| 1 4 6 4 1 |
| 1 2 1     |
| 1 2 1     |
| -2 -4 -2  |
| -2 -4 -2  |

or

| 0 0 0 0 0 |
| 0 0 0 0 0 |
| 0 0 0 0 0 |
| -2 -8 -12 -8 -2 |
| -2 -8 -12 -8 -2 |

The function requires the working buffer pBuffer which size should be computed by the function ippiFilterSobelHorizSecondBorderGetBufferSize beforehand.

Return Values

ippStsNoErr
Indicates no error. Any other value indicates an error or a warning.

ippStsNullPtrErr
Indicates an error condition if one of the specified pointers is NULL.

ippStsSizeErr
Indicates an error condition if roiSize has a field with a zero or negative value.

ippStsStepErr
Indicates an error condition if srcStep or dstStep is less than roiSize.width * pixelSize.
FilterSobelVertBorderGetBufferSize

Computes the size of the work buffer for the Sobel Vertical filter.

Syntax

IppStatus ippiFilterSobelVertBorderGetBufferSize (IppiSize dstRoiSize, IppiMaskSize mask, IppDataType srcDataType, IppDataType dstDataType, int numChannels, int* pBufferSize);

Include Files
ippi.h

Parameters

dstRoiSize
Size of the destination ROI in pixels.

mask
Predefined mask of IppiMaskSize type. Possible values are ippMskSize3x3 or ippMskSize5x5.

srcDataType
Data type of the source image.

dstDataType
Data type of the destination image.

numChannels
Number of channels in the image. Possible value is 1.

pBufferSize
Pointer to the size of the external work buffer.

Description

The ippiFilterSobelVertBorderGetBufferSize function computes the size, in bytes, of the external work buffer needed for the ippiFilterSobelVertBorder function. The result is stored in the pBufferSize parameter.

For an example on how to use this function, see the code example provided with the ippiFilterSobelVertBorder function description.

Return Values

ippStsNoErr
Indicates no error.

ippStsNullPtrErr
Indicates an error when pBufferSize is NULL.

ippStsSizeErr
Indicates an error when dstRoiSize is negative, or equal to zero.

ippStsMaskSizeErr
Indicates an error when mask has an illegal value.

ippStsDataTypeErr
Indicates an error when srcDataType or dstDataType has an illegal value.
ippStsNumChannelsError Indicates an error when numChannels has an illegal value.

See Also
FilterSobelVertBorder Filters an image using a vertical Sobel filter.

FilterSobelVertBorder
Filters an image using a vertical Sobel filter.

Syntax
IppStatus ippiFilterSobelVertBorder_<mod>(const Ipp<srcDatatype>* pSrc, int srcStep, Ipp<dstDatatype>* pDst, int dstStep, IppiSize dstRoiSize, IppiMaskSize mask, IppiBorderType borderType, Ipp<srcDatatype> borderValue, Ipp8u* pBuffer);

Supported values for mod:
8u16s_C1R 16s_C1R 32f_C1R

Include Files
ippi.h

Domain Dependencies
Headers: ippcore.h, ippvm.h, ipps.h
Libraries: ippcore.lib, ippvm.lib, ipps.lib

Parameters
pSrc
Pointer to the source image ROI.
srcStep
Distance, in bytes, between the starting points of consecutive lines in the source image.
pDst
Pointer to the destination image ROI.
dstStep
Distance, in bytes, between the starting points of consecutive lines in the destination image.
dstRoiSize
Size of the source and destination ROI in pixels.
mask
Predefined mask of IppiMaskSize. Possible values are ippMskSize3x3 or ippMskSize5x5.
borderType
Type of border. Possible values are:
ippBorderConst Values of all border pixels are set to constant.
ippBorderRepl Border is replicated from the edge pixels.
ippBorderMirror Mirrored border is used.
ippBorderMirrorR Mirrored border with replication is used.
ippBorderInMem Border is obtained from the source image pixels in memory.
Mixed borders are also supported. They can be obtained by the bitwise operation OR between any of the ippBorderRepl, ippBorderConst, ippBorderMirror, or ippBorderMirrorR values and the ippBorderInMemTop, ippBorderInMemBottom, ippBorderInMemLeft, ippBorderInMemRight values.

borderValue
Constant value to assign to pixels of the constant border. This parameter is applicable only to the ippBorderConst border type.

pBuffer
Pointer to the work buffer.

Description
Before using this function, you need to compute the size of the work buffer pBuffer using the FilterSobelVertBorderGetBufferSize function.

This function operates with ROI.

This function applies a vertical Sobel filter to the pSrc source image ROI. The size of the source image ROI is equal to the destination image ROI size dstRoiSize. The values of border pixels are assigned in accordance with the borderType and borderValue parameters. The kernel of the filter is a matrix of 3x3 or 5x5 size depending on the mask value. The kernels have the following values:

<table>
<thead>
<tr>
<th>-1</th>
<th>-2</th>
<th>0</th>
<th>2</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td>-1</td>
<td>0</td>
<td>1</td>
<td>-4</td>
<td>-8</td>
</tr>
<tr>
<td>-2</td>
<td>0</td>
<td>2</td>
<td>or</td>
<td>-6</td>
</tr>
<tr>
<td>-1</td>
<td>0</td>
<td>1</td>
<td>-4</td>
<td>-8</td>
</tr>
<tr>
<td>-1</td>
<td>-2</td>
<td>0</td>
<td>2</td>
<td>1</td>
</tr>
</tbody>
</table>

The anchor cell is the center cell of the kernel, highlighted in red.

This filter enhances vertical edges of an image.

Return Values

ippStsNoErr
Indicates no error.

ippStsNullPtrErr
Indicates an error when one of the specified pointers is NULL.

ippStsSizeErr
Indicates an error when dstRoiSize is negative, or equal to zero.

ippStsStepErr
Indicates an error when srcStep or dstStep is negative, or equal to zero.

ippStsNotEvenStepErr
Indicates an error when one of the step values is not divisible by 4 for floating-point images, or by 2 for short-integer images.

ippStsBorderErr
Indicates an error when borderType has an illegal value.

Example

The code example below demonstrates how to use the ippiFilterSobelVertBorderGetBufferSize and ippiFilterSobelVertBorder functions.

```
IppStatus fix_sobel_8u16( void ) {
    Ipp8u pSrc[9*8] =
    {
        0, 1, 2, 120, 121, 122, 50, 51, 52,
```
The result is as follows:

```
pDst ->
-4  -8 -483 -483  -8 279 281  -6
-4  -8 -497 -497  -8 274 275  -7
-4  -8 -504 -504  -8 272 272  -8
-2  -4 -505 -505  -4 270 270  -4
 2  4 -507 -507  4 266 266  4
 4  8 -508 -508  8 264 264  8
 4  8 -508 -508  8 264 264  8
```

See Also
Borders in Neighborhood Operations
Regions of Interest in Intel IPP
User-defined Border Types
FilterSobelVertBorderGetBufferSize Computes the size of the work buffer for the Sobel Vertical filter.

**FilterSobelVertSecondBorderGetBufferSize**

*Computes the size of the work buffer for the Sobel vertical (second derivative) filter.*

**Syntax**

```c
IppStatus ippiFilterSobelVertSecondBorderGetBufferSize (IppiSize dstRoiSize,
            IppiMaskSize mask, IppDataType srcDataType, IppDataType dstDataType, int numChannels,
            int* pBufferSize);
```

**Include Files**

ippi.h
**Domain Dependencies**

**Headers:** ippcore.h, ippvm.h, ipps.h

**Libraries:** ippcore.lib, ippvm.lib, ipps.lib

**Parameters**

- **dstRoiSize**
  Size of the destination ROI in pixels.

- **mask**
  Predefined mask of IppiMaskSize type. Possible values are ippMskSize3x3 or ippMskSize5x5.

- **srcDataType**
  Data type of the source image.

- **dstDataType**
  Data type of the destination image.

- **numChannels**
  Number of channels in the image. Possible value is 1.

- **pBufferSize**
  Pointer to the size of the external work buffer.

**Description**

The `ippiFilterSobelVertSecondBorderGetBufferSize` function computes the size, in bytes, of the external work buffer needed for the `ippiFilterSobelVertSecondBorder` function. The result is stored in the `pBufferSize` parameter.

**Return Values**

- **ippStsNoErr**
  Indicates no error.

- **ippStsNullPtrErr**
  Indicates an error when `pBufferSize` is NULL.

- **ippStsSizeErr**
  Indicates an error when `dstRoiSize` is negative, or equal to zero.

- **ippStsMaskSizeErr**
  Indicates an error when `mask` has an illegal value.

- **ippStsDataTypeErr**
  Indicates an error when `srcDataType` or `dstDataType` has an illegal value.

- **ippStsNumChannelsError**
  Indicates an error when `numChannels` has an illegal value.

**See Also**

FilterSobelVertSecondBorder Applies vertical (second derivative) Sobel filter with border.

**FilterSobelNegVertBorderGetBufferSize**

*Computes the size of the work buffer for the Sobel vertical filter.*

**Syntax**

```c
IppStatus ippiFilterSobelNegVertBorderGetBufferSize (IppiSize dstRoiSize, IppiMaskSize mask, IppDataType srcDataType, IppDataType dstDataType, int numChannels, int* pBufferSize);
```

**Include Files**

ippi.h
Domain Dependencies

Headers: ippcore.h, ippvm.h, ipps.h

Libraries: ippcore.lib, ippvm.lib, ipps.lib

Parameters

dstRoiSize
Size of the destination ROI in pixels.

mask
Predefined mask of IppiMaskSize type. Possible values are ippMskSize3x3 or ippMskSize5x5.

srcDataType
Data type of the source image.

dstDataType
Data type of the destination image.

numChannels
Number of channels in the image. Possible value is 1.

pBufferSize
Pointer to the size of the external work buffer.

Description

The ippiFilterSobelVertSecondBorderGetBufferSize function computes the size, in bytes, of the external work buffer needed for the ippiFilterSobelNegVertBorder function. The result is stored in the pBufferSize parameter.

Return Values

ippStsNoErr
Indicates no error.

ippStsNullPtrErr
Indicates an error when pBufferSize is NULL.

ippStsSizeErr
Indicates an error when dstRoiSize is negative, or equal to zero.

ippStsMaskSizeErr
Indicates an error when mask has an illegal value.

ippStsDataTypeErr
Indicates an error when srcDataType or dstDataType has an illegal value.

ippStsNumChannelsError
Indicates an error when numChannels has an illegal value.

See Also

FilterSobelNegVertSecondBorder Applies vertical Sobel filter with border.

Syntax

IppStatus ippiFilterSobelNegVertBorder_<mod>(const Ipp<srcDatatype>* pSrc, int srcStep, Ipp<dstDatatype>* pDst, int dstStep, IppiSize dstRoiSize, IppiMaskSize mask, IppiBorderType borderType, Ipp<srcDatatype> borderValue, Ipp8u* pBuffer);

Supported values for mod:

8u16s_C1R  32f_C1R
Include Files
ippi.h

Domain Dependencies
Headers: ippi.h, ippvm.h, ipps.h
Libraries: ippcore.lib, ippvm.lib, ipps.lib

Parameters

- **pSrc**: Pointer to the source image ROI.
- **srcStep**: Distance in bytes between starts of consecutive lines in the source image.
- **pDst**: Pointer to the destination image ROI.
- **dstStep**: Distance in bytes between starts of consecutive lines in the destination image.
- **dstRoiSize**: Size of the destination image ROI.
- **mask**: Type of the filter kernel.
- **borderType**: Type of border (see Borders in Neighborhood Operations); following values are possible:
  - ippBorderConst: Values of all border pixels are set to constant.
  - ippBorderRepl: Replicated border is used.
  - ippBorderMirror: Mirrored border is used
  - ippBorderMirrorR: Mirrored border with replication is used.
- **borderValue**: The constant value to assign to the pixels in the constant border (not applicable for other border's type).
- **pBuffer**: Pointer to the working buffer.

Description

These functions operate with ROI (see Regions of Interest in Intel IPP). These functions apply the vertical Sobel filter (x-derivative) to the source image ROI **pSrc** and stores results to the destination image ROI of the same size **pDst**. Source image can be used as the destination image if they have the same data type. The values of border pixels are assigned in accordance with the **borderType** and **borderValue** parameters. The kernel of this filter is the matrix of either 3x3 or 5x5 size that is specified by the parameter **mask**. The anchor cell is the center cell of the kernel (red).

The function **ippiFilterSobelVertBorder** uses the kernels with the following coefficients:

<p>| | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>-1</td>
<td>-2</td>
<td>0</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>-2</td>
<td>0</td>
<td>2</td>
<td></td>
<td></td>
</tr>
<tr>
<td>-1</td>
<td>0</td>
<td>1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>-1</td>
<td>-2</td>
<td>0</td>
<td>2</td>
<td>1</td>
</tr>
</tbody>
</table>
The function \texttt{ippiFilterSobelNegVertBorder} uses the kernels which coefficients are the same in magnitude but opposite in sign:

\begin{verbatim}
1 2 0 -2 -1
1 0 -1
2 0 -2 or 6 12 0 -12 -6
1 0 -1
1 2 0 -2 -1
\end{verbatim}

Before using this function, compute the size of the work buffer \texttt{pBuffer} using the \texttt{FilterSobelNegVertBorderGetBufferSize} function.

Example shows how the function \texttt{ippiFilterSobelNegVertBorder\_8u16s\_C1R} can be used for edge detection.

Return Values

- \texttt{ippStsNoErr}: Indicates no error. Any other value indicates an error or a warning.
- \texttt{ippStsNullPtrErr}: Indicates an error condition if one of the specified pointers is NULL.
- \texttt{ippStsSizeErr}: Indicates an error condition if \texttt{roiSize} has a field with a zero or negative value.
- \texttt{ippStsStepErr}: Indicates an error condition if \texttt{srcStep} or \texttt{dstStep} is less than \texttt{roiSize.width} * \texttt{<pixelSize>}
- \texttt{ippStsNotEvenStepErr}: Indicates an error condition if one of the step values is not divisible by 4 for floating-point images, or by 2 for short-integer images.
- \texttt{ippStsBadArgErr}: Indicates an error if \texttt{borderType} or \texttt{divisor} has a wrong value.
- \texttt{ippStsMaskErr}: Indicates an error condition if \texttt{mask} has a wrong value.

See Also

User-defined Border Types

\texttt{FilterSobelVertSecondBorder}

Applies vertical (second derivative) Sobel filter with border.

Syntax

\begin{verbatim}
IppStatus ippiFilterSobelVertSecondBorder\_<mod>(const Ipp<srcDatatype>\* pSrc, int srcStep, Ipp<dstDatatype>\* pDst, int dstStep, IppiSize dstRoiSize, IppiMaskSize mask, IppiBorderType borderType, Ipp<srcDatatype> borderValue, Ipp8u\* pBuffer);
\end{verbatim}

Supported values for \texttt{mod}:

- 8u16s\_C1R
- 32f\_C1R

Include Files

\texttt{ippi.h}
Domain Dependencies

Headers: ippcore.h, ippvm.h, ipps.h
Libraries: ippcore.lib, ippvm.lib, ipps.lib

Parameters

pSrc  
Pointer to the source image ROI.

srcStep  
Distance in bytes between starts of consecutive lines in the source image.

pDst  
Pointer to the destination image ROI.

dstStep  
Distance in bytes between starts of consecutive lines in the destination image.

dstRoiSize  
Size of the destination image ROI.

mask  
Type of the filter kernel.

borderType  
Type of border (see Borders in Neighborhood Operations); following values are possible:

ippBorderConst  
Values of all border pixels are set to constant.

ippBorderRepl  
Replicated border is used.

ippBorderMirror  
Mirrored border is used

ippBorderMirrorR  
Mirrored border with replication is used

borderValue  
The constant value to assign to the pixels in the constant border (not applicable for other border's type).

pBuffer  
Pointer to the working buffer.

Description

This function operates with ROI (see Regions of Interest in Intel IPP). This function applies the second derivative vertical Sobel filter (x-derivative) to the source image pSrc and stores results to the destination image of the same size pDst. Source image can be used as the destination image if they both have the same data type. The values of border pixels are assigned in accordance with the borderType and borderValue parameters. The kernel of this filter is the matrix of either 3x3 or 5x5 size that is specified by the parameter mask. The kernels have the following values with the anchor in the center cell (red):

<p>| | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0</td>
<td>-2</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>-2</td>
<td>1</td>
<td>4</td>
<td>0</td>
</tr>
<tr>
<td>2</td>
<td>-4</td>
<td>2</td>
<td>6</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>-2</td>
<td>1</td>
<td>4</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>-2</td>
<td>0</td>
<td>1</td>
</tr>
</tbody>
</table>

The function requires the working buffer pBuffer which size should be computed by the function ippiFilterSobelVertSecondBorderGetBufferSize beforehand.

Return Values

ippStsNoErr  
Indicates no error. Any other value indicates an error or a warning.
Indicates an error condition if one of the specified pointers is NULL.

Indicates an error condition if roiSize has a field with a zero or negative value.

Indicates an error condition if srcStep or dstStep is less than roiSize.width * pixelSize.

Indicates an error condition if one of the step values is not divisible by 4 for floating-point images, or by 2 for short-integer images.

Indicates an error if borderType or divisor has a wrong value.

Indicates an error condition if mask has a wrong value.

**Example**

To better understand usage of this function, refer to the following example in the examples archive available for download from https://software.intel.com/en-us/ipp-manual-examples:

FilterSobelVertSecondBorder.c

---

**FilterSobelCrossGetBufferSize**

*Computes the size of the external buffer for the cross Sobel filter with border.*

**Syntax**

IppStatus ippiFilterSobelCrossGetBufferSize_<mod>(IppiSize roiSize, IppiMaskSize mask, int* pBufferSize);

**Supported values for mod:**

8u16s_C1R  32f_C1R

**Include Files**

ippcv.h

**Domain Dependencies**

Headers: ippcore.h, ippvm.h, ipps.h,ippi.h

Libraries: ippcore.lib, ippvm.lib, ipps.lib,ippi.lib

**Parameters**

- **roiSize**
  
  Maximum size of the source and destination image ROI.

- **mask**
  
  Predefined mask of IppiMaskSize type.

- **pBufferSize**
  
  Pointer to the buffer size.

**Description**

This function computes the size of the external buffer that is required for the filter function ippiFilterSobelCrossBorder. The kernel of the filter is the matrix of either 3x3 or 5x5 size that is specified by the parameter mask (see Table “Types of the Fixed Filter Functions”). This buffer pBufferSize[0] can be used to filter an image whose width and height are equal to or less than corresponding fields of roiSize.
Return Values

ippStsNoErr Indicates no error. Any other value indicates an error or a warning.

ippStsNullPtrErr Indicates an error condition if the pBufferSize pointer is NULL.

ippStsSizeErr Indicates an error condition if roiSize has a field with a zero or negative value.

ippStsMaskSizeErr Indicates an error condition if mask has a wrong value.

FilterSobelCrossBorder

Applies second derivative cross Sobel filter with border.

Syntax

IppStatus ippiFilterSobelCrossBorder_<mod>(const Ipp<srcDatatype>* pSrc, int srcStep, Ipp<dstDatatype>* pDst, int dstStep, IppiSize roiSize, IppiMaskSize mask, IppiBorderType borderType, Ipp<srcDatatype> borderValue, Ipp8u* pBuffer);

Supported values for mod:

8u16s_C1R 32f_C1R

Include Files

ippcv.h

Domain Dependencies

Headers: ippcore.h, ippvm.h, ipps.h,ippi.h
Libraries: ippcore.lib, ippvm.lib, ipps.lib,ippi.lib

Parameters

pSrc Pointer to the source image ROI.

csrcStep Distance in bytes between starts of consecutive lines in the source image.

pDst Pointer to the destination image ROI.

dstStep Distance in bytes between starts of consecutive lines in the destination image.

roiSize Size of the source and destination image ROI.

mask Type of the filter kernel.

borderType Type of border (see Borders in Neighborhood Operations); following values are possible:

ippBorderConst Values of all border pixels are set to constant.

ippBorderRepl Replicated border is used.

borderValue The constant value to assign to the pixels in the constant border (not applicable for other border's type).
**pBuffer**

Pointer to the working buffer.

**Description**

This function operates with ROI (see Regions of Interest in Intel IPP).

This function applies the second derivative cross Sobel filter (xy-derivative) to the source image pSrc and stores results to the destination image of the same size pDst. Source image can be used as the destination image if they both have the same data type. The values of border pixels are assigned in accordance with the borderType and borderValue parameters. The kernel of this filter is the matrix of either 3x3 or 5x5 size that is specified by the parameter mask. The kernels have the following values with the anchor in the center cell (red):

<table>
<thead>
<tr>
<th></th>
<th>-1</th>
<th>-2</th>
<th>0</th>
<th>2</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td>-1</td>
<td>0</td>
<td>1</td>
<td>-2</td>
<td>-4</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>-1</td>
<td>2</td>
<td>4</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>1</td>
<td>2</td>
<td>0</td>
<td>-2</td>
<td>-1</td>
</tr>
</tbody>
</table>

The function requires the working buffer pBuffer whose size should be computed by the functionippiFilterSobelCrossGetBufferSize beforehand.

**Return Values**

- **ippStsNoErr**
  Indicates no error. Any other value indicates an error or a warning.
- **ippStsNullPtrErr**
  Indicates an error condition if one of the specified pointers is NULL.
- **ippStsSizeErr**
  Indicates an error condition if roiSize has a field with a zero or negative value.
- **ippStsStepErr**
  Indicates an error condition if srcStep or dstStep is less than roiSize.width * <pixelSize>
- **ippStsNotEvenStepErr**
  Indicates an error condition if one of the step values is not divisible by 4 for floating-point images, or by 2 for short-integer images.
- **ippStsBadArgErr**
  Indicates an error if borderType or divisor has a wrong value.
- **ippStsMaskErr**
  Indicates an error condition if mask has a wrong value.

**GenSobelKernel**

*Computes kernel for the Sobel filter.*

**Syntax**

IppStatus ippiGenSobelKernel_16s(Ipp16s* pDst, int kernelSize, int dx, int sign);
IppStatus ippiGenSobelKernel_32f(Ipp32f* pDst, int kernelSize, int dx, int sign);

**Include Files**

ippcv.h
Domain Dependencies
Headers: ippcore.h, ippvm.h, ipps.h,ippi.h
Libraries: ippcore.lib, ippvm.lib, ipps.lib,ippi.lib

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>pDst</td>
<td>Pointer to the destination vector.</td>
</tr>
<tr>
<td>kernelSize</td>
<td>Size of the Sobel kernel.</td>
</tr>
<tr>
<td>dx</td>
<td>Order of derivative.</td>
</tr>
<tr>
<td>sign</td>
<td>Specifies signs of kernel elements.</td>
</tr>
</tbody>
</table>

Description
This function computes the one-dimensional Sobel kernel. Kernel coefficients are equal to coefficients of the polynomial

\[ (1 + x)^{\text{kernelSize}} - dx - 1 \cdot (x - 1)^{dx} \]

If the `sign` parameter is negative, then signs of kernel coefficients are changed. Kernel calculated by this function can be used to filter images by a high order Sobel filter.

Return Values

<table>
<thead>
<tr>
<th>Status Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ippStsNoErr</td>
<td>Indicates no error. Any other value indicates an error or a warning.</td>
</tr>
<tr>
<td>ippStsNullPtrErr</td>
<td>Indicates an error condition if the <code>pDst</code> pointer is NULL.</td>
</tr>
<tr>
<td>ippStsSizeErr</td>
<td>Indicates an error condition if <code>kernelSize</code> is less than 3 or is even.</td>
</tr>
<tr>
<td>ippStsBadArgErr</td>
<td>Indicates an error condition if <code>dx</code> is equal to or less than <code>kernelSize</code>, or <code>dx</code> is negative.</td>
</tr>
</tbody>
</table>

Deinterlacing Filters
This section describes functions that perform image deinterlacing.

DeinterlaceFilterCAVT
Performs deinterlacing of two-field image.

Syntax

```c
IppStatus ippiDeinterlaceFilterCAVT_8u_C1R(const Ipp8u* pSrc, int srcStep, Ipp8u* pDst, int dstStep, Ipp16u threshold, IppSize roiSize);
```

Include Files

ippi.h

Domain Dependencies
Headers: ippcore.h, ippvm.h, ipps.h
Libraries: ippcore.lib, ippvm.lib, ipps.lib
Parameters

- **pSrc**: Pointer to the source image ROI.
- **srcStep**: Distance in bytes between starts of consecutive lines in the source image.
- **pDst**: Pointer to the destination image ROI.
- **dstStep**: Distance in bytes between starts of consecutive lines in the destination image.
- **threshold**: Threshold level value.
- **roiSize**: Size of the source and destination image ROI.

Description

This function operates with ROI (see Regions of Interest in Intel IPP). This function performs deinterlacing of a two-field image, pointed to by `pSrc`, using content adaptive vertical temporal (CAVT) filtering.

The field pointed to by `pSrc` is copied to `pDst`, while the other field in the destination image, pointed to by `pDst + dstStep`, is the interpolated one. Note that you can set the pointers to the bottom left corner of the images and use negative steps to have the bottom field unchanged and the top one interpolated.

The **threshold** parameter is the edge detection threshold with the valid range [0-2041] regulating the probability of temporal interpolation: 0 means that all the pixels are interpolated only spatially, from the vertically neighbouring pixels of the copied field, 2041 - that combined spatial-temporal interpolation, involving the pixels from the modified field as well, is applied to all the pixels.

Return Values

- **ippStsNoErr**: Indicates no error. Any other value indicates an error.
- **ippStsNullPtrErr**: Indicates an error condition if one of the specified pointers is NULL.
- **ippStsSizeErr**: Indicates an error condition if `roiSize.width` is less than or equal to 0, or `roiSize.height` is odd or less than 8.

Median

*Creates an image consisting of median values of three source images.*

Syntax

```c
IppStatus ippiMedian_8u_P3C1R(const Ipp8u* pSrc[3], int srcStep, Ipp8u* pDst, int dstStep, IppiSize size);
```

Include Files

`ippi.h`

Domain Dependencies

Headers: `ippicore.h`, `ippvm.h`, `ipps.h`

Libraries: `ippicore.lib`, `ippvm.lib`, `ipps.lib`
Parameters

- **pSrc**: Array of pointers to the ROI in each plane of the source image.
- **srcStep**: Distance in bytes between starts of consecutive lines in each plane of the source image.
- **pDst**: Pointer to the destination image ROI.
- **dstStep**: Distance in bytes between starts of consecutive lines in the destination image.
- **size**: Size of the source and destination image ROI.

Description

This function operates with ROI (see Regions of Interest in Intel IPP). This function sets each pixel in the destination image ROI as the median value of correspondent pixels in the each plane of the source image.

Return Values

- **ippStsNoErr**: Indicates no error. Any other value indicates an error.
- **ippStsNullPtrErr**: Indicates an error condition if one of the specified pointers is NULL.
- **ippStsSizeErr**: Indicates an error condition if \( \text{size} \) has a field with zero or negative value.
This chapter describes the Intel® IPP image processing functions that perform linear transform operations on an image buffer.

These operations include Fast Fourier Transform (FFT), Discrete Fourier Transform (DFT), and Discrete Cosine Transform (DCT).

To speed up performance, linear transform functions use precomputed auxiliary data that is needed for computation of the transforms (that is, tables of twiddle factors for FFT functions). This data is calculated by the respective initialization functions and passed to the transform functions in context structures specific for each type of transform.

Intel IPP linear transform functions can use external work buffers for storing data and intermediate results, which eliminates the need to allocate and free internal memory buffers and thus helps to further increase function performance. To determine the required work buffer size, use one of the respective support functions specific for each transform type. In case when no external buffer is specified, the transform functions handle memory allocation internally.

All Intel IPP linear transform functions except DCT of 8x8 size work on images with floating-point data only.

## Fourier Transforms

Intel IPP functions that compute FFT and DFT can process both real and complex images. Function flavors operating on real data are distinguished by R suffix present in function-specific modifier of their full name, whereas complex flavors' names include C suffix (see Function Naming in Chapter 2).

The results of computing the Fourier transform can be normalized by specifying the appropriate value of flag argument for context initialization. This parameter sets up a pair of matched normalization factors to be used in forward and inverse transforms as listed in the following table:

<table>
<thead>
<tr>
<th>Normalization Factors for Fourier Transform Results</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Value of flag Argument</strong></td>
</tr>
<tr>
<td>IPP_FFT_DIV_FWD_BY_N</td>
</tr>
<tr>
<td>IPP_FFT_DIV_INV_BY_N</td>
</tr>
<tr>
<td>IPP_FFT_DIV_BY_SQRTN</td>
</tr>
<tr>
<td>IPP_FFT_NODIV_BY_ANY</td>
</tr>
</tbody>
</table>

In this table, N and M denote the length of Fourier transform in the x- and y-directions, respectively (or, equivalently, the number of columns and rows in the 2D array being transformed).

For the FFT, these lengths must be integer powers of 2, that is \(N=2^{\text{order}_X}, M=2^{\text{order}_Y}\), where power exponents are known as order of FFT.

For the DFT, N and M can take on arbitrary integer non-negative values.

### Real - Complex Packed (RCPack2D) Format

The forward Fourier transform of a real two-dimensional image data yields a matrix of complex results which has conjugate-symmetric properties. Intel IPP functions use packed format RCPack2D for storing and retrieving data of this type. Accordingly, real flavors of the inverse Fourier transform functions convert packed complex conjugate-symmetric data back to its real origin. The RCPack2D format exploits the complex conjugate symmetry of the transformed data to store only a half of the resulting Fourier coefficients. For the N by M transform, the respective FFT and DFT functions actually store real and imaginary parts of the
complex Fourier coefficients $A(i,j)$ for $i = 0,...,M-1; j = 0,..., N/2$ in a single real array of dimensions $(N, M)$. The RCPack2D storage format is slightly different for odd and even $M$ and is arranged in accordance with the following tables:

### RCPack2D Storage for Odd Number of Rows

<table>
<thead>
<tr>
<th>Re $A(0,0)$</th>
<th>Re $A(0,1)$</th>
<th>Im $A(0,1)$</th>
<th>...</th>
<th>Re $A(0, (N-1)/2)$</th>
<th>Im $A(0, (N-1)/2)$</th>
<th>Re $A(0, N/2)$</th>
</tr>
</thead>
<tbody>
<tr>
<td>Re $A(1,0)$</td>
<td>Re $A(1,1)$</td>
<td>Im $A(1,1)$</td>
<td>...</td>
<td>Re $A(1, (N-1)/2)$</td>
<td>Im $A(1, (N-1)/2)$</td>
<td>Re $A(1, N/2)$</td>
</tr>
<tr>
<td>Im $A(1,0)$</td>
<td>Re $A(2,1)$</td>
<td>Im $A(2,1)$</td>
<td>...</td>
<td>Re $A(2, (N-1)/2)$</td>
<td>Im $A(2, (N-1)/2)$</td>
<td>Im $A(2, N/2)$</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
<td></td>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>Re $A(M/2,0)$</td>
<td>Re $A(M-2,1)$</td>
<td>Im $A(M-2,1)$</td>
<td>Re $A(M-2, (N-1)/2)$</td>
<td>Im $A(M-2, (N-1)/2)$</td>
<td>A($M/2$, $N/2$)</td>
<td></td>
</tr>
</tbody>
</table>

The shaded columns to the right side of the tables indicate values for even $N$ only.

Note the above tables show the arrangement of coefficients for one channel. For multichannel images the channel coefficients are clustered and stored consecutively, for example, for 3-channel image they are stored in the following way:

- $C_1$ - $Re A(0,0)$; $C_2$ - $Re A(0,0)$; $C_3$ - $Re A(0,0)$; $C_1$ - $Re A(0,1)$; $C_2$ - $Re A(0,1)$; $C_3$ - $Re A(0,1)$; $C_1$ - $Im A(0,1)$; $C_2$ - $Im A(0,1)$; ...

The remaining Fourier coefficients are obtained using the following relationships based on conjugate-symmetric properties:

- $A(i,j) = conj(A(M-i,N-j))$, $i = 1,..., M-1; j = 1,..., N-1$
- $A(0,j) = conj(A(0,N-j))$, $j = 1,..., N-1$
- $A(i,0) = conj(A(M-i,0))$, $i = 1,..., M-1$

### FFTGetSize

**Computes the size of the FFT context structure and the size of the work buffer.**

**Syntax**

- IppStatus ippiFFTGetSize_R_32f (int orderX, int orderY, int flag, IppHintAlgorithm hint, int* pSizeSpec, int* pSizeInit, int* pSizeBuf);
- IppStatus ippiFFTGetSize_C_32fc (int orderX, int orderY, int flag, IppHintAlgorithm hint, int* pSizeSpec, int* pSizeInit, int* pSizeBuf);
Include Files
ippi.h

Domain Dependencies
Headers: ippcore.h, ippvm.h, ipps.h
Libraries: ippcore.lib, ippvm.lib, ipps.lib

Parameters
orderX, orderY  Order of the FFT in x- and y- directions, respectively.
flag  Flag to choose the option for results normalization.
hint  This parameter is deprecated. Set the value to ippAlgHintNone.
pSizeSpec  Pointer to the size of the FFT context structure.
pSizeInit  Pointer to the size of the buffer for the FFT initialization function.
pSizeBuf  Pointer to the size of the FFT external work buffer.

Description
This function computes the following:
• Size of the FFT context structure. The result in bytes is stored in the pSpecSize parameter.
• Size of the work buffer for the ippiFFTInit functions. The result in bytes is stored in the pSizeInit parameter.
• Size of the work buffer for the ippiFFTFwd and ippiFFTInv functions. The result in bytes is stored in the pSizeBuf parameter.

The suffix after the function name indicates the flavors of the FFT functions: ippiFFTGetSize_C is for complex flavors and ippiFFTGetSize_R is for real flavors.

Return Values
ippStsNoErr  Indicates no error.
ippStsNullPtrErr  Indicates an error when one of the specified pointers is NULL.
ippStsFftOrderErr  Indicates an error condition when the FFT order value is illegal.
ippStsFFTFlagErr  Indicates an error condition when the flag value is illegal.

See Also
FFTInit  Initializes the context structure for the image FFT functions.
FFTInv  Applies an inverse FFT to complex source data and stores results in a destination image.
FFTFwd  Applies forward Fast Fourier Transform to an image.

FFTInit
 Initializes the context structure for the image FFT functions.

Syntax
IppStatus ippiFFTInit_R_32f (int orderX, int orderY, int flag, IppHintAlgorithm hint, IppiFFTSpec_R_32f* pFFTSpec, Ipp8u* pMemInit);
IppStatus ippiFFTInit_C_32fc (int orderX, int orderY, int flag, IppHintAlgorithm hint, IppiFFTSpec_C_32fc* pFFTSpec, Ipp8u* pMemInit);

Include Files
ippi.h

Domain Dependencies
Headers: ippcore.h, ippvm.h, ipps.h
Libraries: ippcore.lib, ippvm.lib, ipps.lib

Parameters

orderX, orderY  Order of the FFT in x- and y- directions, respectively.
flag  Flag to choose the option for results normalization.
hint  This parameter is deprecated. Set the value to ippAlgHintNone.
pFFTSpec  Pointer to the FFT context structure.
pMemInit  Pointer to the temporary work buffer.

Description
This function initializes the pFFTSpec context structure needed to compute the forward and inverse FFT of a two-dimensional image data.

Before calling this function, you need to allocate memory for the FFT context structure and temporary work buffer (if it is required). To compute the size of the FFT context structure and temporary work buffer, use the ippiFFTFGetSize function.

The pMemInit parameter can be NULL only if the work buffer is not used. After initialization is done, you can free the temporary work buffer.

The ippiFFTfwd and ippiFFTInv functions called with the pointer to the initialized pFFTSpec structure, compute the fast Fourier transform with the following characteristics:
• length \(N=2^{\text{orderX}}\) in x-direction by \(M=2^{\text{orderY}}\) in y-direction
• results normalization mode as set by \(\text{flag}\) (see Table "Normalization Factors for Fourier Transform Results")
• computation algorithm indicated by \(\text{hint}\).

The suffix after the function name indicates the type of the context structure to be initialized: ippiFFTInit_C is for the complex FFT context structure and ippiFFTInit_R is for the real FFT context structure.

Return Values

ippStsNoErr  Indicates no error.
ippStsNullPtrErr  Indicates an error when one of the specified pointers is NULL.
ippStsFftOrderErr  Indicates an error condition when the FFT order value is illegal.
ippStsFFTFlagErr  Indicates an error condition when the \(\text{flag}\) value is illegal.
Example
The code example below demonstrates how to use the `ippiFFTGetSize` and `ippiFFTInit` functions.

```c
/// get sizes for required buffers
ippiFFTGetSize_R_32f( orderX, orderY, IPP_FFT_DIV_INV_BY_N, ippAlgHintNone, &sizeSpec, &sizeInit, &sizeBuffer);

/// allocate memory for required buffers
pMemSpec = (IppiFFTSpec_R_32f*) ippMalloc ( sizeSpec );

if ( sizeInit > 0 )
{
    pMemInit = ippMalloc ( sizeInit );
}

if ( sizeBuffer > 0 )
{
    pMemBuffer = ippMalloc ( sizeBuffer );
}

/// initialize FFT specification structure
ippiFFTInit_R_32f( orderX, orderY, IPP_FFT_DIV_INV_BY_N, ippAlgHintNone, pMemSpec, pMemInit );

/// free initialization buffer
if ( sizeInit > 0 )
{
    ippFree( pMemInit );
}

/// perform forward FFT to put source data to frequency domain
ippiFFTFwd_RToPack_32f_C1R( pSrc, srcStep, pDst, dstStep, pMemSpec, pMemBuffer );

/// ...  

/// free buffers
if ( sizeBuffer > 0 )
{
    ippFree( pMemBuffer );
}

ippFree( pMemSpec );
```

See Also

- **FFTGetSize** Computes the size of the FFT context structure and the size of the work buffer.
- **FFTInv** Applies an inverse FFT to complex source data and stores results in a destination image.
- **FFTFwd** Applies forward Fast Fourier Transform to an image.

**FFTFwd**

*Applies forward Fast Fourier Transform to an image.*
Syntax

Case 1: Not-in-place operation on floating-point data

IppStatus ippiFFTFwd_RToPack_<mod> (const Ipp32f* pSrc, int srcStep, Ipp32f* pDst, int dstStep, const IppiFFTSpec_R_32f* pFFTSpec, Ipp8u* pBuffer);

Supported values for mod:

32f_C1R
32f_C3R
32f_C4R
32f_AC4R

Case 2: Not-in-place operation on complex data

IppStatus ippiFFTFwd_CToC_32fc_C1R(const Ipp32fc* pSrc, int srcStep, Ipp32fc* pDst, int dstStep, const IppiFFTSpec_C_32fc* pFFTSpec, Ipp8u* pBuffer);

Case 3: In-place operation on floating-point data

IppStatus ippiFFTFwd_RToPack_<mod>(Ipp32f* pSrcDst, int srcDstStep, const IppiFFTSpec_R_32f* pFFTSpec, Ipp8u* pBuffer);

Supported values for mod:

32f_C1IR
32f_C3IR
32f_C4IR
32f_AC4IR

Case 4: In-place operation on complex data

IppStatus ippiFFTFwd_CToC_32fc_C1IR(Ipp32fc* pSrcDst, int srcDstStep, const IppiFFTSpec_C_32fc* pFFTSpec, Ipp8u* pBuffer);

Include Files

ippi.h

Domain Dependencies

Headers: ippcore.h, ippvm.h, ipps.h
Libraries: ippcore.lib, ippvm.lib, ipps.lib

Parameters

pSrc
Pointer to the source image ROI.
srcStep
Distance in bytes between starts of consecutive lines in the source image.
pDst
Pointer to the destination image ROI.
dstStep
Distance in bytes between starts of consecutive lines in the destination image.
**pSrcDst**  
Pointer to the source and destination image ROI for the in-place operation.

**srcDstStep**  
Distance in bytes between starts of consecutive lines in the source and destination image for the in-place operation.

**pFFTSpec**  
Pointer to the FFT specification structure.

**pBuffer**  
Pointer to the external work buffer.

**Description**

This function operates with ROI.

This function performs a forward FFT on each channel of the source image ROI \( pSrc \) (\( pSrcDst \) for in-place flavors) and writes the Fourier coefficients into the corresponding channel of the destination buffer \( pDst \) (\( pSrcDst \) for in-place flavors). The size of ROI is \( N \times M \), it is specified by the parameters \( \text{orderX} \), \( \text{orderY} \).

The function flavor \( \text{ippiFFTfwd\_RToPack} \) that operates on images with real data takes advantage of the symmetry property and stores the output data in RCPack2D format. It supports processing of the 1-, 3-, and 4-channel images. Note that the functions with \( \text{AC4} \) descriptor do not process alpha channel.

The function flavor \( \text{ippiFFTfwd\_CToC} \) that operates on images with complex data does not perform any packing of the transform results as no symmetry with respect to frequency domain data is observed in this case. Memory layout of images with complex data follows the same conventions as for real images provided that each pixel value consists of two numbers: imaginary and real part.

Before using the forward FFT functions, you need to compute the size of the work buffer by \( \text{ippiFFTGetSize} \) and initialize the context structure by the \( \text{ippiFFTInit} \) function. The forward FFT functions use the \( pFFTSpec \) context structure to set the mode of calculations and retrieve support data.

**Return Values**

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ippStsNoErr</td>
<td>Indicates no error. Any other value indicates an error or a warning.</td>
</tr>
<tr>
<td>ippStsNullPtrErr</td>
<td>Indicates an error condition if ( pSrc ), ( pDst ), or ( pFFTSpec ) pointer is NULL.</td>
</tr>
<tr>
<td>ippStsStepErr</td>
<td>Indicates an error condition if ( srcStep ) or ( dstStep ) value is zero or negative.</td>
</tr>
<tr>
<td>ippStsContextMatchErr</td>
<td>Indicates an error condition if a pointer to an invalid ( pFFTSpec ) structure is passed.</td>
</tr>
<tr>
<td>ippStsMemAllocErr</td>
<td>Indicates an error condition if memory allocation fails.</td>
</tr>
</tbody>
</table>

**Example**

To better understand usage of these functions, refer to the following examples in the examples archive available for download from https://software.intel.com/en-us/ipp-manual-examples:

- FFTFwd\_CToC.c
- FFTFwd\_RToPack.c

**See Also**

- Regions of Interest in Intel IPP
- FFTGetSize: Computes the size of the FFT context structure and the size of the work buffer.
- FFTInit: Initializes the context structure for the image FFT functions.
**FFTInv**

Applies an inverse FFT to complex source data and stores results in a destination image.

**Syntax**

**Case 1: Not-in-place operation on floating-point data**

IppStatus ippiFFTInv_PackToR_<mod>(const Ipp32f* pSrc, int srcStep, Ipp32f* pDst, int dstStep, const IppiFFTSpec_R_32f* pFFTSpec, Ipp8u* pBuffer);

Supported values for mod:

- 32f_C1R
- 32f_C3R
- 32f_C4R
- 32f_AC4R

**Case 2: Not-in-place operation on complex data**

IppStatus ippiFFTInv_CToC_32fc_C1R(const Ipp32fc* pSrc, int srcStep, Ipp32fc* pDst, int dstStep, const IppiFFTSpec_C_32fc* pFFTSpec, Ipp8u* pBuffer);

**Case 3: In-place operation on floating-point data**

IppStatus ippiFFTInv_PackToR_<mod>(Ipp32f* pSrcDst, int srcDstStep, const IppiFFTSpec_R_32f* pFFTSpec, Ipp8u* pBuffer);

Supported values for mod:

- 32f_C1IR
- 32f_C3IR
- 32f_C4IR
- 32f_AC4IR

**Case 4: In-place operation on complex data**

IppStatus ippiFFTInv_CToC_32fc_C1IR(Ipp32fc* pSrcDst, int srcDstStep, const IppiFFTSpec_C_32fc* pFFTSpec, Ipp8u* pBuffer);

**Include Files**

ippi.h

**Domain Dependencies**

Headers: ippcore.h, ippvm.h, ipps.h

Libraries: ippcore.lib, ippvm.lib, ipps.lib
Parameters

- **pSrc**: Pointer to the source image ROI.
- **srcStep**: Distance in bytes between starts of consecutive lines in the source image.
- **pDst**: Pointer to the destination image ROI.
- **dstStep**: Distance in bytes between starts of consecutive lines in the destination image.
- **pSrcDst**: Pointer to the source and destination image ROI for the in-place operation.
- **srcDstStep**: Distance in bytes between starts of consecutive lines in the source and destination image for the in-place operation.
- **pFFTSpec**: Pointer to the previously initialized FFT context structure.
- **pBuffer**: Pointer to the external work buffer.

Description

This function operates with ROI.

This function performs an inverse FFT on each channel of the source image `pSrc` (`pSrcDst` for in-place flavors) and writes the restored image data into the corresponding channel of the destination image buffer `pDst` (`pSrcDst` for in-place flavors). The size of ROI is `N x M`, it is specified by the parameters `orderX`, `orderY`.

For the `ippiFFTInv_PackToR` function flavor, the input buffer must contain data in RCPack2D format.

Before using the inverse FFT functions, you need to compute the size of the work buffer by `ippiFFTGetSize` and initialize the context structure by the `ippiFFTInit` function. The inverse FFT functions use the `pFFTSpec` context structure to set the mode of calculations and retrieve support data.

Return Values

- **ippStsNoErr**: Indicates no error. Any other value indicates an error or a warning.
- **ippStsNullPtrErr**: Indicates an error condition if `pSrc`, `pDst`, or `pFFTSpec` pointer is NULL.
- **ippStsStepErr**: Indicates an error condition if `srcStep` or `dstStep` value is zero or negative.
- **ippStsContextMatchErr**: Indicates an error condition if a pointer to an invalid `pFFTSpec` structure is passed.
- **ippStsMemAllocErr**: Indicates an error condition if memory allocation fails.

Example

To better understand usage of these functions, refer to the following examples in the examples archive available for download from https://software.intel.com/en-us/ipp-manual-examples:

- FFTInv_CToC.c
- FFTInv_RToPack.c

See Also

Regions of Interest in Intel IPP
- **FFTGetSize**: Computes the size of the FFT context structure and the size of the work buffer.
FFTInit Initializes the context structure for the image FFT functions.

**DFTGetSize**

Computes the size of the FFT context structure and the size of the work buffer.

**Syntax**

```c
IppStatus ippiDFTGetSize_R_32f (IppiSize roiSize, int flag, IppHintAlgorithm hint, int* pSizeSpec, int* pSizeInit, int* pSizeBuf);

IppStatus ippiDFTGetSize_C_32fc (IppiSize roiSize, int flag, IppHintAlgorithm hint, int* pSizeSpec, int* pSizeInit, int* pSizeBuf);
```

**Include Files**

ippi.h

**Domain Dependencies**

**Headers:** ippcore.h, ippvm.h, ipps.h

**Libraries:** ippcore.lib, ippvm.lib, ipps.lib

**Parameters**

- **roiSize**
  - Size of the source and destination ROI in pixels.

- **flag**
  - Flag to choose the option for results normalization. For more information, see Table "Normalization Factors for Fourier Transform Results"

- **hint**
  - This parameter is deprecated. Set the value to ippAlgHintNone.

- **pSizeSpec**
  - Pointer to the size of the DFT context structure.

- **pSizeInit**
  - Pointer to the size of the buffer for the DFT initialization function.

- **pSizeBuf**
  - Pointer to the size of the DFT external work buffer.

**Description**

This function computes the following:

- Size of the DFT context structure. The result in bytes is stored in the `pSpecSize` parameter.
- Size of the work buffer for the `ippiDFTInit` functions. The result, in bytes, is stored in the `pSizeInit` parameter.
- Size of the work buffer for the `ippiDFTFwd` and `ippiDFTInv` functions. The result, in bytes, is stored in the `pSizeBuf` parameter.

The suffix after the function name indicates the flavors of the DFT functions: `ippiDFTGetSize_C` is for complex flavors and `ippiDFTGetSize_R` is for real flavors.

**Return Values**

- ippStsNoErr: Indicates no error.
- ippStsNullPtrErr: Indicates an error when one of the specified pointers is NULL.
- ippStsFftFlagErr: Indicates an error condition when the `flag` value is illegal.
Indicates an error when the amount of memory needed to compute the DFT for points in the ROI of size \( \text{roiSize} \) exceeds the limit.

Indicates an error condition when the \( \text{roiSize} \) has a field with a zero or negative value.

**See Also**

**DFTInit** Initializes the context structure for the image DFT functions.

---

**DFTInit**  
*Initializes the context structure for the image DFT functions.*

**Syntax**

```c
IppStatus ippiDFTInit_R_32f (IppiSize roiSize, int flag, IppHintAlgorithm hint, IppiDFTSpec_R_32f* pDFTSpec, Ipp8u* pMemInit);
IppStatus ippiDFTInit_C_32fc (IppiSize roiSize, int flag, IppHintAlgorithm hint, IppiDFTSpec_C_32fc* pDFTSpec, Ipp8u* pMemInit);
```

**Include Files**

```c
ippi.h
```

**Domain Dependencies**

**Headers:** ippcore.h, ippvm.h, ipps.h

**Libraries:** ippcore.lib, ippvm.lib, ipps.lib

**Parameters**

- **roiSize**  
  Size of the source and destination ROI in pixels.

- **flag**  
  Flag to choose the option for results normalization.

- **hint**  
  This parameter is deprecated. Set the value to \( \text{ippAlgHintNone} \).

- **pDFTSpec**  
  Pointer to the DFT context structure.

- **pMemInit**  
  Pointer to the temporary work buffer.

**Description**

This function initializes the \( pDFTSpec \) context structure needed to compute the forward and inverse DFT of a two-dimensional image data.

Before calling this function, you need to allocate memory for the FFT context structure and temporary work buffer (if it is required). To compute the size of the FFT context structure and temporary work buffer, use the \( \text{ippiDFTGetSize} \) function.

The \( pMemInit \) parameter can be **NULL** only if the work buffer is not used. After initialization is done, you can free the temporary work buffer.

The \( \text{ippiDFTFwd} \) and \( \text{ippiDFTInv} \) functions called with the pointer to the initialized \( pDFTSpec \) structure compute the discrete Fourier transform with the following characteristics:

- **ROI of the \( \text{roiSize} \) size**
- **results normalization mode set by \( \text{flag} \) (see Table "Normalization Factors for Fourier Transform Results")**
The suffix after the function name indicates the type of the context structure to be initialized: `ippiDFTInit_C` is for the complex DFT context structure and `ippiDFTInit_R` is for the real DFT context structure.

**Return Values**

- `ippStsNoErr` indicates no error.
- `ippStsNullPtrErr` indicates an error when one of the specified pointers is `NULL`.
- `ippStsFftFlagErr` indicates an error condition when the flag value is illegal.
- `ippStsFftOrderErr` indicates an error when the amount of memory needed to compute the DFT for points in the ROI of size `roiSize` exceeds the limit.
- `ippStsSizeErr` indicates an error condition when the `roiSize` has a field with a zero or negative value.

**Example**

The code example below demonstrates how to use the `ippiDFTGetSize` and `ippiDFTInit` functions.

```c
/// get sizes for required buffers
ippiDFTGetSize_R_32f( roiSize, IPP_FFT_DIV_INV_BY_N, ippAlgHintNone, &sizeSpec, &sizeInit, &sizeBuffer );

/// allocate memory for required buffers
pMemSpec = (IppiDFTSpec_R_32f*) ippMalloc ( sizeSpec );

if ( sizeInit > 0 )
{
    pMemInit = ippMalloc ( sizeInit );
}

if ( sizeBuffer > 0 )
{
    pMemBuffer = ippMalloc ( sizeBuffer );
}

/// initialize DFT specification structure
ippiDFTInit_R_32f( roiSize, IPP_FFT_DIV_INV_BY_N, ippAlgHintNone, pMemSpec, pMemInit );

/// free initialization buffer
if ( sizeInit > 0 )
{
    ippFree( pMemInit );
}

/// perform forward DFT to put source data to frequency domain
ippiDFTFwd_RToPack_32f_C1R( pSrc, srcStep, pDst, dstStep, pMemSpec, pMemBuffer );

/// ...

/// free buffers
if ( sizeBuffer > 0 )
{
    ippFree( pMemBuffer );
}```
See Also

Regions of Interest in Intel IPP

DFTGetSize Computes the size of the FFT context structure and the size of the work buffer.

DFTFwd Applies forward discrete Fourier transform to an image.

DFTInv Applies an inverse DFT to complex source data and stores results in a destination image.

**DFTFwd**

*Applies forward discrete Fourier transform to an image.*

**Syntax**

**Case 1: Not-in-place operation on floating-point data**

IppStatus ippiDFTFwd_RToPack_<mod>(const Ipp32f* pSrc, int srcStep, Ipp32f* pDst, int dstStep, const IppiDFTSpec_R_32f* pDFTSpec, Ipp8u* pBuffer);

Supported values for mod:

- 32f_C1R
- 32f_C3R
- 32f_C4R
- 32f_AC4R

**Case 2: Not-in-place operation on complex data**

IppStatus ippiDFTFwd_CToC_32fc_C1R(const Ipp32fc* pSrc, int srcStep, Ipp32fc* pDst, int dstStep, const IppiDFTSpec_C_32fc* pDFTSpec, Ipp8u* pBuffer);

**Case 3: In-place operation on floating-point data**

IppStatus ippiDFTFwd_RToPack_<mod>(Ipp32f* pSrcDst, int srcDstStep, const IppiDFTSpec_R_32f* pDFTSpec, Ipp8u* pBuffer);

Supported values for mod:

- 32f_C1IR
- 32f_C3IR
- 32f_C4IR
- 32f_AC4IR

**Case 4: In-place operation on complex data**

IppStatus ippiDFTFwd_CToC_32fc_C1IR(Ipp32fc* pSrcDst, int srcDstStep, const IppiDFTSpec_C_32fc* pDFTSpec, Ipp8u* pBuffer);

**Include Files**

ippi.h
Domain Dependencies

Headers: ippcore.h, ippvm.h, ipps.h
Libraries: ippcore.lib, ippvm.lib, ipps.lib

Parameters

pSrc
Pointer to the source image ROI.

srcStep
Distance in bytes between starts of consecutive lines in the source image.

pDst
Pointer to the destination image ROI.

dstStep
Distance in bytes between starts of consecutive lines in the destination image.

pSrcDst
Pointer to the source and destination image ROI for the in-place operation.

srcDstStep
Distance in bytes between starts of consecutive lines in the source and destination image for the in-place operation.

pDFTSpec
Pointer to the previously initialized DFT context structure.

pBuffer
Pointer to the external work buffer.

Description

This function operates with ROI.

This function performs a forward DFT on each channel of the source image ROI pSrc (pSrcDst for in-place flavors) and writes the Fourier coefficients into the corresponding channel of the destination buffer pDst (pSrcDst for in-place flavors).

The function flavorippiDFTFwd_RToPack that operates on images with real data takes advantage of the symmetry property and stores the output data in RCPack2D format. It supports processing of the 1-, 3-, and 4-channel images. Note that the functions with AC4 descriptor do not process alpha channel.

The function flavorippiDFTFwd_CToC that operates on images with complex data performs no packing of the transform results as no symmetry with respect to frequency domain data is observed in this case. Memory layout of images with complex data follows the same conventions as for real images provided that each pixel value consists of two numbers: imaginary and real part.

Before using the forward DFT functions, you need to compute the size of the work buffer byippiDFTGetSize and initialize the context structure by theippiDFTInit function. The forward DFT functions use the pDFTSpec context structure to set the mode of calculations and retrieve support data.

Return Values

ippStsNoErr
Indicates no error. Any other value indicates an error or a warning.

ippStsNullPtrErr
Indicates an error condition if pSrc, pDst, or pDFTSpec pointer is NULL.

ippStsStepErr
Indicates an error condition if srcStep or dstStep value is zero or negative.

ippStsContextMatchErr
Indicates an error condition if a pointer to an invalid pDFTSpec structure is passed.

ippStsMemAllocErr
Indicates an error condition if memory allocation fails.
See Also
Regions of Interest in Intel IPP
DFTGetSize Computes the size of the FFT context structure and the size of the work buffer.
DFTInit Initializes the context structure for the image DFT functions.

DFTInv
Applies an inverse DFT to complex source data and stores results in a destination image.

Syntax
Case 1: Not-in-place operation on floating-point data
IppStatus ippiDFTInv_PackToR_<mod>(const Ipp32f* pSrc, int srcStep, Ipp32f* pDst, int dstStep, const IppiDFTSpec_R_32f* pDFTSpec, Ipp8u* pBuffer);

Supported values for mod:

- 32f_C1R
- 32f_C3R
- 32f_C4R
- 32f_AC4R

Case 2: Not-in-place operation on complex data
IppStatus ippiDFTInv_CToC_32fc_C1R(const Ipp32fc* pSrc, int srcStep, Ipp32fc* pDst, int dstStep, const IppiDFTSpec_C_32fc* pDFTSpec, Ipp8u* pBuffer);

Case 3: In-place operation on floating-point data
IppStatus ippiDFTInv_PackToR_<mod>(Ipp32f* pSrcDst, int srcDstStep, const IppiDFTSpec_R_32f* pDFTSpec, Ipp8u* pBuffer);

Supported values for mod:

- 32f_C1IR
- 32f_C3IR
- 32f_C4IR
- 32f_AC4IR

Case 5: In-place operation on complex data
IppStatus ippiDFTInv_CToC_32fc_C1IR(Ipp32fc* pSrcDst, int srcDstStep, const IppiDFTSpec_C_32fc* pDFTSpec, Ipp8u* pBuffer);

Include Files
ippi.h

Domain Dependencies
Headers: ippcore.h, ippvm.h, ipps.h
Libraries: ippcore.lib, ippvm.lib, ipps.lib

Parameters

- **pSrc**
  Pointer to the source image ROI.
- **srcStep**
  Distance in bytes between starts of consecutive lines in the source image.
- **pDst**
  Pointer to the destination image ROI.
- **dstStep**
  Distance in bytes between starts of consecutive lines in the destination image.
- **pSrcDst**
  Pointer to the source and destination image ROI for the in-place operation.
- **srcDstStep**
  Distance in bytes between starts of consecutive lines in the source and destination image for the in-place operation.
- **pDFTSpec**
  Pointer to the previously initialized DFT context structure.
- **pBuffer**
  Pointer to the external work buffer.

Description

This function operates with ROI.

This function performs an inverse DFT on each channel of the input buffer \( pSrc \) (\( pSrcDst \) for in-place flavors) and writes the restored image data into the corresponding channel of the output image buffer \( pDst \) (\( pSrcDst \) for in-place flavors).

For function flavor ippiDFTInv_PackToR, the input buffer must contain data in RCPack2D format.

Before using the inverse DFT functions, you need to compute the size of the work buffer by ippiDFTGetSize and initialize the context structure by the ippiDFTInit function. The inverse DFT functions use the \( pDFTSpec \) context structure to set the mode of calculations and retrieve support data.

Return Values

- **ippStsNoErr**
  Indicates no error. Any other value indicates an error or a warning.
- **ippStsNullPtrErr**
  Indicates an error condition if \( pSrc, pDst, \) or \( pDFTSpec \) pointer is NULL.
- **ippStsStepErr**
  Indicates an error condition if \( srcStep \) or \( dstStep \) value is zero or negative.
- **ippStsContextMatchErr**
  Indicates an error condition if a pointer to an invalid \( pDFTSpec \) structure is passed.
- **ippStsMemAllocErr**
  Indicates an error condition if memory allocation fails.

See Also

- **Regions of Interest in Intel IPP**
- **DFTGetSize** Computes the size of the FFT context structure and the size of the work buffer.
- **DFTInit** Initializes the context structure for the image DFT functions.

**MulPack**

*Multiplies two source images in packed format.*
Syntax

IppStatus ippiMulPack_<mod>(const Ipp32f* pSrc1, int src1Step, const Ipp32f* pSrc2, int src2Step, Ipp32f* pDst, int dstStep, IppiSize roiSize);

Supported values for mod:

32f_C1R
32f_C3R
32f_C4R
32f_AC4R

IppStatus ippiMulPack_<mod>(const Ipp32f* pSrc, int srcStep, Ipp32f* pSrcDst, int srcDstStep, IppiSize roiSize);

Supported values for mod:

32f_C1IR
32f_C3IR
32f_C4IR
32f_AC4IR

Include Files

ippi.h

Domain Dependencies

Headers: ippcore.h, ippvm.h, ipps.h
Libraries: ippcore.lib, ippvm.lib, ipps.lib

Parameters

pSrc1, pSrc2
src1Step, src2Step
pDst
dstStep
pSrc
srcStep
pSrcDst
srcDstStep
roiSize

Pointer to the ROI in the source images.
Distance in bytes between starts of consecutive lines in the source images.
Pointer to the destination image ROI.
Distance in bytes between starts of consecutive lines in the destination image.
Pointer to the first source image ROI for the in-place operation.
Distance in bytes between starts of consecutive lines in the first source image for the in-place operation.
Pointer to the second source and destination image ROI for the in-place operation.
Distance in bytes between starts of consecutive lines in the source and destination image for the in-place operation.
Size of the source and destination ROI in pixels.
Scale factor (see Integer Result Scaling).

Description
This function operates with ROI (see Regions of Interest in Intel IPP).
This function multiplies corresponding pixel values of two source images, \( A \) and \( B \) represented in RCPack2D format and stores the result into the destination image \( C \) in packed format also. The multiplying is performed according to the following formulas:
\[
\begin{align*}
\text{Re} C &= \text{Re} A \times \text{Re} B - \text{Im} A \times \text{Im} B; \\
\text{Im} C &= \text{Im} A \times \text{Re} B + \text{Im} B \times \text{Re} A.
\end{align*}
\]
Not-in-place flavors multiply pixel values of ROI in the source images \( pSrc1 \) and \( pSrc2 \), and store result in the \( pDst \).
In-place flavors multiply pixel values of ROI in the source images \( pSrc \) and \( pSrcDst \), and store result in the \( pSrcDst \).
This function can be used in image filtering operations that include FFT transforms.

Example
To better understand usage of this function, refer to the following example in the examples archive available for download from https://software.intel.com/en-us/ipp-manual-examples:

MulPack.c

Return Values
- ippStsNoErr: Indicates no error. Any other value indicates an error or a warning.
- ippStsNullPtrErr: Indicates an error condition if any of the specified pointers is NULL.
- ippStsSizeErr: Indicates an error condition if \( roiSize \) has a field with zero or negative value.
- ippStsStepErr: Indicates an error condition if any of the specified buffer step values is zero or negative.

MulPackConj
Multiplying a source image by the complex conjugate image with data in packed format and stores the result in the destination buffer in the packed format.

Syntax
Case 1: Not-in-place operation
\[
\text{IppStatus iippiMulPackConj}_<\text{mod}>(\text{const Ipp32f*} \ pSrc1, \text{int src1Step}, \text{const Ipp32f*} \ pSrc2, \text{int src2Step}, \text{Ipp32f*} \ pDst, \text{int dstStep}, \text{IppiSize roiSize});
\]
Supported values for \( \text{mod} \):
- 32f_C1R
- 32f_C3R
Case 2: In-place operation

IppStatus ippiMulPackConj_mod(const Ipp32f* pSrc, int srcStep, Ipp32f* pSrcDst, int srcDstStep, IppiSize roiSize);

Supported values for mod:

- 32f_C1IR
- 32f_C3IR
- 32f_C4IR
- 32f_AC4IR

Include Files

ippi.h

Domain Dependencies

Headers: ippcore.h, ippvm.h, ipps.h
Libraries: ippcore.lib, ippvm.lib, ipps.lib

Parameters

- pSrc1, pSrc2: Pointer to the ROI in the source images.
- src1Step, src2Step: Distance in bytes between starts of consecutive lines in the source images.
- pDst: Pointer to the destination image ROI.
- dstStep: Distance in bytes between starts of consecutive lines in the destination image.
- pSrc: Pointer to the first source image ROI for the in-place operation.
- srcStep: Distance in bytes between starts of consecutive lines in the first source image for the in-place operation.
- pSrcDst: Pointer to the second source and destination image ROI for the in-place operation.
- srcDstStep: Distance in bytes between starts of consecutive lines in the source and destination image for the in-place operation.
- roiSize: Size of the source and destination ROI in pixels.

Description

This function operates with ROI (see Regions of Interest in Intel IPP).

This function multiplies pixel values of the source image \( A \) by the corresponding pixel values of the complex conjugate image \( A^* \), represented in RCPack2D format. The result of the operation is written into the destination buffer in packed format also.

Not-in-place flavors multiply pixel values of ROI in the source images \( pSrc1 \) and \( pSrc2 \), and store result in the \( pDst \).
In-place flavors multiply pixel values of ROI in the source images \( p_{Src} \) and \( p_{SrcDst} \), and store result in the \( p_{SrcDst} \).

**Return Values**

- `ippStsNoErr`: Indicates no error. Any other value indicates an error or a warning.
- `ippStsNullPtrErr`: Indicates an error condition if any of the specified pointers is NULL.
- `ippStsSizeErr`: Indicates an error condition if \( roiSize \) has a field with zero or negative value.
- `ippStsStepErr`: Indicates an error condition if any of the specified buffer step values is zero or negative.

**Magnitude**

*Computes magnitude of elements of a complex data image.*

**Syntax**

\[
\text{IppStatus ippiMagnitude\_<mod>(const Ipp<srcDatatype>* pSrc, int srcStep, Ipp<dstDatatype>* pDst, int dstStep, IppiSize roiSize);}\
\]

Supported values for `mod`:

- `32fc32f_C1R`
- `32fc32f_C3R`

**Include Files**

ippi.h

**Domain Dependencies**

Headers: ippcore.h, ippvm.h, ipps.h

Libraries: ippcore.lib, ippvm.lib, ipps.lib

**Parameters**

- `pSrc`: Pointer to the source image ROI.
- `srcStep`: Distance in bytes between starts of consecutive lines in the source image.
- `pDst`: Pointer to the destination image ROI.
- `dstStep`: Distance in bytes between starts of consecutive lines in the destination image.
- `roiSize`: Size of the source and destination ROI in pixels.

**Description**

This function operates with ROI (see Regions of Interest in Intel IPP).

This function computes magnitude of elements of the source image \( p_{Src} \) given in complex data format, and stores results in the destination image \( p_{Dst} \).
Return Values

<table>
<thead>
<tr>
<th>Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ippStsNoErr</td>
<td>Indicates no error. Any other value indicates an error or a warning.</td>
</tr>
<tr>
<td>ippStsNullPtrErr</td>
<td>Indicates an error condition if pSrc or pDst pointer is NULL.</td>
</tr>
<tr>
<td>ippStsStepErr</td>
<td>Indicates an error condition if srcStep or dstStep value is zero or negative.</td>
</tr>
<tr>
<td>ippStsSizeErr</td>
<td>Indicates an error condition if width or height of images is less than or equal to zero.</td>
</tr>
</tbody>
</table>

**MagnitudePackGetBufferSize**

*Computes the size of the work buffer for the ippiMagnitudePack function.*

**Syntax**

```c
IppStatus ippiMagnitudePackGetBufferSize_32f (int numChannels, IppiSize dstRoiSize, int* pSize);
```

**Include Files**

ippi.h

**Domain Dependencies**

*Headers:* ippcore.h, ippvm.h, ipps.h  
*Libraries:* ippcore.lib, ippvm.lib, ipps.lib

**Parameters**

- **numChannels**
  - Number of channels in the image. Possible values are 1 and 3.
- **dstRoiSize**
  - Size, in pixels, of the destination image ROI.
- **pSize**
  - Pointer to the computed size of the external work buffer, in bytes.

**Description**

The `ippiMagnitudePackGetBufferSize` function computes the size, in bytes, of the external work buffer needed for the `ippiMagnitudePack` function. The result is stored in the `pSize` parameter.

**Return Values**

<table>
<thead>
<tr>
<th>Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ippStsNoErr</td>
<td>Indicates no error.</td>
</tr>
<tr>
<td>ippStsNullPtrErr</td>
<td>Indicates an error when <code>pSize</code> is NULL.</td>
</tr>
<tr>
<td>ippStsSizeErr</td>
<td>Indicates an error when <code>dstRoiSize</code> has a field with a value less than 1.</td>
</tr>
<tr>
<td>ippStsNumChannelsErr</td>
<td>Indicates an error when <code>numChannels</code> has an illegal value.</td>
</tr>
</tbody>
</table>

**See Also**

MagnitudePackMODIFIED API. Computes magnitude of elements of an image in packed format.
MagnitudePack

MODIFIED API. Computes magnitude of elements of an image in packed format.

Syntax

IppStatus ippiMagnitudePack_<mod>(const Ipp<datatype>* pSrc, int srcStep,
Ipp<datatype>* pDst, int dstStep, IppiSize dstRoiSize, Ipp8u* pBuffer);

Supported values for mod:

32f_C1R  32f_C3R

Include Files

ippi.h

Domain Dependencies

Headers: ippcore.h, ippvm.h, ipps.h
Libraries: ippcore.lib, ippvm.lib, ipps.lib

Parameters

pSrc Pointer to the source image ROI.
srcStep Distance in bytes between starts of consecutive lines in the source image.
pDst Pointer to the destination image ROI.
dstStep Distance in bytes between starts of consecutive lines in the destination image.
dstRoiSize Size of the destination ROI in pixels.
pBuffer Pointer to the work buffer. To compute the size of the buffer, use the ippiMagnitudePackGetBufferSize function.

Description

Important The API of this function has been modified in Intel IPP 9.0 release.

This function operates with ROI (see Regions of Interest in Intel IPP).

This function computes magnitude of elements of the source image pSrc given in RCPack2D format and stores results in the destination image pDst.

Return Values

ippStsNoErr Indicates no error. Any other value indicates an error or a warning.

ippStsNullPtrErr Indicates an error when pSrc or pDst pointer is NULL.

ippStsStepErr Indicates an error when srcStep or dstStep value is zero or negative.
ippStsSizeErr  Indicates an error when width or height of images is less than, or equal to zero.

See Also
MagnitudePackGetBufferSize Computes the size of the work buffer for theippiMagnitudePack function.
Regions of Interest in Intel IPP
Real - Complex Packed (RCPack2D) Format

Phase
Computes the phase of elements of a complex data image.

Syntax
IppStatus ippiPhase_<mod>(const Ipp<srcDatatype>* pSrc, int srcStep, Ipp<dstDatatype>* pDst, int dstStep, IppiSize roiSize);

Supported values for mod:
32fc32f_C1R  32fc32f_C3R

Include Files
ippi.h

Domain Dependencies
Headers: ippcore.h, ippvm.h, ipps.h
Libraries: ippcore.lib, ippvm.lib, ipps.lib

Parameters
pSrc  Pointer to the source image ROI.
srcStep  Distance in bytes between starts of consecutive lines in the source image.
pDst  Pointer to the destination image ROI.
dstStep  Distance in bytes between starts of consecutive lines in the destination image.
roiSize  Size of the source and destination ROI in pixels.

Description
This function operates with ROI (see Regions of Interest in Intel IPP).
This function computes the phase in radians of elements of a source image pSrc given in complex data format, and stores results in the destination image pDst.

Return Values
ippStsNoErr  Indicates no error. Any other value indicates an error or a warning.
ippStsNullPtrErr  Indicates an error condition if pSrc or pDst pointer is NULL.
ippStsStepErr Indicates an error condition if srcStep or dstStep value is zero or negative.

ippStsSizeErr Indicates an error condition if width or height of images is less than or equal to zero.

PhasePackGetBufferSize
Computes the size of the work buffer for theippiPhasePack function.

Syntax
IppStatus ippiPhasePackGetBufferSize_32f (int numChannels, IppiSize dstRoiSize, int* pSize);

Include Files
ippi.h

Domain Dependencies
Headers: ippcore.h, ippvm.h, ipps.h
Libraries: ippcore.lib, ippvm.lib, ipps.lib

Parameters
numChannels Number of channels in the image. Possible values are 1 and 3.
dstRoiSize Size, in pixels, of the destination image ROI.
pSize Pointer to the computed size of the external work buffer, in bytes.

Description
TheippiPhasePackGetBufferSize function computes the size, in bytes, of the external work buffer needed for theippiPhasePack function. The result is stored in the pSize parameter.

Return Values
ippStsNoErr Indicates no error.
ippStsNullPtrErr Indicates an error when pSize is NULL.
ippStsSizeErr Indicates an error when dstRoiSize has a field with a value less than 1.
ippStsNumChannelsErr Indicates an error when numChannels has an illegal value.

See Also
PhasePackMODIFIED API. Computes the phase of elements of an image in packed format.

PhasePack
MODIFIED API. Computes the phase of elements of an image in packed format.
Syntax

IppStatus ippiPhasePack_mod(const Ippdatatype*pSrc, int srcStep, Ippdatatype*pDst, int dstStep, IppSize dstRoiSize, Ipp8u*pBuffer);

Supported values for mod:

32f_C1R  32f_C3R

Include Files

ippi.h

Domain Dependencies

Headers: ippcore.h, ippvm.h, ipps.h
Libraries: ippcore.lib, ippvm.lib, ipps.lib

Parameters

pSrc
srcStep
pDst
dstStep
dstRoiSize
pBuffer

Description

Important The API of this function has been modified in Intel IPP 9.0 release.

This function operates with ROI (see Regions of Interest in Intel IPP).
This function computes the phase of elements of a source image pSrc given in RCPack2D format and stores the results in the destination image pDst.

Return Values

ippStsNoErr Indicates no error. Any other value indicates an error or a warning.
ippStsNullPtrErr Indicates an error when pSrc or pDst pointer is NULL.
ippStsStepErr Indicates an error when srcStep or dstStep value is zero or negative.
ippStsSizeErr Indicates an error when width or height of images is less than, or equal to zero.

See Also

PhasePackGetBufferSize Computes the size of the work buffer for the ippiPhasePack function.
Regions of Interest in Intel IPP
Real - Complex Packed (RCPack2D) Format
PolarToCart

Converts an image in the polar coordinate form to Cartesian coordinate form.

Syntax

IppStatus ippiPolarToCart_<mod>(const Ipp32f* pSrcMagn, const Ipp32f* pSrcPhase, int srcStep, IppiSize roiSize, Ipp32fc* pDst, int dstStep);

Supported values for mod:

32fc_C1R  32fc_C3R

Include Files

ippi.h

Domain Dependencies

Headers: ippcore.h, ippvm.h, ipps.h
Libraries: ippcore.lib, ippvm.lib, ipps.lib

Parameters

pSrcMagn  Pointer to the buffer containing magnitudes of the source image.
pSrcPhase  Pointer to the buffer containing phase values of the source image.
srcStep  Distance in bytes between starts of consecutive lines in the source buffers.
roiSize  Size of the source and destination image ROI.
pDst  Pointer to the destination image ROI.
dstStep  Distance in bytes between starts of consecutive lines in the destination image buffer.

Description

This function operates with ROI (see Regions of Interest in Intel IPP).

This function converts the polar coordinate of the source image stored in the arrays of magnitudes pSrcMagn and phase values pSrcPhase to the destination image pDst in complex-data format (in Cartesian coordinate form).

Return Values

ippStsNoErr  Indicates no error. Any other value indicates an error or a warning.
ippStsNullPtrErr  Indicates an error condition if one of the specified pointers is NULL.
ippStsStepErr  Indicates an error condition if srcStep or dstStep value is zero or negative.
ippStsSizeErr  Indicates an error condition if srcSize has a field with value less than 1.
Example

The code example below shows how to use the function ippiPolarToCart_32fc_C1R.

```c
void func_polartocart()
{
    Ipp32f pSrcMagn[2*2] = {1.0, 0.0, 2.1, 3.2};
    Ipp32f pSrcPhase[2*2] = {0.0, 2.0, 1.6, -1.0};
    Ipp32fc pDst[2*2] = {0};
    int srcStep = 2*sizeof(Ipp32f);
    int dstStep = 2*sizeof(Ipp32fc);
    IppiSize roiSize = {2, 2};

    ippiPolarToCart_32fc_C1R(pSrcMagn, pSrcPhase, srcStep, roiSize, pDst, dstStep);
}

Result: pDst -> (1.0, 0.0) (0.0, 0.0) (-0.1, 2.1) (1.7, -2.7)
```

PackToCplxExtend

Converts an image in packed format to a complex data image.

**Syntax**

```c
IppStatus ippiPackToCplxExtend_32f32fc_C1R(const Ipp32f* pSrc, IppiSize srcSize, int srcStep, Ipp32fc* pDst, int dstStep);
```

**Include Files**

ippi.h

**Domain Dependencies**

**Headers**: ippcore.h, ippvm.h, ipps.h

**Libraries**: ippcore.lib, ippvm.lib, ipps.lib

**Parameters**

- **pSrc**
  Pointer to the source image ROI.
- **srcSize**
  Size in pixels of the source image ROI.
- **srcStep**
  Distance in bytes between starts of consecutive lines in the source buffer.
- **pDst**
  Pointer to the destination image buffer.
- **dstStep**
  Distance in bytes between starts of consecutive lines in the destination image buffer.
Description

This function operates with ROI (see Regions of Interest in Intel IPP).

This function converts the source image pSrc in RCPack2D format to complex data format and stores the results in pDst, which is a matrix with complete set of the Fourier coefficients. Note that if the pSrc in RCPack2D format is a real array of dimensions (N x M), then the pDst is a real array of dimensions (2 x N x M). This should be taken into account when allocating memory for the function operation.

Return Values

- **ippStsNoErr**: Indicates no error. Any other value indicates an error or a warning.
- **ippStsNullPtrErr**: Indicates an error condition if pSrc or pDst pointer is NULL.
- **ippStsStepErr**: Indicates an error condition if srcStep or dstStep value is zero or negative.
- **ippStsSizeErr**: Indicates an error condition if srcSize has field with zero or negative value.

**CplxExtendToPack**

Converts a complex data image to an image in packed format.

**Syntax**

```c
IppStatus ippiCplxExtendToPack_<mod>(const Ipp<srcDatatype>* pSrc, int srcStep, IppiSize srcSize, Ipp<dstDatatype>* pDst, int dstStep);
```

Supported values for **mod**:

- `32fc32f_C1R`
- `32fc32f_C3R`

**Include Files**

ippi.h

**Domain Dependencies**

- **Headers**: ippcore.h, ippvm.h, ipps.h
- **Libraries**: ippcore.lib, ippvm.lib, ipps.lib

**Parameters**

- **pSrc**: Pointer to the source image ROI.
- **srcSize**: Size in pixels of the source image ROI.
- **srcStep**: Distance in bytes between starts of consecutive lines in the source buffer.
- **pDst**: Pointer to the destination image buffer.
- **dstStep**: Distance in bytes between starts of consecutive lines in the destination image buffer.
Description
This function operates with ROI (see Regions of Interest in Intel IPP).
This function converts the source image pSrc in complex data format to RCPack2D format and stores the results in pDst, which is a real array of dimensions (N*M). The pSrc is a matrix with complete set of the Fourier coefficients.

Return Values

ippStsNoErr  
Indicates no error. Any other value indicates an error or a warning.

ippStsNullPtrErr  
Indicates an error condition if pSrc or pDst pointer is NULL.

ippStsStepErr  
Indicates an error condition if srcStep or dstStep value is zero or negative.

ippStsSizeErr  
Indicates an error condition if srcSize has field with zero or negative value.

Windowing Functions

This section describes Intel IPP windowing functions used in image processing. A window is a mathematical function by which pixel values are multiplied to prepare an image for the subsequent analysis. This procedure is often called ‘windowing’. In fact, a window function approaches zero towards the edges of the image avoiding strong distortions of spectral densities in the Fourier domain.

The Intel IPP provides two following types of window functions:

- Bartlett window function
- Hamming window function

These functions generate the window samples and applied them to the specified image. To obtain the window samples themselves, you should apply the desired function to the image with all pixel values set to 1.0. As the windowing operation is very time consuming, it may be useful if you want to apply the same window to the multiple images. In this case use one of the image multiplication functions (ippiMul) to multiply the pixel values of the image by the window samples.

WinBartlettGetBufferSize, WinBartlettSepGetBufferSize,

*Compute the size of the work buffer for the*  
ippiWinBartlett orippiWinBartlettSep function.*

Syntax

IppStatus ippiWinBartlettGetBufferSize (IppDataType dataType, IppiSize roiSize, int* pSize);

IppStatus ippiWinBartlettSepGetBufferSize (IppDataType dataType, IppiSize roiSize, int* pSize);

Include Files

ippi.h

Domain Dependencies

Headers: ippcore.h, ippvm.h, ipps.h

Libraries: ippcore.lib, ippvm.lib, ipps.lib
Parameters

**dataType**
Data type for the Bartlett window function. Possible values are: ipp8u, ipp16u, or ipp32f.

**roiSize**
Size, in pixels, of the image ROI.

**pSize**
Pointer to the computed size of the external work buffer, in bytes.

Description
The `ippiWinBartlettGetBufferSize` and `ippiWinBartlettSepGetBufferSize` functions compute the size, in bytes, of the external work buffer needed for the `ippiWinBartlett` or `ippiWinBartlettSep` function. The result is stored in the `pSize` parameter.

Return Values

- **ippStsNoErr**: Indicates no error.
- **ippStsNullPtrErr**: Indicates an error when `pSize` is NULL.
- **ippStsSizeErr**: Indicates an error when `roiSize` has a field with a value less than 3.
- **ippStsNumChannelsErr**: Indicates an error when `dataType` has an illegal value.

See Also

WinBartlett, WinBartlettSep
MODIFIED API. Apply Bartlett window function to the image.

**WinBartlett, WinBartlettSep**

MODIFIED API. Apply Bartlett window function to the image.

Syntax

**Case 1: Not-in-place operation**

```c
IppStatus ippiWinBartlett_<mod>(const Ipp<datatype>* pSrc, int srcStep, Ipp<datatype>* pDst, int dstStep, IppiSize roiSize, Ipp8u* pBuffer);
```

Supported values for `mod`:

- 8u_C1R
- 16u_C1R
- 32f_C1R

```c
IppStatus ippiWinBartlettSep_<mod>(const Ipp<datatype>* pSrc, int srcStep, Ipp<datatype>* pDst, int dstStep, IppiSize roiSize, Ipp8u* pBuffer);
```

Supported values for `mod`:

- 8u_C1R
- 16u_C1R
- 32f_C1R
Case 2: In-place operation

IppStatus ippiWinBartlett_<mod>(Ipp<datatype>* pSrcDst, int srcDstStep, IppiSize roiSize, Ipp8u* pBuffer);

Supported values for mod:
8u_C1IR   16u_C1IR   32f_C1IR

IppStatus ippiWinBartlettSep_<mod>(Ipp<datatype>* pSrcDst, int srcDstStep, IppiSize roiSize, Ipp8u* pBuffer);

Supported values for mod:
8u_C1IR   16u_C1IR   32f_C1IR

Include Files
ippi.h

Domain Dependencies

Headers: ippcore.h, ippvm.h, ipps.h
Libraries: ippcore.lib, ippvm.lib, ipps.lib

Parameters

pSrc
Pointer to the source image ROI.
srcStep
Distance, in bytes, between the starting points of consecutive lines in the source image.
pDst
dstStep
Pointer to the destination image ROI.
Distance, in bytes, between the starting points of consecutive lines in the destination image.
pSrcDst
srcDstStep
 Pointer to the source and destination image ROI for the in-place operation.
Distance, in bytes, between the starting points of consecutive lines in the source and destination image for the in-place operation.
roiSize
Size of the source and destination ROI, in pixels.
pBuffer

Description

These functions operate with ROI (see Regions of Interest in Intel IPP).
These functions compute the Bartlett (triangle) window samples, multiply pixel values of the source image pSrc (pSrcDst for in-place flavors) with these samples, and store results in the destination image pDst (pSrcDst for in-place flavors).

The Bartlett window function for one-dimensional case with $M$ elements is defined as follows:

$$W_{\text{bartlett}}(i) = \begin{cases} \frac{2i}{M-1}, & 0 \leq i \leq \frac{M-1}{2} \\ \frac{2(M-1)-2i}{M-1}, & \frac{M-1}{2} < i \leq M-1 \end{cases}$$
The `ippiWinBartlettSep` flavor applies the window function successively to the rows and then to the columns of the image.

**Return Values**

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ippStsNoErr</td>
<td>Indicates no error. Any other value indicates an error or a warning.</td>
</tr>
<tr>
<td>ippStsNullPtrErr</td>
<td>Indicates an error when <code>pSrc</code> or <code>pDst</code> pointer is <code>NULL</code>.</td>
</tr>
<tr>
<td>ippStsStepErr</td>
<td>Indicates an error when <code>srcStep</code> or <code>dstStep</code> value is zero or negative.</td>
</tr>
<tr>
<td>ippStsSizeErr</td>
<td>Indicates an error when width or height of images is less than, or equal to zero.</td>
</tr>
</tbody>
</table>

**See Also**

`WinHammingGetBufferSize`, `WinHammingSepGetBufferSize` Compute the size of the work buffer for the `ippiWinHamming` or `ippiWinHammingSep` function.

**Regions of Interest in Intel IPP**

`WinHammingGetBufferSize`, `WinHammingSepGetBufferSize`,  
*Compute the size of the work buffer for the*  
`ippiWinHamming` or `ippiWinHammingSep` *function.*

**Syntax**

```c
IppStatus ippiWinHammingGetBufferSize (IppDataType dataType, IppSize roiSize, int* pSize);
IppStatus ippiWinHammingSepGetBufferSize (IppDataType dataType, IppSize roiSize, int* pSize);
```

**Include Files**

`ippi.h`

**Domain Dependencies**

- **Headers:** ippcore.h, ippvm.h, ipps.h
- **Libraries:** ippcore.lib, ippvm.lib, ipps.lib

**Parameters**

- **dataType**  
  Data type for the Bartlett window function. Possible values are: ipp8u, ipp16u, or ipp32f.

- **roiSize**  
  Size, in pixels, of the image ROI.

- **pSize**  
  Pointer to the computed size of the external work buffer, in bytes.

**Description**

The `ippiWinHammingGetBufferSize` and `ippiWinHammingSepGetBufferSize` functions compute the size, in bytes, of the external work buffer needed for the `ippiWinHamming` or `ippiWinHammingSep` function. The result is stored in the `pSize` parameter.
Return Values

ippStsNoErr  Indicates no error.
ippStsNullPtrErr  Indicates an error when pSize is NULL.
ippStsSizeErr  Indicates an error when roiSize has a field with a value less than 3.
ippStsNumChannelsErr  Indicates an error when dataType has an illegal value.

See Also
WinHamming, WinHammingSepMODIFIED API. Apply Hamming window function to the image.

WinHamming, WinHammingSep
MODIFIED API. Apply Hamming window function to the image.

Syntax

Case 1: Not-in-place operation
IppStatus ippiWinHamming_<mod>(const Ipp<datatype>* pSrc, int srcStep, Ipp<datatype>* pDst, int dstStep, IppiSize roiSize, Ipp8u* pBuffer);

Supported values for mod:

8u_C1R  16u_C1R  32f_C1R

IppStatus ippiWinHammingSep_<mod>(const Ipp<datatype>* pSrc, int srcStep, Ipp<datatype>* pDst, int dstStep, IppiSize roiSize, Ipp8u* pBuffer);

Supported values for mod:

8u_C1R  16u_C1R  32f_C1R

Case 2: In-place operation
IppStatus ippiWinHamming_<mod>(Ipp<datatype>* pSrcDst, int srcDstStep, IppiSize roiSize, Ipp8u* pBuffer);

Supported values for mod:

8u_C1IR  16u_C1IR  32f_C1IR

IppStatus ippiWinHammingSep_<mod>(Ipp<datatype>* pSrcDst, int srcDstStep, IppiSize roiSize, Ipp8u* pBuffer);

Supported values for mod:

8u_C1IR  16u_C1IR  32f_C1IR

Include Files
ippi.h

Domain Dependencies
Headers: ippcore.h, ippvm.h, ipps.h
Libraries: ippcore.lib, ippvm.lib, ipps.lib

Parameters

- **pSrc**: Pointer to the source image ROI.
- **srcStep**: Distance, in bytes, between the starting points of consecutive lines in the source image.
- **pDst**: Pointer to the destination image ROI.
- **dstStep**: Distance, in bytes, between the starting points of consecutive lines in the destination image.
- **pSrcDst**: Pointer to the source and destination image ROI for the in-place operation.
- **srcDstStep**: Distance, in bytes, between the starting points of consecutive lines in the source and destination image for the in-place operation.
- **roiSize**: Size of the source and destination ROI, in pixels.
- **pBuffer**: Pointer to the work buffer. To compute the size of the buffer, use the `ippiWinHammingGetBufferSize` or `ippiWinHammingSepGetBufferSize` function.

Description

These functions operate with ROI (see Regions of Interest in Intel IPP).

These functions compute the Hamming (triangle) window samples, multiply pixel values of the source image `pSrc` (`pSrcDst` for in-place flavors) with these samples, and store results in the destination image `pDst` (`pSrcDst` for in-place flavors).

The Hamming window function for one-dimensional case with M elements is defined as follows:

\[ w_{\text{hamming}}(n) = 0.54 - 0.46 \cos \left( \frac{2 \pi n}{M-1} \right) \]

The `ippiWinHammingSep` flavor applies the window function successively to the rows and then to the columns of the image.

Return Values

- **ippStsNoErr**: Indicates no error. Any other value indicates an error or a warning.
- **ippStsNullPtrErr**: Indicates an error when `pSrc` or `pDst` pointer is NULL.
- **ippStsStepErr**: Indicates an error when `srcStep` or `dstStep` value is zero or negative.
- **ippStsSizeErr**: Indicates an error when width or height of images is less than, or equal to zero.

See Also

WinHammingGetBufferSize, WinHammingSepGetBufferSize Compute the size of the work buffer for the `ippiWinHamming` or `ippiWinHammingSep` function.

Regions of Interest in Intel IPP
Discrete Cosine Transforms

Discrete Cosine Transform (DCT) of a real 2D image yields output results that are also real, which eliminates the need to use packed format for storing the transformed data. However, forward and inverse DCT functions ippiDCTFwd and ippiDCTInv need different context data structures to be initialized and filled in prior to their use. Consequently, the required workspace buffer size is different for these functions. In case of using an external buffer, its size must be determined by previously calling the respective support function. DCT functions that use context structures implement the modified computation algorithm proposed in [Rao90].

The DCT functions ippiDCT8x8Fwd and ippiDCT8x8Inv working on a fixed 8x8 image buffer need no context data or external workspace buffers. Functions ippiDCT8x8Inv meet IEEE-1180 standard requirements (see [IEEE]).

Intel IPP Discrete Cosine Transform functions working on a fixed 8x8 image buffer use Feig and Winograd algorithm ([Feig92]) modified for taking advantage of SIMD instructions. For details on algorithms used in DCT transforms and for more references, see [AP922].

Optimization Notice

Intel's compilers may or may not optimize to the same degree for non-Intel microprocessors for optimizations that are not unique to Intel microprocessors. These optimizations include SSE2, SSE3, and SSSE3 instruction sets and other optimizations. Intel does not guarantee the availability, functionality, or effectiveness of any optimization on microprocessors not manufactured by Intel. Microprocessor-dependent optimizations in this product are intended for use with Intel microprocessors. Certain optimizations not specific to Intel microarchitecture are reserved for Intel microprocessors. Please refer to the applicable product User and Reference Guides for more information regarding the specific instruction sets covered by this notice.

Notice revision #20110804

DCTFwdGetSize, DCTInvGetSize

Compute the size of the DCT context structure and the size of the required work buffers.

Syntax

IppStatus ippiDCTFwdGetSize_32f (IppiSize roiSize, int* pSizeSpec, int* pSizeInit, int* pSizeBuf);
IppStatus ippiDCTInvGetSize_32f (IppiSize roiSize, int* pSizeSpec, int* pSizeInit, int* pSizeBuf);

Include Files

ippi.h

Domain Dependencies

Headers: ippcore.h, ippvm.h, ipps.h
Libraries: ippcore.lib, ippvm.lib, ipps.lib

Parameters

roiSize Size of the source and destination ROI, in pixels.
pSizeSpec Pointer to the size of the DCT context structure.
pSizeInit Pointer to the size of the buffer for the DCT initialization function.
pSizeBuf

Pointer to the size of the DCT external work buffer.

**Description**

These functions compute the following:

1. Size of the DCT context structure. The result, in bytes, is stored in the pSizeSpec parameter.
2. Size of the work buffer for the ippiDCTFwdInit and ippiDCTInvInit functions. The result, in bytes, is stored in the pSizeInit parameter.
3. Size of the work buffer for the ippiDCTFwd and ippiDCTInv functions. The result, in bytes, is stored in the pSizeBuf parameter.

**Return Values**

- ippStsNoErr: Indicates no error.
- ippStsNullPtrErr: Indicates an error when one of the specified pointers is NULL.
- ippStsSizeErr: Indicates an error when roiSize has a field with a zero or negative value.

**See Also**

DCTFwdInit, DCTInvInit Initialize the context structure for the forward or inverse DCT operation.
DCTFwd Applies a forward discrete cosine transform to an image.
DCTInv Applies an inverse discrete cosine transform to an image.

**DCTFwdInit, DCTInvInit**

*Initialize the context structure for the forward or inverse DCT operation.*

**Syntax**

IppStatus ippiDCTFwdInit_32f (IppiDCTFwdSpec_32f* pDCTSpec, IppiSize roiSize, Ipp8u* pMemInit);
IppStatus ippiDCTInvInit_32f (IppiDCTInvSpec_32f* pDCTSpec, IppiSize roiSize, Ipp8u* pMemInit);

**Include Files**

ippi.h

**Domain Dependencies**

- **Headers:** ippcore.h, ippvm.h, ipps.h
- **Libraries:** ippcore.lib, ippvm.lib, ipps.lib

**Parameters**

- **pDCTSpec**: Pointer to the forward or inverse DCT context structure for initialization.
- **roiSize**: Size of the source and destination ROI, in pixels.
- **pMemInit**: Pointer to the temporary work buffer.
Description
These functions initialize the pDCTSpec context structure to apply the forward or inverse DCT to two-dimensional image data. TheippiDCTFwd andippiDCTInv functions use the pointer to the initialized DCT context structure as an argument to compute the forward or inverse DCT for points in the ROI of size roiSize.

Return Values
ippStsNoErr Indicates no error.
ippStsNullPtrErr Indicates an error when one of the specified pointers is NULL.
ippStsSizeErr Indicates an error when roiSize has a field with a zero or negative value.

See Also
DCTFwd Applies a forward discrete cosine transform to an image.
DCTInv Applies an inverse discrete cosine transform to an image.

DCTFwd
Applies a forward discrete cosine transform to an image.

Syntax
IppStatus ippiDCTFwd_<mod> (const Ipp32f* pSrc, int srcStep, Ipp32f* pDst, int dstStep, const IppiDCTFwdSpec_32f* pDCTSpec, Ipp8u* pBuffer);

Supported values for mod:

32f_C1R
32f_C3R
32f_C4R
32f_AC4R

Include Files
ippi.h

Domain Dependencies
Headers: ippcore.h, ippvm.h, ipps.h
Libraries: ippcore.lib, ippvm.lib, ipps.lib

Parameters
pSrc Pointer to the source image ROI.
srcStep Distance, in bytes, between the starting points of consecutive lines in the source image.
pDst Pointer to the destination image ROI.
dstStep Distance, in bytes, between the starting points of consecutive lines in the destination image.
pDCTSpec  Pointer to the previously initialized forward DCT context structure.

pBuffer  Pointer to the external work buffer.

Description
This function operates with ROI (see Regions of Interest in Intel IPP) of the size that is specified by the ippiDFTInit function.

This function performs a forward DCT on each channel of the source image pSrc and writes the result into the corresponding channel of the destination image buffer pDst. Note that the function flavor with AC4 descriptor does not process alpha channel. This function uses the previously initialized pDCTSpec context structure to set the mode of calculations and retrieve support data.

You can use this function with the external work buffer pBuffer to avoid memory allocation within the functions. Once the work buffer is allocated, it can be used for all following calls to the functions computing DCT. As internal allocation of memory is too expensive operation and depends on operating system and/or runtime libraries used - the use of an external buffer improves performance significantly, especially for the small size transforms.

Before using the forward DCT functions, you need to compute the size of the required buffers and the external work buffer using the ippiDCTFwdGetSize function.

Return Values

ippStsNoErr  Indicates no error. Any other value indicates an error or a warning.

ippStsNullPtrErr  Indicates an error condition if pSrc, pDst, or pDCTSpec pointer is NULL.

ippStsStepErr  Indicates an error condition if srcStep or dstStep value is zero or negative.

ippStsContextMatchErr  Indicates an error condition if a pointer to an invalid pDCTSpec structure is passed.

ippStsMemAllocErr  Indicates an error condition if memory allocation fails.

Example
The code example below demonstrates how to use the ippiDCTFwdGetSize, ippiDCTFwdInit, and ippiDCTFwd functions.

```c
void DCT_example( void )
{
    Ipp32f Src[8*8] = {
        0.0, 1.0, 2.0, 3.0, 4.0, 5.0, 6.0, 7.0,
        0.0, 1.0, 2.0, 3.0, 4.0, 5.0, 6.0, 7.0,
        0.0, 1.0, 2.0, 3.0, 4.0, 5.0, 6.0, 7.0,
        0.0, 1.0, 2.0, 3.0, 4.0, 5.0, 6.0, 7.0,
        0.0, 1.0, 2.0, 3.0, 4.0, 5.0, 6.0, 7.0,
        0.0, 1.0, 2.0, 3.0, 4.0, 5.0, 6.0, 7.0,
        0.0, 1.0, 2.0, 3.0, 4.0, 5.0, 6.0, 7.0,
        0.0, 1.0, 2.0, 3.0, 4.0, 5.0, 6.0, 7.0
    };
    Ipp32f Dst[8*8];
    IppSize roiSize = {8, 8};
    int srcStep;
    int dstStep;
```
int sizeSpec;
int sizeInit;
int sizeBuffer;

IppiDCTFwdSpec_32f *pMemSpec;
Ipp8u *pMemInit = 0;
Ipp8u *pMemBuffer = 0;

srcStep = dstStep = 8 * sizeof(Ipp32f);

/// get sizes for required buffers
ippiDCTFwdGetSize_32f( roiSize, &sizeSpec, &sizeInit, &sizeBuffer );

/// allocate memory for required buffers
pMemSpec = (IppiDCTFwdSpec_32f*) ippMalloc ( sizeSpec );

if ( sizeInit > 0 )
{
    pMemInit = (Ipp8u*) ippMalloc ( sizeInit );
}

if ( sizeBuffer > 0 )
{
    pMemBuffer = (Ipp8u*) ippMalloc ( sizeBuffer );
}

/// initialize DCT specification structure
ippiDCTFwdInit_32f( pMemSpec, roiSize, pMemInit );

/// free initialization buffer
if ( sizeInit > 0 )
{
    ippFree( pMemInit );
}

/// perform forward DCT
ippiDCTFwd_32f_C1R( Src, srcStep, Dst, dstStep, pMemSpec, pMemBuffer );

/// ...

/// free buffers
if ( sizeBuffer > 0 )
{
    ippFree( pMemBuffer );
}

ippFree( pMemSpec );

Result:

Dst ->  
28.0  -18.2  0.0  -1.9  0.0  -0.57  0.0  -0.14
       0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0
       0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0
       0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0
       0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0
       0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0
       0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0

Image Linear Transforms
DCTInv

Applies an inverse discrete cosine transform to an image.

Syntax

IppStatus ippiDCTInv_<mod> (const Ipp32f* pSrc, int srcStep, Ipp32f* pDst, int dstStep, const IppiDCTInvSpec_32f* pDCTSpec, Ipp8u* pBuffer);

Supported values for mod :

- 32f_C1R
- 32f_C3R
- 32f_C4R
- 32f_AC4R

Include Files

ippi.h

Domain Dependencies

Headers: ippicore.h, ippvm.h, ipps.h
Libraries: ippcore.lib, ippvm.lib, ipps.lib

Parameters

- pSrc : Pointer to the source image ROI.
- srcStep : Distance, in bytes, between the starting points of consecutive lines in the source image.
- pDst : Pointer to the destination image ROI.
- dstStep : Distance, in bytes, between the starting points of consecutive lines in the destination image.
- pDCTSpec : Pointer to the previously initialized inverse DCT context structure.
- pBuffer : Pointer to the external work buffer.

Description

This function operates with ROI (see Regions of Interest in Intel IPP) of the size that is specified by the ippiDCTInvInit function.

This function performs an inverse DCT on each channel of the input image pSrc and writes the result into the corresponding channel of the output image buffer pDst. Note that the function flavor with AC4 descriptor does not process alpha channel. This function uses the previously initialized pDCTSpec context structure to set the mode of calculations and retrieve support data.

The function may be used with the external work buffer pBuffer to avoid memory allocation within the functions. Once the work buffer is allocated, it can be used for all following calls to the functions computing DCT. As internal allocation of memory is too expensive operation and depends on operating system and/or runtime libraries used - the use of an external buffer improves performance significantly, especially for the small size transforms.
Before using the inverse DCT functions, you need to compute the size of the required buffers and the 
external work buffer using the `ippiDCTInvGetSize` function.

**Return Values**

- **ippStsNoErr**
  Indicates no error. Any other value indicates an error or a warning.

- **ippStsNullPtrErr**
  Indicates an error condition if `pSrc`, `pDst`, or `pDCTSpec` pointer is NULL.

- **ippStsStepErr**
  Indicates an error condition if `srcStep` or `dstStep` value is zero or negative.

- **ippStsContextMatchErr**
  Indicates an error condition if a pointer to an invalid `pDCTSpec` structure is passed.

- **ippStsMemAllocErr**
  Indicates an error condition if memory allocation fails.

**DCT8x8Fwd**

*Performs a forward DCT on a 2D buffer of 8x8 size.*

**Syntax**

**Case 1: Not-in-place operation**

```c
IppStatus ippiDCT8x8Fwd_<mod>(const Ipp<datatype>* pSrc, Ipp<datatype>* pDst);
```

Supported values for `mod`:

- 16s_C1
- 32f_C1

**Case 2: Not-in-place operation with ROI**

```c
IppStatus ippiDCT8x8Fwd_<mod>(const Ipp<srcDatatype>* pSrc, int srcStep, Ipp<dstDatatype>* pDst);
```

Supported values for `mod`:

- 16s_C1R
- 8u16s_C1R

**Case 3: In-place operation**

```c
IppStatus ippiDCT8x8Fwd_<mod>(Ipp<datatype>* pSrcDst);
```

Supported values for `mod`:

- 16s_C1I
- 32f_C1I

**Include Files**

`ippi.h`

**Domain Dependencies**

- **Headers:** `ippcore.h`, `ippvm.h`, `ipps.h`
- **Libraries:** `ippcore.lib`, `ippvm.lib`, `ipps.lib`
**Parameters**

- `pSrc`  
  Pointer to the source image buffer.

- `srcStep`  
  Distance in bytes between starts of consecutive lines in the source image buffer for operations with ROI.

- `pDst`  
  Pointer to the destination image buffer.

- `pSrcDst`  
  Pointer to the source and destination image for in-place operations.

**Description**

Some flavors operate with ROI (see Regions of Interest in Intel IPP).

This function computes the forward discrete cosine transform of short integer or floating-point data in a 2D buffer of 8x8 size. No prerequisites are needed to use this transform function.

**Return Values**

- `ippStsNoErr`  
  Indicates no error. Any other value indicates an error or a warning.

- `ippStsNullPtrErr`  
  Indicates an error condition if `pSrc`, `pDst`, or `pSrcDst` pointer is NULL.

- `ippStsStepErr`  
  Indicates an error condition if `srcStep` value is zero or negative.

**Example**

The code example below illustrates the use of `ippiDCT8x8Fwd` function.

```c
IppStatus dct16s( void ) {
    Ipp16s x[64] = {0};
    IppiSize roi = {8,8};
    int i;
    for( i=0; i<8; ++i ) {
        ippiSet_16s_C1R( (Ipp16s)i, x+8*i+i, 8*sizeof(Ipp16s), roi );
        --roi.width;
        --roi.height;
    }
    return ippiDCT8x8Fwd_16s_C1I( x );
}
```
The destination image \( x \) contains:

\[
\begin{bmatrix}
18 & -9 & -2 & -1 & 0 & 0 & 0 & 0 \\
-9 & 7 & 0 & 0 & 0 & 0 & 0 & 0 \\
-2 & 0 & 2 & 0 & 0 & 0 & 0 & 0 \\
-1 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0
\end{bmatrix}
\]

**DCT8x8Inv, DCT8x8Inv_A10**

*Performs an inverse DCT on a 2D buffer of 8x8 size.*

**Syntax**

**Case 1: Not-in-place operation**

\[
\text{IppStatus ippiDCT8x8Inv\_mod}(\text{const Ipp<datatype>* pSrc, Ipp<datatype>* pDst});
\]

Supported values for \( \text{mod} \):

16s\_C1 32f\_C1

\[
\text{IppStatus ippiDCT8x8Inv\_A10\_16s\_C1}(\text{const Ipp16s* pSrc, Ipp16s* pDst});
\]

**Case 2: Not-in-place operation with ROI**

\[
\text{IppStatus ippiDCT8x8Inv\_mod}(\text{const Ipp<srcDatatype>* pSrc, Ipp<dstDatatype>* pDst, int dstStep});
\]

Supported values for \( \text{mod} \):

16s\_C1R 16s8u\_C1R

**Case 3: In-place operation**

\[
\text{IppStatus ippiDCT8x8Inv\_mod}(\text{Ipp<datatype>* pSrcDst});
\]

Supported values for \( \text{mod} \):

16s\_C1I 32f\_C1I

\[
\text{IppStatus ippiDCT8x8Inv\_A10\_16s\_C1I}(\text{Ipp16s* pSrcDst});
\]

**Include Files**

ippi.h

**Domain Dependencies**

**Headers:** ippcore.h, ippvm.h, ipps.h

**Libraries:** ippcore.lib, ippvm.lib, ipps.lib
Parameters

- `pSrc`: Pointer to the source image.
- `pDst`: Pointer to the destination buffer.
- `dstStep`: Distance in bytes between starts of consecutive lines in the destination buffer for operations with ROI.
- `pSrcDst`: Pointer to the source and destination image for in-place operations.

Description

Some flavors operate with ROI (see Regions of Interest in Intel IPP).

This function computes the inverse discrete cosine transform of data in a 2D buffer of 8x8 size. No prerequisites are needed to use this transform function.

Caution

Source data for 16s functions must be the result of the forward discrete cosine transform of data from the range \([-512, 511]\) for flavors with A10 modifier (ippiDCT8x8Inv_A10), and from the range \([-256, 255]\) for flavors without A10 modifier (ippiDCT8x8Inv); they cannot be arbitrary data from the range \([-32768, 32767]\).

Return Values

- `ippStsNoErr`: Indicates no error. Any other value indicates an error or a warning.
- `ippStsNullPtrErr`: Indicates an error condition if `pSrc`, `pDst`, or `pSrcDst` pointer is NULL.
- `ippStsStepErr`: Indicates an error condition if `dstStep` value is zero or negative.

DCT8x8FwdLS

Performs a forward DCT on a 2D buffer of 8x8 size with prior data conversion and level shift.

Syntax

```c
IppStatus ippiDCT8x8FwdLS_8u16s_C1R(const Ipp8u* pSrc, int srcStep, Ipp16s* pDst, Ipp16s addVal);
```

Include Files

- ippi.h

Domain Dependencies

Headers: ippcore.h, ippvm.h, ipps.h
Libraries: ippcore.lib, ippvm.lib, ipps.lib

Parameters

- `pSrc`: Pointer to the source image buffer.
pDst
Pointer to the destination buffer.

srcStep
Distance in bytes between starts of consecutive lines in the source image buffer.

addVal
The level shift value.

Description
This function operates with ROI (see Regions of Interest in Intel IPP) that is a 2D buffer of 8x8 size in this case, thus there is no need to specify its size.

This function first converts data in the buffer pSrc from unsigned Ipp8u type to the signed Ipp16s type and then performs level shift operation by adding the constant value addVal to each sample. After that, the function performs the forward discrete cosine transform of the modified data. The result is stored in pDst.

Return Values
ippStsNoErr Indicates no error. Any other value indicates an error or a warning.
ippStsNullPtrErr Indicates an error condition if pSrc or pDst pointer is NULL.
ippStsStepErr Indicates an error condition if srcStep value is zero or negative.

DCT8x8InvLSClip
Performs an inverse DCT on a 2D buffer of 8x8 size with further data conversion and level shift.

Syntax
IppStatus ippiDCT8x8InvLSClip_16s8u_C1R(const Ipp16s* pSrc, Ipp8u* pDst, int dstStep, Ipp16s addVal, Ipp8u clipDown, Ipp8u clipUp);

Include Files
ippi.h

Domain Dependencies
Headers: ippcore.h, ippvm.h, ipps.h
Libraries: ippcore.lib, ippvm.lib, ipps.lib

Parameters
pSrc Pointer to the source image buffer.
PDst Pointer to the destination image buffer.
dstStep Distance in bytes between starts of consecutive lines in the destination image buffer.
addVal The level shift value.
clipDown The lower bound for the range of output values.
clipUp The upper bound for the range of output values.
Description
This function operates with ROI (see Regions of Interest in Intel IPP) that is a 2D buffer of 8x8 size in this case, thus there is no need to specify its size.

This function performs the inverse discrete cosine transform of the buffer pSrc. After completing the DCT, this function performs level shift operation by adding the constant value addVal to each sample. Finally, the function converts data from the signed Ipp16s type to the unsigned Ipp8u type. The output data are clipped to the range [clipDown..clipUp]. The result is stored in the destination buffer pDst.

Caution
Source data for 16s flavors must be the result of the forward discrete cosine transform of data from the range [-256, 255], they cannot be arbitrary data from the range [-32768, 32767].

Return Values
- ippStsNoErr: Indicates no error. Any other value indicates an error or a warning.
- ippStsNullPtrErr: Indicates an error condition if pSrc or pDst pointer is NULL.
- ippStsStepErr: Indicates an error condition if dstStep value is zero or negative.

DCT8x8Inv_2x2, DCT8x8Inv_4x4
Perform an inverse DCT on a top left quadrant of size 2x2 or 4x4 in the 2D buffer of size 8x8.

Syntax
IppStatus ippiDCT8x8Inv_2x2_16s_C1(const Ipp16s* pSrc, Ipp16s* pDst);
IppStatus ippiDCT8x8Inv_4x4_16s_C1(const Ipp16s* pSrc, Ipp16s* pDst);
IppStatus ippiDCT8x8Inv_2x2_16s_C1I(Ipp16s* pSrcDst);
IppStatus ippiDCT8x8Inv_4x4_16s_C1I(Ipp16s* pSrcDst);

Include Files
ippi.h

Domain Dependencies
Headers: ippcore.h, ippvm.h, ipps.h
Libraries: ippcore.lib, ippvm.lib, ipps.lib

Parameters
- pSrc: Pointer to the source image.
- pDst: Pointer to the destination buffer.
- pSrcDst: Pointer to the source and destination buffer for in-place operations.
Description
These functions compute the inverse discrete cosine transform of non-zero elements in the top left quadrant of size 2x2 or 4x4 in the 2D buffer of 8x8 size. No prerequisites are needed to use this transform function.

Caution
Source data for 16s flavors must be the result of the forward discrete cosine transform of data from the range [-256, 255], they cannot be arbitrary data from the range [-32768, 32767].

Return Values
ippStsNoErr Indicates no error. Any other value indicates an error.
ippStsNullPtrErr Indicates an error condition if one of the specified pointers is NULL.

DCT8x8To2x2Inv, DCT8x8To4x4Inv
Perform an inverse DCT on a 2D buffer of 8x8 size with further downsampling to 2x2 or 4x4 size.

Syntax
IppStatusippiDCT8x8To2x2Inv_16s_C1(const Ipp16s*pSrc, Ipp16s*pDst);
IppStatusippiDCT8x8To4x4Inv_16s_C1(const Ipp16s*pSrc, Ipp16s*pDst);
IppStatusippiDCT8x8To2x2Inv_16s_C1I(Ipp16s*pSrcDst);
IppStatusippiDCT8x8To4x4Inv_16s_C1I(Ipp16s*pSrcDst);

Include Files
ippi.h

Domain Dependencies
Headers: ippcore.h, ippvm.h, ipps.h
Libraries: ippcore.lib, ippvm.lib, ipps.lib

Parameters
pSrc Pointer to the source image.
pDst Pointer to the destination buffer.
pSrcDst Pointer to the source and destination buffer for in-place operations.

Description
These functions compute the inverse discrete cosine transform of the 2D buffer pSrc of 8x8 size. Then the functions perform downsampling of the result by averaging to the destination buffer pdst of size 2x2 or 4x4. In-place flavors of the functions perform operations on the source and destination buffer pSrcDst.
Caution
Source data for 16s flavors must be the result of the forward discrete cosine transform of
data from the range [-256, 255], they cannot be arbitrary data from the range [-32768,
32767].

Return Values
ippStsNoErr
Indicates no error. Any other value indicates an error.
ippStsNullPtrErr
Indicates an error condition if one of the specified pointers is
NULL.
This chapter describes the Intel® IPP image processing functions that can be used to compute the following statistical parameters of an image:

- sum, integrals, mean and standard deviation of pixel values
- intensity histogram of pixel values
- minimum and maximum pixel values
- spatial and central moments of order 0 to 3
- the infinity, $L_1$, and $L_2$ norms of the image pixel values and of the differences between pixel values of two images
- relative error values for the infinity, $L_1$, and $L_2$ norms of differences between pixel values of two images
- universal image quality index
- proximity measures of an image and a template (another image).

**Sum**

*Computes the sum of image pixel values.*

**Syntax**

**Case 1: Operation on one-channel integer data**

```c
IppStatus ippiSum_<mod>(const Ipp<datatype>* pSrc, int srcStep, IppiSize roiSize, Ipp64f* pSum);
```

Supported values for `mod`:

- 8u_C1R
- 16u_C1R
- 16s_C1R

**Case 2: Operation on one-channel floating-point data**

```c
IppStatus ippiSum_32f_C1R(const Ipp32f* pSrc, int srcStep, IppiSize roiSize, Ipp64f* pSum, IppHintAlgorithm hint);
```

**Case 3: Operation on multi-channel integer data**

```c
IppStatus ippiSum_<mod>(const Ipp<datatype>* pSrc, int srcStep, IppiSize roiSize, Ipp64f sum[3]);
```

Supported values for `mod`:

- 8u_C3R
- 16u_C3R
- 16s_C3R

```c
IppStatus ippiSum_<mod>(const Ipp<datatype>* pSrc, int srcStep, IppiSize roiSize, Ipp64f sum[4]);
```

Supported values for `mod`:

- 8u_C4R
- 16u_C4R
- 16s_C4R
Case 4: Operation on multi-channel floating-point data

IppStatus ippiSum_32f_C3R(const Ipp32f* pSrc, int srcStep, IppSize roiSize, Ipp64f sum[3], IppHintAlgorithm hint);
IppStatus ippiSum_32f_C4R(const Ipp32f* pSrc, int srcStep, IppSize roiSize, Ipp64f sum[4], IppHintAlgorithm hint);

Include Files
ippi.h

Domain Dependencies
Headers: ippcore.h, ippvm.h, ipps.h
Libraries: ippcore.lib, ippvm.lib, ipps.lib

Parameters

- pSrc: Pointer to the source image ROI.
- srcStep: Distance in bytes between starts of consecutive lines in the source image.
- roiSize: Size of the source ROI in pixels.
- pSum: Pointer to the computed sum of pixel values.
- sum: Array containing computed sums of channel values of pixels in the source buffer.
- hint: Option to select the algorithmic implementation of the function.

Description
This function operates with ROI (see Regions of Interest in Intel IPP). This function computes the sum of pixel values pSum for the source image pSrc using algorithm indicated by the hint argument (see Table "Hint Arguments for Image Moment Functions"). In case of a multi-channel image, the sum is computed over each channel and stored in the array sum.

Return Values

- ippStsNoErr: Indicates no error. Any other value indicates an error or a warning.
- ippStsNullPtrErr: Indicates an error condition if the pSrc or pSum pointer is NULL.
- ippStsSizeErr: Indicates an error condition if roiSize has a field with zero or negative value.
Example

The code example below demonstrates the use of `ippiSum` function:

```c
IppStatus sum( void ) {
    Ipp64f sum;
    Ipp8u x[5*4];
    IpplSize roi = {5,4};
    ippiSet_8u_C1R( 1, x, 5, roi );
    return ippiSum_8u_C1R( x, 5, roi, &sum);
}
```

Integral

*Transforms an image to the integral representation.*

**Syntax**

```c
IppStatus ippiIntegral_8u32s_C1R(const Ipp8u* pSrc, int srcStep, Ipp32s* pDst, int dstStep, IpplSize srcRoiSize, Ipp32s val);
IppStatus ippiIntegral_8u32f_C1R(const Ipp8u* pSrc, int srcStep, Ipp32f* pDst, int dstStep, IpplSize srcRoiSize, Ipp32f val);
IppStatus ippiIntegral_32f_C1R(const Ipp32f* pSrc, int srcStep, Ipp32f* pDst, int dstStep, IpplSize srcRoiSize);
```

**Include Files**

ippcv.h

**Domain Dependencies**

**Headers:** ippcore.h, ippvm.h, ipps.h,ippi.h

**Libraries:** ippcore.lib, ippvm.lib, ipps.lib,ippi.lib

**Parameters**

- `pSrc`
  - Pointer to the source image ROI.
- `srcStep`
  - Distance in bytes between starts of consecutive lines in the source image.
- `pDst`
  - Pointer to the ROI in the destination integral image.
- `dstStep`
  - Distance in bytes between starts of consecutive lines in the destination image.
- `srcRoiSize`
  - Size of source and destination image ROI in pixels.
- `val`
  - The value to add to `pDst` image pixels.
Description

This function operates with ROI (see Regions of Interest in Intel IPP). This function transforms a source image \( pSrc \) to the integral image \( pDst \). Pixel values of the destination image \( pDst \) are computed using pixel values of the source image \( pSrc \) and the specified value \( val \) in accordance with the following formula:

\[
pDst[i, j] = val + \sum_{k \leq i} \sum_{l \leq j} pSrc[k, l]
\]

where \( i,j \) are coordinates of the destination image pixels (see Figure 11-1) varying in the range \( i = 1, ..., srcRoiSize.height, j = 0, ..., srcRoiSize.width \). Pixel values of zero row and column of \( pDst \) \((i=0)\) is set to \( val \).

For the \( ippiIntegral_32f_C1 \) function flavor the value of \( val \) is considered to be equal to zero.

The size of the destination images is \((srcRoiSize.width + 1) \times (srcRoiSize.height + 1)\).

Figure “Operation of the Integral and TiltedIntegral functions” shows what pixels (red circles) of the source image are used in computation new pixel values in the \( i, j \) coordinates.

For large images the result of summation can exceed the upper bound of the output data type. Table “Maximum Image Size for Integral Functions” lists the maximum image size for different function flavors and values.

Operation of the Integral and TiltedIntegral functions

<table>
<thead>
<tr>
<th>Maximum Image Size for Integral Functions</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Function Flavor</strong></td>
</tr>
<tr>
<td>----------------------</td>
</tr>
<tr>
<td>( ippiIntegral_8u32a_C1R )</td>
</tr>
<tr>
<td>( ippiIntegral_8u32a_C1R )</td>
</tr>
<tr>
<td>( ippiIntegral_8u32f_C1R )</td>
</tr>
<tr>
<td>( ippiIntegral_8u32f_C1R )</td>
</tr>
</tbody>
</table>

**Return Values**

- \( ippStsNoErr \): Indicates no error.
- \( ippStsNullPtrErr \): Indicates an error if \( pSrc \) or \( pDst \) is \( NULL \).
- \( ippStsSizeErr \): Indicates an error condition if \( srcRoiSize \) has a field with zero or negative value.
Indicates an error condition if `srcStep` is less than `srcRoiSize.width * <pixelSize>`, or `dstStep` is less than `(srcRoiSize.width+1) * <pixelSize>`.

Indicates an error condition if `srcStep` or `dstStep` is not divisible by `<pixelSize>`.

**Example**

The code example below demonstrates how to use the `ippiIntegral_8u32s_C1R` function:

```c
void func_integral_8u32s_C1R()
{
    Ipp8u pSrc[5*4];
    Ipp32s pDst[6*5];
    IppiSize ROI = {5,4};
   ippiSet_8u_C1R(1,pSrc,5,ROI);
    Ipp32s val = 1;
   ippiIntegral_8u32s_C1R(pSrc, 5, pDst, 6*sizeof(Ipp32s), ROI, val);
}
```

Result:

```
pSrc -> 1 1 1 1 1          pDst -> 1 1 1 1 1 1
    1 1 1 1 1
    1 1 1 1 1
    1 1 1 1 1
    1 1 1 1 1
    1 1 1 1 1

1 2 3 4 5 6
1 3 5 7 9 11
1 4 7 10 13 16
1 5 9 13 17 21
```

The code example below demonstrates how to use the `ippiIntegral_32f_C1R` function:

```c
void func_integral_32f_C1R()
{
    Ipp32f pSrc[5*4];
    Ipp32f pDst[6*5];
    IppiSize ROI = {5,4};
   ippiSet_32f_C1R(1,pSrc,5*sizeof(Ipp32f),ROI);
   ippiIntegral_32f_C1R(pSrc, 5*sizeof(Ipp32f), pDst, 6*sizeof(Ipp32f), ROI);
}
```

Result:

```
pSrc -> 1.0 1.0 1.0 1.0 1.0        pDst -> 0.0 0.0 0.0 0.0 0.0 0.0
    1.0 1.0 1.0 1.0 1.0
    1.0 1.0 1.0 1.0 1.0
    1.0 1.0 1.0 1.0 1.0

0.0 1.0 2.0 3.0 4.0 5.0
0.0 2.0 4.0 6.0 8.0 10.0
0.0 3.0 6.0 9.0 12.0 15.0
0.0 4.0 8.0 12.0 16.0 20.0
```

**SqrIntegral**

Transforms an image to integral and integral of pixel squares representations.

**Syntax**

```c
IppStatus ippiSqrIntegral_8u32s64f_C1R(const Ipp8u* pSrc, int srcStep, Ipp32s* pDst, int dstStep, Ipp64f* pSqr, int sqrStep, IppiSize roiSize, Ipp32s val, Ipp64f valSqr);
IppStatus ippiSqrIntegral_8u32f64f_C1R(const Ipp8u* pSrc, int srcStep, Ipp32f* pDst, int dstStep, Ipp64f* pSqr, int sqrStep, IppiSize roiSize, Ipp32f val, Ipp64f valSqr);
```
IppStatus ippiSqrIntegral_8u32s_C1R(const Ipp8u* pSrc, int srcStep, Ipp32s* pDst, int dstStep, Ipp32s* pSqr, int sqrStep, IppSize roiSize, Ipp32s val, Ipp32s valSqr);

Include Files
ippcv.h

Domain Dependencies
Headers: ippcore.h, ippvm.h, ipps.h,ippi.h
Libraries: ippcore.lib, ippvm.lib, ipps.lib,ippi.lib

Parameters

pSrc
Pointer to the source image ROI.
srcStep
Distance in bytes between starts of consecutive lines in the source image.
pDst
Pointer to the ROI in the destination integral image.
dstStep
Distance in bytes between starts of consecutive lines in the destination image.
pSqr
Pointer to the ROI of the destination integral image of pixel squares.
sqrStep
Distance in bytes between starts of consecutive lines in the destination integral image of pixel squares.
roiSize
Size of source image ROI in pixels.
val
The value to add to pDst image pixels.
valSqr
The value to add to pSqr image pixels

Description
This function operates with ROI (see Regions of Interest in Intel IPP). This function builds two destination images: integral image pDst and integral image of pixel squares pSqr. Pixel values of pDst are computed using pixel values of the source image pSrc and the specified value val in accordance with the following formula:

\[ p_{Dst}[i, j] = val + \sum_{k<i} \sum_{l<j} p_{Src}[k, l] \]

Pixel values of pSqr are computed using pixel values of the source image pSrc and the specified value valSqr in accordance with the following formula:

\[ p_{Sqr}[i, j] = valSqr + \sum_{k<i} \sum_{l<j} p_{Src}[k, l]^2 \]

where \(i,j\) are coordinates of the destination image pixels (see Figure "Operation of the Integral and TiltedIntegral functions") varying in the range \(i = 1, ..., roiSize.height, j = 0, ..., roiSize.width\). Pixel values of zero row and column are set to \(val\) for pDst, and to \(valSqr\) for pSqr. The size of both destination images is \((roiSize.width + 1) \times (roiSize.height + 1)\).

Figure "Operation of the Integral and TiltedIntegral functions" shows what pixels (red circles) of the source image are used in computation new pixel values in the \(i,j\) coordinates.
Return Values

ippStsNoEr Indicates no error.
ippStsNullPtrErr Indicates an error if one of the specified pointers is NULL.
ippStsSizeErr Indicates an error condition if roiSize has a field with zero or negative value.
ippStsStepErr Indicates an error condition if srcStep is less than roiSize.width * <pixelSize>, or dstStep or sqrStep is less than (roiSize.width+1) * <pixelSize>.
ippStsNotEvenStepErr Indicates an error condition if dstStep is not divisible by 4, or sqrStep is not divisible by 8.

TiltedIntegral

Transforms an image to the tilted integral representation.

Syntax

IppStatus ippiTiltedIntegral_8u32f_C1R(const Ipp8u* pSrc, int srcStep, Ipp32f* pDst, int dstStep, IppiSize roiSize, Ipp32f val);
IppStatus ippiTiltedIntegral_8u32s_C1R(const Ipp8u* pSrc, int srcStep, Ipp32s* pDst, int dstStep, IppiSize roiSize, Ipp32s val);

Include Files

ippcv.h

Domain Dependencies

Headers: ippcore.h, ippvm.h, ipps.h,ippi.h
Libraries: ippcore.lib, ippvm.lib, ipps.lib,ippi.lib

Parameters

pSrc Pointer to the source image ROI.
srcStep Distance in bytes between starts of consecutive lines in the source image.
pDst Pointer to the ROI in the destination integral image.
dstStep Distance in bytes between starts of consecutive lines in the destination image.
roiSize Size of source image ROI in pixels.
val The value to add to pDst image pixels

Description

This function operates with ROI (see Regions of Interest in Intel IPP). This function transforms a source image pSrc to the tilted integral image pDst. Pixel values of the destination image pDst are computed using pixel values of the source image pSrc and the specified value val in accordance with the following formula:
where \( i, j \) are coordinates of the destination image pixels (see Figure "Operation of the Integral and TiltedIntegral functions") varying in the range \( i = 2, \ldots, roiSize.height + 1, j = 0, \ldots, roiSize.width + 1 \). Pixel values of rows 0 and 1 of the destination image \( pDst(i=0) \) is set to \( val \).

The size of the destination images is \((roiSize.width + 2) \times (roiSize.height + 2)\).

Figure "Operation of the Integral and TiltedIntegral functions" shows what pixels (red circles) of the source image are used in computation new pixel values in the \( i, j \) coordinates.

### Return Values

- **ippStsNoErr**
  Indicates no error.

- **ippStsNullPtrErr**
  Indicates an error if \( pSrc \) or \( pDst \) is NULL.

- **ippStsSizeErr**
  Indicates an error condition if \( roiSize \) has a field with zero or negative value.

- **ippStsStepErr**
  Indicates an error condition if \( srcStep \) is less than \( roiSize.width \times \langle \text{pixelSize} \rangle \), or \( dstStep \) is less than \( (roiSize.width+2) \times \langle \text{pixelSize} \rangle \).

- **ippStsNotEvenStepErr**
  Indicates an error condition if one \( dstStep \) is not divisible by 4.

### TiltedSqrIntegral

**Transforms an image to tilted integral and tilted integral of pixel squares representations.**

**Syntax**

```c
IppStatus ippiTiltedSqrIntegral_8u32s_C1R(const Ipp8u* pSrc, int srcStep, Ipp32s* pDst, int dstStep, Ipp32s* pSqr, int sqrStep, IppiSize roiSize, Ipp32s val, Ipp32s valSqr);
IppStatus ippiTiltedSqrIntegral_8u32s64f_C1R(const Ipp8u* pSrc, int srcStep, Ipp32s* pSrc, int srcStep, Ipp64f* pSqr, int sqrStep, IppiSize roiSize, Ipp32s val, Ipp64f valSqr);
IppStatus ippiTiltedSqrIntegral_8u32f64f_C1R(const Ipp8u* pSrc, int srcStep, Ipp32f* pDst, int dstStep, Ipp64f* pSqr, int sqrStep, IppiSize roiSize, Ipp32f val, Ipp64f valSqr);
```

**Include Files**

ippcv.h

**Domain Dependencies**

- Headers: ippcore.h, ippvm.h, ipps.h,ippi.h
- Libraries: ippcore.lib, ippvm.lib, ipps.lib,ippi.lib

**Parameters**

- \( pSrc \)
  Pointer to the source image ROI.
Description
This function operates with ROI (see Regions of Interest in Intel IPP). This function builds two destination images: tilted integral image \( pDst \) and tilted integral image of pixel squares \( pSqr \).

Pixel values of \( pDst \) are computed using pixel values of the source image \( pSrc \) and the specified value \( \text{val} \) in accordance with the following formula:

\[
pDst[i, j] = \text{val} + \sum_{k=-1}^{1} \sum_{j=-1}^{1} pSrc[k, l]
\]

Pixel values of \( pSqr \) are computed using pixel values of the source image \( pSrc \) and the specified value \( \text{valSqr} \) in accordance with the following formula:

\[
pSqr[i, j] = \text{valSqr} + \sum_{k=-1}^{1} \sum_{j=-1}^{1} pSrc[k, l]^2
\]

where \( i, j \) are coordinates of the destination image pixels (see Figure “Operation of the Integral and TiltedIntegral functions”) varying in the range \( i = 2, \ldots, \text{roiSize.height} \), \( j = 0, \ldots, \text{roiSize.width} \). Pixel values of zero and first rows (\( i=0,1 \)) are set to \( \text{val} \) for \( pDst \), and to \( \text{valSqr} \) for \( pSqr \). The size of both destination images is \( (\text{roiSize.width}+2) \times (\text{roiSize.height}+2) \).

Figure “Operation of the Integral and TiltedIntegral functions” shows what pixels (red circles) of the source image are used in computation new pixel values in the \( i, j \) coordinates.

Return Values
- ippStsNoErr: Indicates no error.
- ippStsNullPtrErr: Indicates an error if one of the specified pointers is NULL.
- ippStsSizeErr: Indicates an error condition if \( \text{roiSize} \) has a field with zero or negative value.
- ippStsStepErr: Indicates an error condition if \( \text{srcStep} \) is less than \( \text{roiSize.width} \times <\text{pixelSize}> \), or \( \text{dstStep} \) or \( \text{sqrStep} \) is less than \( (\text{roiSize.width}+2) \times <\text{pixelSize}> \).
- ippStsNotEvenStepErr: Indicates an error condition if \( \text{dstStep} \) is not divisible by 4, or \( \text{sqrStep} \) is not divisible by 8.
Mean

Computes the mean of image pixel values.

Syntax

Case 1: Operation on one-channel integer data

IppStatus ippiMean_<mod>(const Ipp<datatype>* pSrc, int srcStep, IppSize roiSize, Ipp64f* pMean);

Supported values for mod:

8u_C1R     16u_C1R     16s_C1R

Case 2: Operation on one-channel floating-point data

IppStatus ippiMean_32f_C1R(const Ipp32f* pSrc, int srcStep, IppSize roiSize, Ipp64f* pMean, IppHintAlgorithm hint);

Case 3: Masked operation on one-channel data

IppStatus ippiMean_<mod>(const Ipp<datatype>* pSrc, int srcStep, const Ipp8u* pMask, int maskStep, IppSize roiSize, Ipp64f* pMean);

Supported values for mod:

8u_C1MR    16u_C1MR    32f_C1MR

Case 4: Operation on multi-channel integer data

IppStatus ippiMean_<mod>(const Ipp<datatype>* pSrc, int srcStep, IppSize roiSize, Ipp64f mean[3]);

Supported values for mod:

8u_C3R     16u_C3R     16s_C3R

IppStatus ippiMean_<mod>(const Ipp<datatype>* pSrc, int srcStep, IppSize roiSize, Ipp64f mean[4]);

Supported values for mod:

8u_C4R     16u_C4R     16s_C4R

Case 5: Operation on multi-channel floating-point data

IppStatus ippiMean_<mod>(const Ipp32f* pSrc, int srcStep, IppSize roiSize, Ipp64f mean[3], IppHintAlgorithm hint);

Supported values for mod:

32f_C3R

IppStatus ippiMean_32f_C4R(const Ipp32f* pSrc, int srcStep, IppSize roiSize, Ipp64f mean[4], IppHintAlgorithm hint);
Case 6: Masked operation on multi-channel data

IppStatus ippiMean_<mod>(const Ipp<datatype>* pSrc, int srcStep, const Ipp8u* pMask, int maskStep, IppiSize roiSize, int coi, Ipp64f* pMean);

Supported values for mod:

8u_C3CMR  16u_C3CMR  32f_C3CMR

Include Files
ippi.h
ippi.cv.h

Domain Dependencies
Flavors declared inippi.h:
Headers: ippcore.h, ippvm.h, ipps.h
Libraries: ippcore.lib, ippvm.lib, ipps.lib
Flavors declared inippi.cv.h:
Headers: ippcore.h, ippvm.h, ipps.h, ippi.h
Libraries: ippcore.lib, ippvm.lib, ipps.lib, ippi.lib

Parameters
pSrc
Pointer to the source image ROI.

srcStep
Distance in bytes between starts of consecutive lines in the source image.

pMask
Pointer to the mask image.

maskStep
Distance in bytes between starts of consecutive lines in the mask image.

roiSize
Size of the source ROI in pixels.

coi
Channel of interest (for color images only); can be 1, 2, or 3.

pMean
Pointer to the computed mean of pixel values.

mean
Array containing computed mean values for each channel of a multi-channel image.

hint
Option to select the algorithmic implementation of the function.

Description
The flavors of the functionippiMean that perform masked operations are declared in the ippi.cv.h file. All other function flavors are declared in the ippi.h file. This function operates with ROI (see Regions of Interest in Intel IPP). It computes the mean (average) of pixel values pMean for the source image pSrc. Computation algorithm is specified by the hint argument (see Table "Hint Arguments for Image Moment Functions"). For non-masked operations on a multi-channel image (Case 4, 5), the mean is computed over each channel and stored in the array mean. In the mask multi-channel mode (Case 6), the mean is computed for a single channel of interest specified by coi.
Return Values

ippStsNoErr
Indicates no error. Any other value indicates an error or a warning.

ippStsNullPtrErr
Indicates an error when any of the specified pointers is NULL.

ippStsSizeErr
Indicates an error condition if roiSize has a field with zero or negative value.

ippStsStepErr
Indicates an error condition in mask mode, if srcStep or maskStep is less than roiSize.width * <pixelSize>.

ippStsNotEvenStepErr
Indicates an error condition in mask mode if steps for floating-point images cannot be divided by 4.

ippStsCOIErr
Indicates an error when coi is not 1, 2, or 3.

Example
The code example below shows how to use the ippiMean function.

```c
IppStatus mean( void ) {
    Ipp64f mean;
    Ipp8u x[5*4];
    IppiSize roi = {5,4};
    ippiSet_8u_C1R( 3, x, 5, roi );
    return ippiMean_8u_C1R( x, 5, roi, &mean );
}
```

Mean_StdDev

*Computes the mean and standard deviation of image pixel values.*

Syntax

Case 1: Operation on one-channel data

```c
IppStatus ippiMean_StdDev_<mod>(const Ipp<datatype>* pSrc, int srcStep, IppiSize roiSize, Ipp64f* pMean, Ipp64f* pStdDev);
```

Supported values for mod:

8u_C1R 16u_C1R 32f_C1R

Case 2: Masked operation on one-channel data

```c
IppStatus ippiMean_StdDev_<mod>(const Ipp<datatype>* pSrc, int srcStep, const Ipp8u* pMask, int maskStep, IppiSize roiSize, Ipp64f* pMean, Ipp64f* pStdDev);
```

Supported values for mod:

8u_C1MR 16u_C1MR 32f_C1MR
Case 3: Operation on multi-channel data

IppStatus ippiMean_StdDev_<mod>(const Ipp<datatype>* pSrc, int srcStep, IppSize roiSize, int coi, Ipp64f* pMean, Ipp64f* pStdDev);

Supported values for mod:

  8u_C3CR     16u_C3CR     32f_C3CR

Case 4: Masked operation on multi-channel data

IppStatus ippiMean_StdDev_<mod>(const Ipp<datatype>* pSrc, int srcStep, const Ipp8u* pMask, int maskStep, IppSize roiSize, int coi, Ipp64f* pMean, Ipp64f* pStdDev);

Supported values for mod:

  8u_C3CMR    16u_C3CMR    32f_C3CMR

Include Files

ippcv.h

Domain Dependencies

Headers: ippcore.h, ippvm.h, ipps.h,ippi.h
Libraries: ippcore.lib, ippvm.lib, ipps.lib, ippi.lib

Parameters

- **pSrc**
  Pointer to the source image ROI.

- **srcStep**
  Distance in bytes between starts of consecutive lines in the source image.

- **pMask**
  Pointer to the mask image.

- **maskStep**
  Distance in bytes between starts of consecutive lines in the mask image.

- **roiSize**
  Size of the source ROI in pixels

- **coi**
  Channel of interest (for color images only); can be 1, 2, or 3.

- **pMean**
  Pointer to the computed mean of pixel values.

- **pStdDev**
  Pointer to the computed standard deviation of pixel values in the image.

Description

This function operates with ROI (see Regions of Interest in Intel IPP). This function computes the mean and standard deviation of pixel values in the ROI of the source image pSrc. In the mask mode, the computation is done over pixels selected by nonzero mask values. In the multi-channel mode, the mean is computed for a single channel of interest specified by coi. If any of the parameters pMean or pStdDev is not required, the zero pointer is to be passed to the corresponding parameter.

Return Values

- **ippStsNoErr**
  Indicates no error. Any other value indicates an error or a warning.

- **ippStsNullPtrErr**
  Indicates an error when pSrc or pMask pointer is NULL.
Indicates an error condition if \( \text{roiSize} \) has a field with zero or negative value.

Indicates an error condition if \( \text{srcStep} \) or \( \text{maskStep} \) is less than \( \text{roiSize.width} \times \text{pixelSize} \).

Indicates an error condition if steps for floating-point images cannot be divided by 4.

Indicates an error when \( \text{coi} \) is not 1, 2, or 3.

RectStdDev

*Computes the standard deviation of the integral images.*

**Syntax**

IppStatus ippiRectStdDev_32f_C1R(const Ipp32f* \( pSrc \), int \( srcStep \), const Ipp64f* \( pSqr \), int \( sqrStep \), Ipp32f* \( pDst \), int \( dstStep \), IppiSize \( roiSize \), IppiRect \( rect \));

IppStatus ippiRectStdDev_32s32f_C1R(const Ipp32s* \( pSrc \), int \( srcStep \), const Ipp64f* \( pSqr \), int \( sqrStep \), Ipp32f* \( pDst \), int \( dstStep \), IppiSize \( roiSize \), IppiRect \( rect \));

IppStatus ippiRectStdDev_32s_C1RSfs(const Ipp32s* \( pSrc \), int \( srcStep \), const Ipp32s* \( pSqr \), int \( sqrStep \), Ipp32s* \( pDst \), int \( dstStep \), IppiSize \( roiSize \), IppiRect \( rect \), int \( scaleFactor \));

**Include Files**

ippcv.h

**Domain Dependencies**

Headers: ippcore.h, ippvm.h, ipps.h,ippi.h

Libraries: ippcore.lib, ippvm.lib, ipps.lib,ippi.lib

**Parameters**

\( pSrc \) Pointer to the ROI in the source integral image.

\( srcStep \) Distance in bytes between starts of consecutive lines in the source integral image.

\( pSqr \) Pointer to the ROI in the source integral image of pixel squares.

\( sqrStep \) Distance in bytes between starts of consecutive lines in the source integral image of pixel squares.

\( pDst \) Pointer to the destination image ROI.

\( dstStep \) Distance in bytes between starts of consecutive lines in the destination image.

\( roiSize \) Size of destination image ROI in pixels.

\( rect \) Rectangular window.

\( scaleFactor \) Scale factor (see Integer Result Scaling).
This function operates with ROI (see Regions of Interest in Intel IPP). This function computes the standard deviation for each pixel in the rectangular window rect using the integral image pSrc and integral image of pixel squares pSqr. The computations are performed in accordance with the following formulas:

\[
    pDst[i, j] = \sqrt{\max\left(0, \frac{\text{sumSqr} \cdot \text{numPix} - \text{sum}^2}{\text{numPix}}\right)}
\]

where \(i, j\) are coordinates of the destination image pixels varying in the range \(i = 0, \ldots, \text{roiSize.height} - 1, j = 0, \ldots, \text{roiSize.width} - 1\);

\[
    \text{sum} = pSrc[i + \text{rect.y} + \text{rect.height}, j + \text{rect.x} + \text{rect.width}] - pSrc[i + \text{rect.y}, j + \text{rect.x} + \text{rect.width}] - pSrc[i + \text{rect.y} + \text{rect.height}, j + \text{rect.x}] + pSrc[i + \text{rect.y}, j + \text{rect.x}];
\]

\[
    \text{sumSqr} = pSqr[i + \text{rect.y} + \text{rect.height}, j + \text{rect.x} + \text{rect.width}] - pSqr[i + \text{rect.y}, j + \text{rect.x} + \text{rect.width}] - pSqr[i + \text{rect.y} + \text{rect.height}, j + \text{rect.x}] + pSqr[i + \text{rect.y}, j + \text{rect.x}];
\]

\[
    \text{numPix} = \text{rect.height} \cdot \text{rect.width}.
\]

The minimum size of each source images pSrc and pSqr should be \((\text{roiSize.width} + \text{rect.x} + \text{rect.width}) \times (\text{roiSize.height} + \text{rect.y} + \text{rect.height})\).

The source images pSrc and pSqr can be obtained by using the functions ippiIntegral or ippiSqrIntegral.

Return Values

- ippStsNoErr: Indicates no error.
- ippStsNullPtrErr: Indicates an error if one of the specified pointers is NULL.
- ippStsSizeErr: Indicates an error condition if roiSize has a field with zero or negative value.
- ippStsSizeErr: Indicates an error condition if rect.width or rect.height is less than or equal to zero, or if rect.x or rect.y is less than zero.
- ippStsStepErr: Indicates an error condition if srcStep or sqrStep is less than \((\text{roiSize.width} + \text{rect.x} + \text{rect.width}) \times <\text{pixelSize}>\), or dstStep is less than \(\text{roiSize.width} \times <\text{pixelSize}>\).
- ippStsNotEvenStepErr: Indicates an error condition if sqrStep is not divisible by 8, or one of pSrc and dstStep is not divisible by 4.

**TiltedRectStdDev**

**Computation of the standard deviation of the tilted integral images.**

**Syntax**

IppStatus ippiTiltedRectStdDev_32f_C1R(const Ipp32f* pSrc, int srcStep, const Ipp64f* pSqr, int sqrStep, Ipp32f* pDst, int dstStep, IppiSize roiSize, IppiRect rect);

IppStatus ippiTiltedRectStdDev_32s_C1RSfs(const Ipp32s* pSrc, int srcStep, const Ipp32s* pSqr, int sqrStep, Ipp32s* pDst, int dstStep, IppiSize roiSize, IppiRect rect, int scaleFactor);
IppStatusippiTiltedRectStdDev_32s32f_C1R(const Ipp32s* pSrc, int srcStep, const Ipp64f* pSqr, int sqrStep, Ipp32f* pDst, int dstStep, IppiSize roiSize, IppiRect rect);

Include Files
ippcv.h

Domain Dependencies
Headers: ippcore.h, ippvm.h, ipps.h,ippi.h
Libraries: ippcore.lib, ippvm.lib, ipps.lib,ippi.lib

Parameters

- **pSrc**
  Pointer to the ROI in the source integral image.
- **srcStep**
  Distance in bytes between starts of consecutive lines in the source integral image.
- **pSqr**
  Pointer to the ROI in the source integral image of pixel squares.
- **sqrStep**
  Distance in bytes between starts of consecutive lines in the source integral image of pixel squares.
- **pDst**
  Pointer to the destination image ROI.
- **dstStep**
  Distance in bytes between starts of consecutive lines in the destination image.
- **roiSize**
  Size of destination image ROI in pixels.
- **rect**
  Rectangular window.
- **scaleFactor**
  Scale factor (see Integer Result Scaling).

Description

This function operates with ROI (see Regions of Interest in Intel IPP). This function computes the standard deviation for each pixel in the rectangular window `rect` using the tilted integral image `pSrc` and tilted integral image of pixel squares `pSqr`. The computations are performed in accordance with the following formulas:

\[
\text{pDst}[i, j] = \sqrt{\max\left(0, \frac{\text{sumSqr} \cdot \text{numPix} - \text{sum}^2}{\text{numPix}^2}\right)}
\]

where \(i, j\) are coordinates of the destination image pixels varying in the range \(i = 0, ..., \text{roiSize}.\text{height} - 1; j = 0, ..., \text{roiSize}.\text{width} - 1\);

\[
\text{sum} = \text{pSrc}[i + \text{rect}.x - \text{rect}.y + \text{rect}.height + \text{rect}.width, j + \text{rect}.x + \text{rect}.y - \text{rect}.height + \text{rect}.width] - \text{pSrc}[i + \text{rect}.x - \text{rect}.y + \text{rect}.width, j + \text{rect}.x + \text{rect}.y + \text{rect}.width] - \text{pSrc}[i + \text{rect}.x - \text{rect}.y + \text{rect}.height, j + \text{rect}.x - \text{rect}.y - \text{rect}.height] + \text{pSrc}[i + \text{rect}.x - \text{rect}.y, j + \text{rect}.x + \text{rect}.y];
\]

\[
\text{sumSqr} = \text{pSqr}[i + \text{rect}.x - \text{rect}.y + \text{rect}.height + \text{rect}.width, j + \text{rect}.x + \text{rect}.y - \text{rect}.height + \text{rect}.width] - \text{pSqr}[i + \text{rect}.x - \text{rect}.y + \text{rect}.width, j + \text{rect}.x + \text{rect}.y + \text{rect}.width] - \text{pSqr}[i + \text{rect}.x - \text{rect}.y + \text{rect}.height, j + \text{rect}.x - \text{rect}.y - \text{rect}.height] + \text{pSqr}[i + \text{rect}.x - \text{rect}.y, j + \text{rect}.x + \text{rect}.y];
\]

\[
\text{numPix} = 2 \cdot \text{rect}.height \cdot \text{rect}.width.\]
The minimum size of each source images \( pSrc \) and \( pSqr \) should be \((roiSize.width + rect.height + rect.width - 2) \times (roiSize.height + rect.x + rect.y + rect.height + rect.width - 2)\). The source images \( pSrc \) and \( pSqr \) can be obtained by using the functions ippiTiltedIntegral or ippiTiltedSqrIntegral.

**Return Values**

- **ippStsNoErr**: Indicates no error.
- **ippStsNullPtrErr**: Indicates an error if one of the specified pointers is NULL.
- **ippStsSizeErr**: Indicates an error condition if \( roiSize \) has a field with zero or negative value.
- **ippStsSizeErr**: Indicates an error condition if \( rect.width \) or \( rect.height \) is less than or equal to zero, or if \( rect.x \) or \( rect.y \) is less than zero.
- **ippStsStepErr**: Indicates an error condition if \( srcStep \) or \( sqrStep \) is less than \((roiSize.width+rect.x+rect.width+1) \times <pixelSize>\), or \( dstStep \) is less than \( roiSize.width \times <pixelSize>\).
- **ippStsNotEvenStepErr**: Indicates an error condition if \( sqrStep \) is not divisible by 8, or one of \( pSrc \) and \( dstStep \) is not divisible by 4.

---

**HistogramGetBufferSize**

*Computes the size of the specification structure and work buffer for the ippiHistogram function.*

**Syntax**

```c
IppStatus ippiHistogramGetBufferSize(IppDataType dataType, IppiSize roiSize, const int nLevels[], int numChannels, int uniform, int* pSpecSize, int* pBufferSize);
```

**Include Files**

ippi.h

**Domain Dependencies**

Headers: ippcore.h, ippvm.h, ipps.h

Libraries: ippcore.lib, ippvm.lib, ipps.lib

**Parameters**

- **dataType**: Data type of the source image. Supported values are: ipp8u, ipp16u, ipp16s, and ipp32f.
- **roiSize**: Size of the source image ROI, in pixels.
- **nLevels**: Number of level values. Each channel has a separate number of levels.
- **numChannels**: Number of image channels. Supported values are: 1, 3, and 4.
- **uniform**: Type of levels distribution: 0 - with random step, 1 - with uniform step.
pSpecSize

Pointer to the computed value of the specification structure size, in bytes.

pBufferSize

Pointer to the computed value of the external buffer size.

Description

The ippiHistogramGetBufferSize function computes the size of the histogram specification structure and the size of the external work buffer (in bytes) needed for the Histogram function.

For an example on how to use this function, refer to the example provided with the Histogram function description.

Return Values

ippStsNoErr

Indicates no error. Any other value indicates an error or a warning.

ippStsNullPtrErr

Indicates an error when one of the specified pointers is NULL.

ippStsSizeErr

Indicates an error when roiSize is less than, or equal to zero.

ippStsHistoNofLevelsErr

Indicates an error when the number of levels is less than 2.

ippStsNumChannelsErr

Indicates an error when the numChannels value differs from 1, 3, or 4.

ippStsDataTypeErr

Indicates an error when the dataType value differs from ipp8u, ipp16u, ipp16s, or ipp32f.

See Also

Histogram Computes the intensity histogram of an image.

HistogramGetLevels

* Returns the array with level values stored in the specification structure.*

Syntax

IppStatus ippiHistogramGetLevels(const IppiHistogramSpec* pSpec, Ipp32f* pLevels[]);

Include Files

ippi.h

Domain Dependencies

Headers: ippcore.h, ippvm.h, ipps.h
Libraries: ippcore.lib, ippvm.lib, ipps.lib

Parameters

pSpec

Pointer to the specification structure.

pLevels

Pointer to the array of pointers to the level values vectors for each channel.

Description

The ippiHistogramGetLevels function returns the level values stored in the histogram specification structure.
For an example on how to use this function, refer to the example provided with the `Histogram` function description.

**Return Values**

- `ippStsNoErr` Indicates no error. Any other value indicates an error or a warning.
- `ippStsNullPtrErr` Indicates an error when one of the specified pointers is `NULL`.
- `ippStsBadArgErr` Indicates an error when the `pSpec` object is not initialized.

**See Also**

`Histogram` Computes the intensity histogram of an image.

**ippiHistogramInit, ippiHistogramUniformInit**

*Initializes the specification structure for the `ippiHistogram` function.*

**Syntax**

```c
IppStatus ippiHistogramInit(IppDataType dataType, const Ipp32f* pLevels[], int nLevels[], int numChannels, IppiHistogramSpec* pSpec);
IppStatus ippiHistogramUniformInit(IppDataType dataType, Ipp32f lowerLevel[], Ipp32f upperLevel[], int nLevels[], int numChannels, IppiHistogramSpec* pSpec);
```

**Include Files**

`ippi.h`

**Domain Dependencies**

*Headers: ippcore.h, ippvm.h, ipps.h*

*Libraries: ippcore.lib, ippvm.lib, ipps.lib*

**Parameters**

- `dataType` Data type of the source image. Supported values are: `ipp8u`, `ipp16u`, `ipp16s`, and `ipp32f`.
- `pLevels` Pointer to the array of pointers to the level values vectors for each channel.
- `lowerLevel` Lower levels for uniform histogram, separate for each channel.
- `upperLevel` Upper levels for uniform histogram, separate for each channel.
- `nLevels` Number of level values. Each channel has a separate number of levels.
- `numChannels` Number of image channels. Supported values are: 1, 3, and 4.
- `pSpec` Pointer to the specification structure.

**Description**

The `ippiHistogramInit` function initializes the specification structure for the `Histogram` function.

For an example on how to use these functions, refer to the example provided with the `Histogram` function description.
Return Values

- ippStsNoErr: Indicates no error. Any other value indicates an error or a warning.
- ippStsNullPtrErr: Indicates an error when one of the specified pointers is NULL.
- ippStsSizeErr: Indicates an error when roiSize is less than, or equal to zero.
- ippStsHistoNofLevelsErr: Indicates an error when the number of levels is less than 2.
- ippStsNumChannelsErr: Indicates an error when the numChannels value differs from 1, 3, or 4.
- ippStsDataTypeErr: Indicates an error when the dataType value differs from ipp8u, ipp16u, ipp16s, or ipp32f.

See Also
Histogram Computes the intensity histogram of an image.

Histogram

*Computes the intensity histogram of an image.*

**Syntax**

**Case 1: One-channel data**

IppStatus ippiHistogram_<mod>(const Ipp<dataType>* pSrc, int srcStep, IppiSize roiSize, Ipp32u* pHist, const IppiHistogramSpec* pSpec, Ipp8u* pBuffer);

Supported values for mod:

8u_C1R 16u_C1R 16s_C1R 32f_C1R

**Case 2: Three-channel data**

IppStatus ippiHistogram_<mod>(const Ipp<dataType>* pSrc, int srcStep, IppiSize roiSize, Ipp32u* pHist[3], const IppiHistogramSpec* pSpec, Ipp8u* pBuffer);

Supported values for mod:

8u_C3R 16u_C3R 16s_C3R 32f_C3R

**Case 3: Four-channel data**

IppStatus ippiHistogram_<mod>(const Ipp<dataType>* pSrc, int srcStep, IppiSize roiSize, Ipp32u* pHist[4], const IppiHistogramSpec* pSpec, Ipp8u* pBuffer);

Supported values for mod:

8u_C4R 16u_C4R 16s_C4R 32f_C4R

**Include Files**

ippi.h

**Domain Dependencies**

Headers: ippcore.h, ippvm.h, ipps.h
**Libraries:** ippcore.lib, ippvm.lib, ipps.lib

**Parameters**

- **pSrc**
  Pointer to the source image ROI.
- **srcStep**
  Distance, in bytes, between the starting points of consecutive lines in the source image.
- **roiSize**
  Size of the source image ROI, in pixels.
- **pHist**
  Pointer to the computed histogram. In case of multi-channel data, `pHist` is an array of pointers to the histogram for each channel.
- **pSpec**
  Pointer to the specification structure.
- **pBuffer**
  Pointer to the work buffer.

**Description**

The `ippiHistogram` function operates with ROI (see Regions of Interest in Intel IPP).

This function computes the intensity histogram for each channel of the source image and stores the result in the `pHist` array.

Before calling this function, initialize the specification structure using the `HistogramInit` or `HistogramUniformInit` functions. The specification structure defines the following parameters for histogram calculation:

- Histogram type: with uniform or random levels step
- Number of levels
- Level values

Length of the `pHist` array is defined by the `nLevels` parameter passed to the `HistogramInit` or `HistogramUniformInit` function.

As `nLevels` is the number of levels, the number of values in the `pHist` array, which is the number of histogram bins, is `nLevels - 1`. The meaning of the `pHist` and `pLevels` values can be illustrated by the following example: `pHist[k]` is the number of the source image pixels `pSrc(x, y)` that satisfy the condition `pLevels[k]<=pSrc(x, y)<pLevels(k+1)`.

**Return Values**

- **ippStsNoErr**
  Indicates no error. Any other value indicates an error or a warning.
- **ippStsNullPtrErr**
  Indicates an error when one of the specified pointers is NULL.
- **ippStsSizeErr**
  Indicates an error when `roiSize` has a zero or negative value.
- **ippStsStepErr**
  Indicates an error when `srcStep` is less than `roiSize.width*sizeOf(*pSrc)*nChannels`.
- **ippStsBadArgErr**
  Indicates an error when the `pSpec` object is not initialized.

**Example**

The code example below demonstrates how to use the `HistogramGetBufferSize`, `HistogramUniformInit`, `HistogramGetLevels`, and `Histogram` functions.

```c
void HistogramExample()
{
    const int HEIGHT = 8;
    // Code...
}
```
const int WIDTH = 8;
Ipp8u pImg[WIDTH*HEIGHT];
IppiSize roi = {WIDTH, HEIGHT};
int 1;
IppStatus sts;
{
  // fill image with random values in [0..255] range with uniform distribution.
  IppsRandUniState_8u* pRndObj;
  int sizeRndObj;

  // get spec size
  ippsRandUniformGetSize_8u( &sizeRndObj );
  pRndObj = (IppsRandUniState_8u*)ippsMalloc_8u( sizeRndObj );
  // initialize rnd spec
  ippsRandUniformInit_8u(pRndObj, 0/*low*/, 255/*high*/, 0/*seed*/ );

  // fill image
  for ( i=0; i<HEIGHT; i++ ) {
    sts = ippsRandUniform_8u(pImg + i*WIDTH, WIDTH, pRndObj);
  }
  ippsFree( pRndObj );
}

printf_8u_2D("pImg:", pImg, roi, WIDTH, sts);
{
  const int nBins = 5;
  int nLevels[] = { nBins+1 };  
  Ipp32f lowerLevel[] = {0};
  Ipp32f upperLevel[] = {255};
  Ipp32f pLevels[nBins+1], *ppLevels[1];
  int sizeHistObj, sizeBuffer;
  IppiHistogramSpec* pHistObj;
  Ipp8u* pBuffer;
  Ipp32u pHistVec[nBins];

  // get sizes for spec and buffer
  ippiHistogramGetBufferSize(ipp8u, roi, nLevels, 1/*nChan*/, 1/*uniform*/, &sizeHistObj, &sizeBuffer);
  pHistObj = (IppiHistogramSpec*)ippsMalloc_8u( sizeHistObj );
  pBuffer = (Ipp8u*)ippsMalloc_8u( sizeBuffer );
  // initialize spec
  ippiHistogramUniformInit( ipp8u, lowerLevel, upperLevel, nLevels, 1, pHistObj );

  // check levels of bins
  ppLevels[0] = pLevels;
  sts = ippiHistogramGetLevels( pHistObj, ppLevels );
  printf_32f( "pLevels:", pLevels, nBins+1, sts );

  // calculate histogram
  sts = ippiHistogram_8u_C1R( pImg, WIDTH, roi, pHistVec, pHistObj, pBuffer );
  ippsFree( pHistObj );
  ippsFree( pBuffer );
printf_32u( "Histogram:", pHistVec, nBins, sts );
}
}

Output:

pImg:
0 33 53 102 90 188 210 60
195 137 247 137 7 15 65 244
149 44 210 20 170 140 183 144
133 61 191 32 212 108 178 89
86 30 54 93 168 93 2 114
30 145 216 42 86 113 148 205
148 181 217 99 219 31 156 144
237 36 74 80 208 121 118 106

pLevels:
0.0 51.0 102.0 153.0 204.0 255.0

Histogram:
13 14 16 10 11

See Also

Regions of Interest in Intel IPP

Histogram Computes the intensity histogram of an image.
HistogramBufferSize Computes the size of the specification structure and work buffer for the
ippiHistogram function.
HistogramGetLevels Returns the array with level values stored in the specification structure.
HistogramInit, HistogramUniformInit Initializes the specification structure for the ippiHistogram
function.

CountInRange

Computes the number of pixels within the given
intensity range.

Syntax

Case 1: Operation on one-channel data

IppStatus ippiCountInRange_<mod>(const Ipp<datatype>* pSrc, int srcStep, IppiSize roiSize, int* counts, Ipp<datatype> lowerBound, Ipp<datatype> upperBound);

Supported values for mod:

8u_C1R 32f_C1R

Case 2: Operation on multi-channel data

IppStatus ippiCountInRange_<mod>(const Ipp<datatype>* pSrc, int srcStep, IppiSize roiSize, int counts[3], Ipp<datatype> lowerBound[3], Ipp<datatype> upperBound[3]);

Supported values for mod:

8u_C3R 32f_C3R
8u_AC4R 32f_AC4R
Include Files
ippi.h

Domain Dependencies
Headers: ippi.h, ippvm.h, ipps.h
Libraries: ippcore.lib, ippvm.lib, ipps.lib

Parameters

$pSrc$  
Pointer to the source image ROI.

$srcStep$  
Distance in bytes between starts of consecutive lines in the source image.

$roiSize$  
Size of the source ROI in pixels.

$counts$  
The computed number of pixels within the given intensity range. An array of 3 values in case of multi-channel data.

$lowerBound$  
Lower limit of the intensity range.

$upperBound$  
Upper limit of the intensity range.

Description
This function operates with ROI (see Regions of Interest in Intel IPP). This function computes the number of pixels in the image which have intensity values in the range between $lowerBound$ and $upperBound$ (inclusive).

In case of a multi-channel image, pixels are counted within intensity range for each color channel separately, and the array $counts$ of three resulting values is returned. The alpha channel values, if present, are not processed.

Return Values

ippStsNoErr  
Indicates no error. Any other value indicates an error or a warning.

ippStsNullPtrErr  
Indicates an error condition if $pSrc$ pointer is NULL.

ippStsSizeErr  
Indicates an error condition if $roiSize$ has a field with zero or negative value.

ippStsStepErr  
Indicates an error condition if $srcStep$ has a zero or negative value.

ippStsRangeErr  
Indicates an error condition if $lowerBound$ exceeds $upperBound$.

BlockMinMax

Finds minimum and maximum values for blocks of an image.
Syntax
IppStatus ippiBlockMinMax_<dataType>_C1R(const Ipp<dataType>* pSrc, int srcStep, IppSize srcSize, Ipp<dataType>* pDstMin, int dstMinStep, Ipp<dataType>* pDstMax, int dstMaxStep, IppSize blockSize, Ipp<dataType>* pGlobalMin, Ipp<dataType>* pGlobalMax);

Supported values for dataType:

<p>| | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>8u</td>
<td>16u</td>
<td>16s</td>
<td>32f</td>
</tr>
</tbody>
</table>

Include Files
ippi.h

Domain Dependencies
Headers: ippcore.h, ippvm.h, ipps.h
Libraries: ippcore.lib, ippvm.lib, ipps.lib

Parameters

- **pSrc**
  Pointer to the source image ROI.
- **srcStep**
  Distance, in bytes, between the starting points of consecutive lines in the source image.
- **srcSize**
  Size, in pixels, of the source image.
- **pDstMin**
  Pointer to the destination image to store minimum values per block.
- **dstMinStep**
  Distance, in bytes, between the starting points of consecutive lines in the pDstMin image.
- **pDstMax**
  Pointer to the destination image to store maximum values per block.
- **dstMaxStep**
  Distance, in bytes, between the starting points of consecutive lines in the pDstMax image.
- **blockSize**
  Size, in pixels, of the image block.
- **pGlobalMin**
  Destination pointer to the minimum value for the entire source image.
- **pGlobalMax**
  Destination pointer to the maximum value for the entire source image.

Description

This function operates with ROI.

This function finds minimum and maximum values for blocks of the source image, which are defined by the blockSize parameter. Minimum and maximum values for blocks are stored in the pDstMin and pDstMax images, respectively. Minimum and maximum values for the entire image are stored in the pGlobalMin and pGlobalMax pointers, respectively.

If pDstMin or pDstMax pointer is NULL, the corresponding component (minimum or maximum value) is not calculated.

The size of the pDstMin and pDstMax images is calculated by the following formulae:

- if srcWidth is divisible by blockSize, the destination width is equal to:
  
  \[ \text{dstWidth} = \text{srcWidth} / \text{blockSize} \]
otherwise:
\[ \text{dstWidth} = \frac{\text{srcWidth}}{\text{blockWidth}} + 1 \]

- if \( \text{srcHeight} \) is divisible by \( \text{blockHeight} \), the destination height is equal to:
\[ \text{dstHeight} = \frac{\text{srcHeight}}{\text{blockHeight}} \]
- otherwise:
\[ \text{dstHeight} = \frac{\text{srcHeight}}{\text{blockHeight}} + 1 \]

**Return Values**

- ippStsNoErr: Indicates no error.
- ippStsNullPtrErr: Indicates an error when \( pSrc \), \( pDstMin \), and \( pDstMax \) pointers are NULL.
- ippStsStepErr: Indicates an error when:
  - \( \text{srcStep} \) is less than \( \text{srcSize.width} \times \text{<pixelSize>} \)
  - \( \text{dstMinStep} \) or \( \text{dstMaxStep} \) is less than \( \text{dstSize.width} \times \text{<pixelSize>} \)
- ippStsSizeErr: Indicates an error when \( \text{srcSize} \) or \( \text{ blockSize} \) has a zero or negative value.

**See Also**
Regions of Interest in Intel IPP

**Min**

*Computes the minimum of image pixel values.*

**Syntax**

**Case 1: Operation on one-channel data**

```c
IppStatusippiMin_<mod>(const Ipp<datatype>\* pSrc, int srcStep, IppiSize roiSize, Ipp<datatype>\* pMin);
```

Supported values for \( \text{mod} \):

- 8u_C1R
- 16u_C1R
- 16s_C1R
- 32f_C1R

**Case 2: Operation on multi-channel data**

```c
IppStatusippiMin_<mod>(const Ipp<datatype>\* pSrc, int srcStep, IppiSize roiSize, Ipp<datatype> min[3]);
```

Supported values for \( \text{mod} \):

- 8u_C3R
- 16u_C3R
- 16s_C3R
- 32f_C3R
- 8u_AC4R
- 16u_AC4R
- 16s_AC4R
- 32f_AC4R
IppStatusippiMin_<mod>(const Ipp<datatype>* pSrc, int srcStep, IppiSize roiSize, Ipp<datatype> min[4]);

Supported values for mod:

- 8u_C4R
- 16u_C4R
- 16s_C4R
- 32f_C4R

Include Files

ippi.h

Domain Dependencies

Headers: ippcore.h, ippvm.h, ipps.h
Libraries: ippcore.lib, ippvm.lib, ipps.lib

Parameters

- pSrc
  Pointer to the source image ROI.
- srcStep
  Distance in bytes between starts of consecutive lines in the source image.
- roiSize
  Size of the source ROI in pixels.
- pMin
  Pointer to the minimum pixel value (for one-channel data).
- min
  Array containing minimum channel values of pixels in the source buffer (for multi-channel data).

Description

This function operates with ROI (see Regions of Interest in Intel IPP). This function computes the minimum pixel value pMin for the source image pSrc. In case of a multi-channel image, the minimum is computed over each channel and stored in the array min.

Return Values

- ippStsNoErr
  Indicates no error. Any other value indicates an error or a warning.
- ippStsNullPtrErr
  Indicates an error condition if pSrc or pMin pointer is NULL.
- ippStsSizeErr
  Indicates an error condition if roiSize has a field with zero or negative value.

Example

The code example below demonstrates how to use the functionippiMin.

```c
Ipp8u src[4*1] = { 40, 20, 60, 80 };
IppiSize roiSize = { 4, 1 };
Ipp8u min;
ippiMin_8u_C1R ( src, 4, roiSize, &min );
result: min = 20
```
MinIndx

Computes the minimum of image pixel values and retrieves the x and y coordinates of pixels with minimal intensity values.

Syntax

Case 1: Operation on one-channel data

IppStatus ippiMinIndx_<mod>(const Ipp<datatype>* pSrc, int srcStep, IppiSize roiSize, Ipp<datatype>* pMin, int* pIndexX, int* pIndexY);

Supported values for mod:

8u_C1R  16u_C1R  16s_C1R  32f_C1R

Case 2: Operation on multi-channel data

IppStatus ippiMinIndx_<mod>(const Ipp<datatype>* pSrc, int srcStep, IppiSize roiSize, Ipp<datatype> min[3], int indexX[3], int indexY[3]);

Supported values for mod:

8u_C3R  16u_C3R  16s_C3R  32f_C3R
8u_AC4R  16u_AC4R  16s_AC4R  32f_AC4R

IppStatus ippiMinIndx_<mod>(const Ipp<datatype>* pSrc, int srcStep, IppiSize roiSize, Ipp<datatype> min[4], int indexX[4], int indexY[4]);

Supported values for mod:

8u_C4R  16u_C4R  16s_C4R  32f_C4R

Include Files

ippi.h

Domain Dependencies

Headers: ippcore.h, ippvm.h, ipps.h
Libraries: ippcore.lib, ippvm.lib, ipps.lib

Parameters

pSrc  
Pointer to the source image ROI.

srcStep  
Distance in bytes between starts of consecutive lines in the source image.

roiSize  
Size of the source ROI in pixels.

pMin  
Pointer to the minimum pixel value (for one-channel data).

min  
Array containing minimum color channel values of pixels in the source buffer (for multi-channel data).

pIndexX, pIndexY  
Pointers to the x and y coordinates of the pixel with minimum value.
**Description**

This function operates with ROI (see Regions of Interest in Intel IPP).

This function computes the minimum pixel value \( p_{\text{Min}} \) for the source image \( p_{\text{Src}} \). In case of a multi-channel image, the minimum is computed over each channel and stored in the array \( \text{min} \). The function also retrieves the \( x \) and \( y \) coordinates of pixels on which the minimum is reached. If several pixels have equal minimum values, the coordinates of the first pixel from the start of the source buffer is returned. For multi-channel data, \( \text{indexX}[k] \) and \( \text{indexY}[k] \) are the \( x \) and \( y \) coordinates of the pixel that has the minimal intensity value of the \( k \)-th channel, \( k = 1,2,3,4 \).

**Return Values**

- **ippStsNoErr**
  - Indicates no error. Any other value indicates an error or a warning.
- **ippStsNullPtrErr**
  - Indicates an error condition if one of \( p_{\text{Src}}, p_{\text{Min}}, p_{\text{IndexX}}, \) or \( p_{\text{IndexY}} \) pointers is NULL.
- **ippStsSizeErr**
  - Indicates an error condition if \( \text{roiSize} \) has a field with zero or negative value.

**Max**

*Computes the maximum of image pixel values.*

**Syntax**

**Case 1: Operation on one-channel data**

\[
\text{IppStatus ippiMax\_<mod>}(\text{const Ipp<datatype>* } p_{\text{Src}}, \text{ int srcStep}, \text{ IppSize roiSize}, \text{ Ipp<datatype>* } p_{\text{Max}});
\]

Supported values for \( \text{mod} \):

- 8u\_C1R
- 16u\_C1R
- 16s\_C1R
- 32f\_C1R

**Case 2: Operation on multi-channel data**

\[
\text{IppStatus ippiMax\_<mod>}(\text{const Ipp<datatype>* } p_{\text{Src}}, \text{ int srcStep}, \text{ IppSize roiSize}, \text{ Ipp<datatype> } \text{max}[3]);
\]

Supported values for \( \text{mod} \):

- 8u\_C3R
- 16u\_C3R
- 16s\_C3R
- 32f\_C3R
- 8u\_AC4R
- 16u\_AC4R
- 16s\_AC4R
- 32f\_AC4R

**Case 3: Operation on multi-channel data**

\[
\text{IppStatus ippiMax\_<mod>}(\text{const Ipp<datatype>* } p_{\text{Src}}, \text{ int srcStep}, \text{ IppSize roiSize}, \text{ Ipp<datatype> } \text{max}[4]);
\]

Supported values for \( \text{mod} \):

- 8u\_C4R
- 16u\_C4R
- 16s\_C4R
- 32f\_C4R
Include Files
ippi.h

Domain Dependencies
Headers: ippcore.h, ippvm.h, ipps.h
Libraries: ippcore.lib, ippvm.lib, ipps.lib

Parameters

pSrc   Pointer to the source image ROI.
srcStep Distance in bytes between starts of consecutive lines in the source image.
roiSize Size of the source ROI in pixels
pMax   Pointer to the maximum pixel value (for one-channel data).
max    Array containing maximum channel values of pixels in the source buffer (for multi-channel data).

Description
This function operates with ROI (see Regions of Interest in Intel IPP). This function computes the maximum pixel value \( pMax \) for the source image \( pSrc \). In case of a multi-channel image, the maximum is computed over each channel and stored in the array \( max \).

Return Values

ippStsNoErr Indicates no error. Any other value indicates an error or a warning.
ippStsNullPtrErr Indicates an error condition if \( pSrc \) or \( pMax \) pointer is NULL.
ippStsSizeErr Indicates an error condition if \( roiSize \) has a field with zero or negative value.

MaxIndx

Computes the maximum of image pixel values and retrieves the x and y coordinates of pixels with maximal intensity values.

Syntax
Case 1: Operation on one-channel data

\[
\text{IppStatus ippiMaxIndx_<mod>}(\text{const Ipp<datatype>* } pSrc, \text{ int srcStep, IppSize roiSize, Ipp<datatype>* } pMax, \text{ int* } pIndexX, \text{ int* } pIndexY);
\]

Supported values for mod:

8u_C1R  16u_C1R  16s_C1R  32f_C1R
Case 2: Operation on multi-channel data

IppStatus ippiMaxIndx_<mod>(const Ipp<datatype>* pSrc, int srcStep, IppiSize roiSize, Ipp<datatype> max[3], int indexX[3], int indexY[3]);

Supported values for mod:

- 8u_C3R
- 16u_C3R
- 16s_C3R
- 32f_C3R
- 8u_AC4R
- 16u_AC4R
- 16s_AC4R
- 32f_AC4R

IppStatus ippiMaxIndx_<mod>(const Ipp<datatype>* pSrc, int srcStep, IppiSize roiSize, Ipp<datatype> max[4], int indexX[4], int indexY[4]);

Supported values for mod:

- 8u_C4R
- 16s_C4R
- 16u_C4R
- 32f_C4R

Include Files

ippi.h

Domain Dependencies

Headers: ippcore.h, ippvm.h, ipps.h
Libraries: ippcore.lib, ippvm.lib, ipps.lib

Parameters

- pSrc: Pointer to the source image ROI.
- srcStep: Distance in bytes between starts of consecutive lines in the source image.
- roiSize: Size of the source ROI in pixels.
- pMax: Pointer to the maximum pixel value (for one-channel data).
- max: Array containing maximum channel values of pixels in the source buffer (for multi-channel data).
- pIndexX, pIndexY: Pointers to the x and y coordinates of the pixel with maximum value.
- indexX, indexY: Arrays containing the x and y coordinates of pixels with maximum channel values.

Description

This function operates with ROI (see Regions of Interest in Intel IPP). This function computes the maximum pixel value \( pMax \) for the source image \( pSrc \). In case of a multi-channel image, the maximum is computed over each channel and stored in the array \( max \). The function also retrieves the \( x \) and \( y \) coordinates of pixels on which the maximum is reached. If several pixels have equal maximum values, the coordinates of the first pixel from the start of the source buffer is returned. For multi-channel data, \( indexX[k] \) and \( indexY[k] \) are the \( x \) and \( y \) coordinates of the pixel that has the maximal intensity value of the \( k \)-th channel, \( k = 1,2,3,4 \).

Return Values

- ippStsNoErr: Indicates no error. Any other value indicates an error or a warning.
Indicates an error condition if any of $pSrc, pMax, pIndexX$, or $pIndexY$ pointers is NULL.

Indicates an error condition if $roiSize$ has a field with zero or negative value.

**Example**

The code example below demonstrates how to use the function `ippiMaxIndx`.

```c
Ipp8u src[4*1] = { 40, 20, 60, 80 };
IppiSize roiSize = { 4, 1 };
Ipp8u max;
int IndexX;
int IndexY;

ippiMaxIndx_8u_C1R ( src, 4, roiSize, &max, &IndexX, &IndexY );
```

result: $max = 80$  $IndexX = 3$  $IndexY = 0$

---

**MinMax**

*Computes the minimum and maximum of image pixel values.*

**Syntax**

**Case 1: Operation on one-channel data**

```c
IppStatus ippiMinMax_<mod>(const Ipp<datatype>* $pSrc$, int $srcStep$, IppiSize $roiSize$, Ipp<datatype>* $pMin$, Ipp<datatype>* $pMax$);
```

Supported values for `mod`:

- 8u_C1R
- 16u_C1R
- 16s_C1R
- 32f_C1R

**Case 2: Operation on multi-channel data**

```c
```

Supported values for `mod`:

- 8u_C3R
- 16u_C3R
- 16s_C3R
- 32f_C3R
- 8u_AC4R
- 16u_AC4R
- 16s_AC4R
- 32f_AC4R

```c
```

Supported values for `mod`:

- 8u_C4R
- 16u_C4R
- 16s_C4R
- 32f_C4R
Include Files
ippi.h

Domain Dependencies
Headers: ippcore.h, ippvm.h, ipps.h
Libraries: ippcore.lib, ippvm.lib, ipps.lib

Parameters
pSrc
srcStep
roiSize
pMin, pMax
min, max

Description
This function operates with ROI (see Regions of Interest in Intel IPP). This function computes the minimum and maximum pixel values \( p_{\text{Min}} \) and \( p_{\text{Max}} \) for the source image \( p_{\text{Src}} \). In case of a multi-channel image, the minimum and maximum is computed over each channel and stored in the arrays \( \text{min} \) and \( \text{max} \).

Return Values
ippStsNoErr
Indicates no error. Any other value indicates an error or a warning.
ippStsNullPtrErr
Indicates an error condition if \( p_{\text{Src}}, p_{\text{Min}}, \) or \( p_{\text{Max}} \) pointer is NULL.
ippStsSizeErr
Indicates an error condition if \( \text{roiSize} \) has a field with zero or negative value.

MinMaxIndx

Calculates minimum and maximum pixel values and their indexes in selected image rectangle.

Syntax
Case 1: Operation on one-channel data
IppStatus ippiMinMaxIndx_<mod>(const Ipp<datatype>* pSrc, int srcStep, IppiSize roiSize, Ipp32f* pMinVal, Ipp32f* pMaxVal, IppiPoint* pMinIndex, IppiPoint* pMaxIndex);

Supported values for \( \text{mod} \):
8u_C1R  16u_C1R  32f_C1R
Case 2: Masked operation on one-channel data

IppStatus ippiMinMaxIndx_<mod>(const Ipp<datatype>* pSrc, int srcStep, const Ipp8u* pMask, int maskStep, IppiSize roiSize, Ipp32f* pMinVal, Ipp32f* pMaxVal, IppiPoint* pMinIndex, IppiPoint* pMaxIndex);

Supported values for mod:

8u_C1MR   16u_C1MR   32f_C1MR

Case 3: Operation on multi-channel data

IppStatus ippiMinMaxIndx_<mod>(const Ipp<datatype>* pSrc, int srcStep, IppiSize roiSize, int coi, Ipp32f* pMinVal, Ipp32f* pMaxVal, IppiPoint* pMinIndex, IppiPoint* pMaxIndex);

Supported values for mod:

8u_C3CR   16u_C3CR   32f_C3CR

Case 4: Masked operation on multi-channel data

IppStatus ippiMinMaxIndx_<mod>(const Ipp<datatype>* pSrc, int srcStep, const Ipp8u* pMask, int maskStep, IppiSize roiSize, int coi, Ipp32f* pMinVal, Ipp32f* pMaxVal, IppiPoint* pMinIndex, IppiPoint* pMaxIndex);

Supported values for mod:

8u_C3CMR  16u_C3CMR  32f_C3CMR

Include Files

ippcv.h

Domain Dependencies

Headers: ippcore.h, ippvm.h, ipps.h, ippi.h
Libraries: ippcore.lib, ippvm.lib, ipps.lib, ippi.lib

Parameters

pSrc
srcStep
pMask
maskStep
roiSize
coi
pMinVal
pMaxVal
pMinIndex

Pointer to the source image ROI.
Distance in bytes between starts of consecutive lines in the source image.
Pointer to the mask image.
Distance in bytes between starts of consecutive lines in the mask image.
Size of the image ROI in pixels.
Channel of interest (for color images only); can be 1, 2, or 3.
Pointer to the variable that returns the value of the minimum pixel.
Pointer to the variable that returns the value of the maximum pixel.
Pointer to the variable that returns the index of the minimum value found.
**pMaxIndex**

Pointer to the variable that returns the index of the maximum value found.

**Description**

This function operates with ROI (see Regions of Interest in Intel IPP). This function finds minimum and maximum pixel values and their indexes in an image ROI or in an arbitrary image region defined by nonzero mask values. If there are several minima and maxima in the selected area, the function returns the top leftmost positions. If the specified region in the mask mode is empty, that is, the mask image is filled with zeros, then the function returns \( \{\text{minIndex}, \text{maxIndex}\} = \{0, 0\}, \text{minVal} = \text{maxVal} = 0. \) If any of the parameters \( p\text{MinVal}, p\text{MaxVal}, p\text{MinIndex}, \) or \( p\text{MaxIndex} \) is not required, the zero pointer is to be passed to the corresponding parameter.

**Return Values**

- **ippStsNoErr**
  Indicates no error. Any other value indicates an error or a warning.
- **ippStsNullPtrErr**
  Indicates an error condition if \( p\text{Src} \) or \( p\text{Mask} \) pointer is NULL.
- **ippStsSizeErr**
  Indicates an error condition if \( \text{roiSize} \) has a field with zero or negative value.
- **ippStsStepErr**
  Indicates an error for masked operations when \( \text{srcStep} \) or \( \text{maskStep} \) is less than \( \text{roiSize}.\text{width} \times \text{pixelSize} \).
- **ippStsNotEvenStepErr**
  Indicates an error when steps for floating-point images cannot be divided by 4.
- **ippStsCOIErr**
  Indicates an error when \( \text{coi} \) is not 1, 2, or 3.

**MaxEvery**

*Computes maximum value for each pair of pixels of two images.*

**Syntax**

**Case 1: Not-in-place operation**

IppStatus ippiMaxEvery_8u_C1R(const Ipp8u* pSrc1, int src1Step, const Ipp8u* pSrc2, int src2Step, Ipp8u* pDst, int dstStep, IppiSize roiSize);

IppStatus ippiMaxEvery_16u_C1R(const Ipp16u* pSrc1, int src1Step, const Ipp16u* pSrc2, int src2Step, Ipp16u* pDst, int dstStep, IppiSize roiSize);

IppStatus ippiMaxEvery_32f_C1R(const Ipp32f* pSrc1, int src1Step, const Ipp32f* pSrc2, int src2Step, Ipp32f* pDst, int dstStep, IppiSize roiSize);

**Case 2: In-place operation**

IppStatus ippiMaxEvery_<mod>(const Ipp<datatype>* pSrc, int srcStep, Ipp<datatype>* pSrcDst, int srcDstStep, IppiSize roiSize);

Supported values for \( \text{mod} \):

- 8u_C1IR
- 16u_C1IR
- 16s_C1IR
- 32f_C1IR
- 8u_C3IR
- 16u_C3IR
- 16s_C3IR
- 32f_C3IR
- 8u_C4IR
- 16u_C4IR
- 16s_C4IR
- 32f_C4IR
Include Files
ippi.h

Domain Dependencies
Headers: ippicore.h, ippvm.h, ipps.h
Libraries: ippcore.lib, ippvm.lib, ipps.lib

Parameters
pSrc1, pSrc2        Pointer to the first and second source image, respectively (for not-in-place operation).
src1Step, src2Step  Distance, in bytes, between the starting points of consecutive lines in the first and second source image, respectively (for not-in-place operation).
pDst                  Pointer to the destination image (for not-in-place operation).
pSrc                  Pointer to the first source image ROI (for in-place operation).
srcStep           Distance, in bytes, between the starting points of consecutive lines in the first source image (for in-place operation).
dstStep                 Distance, in bytes, between the starting points of consecutive lines in the destination image.
pSrcDst              Pointer to the second source and destination image ROI (for in-place operation).
srcDstStep         Distance, in bytes, between the starting points of consecutive lines in the second source and destination image (for in-place operation).
roiSize                Size of the image ROI in pixels.

Description
This function operates with ROI (see Regions of Interest in Intel IPP).

Not-in-place operation:
This function computes the maximum value for each pair of the corresponding pixels of two source images (pSrc1 and pSrc2 for not-in-place operation or pSrc and pSrcDst for in-place), and stores the result in pDst.

In-place operation:
This function computes the maximum value for each pair of the corresponding pixels of two source images pSrc and pSrcDst, and stores the result in pSrcDst:
pSrcDst(i, j) = max(pSrc(i, j), pSrcDst(i, j)).

Return Values
ippStsNoErr          Indicates no error. Any other value indicates an error or a warning.
ippStsNullPtrErr     Indicates an error condition if one of the specified pointers is NULL.
ippStsSizeErr        Indicates an error condition if roiSize has a field with zero or negative value.
ippStsStepErr  Indicates an error condition if srcStep, src1Step, src2Step or srcDstStep is less than roiSize.width*pixelSize.

ippStsNotEvenStepErr  Indicates an error condition if one of step values for floating-point images are not divisible by 4.

**MinEvery**

*Computes minimum value for each pair of pixels of two images.*

**Syntax**

**Case 1: Not-in-place operation**

IppStatus ippiMinEvery_8u_C1R(const Ipp8u* pSrc1, int src1Step, const Ipp8u* pSrc2, int src2Step, Ipp8u* pDst, int dstStep, IppiSize roiSize);

IppStatus ippiMinEvery_16u_C1R(const Ipp16u* pSrc1, int src1Step, const Ipp16u* pSrc2, int src2Step, Ipp16u* pDst, int dstStep, IppiSize roiSize);

IppStatus ippiMinEvery_32f_C1R(const Ipp32f* pSrc1, int src1Step, const Ipp32f* pSrc2, int src2Step, Ipp32f* pDst, int dstStep, IppiSize roiSize);

**Case 2: In-place operation**

IppStatus ippiMinEvery_<mod>(const Ipp<datatype>* pSrc, int srcStep, Ipp<datatype>* pSrcDst, int srcDstStep, IppiSize roiSize);

Supported values for mod:

<p>| | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>8u_C1IR</td>
<td>16u_C1IR</td>
<td>16s_C1IR</td>
<td>32f_C1IR</td>
<td></td>
</tr>
<tr>
<td>8u_C3IR</td>
<td>16u_C3IR</td>
<td>16s_C3IR</td>
<td>32f_C3IR</td>
<td></td>
</tr>
<tr>
<td>8u_C4IR</td>
<td>16u_C4IR</td>
<td>16s_C4IR</td>
<td>32f_C4IR</td>
<td></td>
</tr>
<tr>
<td>8u_AC4IR</td>
<td>16u_AC4IR</td>
<td>16s_AC4IR</td>
<td>32f_AC4IR</td>
<td></td>
</tr>
</tbody>
</table>

**Include Files**

ippi.h

**Domain Dependencies**

Headers: ippcore.h, ippvm.h, ipps.h

Libraries: ippcore.lib, ippvm.lib, ipps.lib

**Parameters**

pSrc1, pSrc2  Pointer to the first and second source image, respectively (for not-in-place operation).

src1Step, src2Step  Distance, in bytes, between the starting points of consecutive lines in the first and second source image, respectively (for not-in-place operation).

pDst  Pointer to the destination image (for not-in-place operation).

pSrc  Pointer to the first source image ROI (for in-place operation).
**Description**

This function operates with ROI (see Regions of Interest in Intel IPP).

**Not-in-place operation:**
This function computes the minimum value for each pair of the corresponding pixels of two source images ($pSrc_1$ and $pSrc_2$ for not-in-place operation or $pSrc$ and $pSrcDst$ for in-place), and stores the result in $pDst$.

**In-place operation:**
This function computes the minimum value for each pair of the corresponding pixels of two source images $pSrc$ and $pSrcDst$, and stores the result in $pSrcDst$:

$$pSrcDst(i, j) = \min(pSrc(i, j), pSrcDst(i, j)).$$

**Return Values**

- ippStsNoErr: Indicates no error. Any other value indicates an error or a warning.
- ippStsNullPtrErr: Indicates an error condition if one of the specified pointers is NULL.
- ippStsSizeErr: Indicates an error condition if roiSize has a field with zero or negative value.
- ippStsStepErr: Indicates an error condition if srcStep, src1Step, src2Step or srcDstStep is less than roiSize.width*<pixelSize>.
- ippStsNotEvenStepErr: Indicates an error condition if one of step values for floating-point images are not divisible by 4.

**FindPeaks3x3GetBufferSize**

*Computes the size of the working buffer for the peak search.*

**Syntax**

```c
IppStatus ippiFindPeaks3x3GetBufferSize_32s_C1R(int roiWidth, int* pBufferSize);
IppStatus ippiFindPeaks3x3GetBufferSize_32f_C1R(int roiWidth, int* pBufferSize);
```

**Include Files**

ippcv.h
Domain Dependencies

**Headers:** ippcore.h, ippvm.h, ipps.h,ippi.h

**Libraries:** ippcore.lib, ippvm.lib, ipps.lib,ippi.lib

Parameters

- *roiWidth*  
  Maximum width of the image, in pixels.

- *pBufferSize*  
  Pointer to the size of the working buffer.

Description

This function operates with ROI (see Regions of Interest in Intel IPP).

This function computes the size of the working buffer required for the function ippiFindPeaks3x3. The buffer with the length $pBufferSize[0]$ can be used to filter images with width that is less than or equal to $roiWidth$.

Example 11-8 shows how to use the function ippiFindPeaks3x3GetBufferSize_32f_C1R.

Return Values

- ippStsNoErr  
  Indicates no error. Any other value indicates an error or a warning.

- ippStsNullPtrErr  
  Indicates an error condition if the pointer $pBufferSize$ is NULL.

- ippStsSizeErr  
  Indicates an error condition if $roiWidth$ is less than 1.

FindPeaks3x3

*Finds coordinates of peaks (maximums or minimums)*

*with absolute value exceeding threshold value.*

Syntax

```c
IppStatus ippiFindPeaks3x3_32s_C1R(const Ipp32s* pSrc, int srcStep, IppiSize roiSize, Ipp32s threshold, IppiPoint* pPeak, int maxPeakCount, int* pPeakCount, IppiNorm norm, int border, Ipp8u* pBuffer);

IppStatus ippiFindPeaks3x3_32f_C1R(const Ipp32f* pSrc, int srcStep, IppiSize roiSize, Ipp32f threshold, IppiPoint* pPeak, int maxPeakCount, int* pPeakCount, IppiNorm norm, int border, Ipp8u* pBuffer);
```

Include Files

ippcv.h

Domain Dependencies

**Headers:** ippcore.h, ippvm.h, ipps.h,ippi.h

**Libraries:** ippcore.lib, ippvm.lib, ipps.lib,ippi.lib

Parameters

- *pSrc*  
  Pointer to the first source image ROI.
**srcStep**

Distance in bytes between starts of consecutive lines in the first source image.

**roiSize**

Size of the image ROI in pixels.

**threshold**

Threshold value.

**pPeak**

Pointer to the coordinates peaks [maxPeakCount].

**maxPeakCount**

Maximum number of peaks.

**pPeakCount**

Pointer to the number of the detected peaks.

**border**

Border value, only pixel with distance from the edge of the image greater than border are processed.

**norm**

Specifies type of the norm to form the mask for extremum search:

-ippiNormInf: Infinity norm (8-connectivity, 3x3 rectangular mask);
-ippiNormL1: L1 norm (4-connectivity, 3x3 cross mask).

**pBuffer**

Pointer to the working buffer.

**Description**

This function operates with ROI (see Regions of Interest in Intel IPP).

This function detects local maximum and minimum pixels in the source image:

\[
p_{Src}(i_m, j_m) = \max_{(k, l) \in O(i_m, j_m)} p_{Src}(k, l),\quad p_{Src}(i_m, j_m) \geq \text{threshold}
\]

\[
p_{Src}(i_m, j_m) = \min_{(k, l) \in O(i_m, j_m)} p_{Src}(k, l),\quad |p_{Src}(i_m, j_m)| \geq \text{threshold}
\]

and stores their coordinates in the pPeak array pPeak[m].x = jm, pPeak[m].y = im, m = 0, ..., pPeakCount[0], pPeakCount[0] ≤ maxPeakCount

The neighborhood O(i, j) for the extremum search is defined by the parameter norm. The number of detected extremums is returned in pPeakCount[0]. The operation is stopped when the maxPeakCount extremums are found.

The function requires the working buffer pBuffer whose size should be computed by the function ippiFindPeaks3x3GetBufferSize beforehand.

**Return Values**

- **ippStsNoErr**: Indicates no error. Any other value indicates an error or a warning.

- **ippStsNullPtrErr**: Indicates an error condition if one of the specified pointers is NULL.

- **ippStsSizeErr**: Indicates an error condition if roiSize has a field with zero or negative value; or if maxPeakCount is less than or equal to 0; or if border is less than 1 or greater than one of 0.5*roiSize.width or of 0.5*roiSize.height.

- **ippStsStepErr**: Indicates an error condition if srcStep is less than roiSize.width*<pixelSize>. 

---

Indicates an error condition if srcStep is not divisible by 4.

**Example**

To better understand usage of this function, refer to the following example in the examples archive available for download from https://software.intel.com/en-us/ipp-manual-examples:

FindPeaks3x3.c

**Image Moments**

Spatial and central moments are important statistical properties of an image. The spatial moment $M_{U}(m,n)$ of order $(m,n)$ is defined as follows:

$$M_{U}(m, n) = \sum_{j} \sum_{k} x_{k}^{m} y_{j}^{n} p_{j, k}$$

where the summation is performed for all rows and columns in the image; $p_{j, k}$ are pixel values; $x_{k}$ and $y_{j}$ are pixel coordinates; $m$ and $n$ are integer power exponents that define the moment order.

The central moment $U_{U}(m,n)$ is the spatial moment computed relative to the “center of gravity” $(x_{0}, y_{0})$:

$$U_{U}(m, n) = \sum_{j} \sum_{k} (x_{k} - x_{0})^{m} (y_{j} - y_{0})^{n} p_{j, k}$$

where $x_{0} = M_{U}(1,0)/ M_{U}(0,0)$ and $y_{0} = M_{U}(0,1)/ M_{U}(0,0)$.

The normalized spatial moment $M(m,n)$ and central moment $U(m,n)$ are defined as follows:

$$M(m, n) = \frac{M_{U}(m, n)}{M_{U}(0, 0)^{m+n+2}}$$

$$U(m, n) = \frac{U_{U}(m, n)}{U_{U}(0, 0)^{m+n+2}}$$

The Intel IPP functions support moments of order $(m, n)$ with $0 \leq m + n \leq 3$. The computation of seven invariant Hu moments derived from the second and third order moments is also supported. All computed moments are stored in context structures of type `IppiMomentState_64s` (for integer versions) or `IppiMomentState_64f` (for floating point versions).

Most Intel IPP functions for computing image moments have code branches that implement different algorithms to compute the results. You can choose the desired code variety to be used by the given function by setting the `hint` argument to one of the following values that are listed in Table "Hint Arguments for Image Moment Functions":

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ippAlgHintNone</td>
<td>The computation algorithm will be chosen by the internal function logic.</td>
</tr>
<tr>
<td>ippAlgHintFast</td>
<td>Fast algorithm must be used. The output results will be less accurate.</td>
</tr>
<tr>
<td>ippAlgHintAccurate</td>
<td>High accuracy algorithm must be used. The function will need more time to execute.</td>
</tr>
</tbody>
</table>
**MomentGetStateSize**

*Computes the size of the external buffer for the moment context structure.*

**Syntax**

```c
IppStatus ippiMomentGetStateSize_64f(IppHintAlgorithm hint, int* pSize);
```

**Include Files**

`ippi.h`

**Domain Dependencies**

**Headers:** ippcore.h, ippvm.h, ipps.h

**Libraries:** ippcore.lib, ippvm.lib, ipps.lib

**Parameters**

- **pSize**: Pointer to the computed value of the buffer size.
- **hint**: Option to select the algorithmic implementation of the function.

**Description**

Use this function to determine the size of the external work buffer for the moment context structure to be initialized by the function `ippiMomentInit`. Computation algorithm is specified by `hint` argument (see Table "Hint Arguments for Image Moment Functions").

**Return Values**

- **ippStsNoErr**: Indicates no error. Any other value indicates an error or a warning.
- **ippStsNullPtrErr**: Indicates an error condition if `pSize` pointer is NULL.

---

**MomentInit**

*Initializes the moment context structure.*

**Syntax**

```c
IppStatus ippiMomentInit_64f(IppiMomentState_64f* pState, IppHintAlgorithm hint);
```

**Include Files**

`ippi.h`

**Domain Dependencies**

**Headers:** ippcore.h, ippvm.h, ipps.h

**Libraries:** ippcore.lib, ippvm.lib, ipps.lib

**Parameters**

- **pState**: Pointer to the structure for storing moment values.
- **hint**: Option to select the algorithmic implementation of the function.
Description
This function initializes the structure that is needed for the function ippiMoments to store the computed image moments. Computation algorithm is specified by hint argument (see Table “Hint Arguments for Image Moment Functions”).

The structure is allocated in the external buffer. The size of this buffer can be computed by the function ippiMomentGetStateSize.

Return Values
ippStsNoErr Indicates no error. Any other value indicates an error or a warning.
ippStsNullPtrErr Indicates an error condition if pState pointer is NULL.

Moments
 Computes all image moments of order 0 to 3 and Hu moment invariants.

Syntax
Case 1: Computation of floating-point results
IppStatus ippiMoments64f_<mod>(const Ipp<datatype>* pSrc, int srcStep, IppiSize roiSize, IppiMomentState_64f* pCtx);

Supported values for mod:

<p>| | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>8u_C1R</td>
<td>16u_C1R</td>
<td>32f_C1R</td>
<td></td>
</tr>
<tr>
<td>8u_C3R</td>
<td>16u_C3R</td>
<td>32f_C3R</td>
<td></td>
</tr>
<tr>
<td>8u_AC4R</td>
<td>16u_AC4R</td>
<td>32f_AC4R</td>
<td></td>
</tr>
</tbody>
</table>

Include Files
ippi.h

Domain Dependencies
Headers: ippcore.h, ippvm.h, ipps.h
Libraries: ippcore.lib, ippvm.lib, ipps.lib

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>pSrc</td>
<td>Pointer to the source image ROI.</td>
</tr>
<tr>
<td>srcStep</td>
<td>Distance in bytes between starts of consecutive lines in the source image.</td>
</tr>
<tr>
<td>roiSize</td>
<td>Size of the source ROI in pixels.</td>
</tr>
<tr>
<td>pCtx</td>
<td>Pointer to the structure that stores image moments.</td>
</tr>
</tbody>
</table>
Description

This function operates with ROI (see Regions of Interest in Intel IPP). This function computes all spatial and central moments of order 0 to 3 for the source image pSrc. The seven Hu moment invariants are also computed. Different functions, ippiMoments64s and ippiMoments64f, are used to compute image moments in integer and floating-point formats, respectively.

The ippiMoments function computes spatial moment values relative to the image point referred to by pSrc. Note that this point is the ROI origin and may not coincide with the entire image origin. If you need to obtain spatial moment values relative to the actual image origin, use ippiGetSpatialMoment functions to recalculate them.

The moments’ values are stored in the pCtx structure. To retrieve a particular moment value, use one of the functions described in the sections that follow.

Return Values

ippStsNoErr Indicates no error. Any other value indicates an error or a warning.
ippStsNullPtrErr Indicates an error condition if pSrc or pCtx is NULL.
ippStsSizeErr Indicates an error condition if roiSize has a field with zero or negative value.
ippStsStepErr Indicates an error condition if srcStep has a zero or negative value.
ippStsContextMatchErr Indicates an error condition if a pointer to an invalid structure is passed.

GetSpatialMoment
Retrieves image spatial moment of the specified order, computed by ippiMoments.

Syntax

IppStatus ippiGetSpatialMoment_64f(const IppiMomentState_64f* pState, int mOrd, int nOrd, int nChannel, IppiPoint roiOffset, Ipp64f* pValue);

Include Files
ippi.h

Domain Dependencies

Headers: ippcore.h, ippvm.h, ipps.h
Libraries: ippcore.lib, ippvm.lib, ipps.lib

Parameters

pState Pointer to the structure that stores image moments.
mOrd, nOrd Integer power exponents defining the moment order. These arguments must satisfy the condition $0 \leq mOrd + nOrd \leq 3$.
nChannel The channel for which the moment is returned.
roiOffset Offset in pixels of the ROI origin (top left corner) from the image origin.
**pValue**

Pointer to the retrieved moment value.

**Description**

This function returns the pointer `pValue` to the spatial moment that was previously computed by the `ippiMoments` function. All spatial moment values are computed by `ippiMoments` relative to the image ROI origin. You may also obtain spatial moment values relative to different point in the image, using the appropriate `roiOffset` settings.

The moment order is specified by the integer exponents `mOrd`, `nOrd`.

**Return Values**

- `ippStsNoErr`: Indicates no error. Any other value indicates an error or a warning.
- `ippStsNullPtrErr`: Indicates an error condition if `pState` or `pValue` pointer is NULL.
- `ippStsContextMatchErr`: Indicates an error condition if a pointer to an invalid structure is passed.
- `ippStsSizeErr`: Indicates an error condition if `mOrd + nOrd` is greater than 3, or `nChannel` has an illegal value.

**GetCentralMoment**

*Retrieves image central moment computed by `ippiMoments`.*

**Syntax**

```c
IppStatus ippiGetCentralMoment_64f(const IppiMomentState_64f* pState, int mOrd, int nOrd, int nChannel, Ipp64f* pValue);
```

**Include Files**

`ippi.h`

**Domain Dependencies**

- **Headers:** `ippcore.h`, `ippvm.h`, `ipps.h`
- **Libraries:** `ippcore.lib`, `ippvm.lib`, `ipps.lib`

**Parameters**

- **pState**: The structure that stores image moments.
- **mOrd**, **nOrd**: Integer power exponents defining the moment order. These arguments must satisfy the condition `0 ≤ mOrd + nOrd ≤ 3`.
- **nChannel**: The channel for which the moment is returned.
- **pValue**: Pointer to the returned moment value.

**Description**

This function returns the pointer `pValue` to the central moment previously computed by the `ippiMoments` function. The moment order is specified by the integer exponents `mOrd`, `nOrd`. 
Return Values

<table>
<thead>
<tr>
<th>Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ippStsNoErr</td>
<td>Indicates no error. Any other value indicates an error or a warning.</td>
</tr>
<tr>
<td>ippStsNullPtrErr</td>
<td>Indicates an error condition if pState or pValue pointer is NULL.</td>
</tr>
<tr>
<td>ippStsContextMatchErr</td>
<td>Indicates an error condition if a pointer to an invalid structure is passed.</td>
</tr>
<tr>
<td>ippStsSizeErr</td>
<td>Indicates an error condition if mOrd + nOrd is greater than 3, or nChannel has an illegal value.</td>
</tr>
</tbody>
</table>

GetNormalizedSpatialMoment

Retrieves the normalized value of the image spatial moment computed by ippiMoments.

Syntax

IppStatus ippiGetNormalizedSpatialMoment_64f(const IppiMomentState_64f* pState, int mOrd, int nOrd, int nChannel, IppiPoint roiOffset, Ipp64f* pValue);

Include Files

ippi.h

Domain Dependencies

Headers: ippcore.h, ippvm.h, ipps.h
Libraries: ippcore.lib, ippvm.lib, ipps.lib

Parameters

- pState
  - The structure that stores image moments.
- mOrd, nOrd
  - Integer power exponents defining the moment order. These arguments must satisfy the condition 0 ≤ mOrd + nOrd ≤ 3.
- nChannel
  - The channel for which the moment is returned.
- roiOffset
  - Offset in pixels of the ROI origin (top left corner) from the image origin.
- pValue
  - Pointer to the returned normalized moment value.

Description

This function normalizes the spatial moment value that was previously computed by the ippiMoments function, and returns the pointer pValue to the normalized moment. See Image Moments for details of moments normalization. The moment order (mOrd, nOrd) is specified by integer power exponents. All spatial moment values are computed by ippiMoments relative to the image ROI origin. You may also obtain normalized spatial moment values relative to different point in the image, using the appropriate roiOffset settings.

Return Values

<table>
<thead>
<tr>
<th>Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ippStsNoErr</td>
<td>Indicates no error. Any other value indicates an error or a warning.</td>
</tr>
<tr>
<td>ippStsNullPtrErr</td>
<td>Indicates an error condition if pState or pValue pointer is NULL.</td>
</tr>
</tbody>
</table>
GetNormalizedCentralMoment

Retrieves the normalized value of the image central moment computed by ippiMoments.

Syntax

IppStatus ippiGetNormalizedCentralMoment_64f(const IppiMomentState_64f* pState, int mOrd, int nOrd, int nChannel, Ipp64f* pValue);

Include Files

ippi.h

Domain Dependencies

Headers: ippcore.h, ippvm.h, ipps.h
Libraries: ippcore.lib, ippvm.lib, ipps.lib

Parameters

pState
The structure that stores image moments.

mOrd, nOrd
Integer power exponents defining the moment order. These arguments must satisfy the condition $0 \leq mOrd + nOrd \leq 3$.

nChannel
The channel for which the moment is returned.

pValue
Pointer to the returned moment value.

Description

This function normalizes the central moment value that was previously computed by the ippiMoments function, and returns the pointer $pValue$ to the normalized moment. The moment order $(mOrd, nOrd)$ is specified by the integer power exponents. See Image Moments for details of moments normalization.

Return Values

ippStsNoErr
Indicates no error. Any other value indicates an error or a warning.

ippStsNullPtrErr
Indicates an error condition if $pState$ or $pValue$ pointer is NULL.

ippStsContextMatchErr
Indicates an error condition if a pointer to an invalid structure is passed.

ippStsMoment00ZeroErr
Indicates an error condition if $M(0,0)$ value is close to zero.

GetHuMoments

Retrieves image Hu moment invariants computed by ippiMoments function.
Syntax

IppStatus ippiGetHuMoments_64f(const IppiMomentState_64f* pState, int nChannel, IppiHuMoment_64f pHm);

Include Files

ippi.h

Domain Dependencies

Headers: ippcore.h, ippvm.h, ipps.h
Libraries: ippcore.lib, ippvm.lib, ipps.lib

Parameters

pState Pointer to the structure that stores image moments.
nChannel The channel for which the moment is returned.
pHm Pointer to the array containing the Hu moment invariants.

Description

This function returns the pointer pHm to the array of seven Hu moment invariants previously computed by the ippiMoments function.

Return Values

ippStsNoErr Indicates no error. Any other value indicates an error or a warning.
ippStsNullPtrErr Indicates an error condition if pState or pHm pointer is NULL.
ippStsContextMatchErr Indicates an error condition if a pointer to an invalid structure is passed.
ippStsMoment00ZeroErr Indicates an error condition if M(0,0) value is close to zero.

Example

To better understand usage of this function, refer to the following example in the examples archive available for download from https://software.intel.com/en-us/ipp-manual-examples :

GetHuMoments.c

Image Norms

The functions described in this section compute the following norms of the image pixel values:

- Infinity norm (the largest absolute pixel value)
- L1 norm (the sum of absolute pixel values)
- L2 norm (the square root of the sum of squared pixel values).

Functions of this group also help you compute the norm of differences in pixel values of two input images as well as the relative error for two input images.

Norm_Inf

*Computes the infinity norm of image pixel values.*
Syntax

Case 1: Operation on one-channel data

IppStatus ippiNorm_Inf_<mod>(const Ipp<datatype>* pSrc, int srcStep, IppiSize roiSize, Ipp64f* pValue);

Supported values for mod:

8u_C1R   16u_C1R   16s_C1R   32f_C1R

Case 2: Masked operation on one-channel data

IppStatus ippiNorm_Inf_<mod>(const Ipp<datatype>* pSrc, int srcStep, const Ipp8u* pMask, int maskStep, IppiSize roiSize, Ipp64f* pNorm);

Supported values for mod:

8u_C1MR  16u_C1MR  32f_C1MR

Case 3: Operation on multi-channel data

IppStatus ippiNorm_Inf_<mod>(const Ipp<datatype>* pSrc, int srcStep, IppiSize roiSize, Ipp64f value[3]);

Supported values for mod:

8u_C3R   16u_C3R   16s_C3R   32f_C3R

IppStatus ippiNorm_Inf_<mod>(const Ipp<datatype>* pSrc, int srcStep, IppiSize roiSize, Ipp64f value[4]);

Supported values for mod:

8u_C4R   16u_C4R   16s_C4R   32f_C4R

Case 4: Masked operation on multi-channel data

IppStatus ippiNorm_Inf_<mod>(const Ipp<datatype>* pSrc, int srcStep, const Ipp8u* pMask, int maskStep, IppiSize roiSize, int coi, Ipp64f* pNorm);

Supported values for mod:

8u_C3CMR  16u_C3CMR  32f_C3CMR

Include Files

ippcv.h
ippi.h

Domain Dependencies

Flavors declared in ippcv.h:

Headers: ippcore.h, ippvm.h, ipps.h,ippi.h
Libraries: ippcore.lib, ippvm.lib, ipps.lib,ippi.lib

Flavors declared inippi.h:
**Parameters**

- **pSrc**: Pointer to the source image ROI.
- **srcStep**: Distance in bytes between starts of consecutive lines in the source image.
- **pMask**: Pointer to the mask image.
- **maskStep**: Distance in bytes between starts of consecutive lines in the mask image.
- **roiSize**: Size of the source ROI in pixels.
- **coi**: Channel of interest (for color images only); can be 1, 2, or 3.
- **pValue**: Pointer to the computed infinity norm of pixel values.
- **value**: An array containing the computed infinity norms of channel values in case of multi-channel data.
- **pNorm**: Pointer to the computed norm value in the mask mode.

**Description**

The flavors of the function `ippiNorm_Inf` that perform masked operation are declared in the `ippcv.h` file. All other function flavors are declared in the `ippi.h` file. This function operates with ROI (see Regions of Interest in Intel IPP) and computes the infinity norm `pValue` (`pNorm` for the mask mode) for the source image `pSrc`. This norm is defined as the largest absolute pixel value in an image. In the mask mode, the computation is done over pixels selected by non-zero mask values.

For non-masked operations on a multi-channel image (Case 3), the norm is computed separately for each channel and stored in the array `value`.

In the mask multi-channel mode (Case 4), the norm is computed for a single channel of interest specified by `coi`.

**Return Values**

- **ippStsNoErr**: Indicates no error. Any other value indicates an error or a warning.
- **ippStsNullPtrErr**: Indicates an error when any of the specified pointers is NULL.
- **ippStsSizeErr**: Indicates an error condition if `roiSize` has a field with zero or negative value.
- **ippStsStepErr**: Indicates an error condition in mask mode, if `srcStep` or `maskStep` is less than `roiSize.width * <pixelSize>`.
- **ippStsNotEvenStepErr**: Indicates an error condition in mask mode if steps for floating-point images cannot be divided by 4.
- **ippStsCOIErr**: Indicates an error when `coi` is not 1, 2, or 3.

**Norm_L1**

*Computes the L1- norm of image pixel values.*
Syntax

Case 1: Operation on one-channel integer data

IppStatus ippiNorm_L1_<mod>(const Ipp<datatype>* pSrc, int srcStep, IppSize roiSize, Ipp64f* pValue);

Supported values for mod:

8u_C1R  16u_C1R  16s_C1R

Case 2: Operation on one-channel floating-point data

IppStatus ippiNorm_L1_32f_C1R(const Ipp32f* pSrc, int srcStep, IppSize roiSize, Ipp64f* pValue, IppHintAlgorithm hint);

Case 3: Masked operation on one-channel data

IppStatus ippiNorm_L1_<mod>(const Ipp<datatype>* pSrc, int srcStep, const Ipp8u* pMask, int maskStep, IppSize roiSize, Ipp64f* pNorm);

Supported values for mod:

8u_C1MR  16u_C1MR  32f_C1MR

Case 4: Operation on multi-channel integer data

IppStatus ippiNorm_L1_<mod>(const Ipp<datatype>* pSrc, int srcStep, IppSize roiSize, Ipp64f value[3]);

Supported values for mod:

8u_C3R  16u_C3R  16s_C3R

IppStatus ippiNorm_L1_<mod>(const Ipp<datatype>* pSrc, int srcStep, IppSize roiSize, Ipp64f value[4]);

Supported values for mod:

8u_C4R  16u_C4R  16s_C4R

Case 5: Operation on multi-channel floating-point data

IppStatus ippiNorm_L1_32f_C3R(const Ipp32f* pSrc, int srcStep, IppSize roiSize, Ipp64f value[3], IppHintAlgorithm hint);

Supported values for mod:

IppStatus ippiNorm_L1_32f_C4R(const Ipp32f* pSrc, int srcStep, IppSize roiSize, Ipp64f value[4], IppHintAlgorithm hint);

Case 6: Masked operation on multi-channel data

IppStatus ippiNorm_L1_<mod>(const Ipp<datatype>* pSrc, int srcStep, const Ipp8u* pMask, int maskStep, IppSize roiSize, int coi, Ipp64f* pNorm);

Supported values for mod:

8u_C3CMR  16u_C3CMR  32f_C3CMR
Include Files
ippcv.h
ippi.h

Domain Dependencies
Flavors declared in ippcv.h:
Headers: ippcore.h, ippvm.h, ipps.h, ippi.h
Libraries: ippcore.lib, ippvm.lib, ipps.lib, ippi.lib
Flavors declared in ippi.h:
Headers: ippcore.h, ippvm.h, ipps.h
Libraries: ippcore.lib, ippvm.lib, ipps.lib

Parameters

- **pSrc**: Pointer to the source image ROI.
- **srcStep**: Distance in bytes between starts of consecutive lines in the source image.
- **pMask**: Pointer to the mask image.
- **maskStep**: Distance in bytes between starts of consecutive lines in the mask image.
- **roiSize**: Size of the source ROI in pixels.
- **coi**: Channel of interest (for color images only); can be 1, 2, or 3.
- **pValue**: Pointer to the computed L1- norm of pixel values.
- **value**: An array containing the computed L1- norms of channel values in case of multi-channel data.
- **pNorm**: Pointer to the computed norm value in the mask mode.
- **hint**: Option to select the algorithmic implementation of the function.

Description
The flavors of the function ippiNorm_L1 that perform masked operation are declared in the ippcv.h file. All other function flavors are declared in the ippi.h file. The function operates with ROI (see Regions of Interest in Intel IPP). It computes the L1- norm pValue (pNorm in mask mode) for the source image pSrc. This norm is defined as the sum of absolute pixel values in an image. Computation algorithm is specified by the hint argument (see Table "Hint Arguments for Image Moment Functions"). In the mask mode, the computation is done over pixels selected by nonzero mask values.

For non-masked operations on a multi-channel image (Case 4, 5), the norm is computed separately for each channel and stored in the array value.

In the mask multi-channel mode (Case 6), the norm is computed for a single channel of interest specified by coi.

Return Values

- ippStsNoErr: Indicates no error. Any other value indicates an error or a warning.
Indicates an error when any of the specified pointers is NULL.
Indicates an error condition if \textit{roiSize} has a field with zero or negative value.
Indicates an error condition in mask mode, if \textit{srcStep} or \textit{maskStep} is less than \textit{roiSize.width} * \textit{pixelSize}.
Indicates an error condition in mask mode if steps for floating-point images cannot be divided by 4.
Indicates an error when \textit{coi} is not 1, 2, or 3.

**Example**

The code example below demonstrates how an image norm can be computed.

```c
IppStatus norm( void ){
    Ipp64f sum, normL1;
    Ipp8u x[5*4];
    IppiSize roi = {5,4};
   ippiSet_8u_C1R( 1, x, 5, roi );
   ippiSum_8u_C1R( x, 5, roi, &sum);
    return ippiNorm_L1_8u_C1R( x, 5, roi, &normL1 );
}
```

**Norm\_L2**

\textit{Computes the L2- norm of image pixel values.}

**Syntax**

**Case 1: Operation on one-channel integer data**

\textbf{IpplStatus ippiNorm\_L2\_<mod>}(const Ipp\<datatype>\* p\textit{Src}, int \textit{srcStep}, IppiSize \textit{roiSize}, Ipp64f\* \textit{pValue});

\textbf{Supported values for} \textit{mod}:

\texttt{8u\_C1R} \hspace{1cm} \texttt{16u\_C1R} \hspace{1cm} \texttt{16s\_C1R}

**Case 2: Operation on one-channel floating-point data**

\textbf{IpplStatus ippiNorm\_L2\_32f\_C1R}(const Ipp32f\* p\textit{Src}, int \textit{srcStep}, IppiSize \textit{roiSize}, Ipp64f\* \textit{pValue}, IppHintAlgorithm \textit{hint});

**Case 3: Masked operation on one-channel data**

\textbf{IpplStatus ippiNorm\_L2\_<mod>}(const Ipp\<datatype>\* p\textit{Src}, int \textit{srcStep}, const Ipp8u\* p\textit{Mask}, int \textit{maskStep}, IppiSize \textit{roiSize}, Ipp64f\* \textit{pNorm});

\textbf{Supported values for} \textit{mod}:

\texttt{8u\_C1MR} \hspace{1cm} \texttt{16u\_C1MR} \hspace{1cm} \texttt{32f\_C1MR}
Case 4: Operation on multi-channel integer data

IppStatus ippiNorm_L2_<mod>(const Ipp<datatype>* pSrc, int srcStep, IppiSize roiSize, Ipp64f value[3]);

Supported values for mod:

8u_C3R    16u_C3R    16s_C3R

IppStatus ippiNorm_L2_<mod>(const Ipp<datatype>* pSrc, int srcStep, IppiSize roiSize, Ipp64f value[4]);

Supported values for mod:

8u_C4R    16u_C4R    16s_C4R

Case 5: Operation on multi-channel floating-point data

IppStatus ippiNorm_L2_32f_C3R(const Ipp32f* pSrc, int srcStep, IppiSize roiSize, Ipp64f value[3], IppHintAlgorithm hint);

IppStatus ippiNorm_L2_32f_C4R(const Ipp32f* pSrc, int srcStep, IppiSize roiSize, Ipp64f value[4], IppHintAlgorithm hint);

Case 6: Masked operation on multi-channel data

IppStatus ippiNorm_L2_<mod>(const Ipp<datatype>* pSrc, int srcStep, const Ipp8u* pMask, int maskStep, IppiSize roiSize, int coi, Ipp64f* pNorm);

Supported values for mod:

8u_C3CMR    16u_C3CMR    32f_C3CMR

Include Files

ippcv.h
ippi.h

Domain Dependencies

Flavors declared in ippcv.h:

Headers: ippcore.h, ippvm.h, ipps.h,ippi.h
Libraries: ippcore.lib, ippvm.lib, ipps.lib,ippi.lib

Flavors declared in ippi.h:

Headers: ippcore.h, ippvm.h, ipps.h
Libraries: ippcore.lib, ippvm.lib, ipps.lib

Parameters

pSrc          Pointer to the source image ROI.
srcStep       Distance in bytes between starts of consecutive lines in the source image.
pMask         Pointer to the mask image.
**Description**

The flavors of the function *ippiNorm_L2* that perform masked operation are declared in the *ippcv.h* file. All other function flavors are declared in the *ippi.h* file. The function operates with ROI (see *Regions of Interest in Intel IPP*). It computes the L2- norm *pValue* (*pNorm* in mask mode) for the source image *pSrc*. This norm is defined as the square root of the sum of squared pixel values in an image. Computation algorithm is specified by the *hint* argument (see Table “Hint Arguments for Image Moment Functions”). In the mask mode, the computation is done over pixels selected by nonzero mask values.

For non-masked operations on a multi-channel image (*Case 4, 5*), the norm is computed separately for each channel and stored in the array *value*.

In the mask multi-channel mode (*Case 6*), the norm is computed for a single channel of interest specified by *coi*.

**Return Values**

- **ippStsNoErr**: Indicates no error. Any other value indicates an error or a warning.
- **ippStsNullPtrErr**: Indicates an error when any of the specified pointers is NULL.
- **ippStsSizeErr**: Indicates an error condition if *roiSize* has a field with zero or negative value.
- **ippStsStepErr**: Indicates an error condition in mask mode, if *srcStep* or *maskStep* is less than *roiSize.width * <pixelSize>.
- **ippStsNotEvenStepErr**: Indicates an error condition in mask mode if steps for floating-point images cannot be divided by 4.
- **ippStsCOIErr**: Indicates an error when *coi* is not 1, 2, or 3.

**NormDiff_Inf**

*Computes the infinity norm of differences between pixel values of two images.*
Syntax

Case 1: Operation on one-channel data

IppStatus ippiNormDiff_Inf_<mod>(const Ipp<datatype>* pSrc1, int src1Step, const Ipp<datatype>* pSrc2, int src2Step, IppiSize roiSize, Ipp64f* pValue);

Supported values for mod:

8u_C1R  16u_C1R  16s_C1R  32f_C1R

Case 2: Masked operation on one-channel data

IppStatus ippiNormDiff_Inf_<mod>(const Ipp<datatype>* pSrc1, int src1Step, const Ipp<datatype>* pSrc2, int src2Step, const Ipp8u* pMask, int maskStep, IppiSize roiSize, Ipp64f* pNorm);

Supported values for mod:

8u_C1MR  16u_C1MR  32f_C1MR

Case 3: Operation on multi-channel data

IppStatus ippiNormDiff_Inf_<mod>(const Ipp<datatype>* pSrc1, int src1Step, const Ipp<datatype>* pSrc2, int src2Step, IppiSize roiSize, Ipp64f value[3]);

Supported values for mod:

8u_C3R  16u_C3R  16s_C3R  32f_C3R

IppStatus ippiNormDiff_Inf_<mod>(const Ipp<datatype>* pSrc1, int src1Step, const Ipp<datatype>* pSrc2, int src2Step, IppiSize roiSize, Ipp64f value[4]);

Supported values for mod:

8u_C4R  16u_C4R  16s_C4R  32f_C4R

Case 4: Masked operation on multi-channel data

IppStatus ippiNormDiff_Inf_<mod>(const Ipp<datatype>* pSrc1, int src1Step, const Ipp<datatype>* pSrc2, int src2Step, const Ipp8u* pMask, int maskStep, IppiSize roiSize, int coi, Ipp64f* pNorm);

Supported values for mod:

8u_C3CMR  16u_C3CMR  32f_C3CMR

Include Files

ippcv.h
ippi.h

Domain Dependencies

Flavors declared in ippcv.h:

Headers: ippcore.h, ippvm.h, ipps.h,ippi.h

Libraries: ippcore.lib, ippvm.lib, ipps.lib,ippi.lib
Flavors declared in ippi.h:

Headers: ippcore.h, ippvm.h, ipps.h

Libraries: ippcore.lib, ippvm.lib, ipps.lib

Parameters

\( pSrc1, pSrc2 \)
Pointers to the source images ROI.

\( src1Step, src2Step \)
Distance in bytes between starts of consecutive lines in the source images.

\( pMask \)
Pointer to the mask image.

\( maskStep \)
Distance in bytes between starts of consecutive lines in the mask image.

\( roiSize \)
Size of the source ROI in pixels.

\( pValue \)
Pointer to the computed infinity norm of difference between pixel values.

\( value \)
An array containing the computed infinity norms of difference between corresponding channel values in case of multi-channel data.

\( coi \)
Channel of interest (for color images only); can be 1, 2, or 3.

\( pNorm \)
Pointer to the computed norm value in the mask mode.

Description

The flavors of the function ippiNormDiff_Inf that perform masked operation are declared in the ippcv.h file. All other function flavors are declared in the ippi.h file. The function operates with ROI (see Regions of Interest in Intel IPP). It computes the infinity norm \( pValue(pNorm) \) in the mask mode) of differences between pixel values of the two source images \( pSrc1 \) and \( pSrc2 \). This norm is defined as the largest absolute value of differences:

\[
\text{norm} = \max |pSrc1-pSrc2|
\]

In the mask mode, the computation is done over pixels selected by nonzero mask values.

For non-masked operations on multi-channel images (Case 3), the norm is computed separately for each pair of corresponding channels and stored in the array \( value \).

In the mask multi-channel mode (Case 4), the norm is computed for a single channel of interest specified by \( coi \).

Return Values

ippStsNoErr
Indicates no error. Any other value indicates an error or a warning.

ippStsNullPtrErr
Indicates an error when any of the specified pointers is NULL.

ippStsSizeErr
Indicates an error condition if \( roiSize \) has a field with zero or negative value.

ippStsStepErr
Indicates an error condition in mask mode, if \( src1Step, src2Step \), or \( maskStep \) is less than \( roiSize.width < \text{pixelSize} \).

ippStsNotEvenStepErr
Indicates an error condition in mask mode if steps for floating-point images cannot be divided by 4.
Indicates an error when coi is not 1, 2, or 3.

**NormDiff_L1**

*Computes the L1- norm of differences between pixel values of two images.*

**Syntax**

**Case 1: Operation on one-channel integer data**

IppStatus ippiNormDiff_L1_<mod>(const Ipp<datatype>* pSrc1, int src1Step, const Ipp<datatype>* pSrc2, int src2Step, IppiSize roiSize, Ipp64f* pValue);

Supported values for mod:

- 8u_C1R
- 16u_C1R
- 16s_C1R

**Case 2: Operation on one-channel floating-point data**

IppStatus ippiNormDiff_L1_32f_C1R(const Ipp32f* pSrc1, int src1Step, const Ipp32f* pSrc2, int src2Step, IppiSize roiSize, Ipp64f* pValue, IppHintAlgorithm hint);

**Case 3: Masked operation on one-channel data**

IppStatus ippiNormDiff_L1_<mod>(const Ipp<datatype>* pSrc1, int src1Step, const Ipp<datatype>* pSrc2, int src2Step, const Ipp8u* pMask, int maskStep, IppiSize roiSize, Ipp64f* pNorm);

Supported values for mod:

- 8u_C1MR
- 16u_C1MR
- 32f_C1MR

**Case 4: Operation on multi-channel integer data**

IppStatus ippiNormDiff_L1_<mod>(const Ipp<datatype>* pSrc1, int src1Step, const Ipp<datatype>* pSrc2, int src2Step, IppiSize roiSize, Ipp64f value[3]);

Supported values for mod:

- 8u_C3R
- 16u_C3R
- 16s_C3R

IppStatus ippiNormDiff_L1_<mod>(const Ipp<datatype>* pSrc1, int src1Step, const Ipp<datatype>* pSrc2, int src2Step, IppiSize roiSize, Ipp64f value[4]);

Supported values for mod:

- 8u_C4R
- 16u_C4R
- 16s_C4R

**Case 5: Operation on multi-channel floating-point data**

IppStatus ippiNormDiff_L1_32f_C3R(const Ipp32f* pSrc1, int src1Step, const Ipp32f* pSrc2, int src2Step, IppiSize roiSize, Ipp64f value[3], IppHintAlgorithm hint);

IppStatus ippiNormDiff_L1_32f_C4R(const Ipp32f* pSrc1, int src1Step, const Ipp32f* pSrc2, int src2Step, IppiSize roiSize, Ipp64f value[4], IppHintAlgorithm hint);
Case 6: Masked operation on multi-channel data

IppStatus ippiNormDiff_L1_mod(const Ipp<datatype>* pSrc1, int src1Step, const Ipp32f* pSrc2, int src2Step, const Ipp8u* pMask, int maskStep, IppiSize roiSize, int coi, Ipp64f* pNorm);

Supported values for mod:

8u_C3CMR   16u_C3CMR   32f_C3CMR

Include Files
ippcv.h
ippi.h

Domain Dependencies
Flavors declared in ippcv.h:
Headers: ippcore.h, ippvm.h, ipps.h,ippi.h
Libraries: ippcore.lib, ippvm.lib, ipps.lib,ippi.lib

Flavors declared in ippi.h:
Headers: ippcore.h, ippvm.h, ipps.h
Libraries: ippcore.lib, ippvm.lib, ipps.lib

Parameters
pSrc1, pSrc2  Pointers to the source images ROI.
src1Step, src2Step  Distance in bytes between starts of consecutive lines in the source images.
pMask  Pointer to the mask image.
maskStep  Distance in bytes between starts of consecutive lines in the mask image.
roiSize  Size of the source ROI in pixels.
coi  Channel of interest (for color images only); can be 1, 2, or 3.
pValue  Pointer to the computed L1- norm of difference between pixel values.
value  An array containing the computed L1- norms of difference between channel values in case of multi-channel data.
pNorm  Pointer to the computed norm value in the mask mode.
hint  Option to select the algorithmic implementation of the function.

Description
The flavors of the function ippiNormDiff_L1 that perform masked operation are declared in the ippcv.h file. All other function flavors are declared in the ippi.h file. The function operates with ROI (see Regions of Interest in Intel IPP). It computes the L1-norm pValue (pNorm in the mask mode) of differences between pixel values of the two source image buffers pSrc1 and pSrc2. This norm is defined as the sum of absolute values of differences:
norm = Σ |pSrc1-pSrc2|

Image Statistics Functions
Computation algorithm is specified by the hint argument (see Table "Hint Arguments for Image Moment Functions"). In the mask mode, the computation is done over pixels selected by nonzero mask values.

For non-masked operations on multi-channel images (Case 4,5), the norm is computed separately for each pair of the corresponding channels and stored in the array value.

In the mask multi-channel mode (Case 6), the norm is computed for a single channel of interest specified by coi.

**Return Values**

- **ippStsNoErr**: Indicates no error. Any other value indicates an error or a warning.
- **ippStsNullPtrErr**: Indicates an error when any of the specified pointers is NULL.
- **ippStsSizeErr**: Indicates an error condition if roiSize has a field with zero or negative value.
- **ippStsStepErr**: Indicates an error condition in mask mode, if src1Step, src2Step, or maskStep is less than roiSize.width * pixelSize.
- **ippStsNotEvenStepErr**: Indicates an error condition in mask mode if steps for floating-point images cannot be divided by 4.
- **ippStsCOIErr**: Indicates an error when coi is not 1, 2, or 3.

**Example**

The code example below shows how to use the function ippiNormDiff_L1_8u_C1R.

```c
void func_normdiff_l1()
{
    Ipp8u pSrc1[8*4];
    Ipp8u pSrc2[8*4];
    Ipp64f Value;
    int src1Step = 8;
    int src2Step = 8;
    IppiSize roi = {8,4};
    IppiSize roiSize = {5,4};

    ippiSet_8u_C1R(1, pSrc1, src1Step, roi);
    ippiSet_8u_C1R(2, pSrc2, src2Step, roi);

    ippiNormDiff_L1_8u_C1R( pSrc1, src1Step, pSrc2, src2Step, roiSize, &Value);
}

Result -> 20.0
```
**NormDiff_L2**

*Computes the L2- norm of differences between pixel values of two images.*

**Syntax**

**Case 1: Operation on one-channel integer data**

IppStatus ippiNormDiff_L2_<mod>(const Ipp<datatype>* pSrc1, int src1Step, const Ipp<datatype>* pSrc2, int src2Step, IppiSize roiSize, Ipp64f* pValue);

**Supported values for** mod:

- 8u_C1R
- 16u_C1R
- 16s_C1R

**Case 2: Operation on one-channel floating-point data**

IppStatus ippiNormDiff_L2_32f_C1R(const Ipp32f* pSrc1, int src1Step, const Ipp32f* pSrc2, int src2Step, IppiSize roiSize, Ipp64f* pValue, IppHintAlgorithm hint);

**Case 3: Masked operation on one-channel data**

IppStatus ippiNormDiff_L2_<mod>(const Ipp<datatype>* pSrc1, int src1Step, const Ipp<datatype>* pSrc2, int src2Step, const Ipp8u* pMask, int maskStep, IppiSize roiSize, Ipp64f* pNorm);

**Supported values for** mod:

- 8u_C1MR
- 16u_C1MR
- 32f_C1MR

**Case 4: Operation on multi-channel integer data**

IppStatus ippiNormDiff_L2_<mod>(const Ipp<datatype>* pSrc1, int src1Step, const Ipp<datatype>* pSrc2, int src2Step, IppiSize roiSize, Ipp64f value[3]);

**Supported values for** mod:

- 8u_C3R
- 16u_C3R
- 16s_C3R

IppStatus ippiNormDiff_L2_<mod>(const Ipp<datatype>* pSrc1, int src1Step, const Ipp<datatype>* pSrc2, int src2Step, IppiSize roiSize, Ipp64f value[4]);

**Supported values for** mod:

- 8u_C4R
- 16u_C4R
- 16s_C4R

**Case 5: Operation on multi-channel floating-point data**

IppStatus ippiNormDiff_L2_32f_C3R(const Ipp32f* pSrc1, int src1Step, const Ipp<datatype>* pSrc2, int src2Step, IppiSize roiSize, Ipp64f value[3], IppHintAlgorithm hint);

IppStatus ippiNormDiff_L2_32f_C4R(const Ipp32f* pSrc1, int src1Step, const Ipp<datatype>* pSrc2, int src2Step, IppiSize roiSize, Ipp64f value[4], IppHintAlgorithm hint);
Case 6: Masked operation on multi-channel data

IppStatus ippiNormDiff_L2_<mod>(const Ipp<datatype>* pSrc1, int src1Step, const Ipp<datatype>* pSrc2, int src2Step, const Ipp8u* pMask, int maskStep, IppiSize roiSize, int coi, Ipp64f* pNorm);

Supported values for mod:

8u_C3CMR  16u_C3CMR  32f_C3CMR

Include Files

ippcv.h
ippi.h

Domain Dependencies

Flavors declared in ippcv.h:

Headers: ippcore.h, ippvm.h, ipps.h,ippi.h
Libraries: ippcore.lib, ippvm.lib, ipps.lib,ippi.lib

Flavors declared in ippi.h:

Headers: ippcore.h, ippvm.h, ipps.h
Libraries: ippcore.lib, ippvm.lib, ipps.lib

Parameters

pSrc1, pSrc2  Pointers to the source images ROI.
src1Step, src2Step  Distance in bytes between starts of consecutive lines in the source images.
pMask  Pointer to the mask image.
maskStep  Distance in bytes between starts of consecutive lines in the mask image.
roiSize  Size of the source ROI in pixels.
coi  Channel of interest (for color images only); can be 1, 2, or 3.
pValue  Pointer to the computed L2- norm of difference between pixel values.
value  An array containing the computed L2- norms of difference between channel values in case of multi-channel data.
pNorm  Pointer to the computed norm value in the mask mode.
hint  Option to select the algorithmic implementation of the function.

Description

The flavors of the function ippiNormDiff_L2 that perform masked operation are declared in the ippcv.h file. All other function flavors are declared in the ippi.h file. The function operates with ROI (see Regions of Interest in Intel IPP). It computes the L2-norm pValue (pNorm in the mask mode) of differences between pixel values of the two source image buffers pSrc1 and pSrc2. This norm is defined as the square root of the sum of squared differences:

\[ \text{norm} = \sqrt{\sum_{\text{pixels}} (p_{\text{Src1}} - p_{\text{Src2}})^2} \]
Computation algorithm is specified by the hint argument (see Table “Hint Arguments for Image Moment Functions”). In the mask mode, the computation is done over pixels selected by nonzero mask values.

For non-masked operations on multi-channel images (Case 4,5), the norm is computed separately for each pair of the corresponding channels and stored in the array value.

In the mask multi-channel mode (Case 6), the norm is computed for a single channel of interest specified by coi.

**Return Values**

- **ippStsNoErr**: Indicates no error. Any other value indicates an error or a warning.
- **ippStsNullPtrErr**: Indicates an error when any of the specified pointers is NULL.
- **ippStsSizeErr**: Indicates an error condition if roiSize has a field with zero or negative value.
- **ippStsStepErr**: Indicates an error condition in mask mode, if src1Step, src2Step, or maskStep is less than roiSize.width * pixelSize.
- **ippStsNotEvenStepErr**: Indicates an error condition in mask mode if steps for floating-point images cannot be divided by 4.
- **ippStsCOIErr**: Indicates an error when coi is not 1, 2, or 3.

**NormRel_Inf**

*Computes the relative error for the infinity norm of differences between pixel values of two images.*

**Syntax**

**Case 1: Operation on one-channel data**

```c
IppStatusippiNormRel_Inf_<mod>(const Ipp<datatype>* pSrc1, int src1Step, const Ipp<datatype>* pSrc2, int src2Step, IppSize roiSize, Ipp64f* pValue);
```

Supported values for mod:

- 8u_C1R
- 16u_C1R
- 16s_C1R
- 32f_C1R

**Case 2: Masked operation on one-channel data**

```c
IppStatusippiNormRel_Inf_<mod>(const Ipp<datatype>* pSrc1, int src1Step, const Ipp<datatype>* pSrc2, int src2Step, const Ipp8u* pMask, int maskStep, IppSize roiSize, Ipp64f* pNorm);
```

Supported values for mod:

- 8u_C1MR
- 16u_C1MR
- 32f_C1MR
Case 3: Operation on multi-channel data

IppStatusippiNormRel_Inf_<mod>(const Ipp<datatype>* pSrc1, int src1Step, const Ipp<datatype>* pSrc2, int src2Step, IppiSize roiSize, Ipp64f value[3]);

Supported values for mod:

8u_C3R  16u_C3R  16s_C3R  32f_C3R

IppStatusippiNormRel_Inf_<mod>(const Ipp<datatype>* pSrc1, int src1Step, const Ipp<datatype>* pSrc2, int src2Step, IppiSize roiSize, Ipp64f value[4]);

Supported values for mod:

8u_C4R  16u_C4R  16s_C4R  32f_C4R

Case 4: Masked operation on multi-channel data

IppStatusippiNormRel_Inf_<mod>(const Ipp<datatype>* pSrc1, int src1Step, const Ipp<datatype>* pSrc2, int src2Step, const Ipp8u* pMask, int maskStep, IppiSize roiSize, int coi, Ipp64f* pNorm);

Supported values for mod:

8u_C3CMR  16u_C3CMR  32f_C3CMR

Include Files

ippcv.h
ippi.h

Domain Dependencies

Flavors declared in ippcv.h:

Headers: ippcore.h, ippvm.h, ipps.h, ippi.h
Libraries: ippcore.lib, ippvm.lib, ipps.lib, ippi.lib

Flavors declared in ippi.h:

Headers: ippcore.h, ippvm.h, ipps.h
Libraries: ippcore.lib, ippvm.lib, ipps.lib

Parameters

pSrc1, pSrc2  Pointers to the source images ROI.
src1Step, src2Step  Distance in bytes between starts of consecutive lines in the source images.
pMask  Pointer to the mask image.
maskStep  Distance in bytes between starts of consecutive lines in the mask image.
roiSize  Size of the source ROI in pixels.
pValue  Pointer to the computed relative error value.
value

An array containing the computed relative error values for separate channels in case of multi-channel data.

coi

Channel of interest (for color images only); can be 1, 2, or 3.

pNorm

Pointer to the computed relative norm value in the mask mode.

Description

The flavors of the function `ippiNormRel_Inf` that perform masked operation are declared in the `ippcv.h` file. All other function flavors are declared in the `ippi.h` file. The function operates with ROI (see Regions of Interest in Intel IPP). It computes the infinity norm of differences between pixel values of two source buffers `pSrc1` and `pSrc2`. This norm is defined as the largest absolute pixel value in an image. The output relative error `pValue` (`pNorm` in the mask mode) is then formed by dividing the computed norm of differences by the infinity norm of the second source image buffer `pSrc2`. In the mask mode, the computation is done over pixels selected by nonzero mask values.

For non-masked operations on multi-channel images (Case 3), the relative norm is computed separately for each pair of corresponding channels and stored in the array `value`.

In the mask multi-channel mode (Case 4), the relative norm is computed for a single channel of interest specified by `coi`.

Return Values

<table>
<thead>
<tr>
<th>Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ippStsNoErr</td>
<td>Indicates no error. Any other value indicates an error or a warning.</td>
</tr>
<tr>
<td>ippStsNullPtrErr</td>
<td>Indicates an error when any of the specified pointers is NULL.</td>
</tr>
<tr>
<td>ippStsSizeErr</td>
<td>Indicates an error condition if <code>roiSize</code> has a field with zero or negative value.</td>
</tr>
<tr>
<td>ippStsStepErr</td>
<td>Indicates an error condition in mask mode, if <code>src1Step</code>, <code>src2Step</code>, or <code>maskStep</code> is less than <code>roiSize.width * &lt;pixelSize&gt;</code>.</td>
</tr>
<tr>
<td>ippStsNotEvenStepErr</td>
<td>Indicates an error condition in mask mode if steps for floating-point images cannot be divided by 4.</td>
</tr>
<tr>
<td>ippStsCOIErr</td>
<td>Indicates an error when <code>coi</code> is not 1, 2, or 3.</td>
</tr>
<tr>
<td>ippStsDivByZero</td>
<td>Indicates a warning when the infinity norm of <code>pSrc2</code> has a zero value.</td>
</tr>
</tbody>
</table>

**NormRel_L1**

*Computes the relative error for the L1 norm of differences between pixel values of two images.*

**Syntax**

**Case 1: Operation on one-channel integer data**

```c
IppStatus ippiNormRel_L1_<mod>(const Ipp<datatype>* pSrc1, int src1Step, const Ipp<datatype>* pSrc2, int src2Step, IppiSize roiSize, Ipp64f* pValue);
```

Supported values for `mod`:

<table>
<thead>
<tr>
<th>Mod</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>8u_C1R</td>
<td>16u_C1R</td>
</tr>
<tr>
<td></td>
<td>16s_C1R</td>
</tr>
</tbody>
</table>
Case 2: Operation on one-channel floating-point data

IppStatus ippiNormRel_L1_32f_C1R(const Ipp32f* pSrc1, int src1Step, const Ipp<datatype>* pSrc2, int src2Step, IppSize roiSize, Ipp64f* pValue, IppHintAlgorithm hint);

Case 3: Masked operation on one-channel data

IppStatus ippiNormRel_L1_<mod>(const Ipp<datatype>* pSrc1, int src1Step, const Ipp<datatype>* pSrc2, int src2Step, const Ipp8u* pMask, int maskStep, IppSize roiSize, Ipp64f* pNorm);

Supported values for mod:

8u_C1MR 16u_C1MR 32f_C1MR

Case 4: Operation on multi-channel integer data

IppStatus ippiNormRel_L1_<mod>(const Ipp<datatype>* pSrc1, int src1Step, const Ipp<datatype>* pSrc2, int src2Step, IppSize roiSize, Ipp64f value[3]);

Supported values for mod:

8u_C3R 16u_C3R 16s_C3R

IppStatus ippiNormRel_L1_<mod>(const Ipp<datatype>* pSrc1, int src1Step, const Ipp<datatype>* pSrc2, int src2Step, IppSize roiSize, Ipp64f value[4]);

Supported values for mod:

8u_C4R 16u_C4R 16s_C4R

Case 5: Operation on multi-channel floating-point data

IppStatus ippiNormRel_L1_32f_C3R(const Ipp32f* pSrc1, int src1Step, const Ipp<datatype>* pSrc2, int src2Step, IppSize roiSize, Ipp64f value[3], IppHintAlgorithm hint);

IppStatus ippiNormRel_L1_32f_C4R(const Ipp32f* pSrc1, int src1Step, const Ipp<datatype>* pSrc2, int src2Step, IppSize roiSize, Ipp64f value[4], IppHintAlgorithm hint);

Case 6: Masked operation on multi-channel data

IppStatus ippiNormRel_L1_<mod>(const Ipp<datatype>* pSrc1, int src1Step, const Ipp<datatype>* pSrc2, int src2Step, const Ipp8u* pMask, int maskStep, IppSize roiSize, int coi, Ipp64f* pNorm);

Supported values for mod:

8u_C3CMR 16u_C3CMR 32f_C3CMR

Include Files

ippcv.h
ippi.h

Domain Dependencies

Flavors declared in ippcv.h:
Parameters

- **pSrc1, pSrc2**
  Pointers to the source images ROI.

- **src1Step, src2Step**
  Distance in bytes between starts of consecutive lines in the source images.

- **pMask**
  Pointer to the mask image.

- **maskStep**
  Distance in bytes between starts of consecutive lines in the mask image.

- **roiSize**
  Size of the source ROI in pixels.

- **coi**
  Channel of interest (for color images only); can be 1, 2, or 3.

- **pValue**
  Pointer to the computed relative error value.

- **value**
  An array containing the computed relative error values for separate channels in case of multi-channel data.

- **pNorm**
  Pointer to the computed relative norm value in the mask mode.

- **hint**
  Option to select the algorithmic implementation of the function.

Description

The flavors of the function `ippiNormRel_L1` that perform masked operation are declared in the `ippcv.h` file. All other function flavors are declared in the `ippi.h` file. The function operates with ROI (see Regions of Interest in Intel IPP). It computes the L1- norm of differences between pixel values of two source buffers `pSrc1` and `pSrc2`. This norm is defined as the sum of absolute pixel values in an image. The output relative error `pValue` (`pNorm` in the mask mode) is then formed by dividing the computed norm of differences by the L1- norm of the second source image buffer `pSrc2`. Computation algorithm is specified by the `hint` argument (see Table "Hint Arguments for Image Moment Functions"). In the mask mode, the computation is done over pixels selected by nonzero mask values.

For non-masked operations on multi-channel images (Cases 4, 5), the relative norm is computed separately for each pair of corresponding channels and stored in the array `value`.

In the mask multi-channel mode (Case 6), the relative norm is computed for a single channel of interest specified by `coi`.

Return Values

- **ippStsNoErr**
  Indicates no error. Any other value indicates an error or a warning.

- **ippStsNullPtrErr**
  Indicates an error when any of the specified pointers is NULL.

- **ippStsSizeErr**
  Indicates an error condition if `roiSize` has a field with zero or negative value.

- **ippStsStepErr**
  Indicates an error condition in mask mode, if `src1Step`, `src2Step`, or `maskStep` is less than `roiSize.width * <pixelSize>`.
ippStsNotEvenStepErr Indicates an error condition in mask mode if steps for floating-point images cannot be divided by 4.
ippStsCOIErr Indicates an error when coi is not 1, 2, or 3.
ippStsDivByZero Indicates a warning when the L1 norm of pSrc2 has a zero value.

Example

The code example below shows how to use the function ippiNormRel_L1_8u_C1R.

```c
void func_normrel_l1()
{
    Ipp8u pSrc1[8*4];
    Ipp8u pSrc2[8*4];
    Ipp64f Value;
    int src1Step = 8;
    int src2Step = 8;
    IppiSize roi = (8,4);
    IppiSize roiSize = (5,4);

    ippiSet_8u_C1R(1, pSrc1, src1Step, roi);
    ippiSet_8u_C1R(2, pSrc2, src2Step, roi);
    ippiNormRel_L1_8u_C1R( pSrc1, src1Step, pSrc2, src2Step, roiSize, &Value);

    Result -> 0.5
}
```

NormRel_L2

_Computes the relative error for the L2 norm of differences between pixel values of two images._

Syntax

Case 1: Operation on one-channel integer data

_IppStatus ippiNormRel_L2_<mod>(const Ipp<datatype>* pSrc1, int src1Step, const Ipp<datatype>* pSrc2, int src2Step, IppiSize roiSize, Ipp64f* pValue);_

Supported values for mod:

- 8u_C1R
- 16u_C1R
- 16s_C1R
Case 2: Operation on one-channel floating-point data

IppStatus ippiNormRel_L2_32f_C1R(const Ipp32f* pSrc1, int src1Step, const Ipp<datatype>* pSrc2, int src2Step, IppiSize roiSize, Ipp64f* pValue, IppHintAlgorithm hint);

Case 3: Masked operation on one-channel data

IppStatus ippiNormRel_L2_<mod>(const Ipp<datatype>* pSrc1, int src1Step, const Ipp<datatype>* pSrc2, int src2Step, const Ipp8u* pMask, int maskStep, IppiSize roiSize, Ipp64f* pNorm);

Supported values for mod:

    8u_C1MR     16u_C1MR     32f_C1MR

Case 4: Operation on multi-channel integer data

IppStatus ippiNormRel_L2_<mod>(const Ipp<datatype>* pSrc1, int src1Step, const Ipp<datatype>* pSrc2, int src2Step, IppiSize roiSize, Ipp64f value[3]);

Supported values for mod:

    8u_C3R     16u_C3R     16s_C3R

IppStatus ippiNormRel_L2_<mod>(const Ipp<datatype>* pSrc1, int src1Step, const Ipp<datatype>* pSrc2, int src2Step, IppiSize roiSize, Ipp64f value[4]);

Supported values for mod:

    8u_C4R     16u_C4R     16s_C4R

Case 5: Operation on multi-channel floating-point data

IppStatus ippiNormRel_L2_32f_C3R(const Ipp32f* pSrc1, int src1Step, const Ipp<datatype>* pSrc2, int src2Step, IppiSize roiSize, Ipp64f value[3], IppHintAlgorithm hint);

IppStatus ippiNormRel_L2_32f_C4R(const Ipp32f* pSrc1, int src1Step, const Ipp<datatype>* pSrc2, int src2Step, IppiSize roiSize, Ipp64f value[4], IppHintAlgorithm hint);

Case 6: Masked operation on multi-channel data

IppStatus ippiNormRel_L2_<mod>(const Ipp<datatype>* pSrc1, int src1Step, const Ipp<datatype>* pSrc2, int src2Step, const Ipp8u* pMask, int maskStep, IppiSize roiSize, int coi, Ipp64f* pNorm);

Supported values for mod:

    8u_C3CMR    16u_C3CMR    32f_C3CMR

Include Files

ippcv.h
ippi.h

Domain Dependencies

Flavors declared in ippcv.h:
Parameters

- **pSrc1, pSrc2**: Pointers to the source images ROI.
- **src1Step, src2Step**: Distance in bytes between starts of consecutive lines in the source images.
- **pMask**: Pointer to the mask image.
- **maskStep**: Distance in bytes between starts of consecutive lines in the mask image.
- **roiSize**: Size of the source ROI in pixels.
- **coi**: Channel of interest (for color images only); can be 1, 2, or 3.
- **pValue**: Pointer to the computed relative error value.
- **value**: An array containing the computed relative error values for separate channels in case of multi-channel data.
- **pNorm**: Pointer to the computed relative norm value in the mask mode.
- **hint**: Option to select the algorithmic implementation of the function.

Description

The flavors of the function `ippiNormRel_L2` that perform masked operation are declared in the `ippcv.h` file. All other function flavors are declared in the `ippi.h` file. The function operates with ROI (see Regions of Interest in Intel IPP). It computes the L2-norm of differences between pixel values of two source buffers `pSrc1` and `pSrc2`. This norm is defined as the square root of the sum of squared pixel values in an image. The output relative error `pValue` (`pNorm` in the mask mode) is then formed by dividing the computed norm of differences by the L2-norm of the second source image buffer `pSrc2`. Computation algorithm is specified by the **hint** argument (see Table “Hint Arguments for Image Moment Functions”). In the mask mode, the computation is done over pixels selected by nonzero mask values.

For non-masked operations on multi-channel images (Cases 4, 5), the relative norm is computed separately for each pair of corresponding channels and stored in the array `value`.

In the mask multi-channel mode (Case 6), the relative norm is computed for a single channel of interest specified by `coi`.

Return Values

- **ippStsNoErr**: Indicates no error. Any other value indicates an error or a warning.
- **ippStsNullPtrErr**: Indicates an error when any of the specified pointers is NULL.
- **ippStsSizeErr**: Indicates an error condition if `roiSize` has a field with zero or negative value.
- **ippStsStepErr**: Indicates an error condition in mask mode, if `src1Step`, `src2Step`, or `maskStep` is less than `roiSize.width * <pixelSize>`.
Indicates an error condition in mask mode if steps for floating-point images cannot be divided by 4.
Indicates an error when coi is not 1, 2, or 3.
Indicates a warning when the L2 norm of pSrc2 has a zero value.

Example

The code example below shows how to use the functionippiNormRel_L2_8u_C1R.

```c
void func_normrel_l1()
{
    Ipp8u pSrc1[8*4];
    Ipp8u pSrc2[8*4];
    Ipp64f Value;
    int src1Step = 8;
    int src2Step = 8;
    IppiSize roi = {8,4};
    IppiSize roiSize = {5,4};

   ippiSet_8u_C1R(1, pSrc1, src1Step, roi);
   ippiSet_8u_C1R(10, pSrc2, src2Step, roi);
   ippiNormRel_L2_8u_C1R( pSrc1, src1Step, pSrc2, src2Step, roiSize, &Value);
}
Result -> 0.9
```

Image Quality Index

Intel IPP functions described in this section compute the universal image quality index [Wang02] that may be used as image and video quality distortion measure. It is mathematically defined by modeling the image distortion relative to the reference image as a combination of three factors: loss of correlation, luminance distortion, and contrast distortion.

If two images $f$ and $g$ are considered as a matrices with $M$ column and $N$ rows containing pixel values $f[i,j]$, $g[i,j]$, respectively ($0 \geq i > M$, $0 \geq j > N$ ), the universal image quality index $Q$ may be calculated as a product of three components:

$$Q = \frac{\sigma_f \sigma_g}{\sigma_f^2 + (\bar{f})^2 + (\bar{g})^2} \cdot \frac{2\bar{f}\bar{g}}{\sigma_f \sigma_g} \cdot \frac{2\sigma_f^2 \sigma_g^2}{2 \sigma_f^2 + 2 \sigma_g^2}$$

where
The first component is the correlation coefficient, which measures the degree of linear correlation between images $f$ and $g$. It varies in the range $[-1, 1]$. The best value 1 is obtained when $f$ and $g$ are linearly related, which means that $g[i, j] = af[i, j] + b$ for all possible values of $i$ and $j$. The second component, with a range of $[0, 1]$, measures how close the mean luminance is between images. Since $\sigma_f$ and $\sigma_g$ can be considered as estimates of the contrast of $f$ and $g$, the third component measures how similar the contrasts of the images are. The value range for this component is also $[0, 1]$. The range of values for the index $\varrho$ is $[-1, 1]$. The best value 1 is achieved if and only if the images are identical.

**QualityIndexGetBufferSize**

*Computes the size of the work buffer for the ippiQualityIndex function.*

**Syntax**

```c
IppStatus ippiQualityIndexGetBufferSize(IppDataType srcType, IppChannels ippChan, IppiSize roiSize, int* pBufferSize);
```

**Include Files**

ippi.h

**Domain Dependencies**

Headers: ippcore.h, ippvm.h, ipps.h

Libraries: ippcore.lib, ippvm.lib, ipps.lib

**Parameters**

- `srcType`
  - Data type of the source images. Possible values: ipp8u, ipp16u, or ipp32f.

- `ippChan`
  - Number of channels in the source images. Possible values: ippC1, ippC3, or ippAC4.

- `roiSize`
  - Size, in pixels, of the source images.

- `pBufferSize`
  - Pointer to the computed value of the buffer size, in bytes.

**Description**

The function computes the size of the work buffer, in bytes, for the ippiQualityIndex function and stores the result in the pBufferSize parameter.
Return Values

ippStsNoErr  Indicates no error. Any other value indicates an error or a warning.
ippStsNullPtrErr  Indicates an error when any of the specified pointers is NULL.
ippStsSizeErr  Indicates an error when roiSize is less than, or equal to zero.
ippStsDataTypeErr  Indicates an error when srcType has an illegal value.
ippStsChannelErr  Indicates an error when ippChan has an illegal value.

See Also
QualityIndex Computes the universal image quality index.

QualityIndex
Computes the universal image quality index.

Syntax

Case 1: Operation on one-channel data
IppStatus ippiQualityIndex_<mod>({const Ipp<srcDatatype>* pSrc1, int src1Step, const Ipp<srcDatatype>* pSrc2, int src2Step, IppiSize roiSize, Ipp<dstDatatype> pQualityIndex[1], Ipp8u* pBuffer);

Supported values for mod:
8u32f_C1R   16u32f_C1R   32f_C1R

Case 2: Operation on multi-channel data
IppStatus ippiQualityIndex_<mod>({const Ipp<srcDatatype>* pSrc1, int src1Step, const Ipp<srcDatatype>* pSrc2, int src2Step, IppiSize roiSize, Ipp<dstDatatype> pQualityIndex[3], Ipp8u* pBuffer);

Supported values for mod:
8u32f_C3R   16u32f_C3R   32f_C3R
8u32f_AC4R   16u32f_AC4R   32f_AC4R

Include Files
ippi.h

Domain Dependencies
Headers: ippcore.h, ippvm.h, ipps.h
Libraries: ippcore.lib, ippvm.lib, ipps.lib

Parameters
pSrc1, pSrc2   Pointers to the source images ROI.
src1Step, src2Step   Distance in bytes between starts of consecutive lines in the source images.
**roiSize**
Size of the source ROI in pixels.

**pQualityIndex**
Pointer to the computed quality index value.

**pBuffer**
Pointer to the buffer for internal calculations. To compute the size of the buffer, use the `QualityIndexGetBufferSize` function.

**Description**
This function operates with ROI (see Regions of Interest in Intel IPP). This function computes the universal image quality index for two images `pSrc1` and `pSrc2` according to the formula in the introduction section above. The computed value of the index is stored in `pQualityIndex`.

**Return Values**
- **ippStsNoErr**: Indicates no error. Any other value indicates an error or a warning.
- **ippStsNullPtrErr**: Indicates an error when any of the specified pointers is NULL.
- **ippStsSizeErr**: Indicates an error condition if `roiSize` has a field with zero or negative value.
- **ippStsStepErr**: Indicates an error condition if `src1Step` or `src2Step` has a zero or negative value.
- **ippStsQualityIndexErr**: Indicates an error condition if pixel values of one of the images are constant.
- **ippStsMemAllocErr**: Indicates an error condition if memory allocation fails.

**Example**
To better understand usage of this function, refer to the following example in the examples archive available for download from https://software.intel.com/en-us/ipp-manual-examples:

QualityIndex.c

Result:

- **image1** (reference)
- **image2**

Universal image quality index of the distorted image2 -> `pQIndex = 0.6`
See Also

QualityIndexGetBufferSize Computes the size of the work buffer for the ippiQualityIndex function.

Image Proximity Measures

The functions described in this section compute the proximity (similarity) measure between an image and a template (another image). These functions may be used as feature detection functions, as well as the components of more sophisticated techniques.

There are several ways to compute the measure of similarity between two images. One way is to compute the Euclidean distance, or sum of the squared distances (SSD), of an image and a template. The smaller is the value of SSD at a particular pixel, the more similarity exists between the template and the image in the neighborhood of that pixel.

The squared Euclidean distance \( S_{_{TX}(r,c)} \) between a template and an image for the pixel in row \( r \) and column \( c \) is given by the equation:

\[
S_{_{TX}(r,c)} = \sum_{j=0}^{tplRows-1} \sum_{i=0}^{tplCols-1} \left[ t(j,i) - x\left( r + j - \frac{tplRows}{2}, \ c + i - \frac{tplCols}{2} \right) \right]^2
\]

where \( x(r,c) \) is the image pixel value in row \( r \) and column \( c \), and \( t(j,i) \) is the template pixel value in row \( j \) and column \( i \); template size is \( tplCols \) by \( tplRows \) and its center is positioned at \((r,c)\).

The other similarity measure is the cross-correlation function: the higher is the cross-correlation at a particular pixel, the more similarity exists between the template and the image in the neighborhood of that pixel.

The cross-correlation \( R_{_{TX}(r,c)} \) between a template and an image at the pixel in row \( r \) and column \( c \) is computed by the equation:

\[
R_{_{TX}(r,c)} = \sum_{j=0}^{tplRows-1} \sum_{i=0}^{tplCols-1} t(j,i) \cdot x\left( r + j - \frac{tplRows}{2}, \ c + i - \frac{tplCols}{2} \right)
\]

The cross-correlation function is dependent on the brightness variation across the image. To avoid this dependence, the correlation coefficient function is used instead. It is defined as:

\[
G_{_{TX}(r,c)} = \sum_{j=0}^{tplRows-1} \sum_{i=0}^{tplCols-1} \left[ t(j,i) - \bar{t} \right] \left[ x\left( r + j - \frac{tplRows}{2}, \ c + i - \frac{tplCols}{2} \right) - \bar{x} \right]
\]

where \( \bar{t} \) with the overline is the mean of the template, and \( \bar{x} \) with the overline is the mean of the image in the region just under the template.

All Intel IPP proximity functions compute normalized values of SSD, cross-correlation and correlation coefficient that are defined as follows:

normalized SSD: \( \sigma_{_{TX}(r,c)} \):

\[
\sigma_{_{TX}(r,c)} = \frac{S_{_{TX}(r,c)}}{\sqrt{R_{_{TX}(r,c)} R_{_{TT}(tplRows,tplCols)}/4}}
\]

normalized cross-correlation \( \rho_{_{TX}(r,c)} \):

\[
\rho_{_{TX}(r,c)} = \frac{R_{_{TX}(r,c)}}{\sqrt{R_{_{XX}(r,c)} R_{_{TT}(tplRows,tplCols)}/4}}
\]

719
Here $R_{xx}$ and $R_{tt}$ denote the autocorrelation of the image and the template, respectively:

$$R_{xx}(r, c) = \sum_{j=-r+\text{tplRows}-1}^{r-\text{tplRows}-1} \sum_{i=-c+\text{tplCols}-1}^{c-\text{tplCols}-1} x_{ij} x_{ij}$$

$$R_{tt}(\frac{\text{tplRows}}{2}, \frac{\text{tplCols}}{2}) = \sum_{j=0}^{\text{tplRows}-1} \sum_{i=0}^{\text{tplCols}-1} t_{ij} t_{ij}$$

Normalized correlation coefficient $\gamma_{tx}(r, c)$:

$$\gamma_{tx}(r, c) = \frac{G_{xx}(r, c)}{\sqrt{G_{xx}(r, c)G_{tt}(\frac{\text{tplRows}}{2}, \frac{\text{tplCols}}{2})}}$$

Here $G_{xx}$ and $G_{tt}$ denote the autocorrelations of the image and the template without constant brightness component, respectively:

$$G_{xx}(r, c) = \sum_{j=-r+\text{tplRows}-1}^{r-\text{tplRows}-1} \sum_{i=-c+\text{tplCols}-1}^{c-\text{tplCols}-1} [x_{ij} - x(r, c)]^2$$

$$G_{tt}(\frac{\text{tplRows}}{2}, \frac{\text{tplCols}}{2}) = \sum_{j=0}^{\text{tplRows}-1} \sum_{i=0}^{\text{tplCols}-1} (t_{ij} - \bar{t})^2$$

**SqrDistanceNormGetBufferSize**

*Computes the size of the work buffer for theippiSqrDistanceNorm function.*

**Syntax**

IppStatus ippSqrDistanceNormGetBufferSize (IppSize srcRoiSize, IppSize tplRoiSize, IppEnum algType, int* pBufferSize);

**Include Files**

ippi.h

**Parameters**

* srcRoiSize, tplRoiSize
  * Size of the source/template ROI in pixels.
* algType
  * Bit-field mask for the algorithm type definition. Possible values are the results of composition of the IppAlgType, IppROIShape, and IppiNormOp values.
* pBufferSize
  * Pointer to the size of the work buffer.
Description
The `ippiSqrDistanceNormGetSizeBufferSize` function computes the size, in bytes, of the external work buffer needed for the function that computes the Euclidean distance between an image and a template. The result is stored in the `pBufferSize` parameter.

Return Values
- `ippStsNoErr`: Indicates no error.
- `ippStsSizeErr`: Indicates an error when:
  - `srcRoiSize` or `tplRoiSize` is negative, or equal to zero
  - the value of `srcRoiSize` is less than the corresponding value of `tplRoiSize`
- `ippStsAlgTypeErr`: Indicates an error when:
  - the result of the bitwise AND operation between the `algType` and `ippAlgMask` values differs from the `ippAlgAuto`, `ippAlgDirect`, or `ippAlgFFT` values.
  - the result of the bitwise AND operation between the `algType` and `ippiROIMask` values differs from the `ippiROIFull`, `ippiROISame`, or `ippiROIValid` values.
  - the result of the bitwise AND operation between the `algType` and `ippiNormMask` values differs from the `ippiNormNone` or `ippiNorm` values.
- `ippStsNullErr`: Indicates an error when the `pBufferSize` is NULL.

See Also
Structures and Enumerators
`SqrDistanceNorm` Computes Euclidean distance between an image and a template.

SqrDistanceNorm
Computes Euclidean distance between an image and a template.

Syntax
Case 1: Operating with integer output
```c
IppStatus ippiSqrDistanceNorm_8u_C1RSfs(const Ipp8u* pSrc, int srcStep, IppiSize srcRoiSize, const Ipp8u* pTpl, int tplStep, IppiSize tplRoiSize, Ipp8u* pDst, int dstStep, int scaleFactor, IppEnum algType, Ipp8u* pBuffer);
```
Case 2: Operating on data with floating-point output
```c
IppStatus ippiSqrDistanceNorm_<mod>(const Ipp<srcDatatype>* pSrc, int srcStep, IppiSize srcRoiSize, const Ipp<srcDatatype>* pTpl, int tplStep, IppiSize tplRoiSize, Ipp32f* pDst, int dstStep, IppEnum algType, Ipp8u* pBuffer);
```

Supported values for `mod`:
- `32f_C1R`
- `8u32f_C1R`
- `16u32f_C1R`

Include Files
`ippi.h`
Domain Dependencies

Headers: ippcore.h, ippvm.h, ipps.h
Libraries: ippcore.lib, ippvm.lib, ipps.lib

Parameters

- **pSrc**: Pointer to the source image ROI.
- **srcStep**: Distance, in bytes, between the starting points of consecutive lines in the source image.
- **srcRoiSize**: Size of the source ROI in pixels.
- **pTpl**: Pointer to the template image.
- **tplStep**: Distance, in bytes, between the starting points of consecutive lines in the template image.
- **tplRoiSize**: Size of the template ROI in pixels.
- **pDst**: Pointer to the destination image ROI.
- **dstStep**: Distance, in bytes, between the starting points of consecutive lines in the destination image.
- **scaleFactor**: Scale factor.
- **algType**: Bit-field mask for the algorithm type definition. Possible values are the results of composition of the IppAlgType, IppiROIShape, and IppiNormOp values.
- **pBuffer**: Pointer to the work buffer.

Description

Before using this function, you need to compute the size of the work buffer using theippiSqrDistanceNormGetBufferSize function.

This function operates with ROI.

Depending on the IppiNormOp value set to the algType parameter, the function calculates the following results:

<table>
<thead>
<tr>
<th>IppiNormOp Value</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>ippiNormNone</td>
<td>Squared Euclidean distances $S_{tx}(r,c)$</td>
</tr>
<tr>
<td>ippiNorm</td>
<td>Normalized squared Euclidean distances $\sigma_{tx}(r,c)$</td>
</tr>
</tbody>
</table>

For more information on how each value is calculated, see Image Proximity Measures.

The size of the resulting matrix depends on the IppiROIShape value:

<table>
<thead>
<tr>
<th>IppiROIShape Value</th>
<th>Matrix Size</th>
</tr>
</thead>
<tbody>
<tr>
<td>ippiROIValid</td>
<td>$(W_S - W_T + 1) \times (H_S \cdot H_T + 1)$</td>
</tr>
<tr>
<td>ippiROISame</td>
<td>$W \times H$</td>
</tr>
<tr>
<td>ippiROIFull</td>
<td>$(W_S + W_T - 1) \times (H_S + H_T - 1)$</td>
</tr>
</tbody>
</table>

where

- $W_S, H_S$ is the width and height of the source image
• $W_t, H_t$ is the width and height of the template image

**Return Values**

- **ippStsNoErr**: Indicates no error.
- **ippStsNullPtrErr**: Indicates an error when any of the specified pointers is NULL.
- **ippStsStepErr**: Indicates an error when the value of `srcStep`, `tplStep`, or `dstStep` is negative, or equal to zero.
- **ippStsSizeErr**: Indicates an error when:
  - `srcRoiSize` or `tplRoiSize` is negative, or equal to zero
  - the value of `srcRoiSize` is less than the corresponding value of `tplRoiSize`
- **ippStsAlgTypeErr**: Indicates an error when:
  - the result of the bitwise AND operation between the `algType` and `ippAlgMask` values differs from the `ippAlgAuto`, `ippAlgDirect`, or `ippAlgFFT` values;
  - the result of the bitwise AND operation between the `algType` and `ippiROIMask` values differs from the `ippiROIFull`, `ippiROISame`, or `ippiROIValid` values;
  - the result of the bitwise AND operation between the `algType` and `ippiNormMask` values differs from the `ippiNormNone` or `ippiNorm` values;

**Example**

The code example below demonstrates how to use the `ippiSqrDistanceNormGetBufferSize` and `ippiSqrDistanceNorm` functions.

```c
IppStatus SqrDistanceNormExample()
{

    IppStatus status;
    Ipp32f pSrc[5*4];
    Ipp32f pTpl[5*4];
    Ipp32f pDst[9*7]; // (5+5-1) x (4+4-1)

    IppiSize srcRoiSize = {5,4};
    IppiSize tplRoiSize = {5,4};
    IppiSize dstRoiSize = {9,7};
    int srcStep = 5*sizeof(Ipp32f);
    int tplStep = 5*sizeof(Ipp32f);
    int dstStep = 9*sizeof(Ipp32f);
    IppEnum funCfg = (IppEnum)(ippAlgAuto | ippiNorm | ippiROIFull);
    Ipp8u * pBuffer;
    int bufSize=0;

    ippSet_32f_C1R(2.0, pSrc, srcStep, srcRoiSize);
    ippSet_32f_C1R(1.0, pTpl, tplStep, tplRoiSize);

    status = ippiSqrDistanceNormGetBufferSize(srcRoiSize, tplRoiSize, ipp32f, funCfg, &bufSize);
    if ( status != ippStsNoErr ) return status;

    pBuffer = ippsMalloc_8u( bufSize );
```
status = ippiSqrDistanceNorm_32f_C1R( pSrc, srcStep, srcRoiSize, pTpl, tplStep, tplRoiSize, pDst, dstStep, funcCfg, pBuffer);
    printf_2D_32f("pDst", pDst, dstRoiSize);
    ippsFree( pBuffer );
    return status;
}

The result is as follows:

<p>| | | | | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>2.24</td>
<td>1.58</td>
<td>1.29</td>
<td>1.12</td>
<td>1.00</td>
<td>1.12</td>
<td>1.29</td>
<td>1.58</td>
<td>2.24</td>
</tr>
<tr>
<td>1.58</td>
<td>1.12</td>
<td>0.91</td>
<td>0.79</td>
<td>0.71</td>
<td>0.79</td>
<td>0.91</td>
<td>1.12</td>
<td>1.58</td>
</tr>
<tr>
<td>1.29</td>
<td>0.91</td>
<td>0.75</td>
<td>0.65</td>
<td>0.58</td>
<td>0.65</td>
<td>0.75</td>
<td>0.91</td>
<td>1.29</td>
</tr>
<tr>
<td>1.12</td>
<td>0.79</td>
<td>0.65</td>
<td>0.56</td>
<td>0.50</td>
<td>0.56</td>
<td>0.65</td>
<td>0.79</td>
<td>1.12</td>
</tr>
<tr>
<td>1.29</td>
<td>0.91</td>
<td>0.75</td>
<td>0.65</td>
<td>0.58</td>
<td>0.65</td>
<td>0.75</td>
<td>0.91</td>
<td>1.29</td>
</tr>
<tr>
<td>1.58</td>
<td>1.12</td>
<td>0.91</td>
<td>0.79</td>
<td>0.71</td>
<td>0.79</td>
<td>0.91</td>
<td>1.12</td>
<td>1.58</td>
</tr>
<tr>
<td>2.24</td>
<td>1.58</td>
<td>1.29</td>
<td>1.12</td>
<td>1.00</td>
<td>1.12</td>
<td>1.29</td>
<td>1.58</td>
<td>2.24</td>
</tr>
</tbody>
</table>

See Also
Integer Result Scaling
Image Proximity Measures
Regions of Interest in Intel IPP
Structures and Enumerators

**SqrDistanceNormGetBufferSize** Computes the size of the work buffer for the *ippiSqrDistanceNorm* function.

### CrossCorrNormGetBufferSize
*Comutes the size of the work buffer for the *ippiCrossCorrNorm* function.*

#### Syntax

```c
IppStatus ippiCrossCorrNormGetBufferSize (IppiSize srcRoiSize, IppiSize tplRoiSize, IppEnum algType, int* pBufferSize);
```

#### Include Files

```c
ippi.h
```

#### Domain Dependencies

**Headers:** ippcore.h, ippvm.h, ipps.h

**Libraries:** ippcore.lib, ippvm.lib, ipps.lib

#### Parameters

- **srcRoiSize, tplRoiSize**
  - Size of the source/template ROI in pixels.
- **algType**
  - Bit-field mask for the algorithm type definition. Possible values are the results of composition of the *IppAlgType*, *IppiROIShape*, and *IppiNormOp* values.
- **pBufferSize**
  - Pointer to the size of the work buffer.
The `ippiCrossCorrNormGetBufferSize` function computes the size, in bytes, of the external work buffer needed for the function that performs two-dimensional cross-correlation. The result is stored in the `pBufferSize` parameter.

**Return Values**

- `ippStsNoErr` Indicates no error.
- `ippStsSizeErr` Indicates an error when:
  - `srcRoiSize` or `tplRoiSize` is negative, or equal to zero
  - the value of `srcRoiSize` is less than the corresponding value of `tplRoiSize`.
- `ippStsAlgTypeErr` Indicates an error when:
  - the result of the bitwise AND operation between the `algType` and `ippAlgMask` differs from the `ippAlgAuto`, `ippAlgDirect`, or `ippAlgFFT` values.
  - the result of the bitwise AND operation between the `algType` and `ippiROIMask` differs from the `ippiROIFull`, `ippiROISame`, or `ippiROIValid` values.
  - the result of the bitwise AND operation between the `algType` and `ippiNormMask` differs from the `ippiNormNone`, `ippiNorm`, or `ippiNormCoefficient` values.
- `ippStsNullPtrErr` Indicates an error when `pBufferSize` is NULL.

**See Also**

- **Structures and Enumerators**
- **CrossCorrNorm** Computes a normalized cross-correlation between an image and a template.

**CrossCorrNorm**

*Computes a normalized cross-correlation between an image and a template.*

**Syntax**

**Case 1: Operating on data with integer output**

```c
IppStatus ippiCrossCorrNorm_8u_C1RSfs(const Ipp8u* pSrc, int srcStep, IippiSize srcRoiSize, const Ipp8u* pTpl, int tplStep, IippiSize tplRoiSize, Ipp8u* pDst, int dstStep, int scaleFactor, IppEnum algType, Ipp8u* pBuffer);
```

**Case 2: Operating on data with floating-point output**

```c
IppStatus ippiCrossCorrNorm_<mod>(const Ipp<srcDatatype>* pSrc, int srcStep, IippiSize srcRoiSize, const Ipp<srcDatatype>* pTpl, int tplStep, IippiSize tplRoiSize, Ipp32f* pDst, int dstStep, IppEnum algType, Ipp8u* pBuffer);
```

Supported values for `mod`:

- `32f_C1R`
- `8u32f_C1R`
- `16u32f_C1R`

**Include Files**

`ippi.h`
Domain Dependencies

Headers: ippcore.h, ippvm.h, ipps.h
Libraries: ippcore.lib, ippvm.lib, ipps.lib

Parameters

- **pSrc**: Pointer to the source image ROI.
- **srcStep**: Distance, in bytes, between the starting points of consecutive lines in the source image.
- **srcRoiSize**: Size of the source ROI in pixels.
- **pTpl**: Pointer to the template image.
- **tplStep**: Distance, in bytes, between the starting points of consecutive lines in the template image.
- **tplRoiSize**: Size of the template ROI in pixels.
- **pDst**: Pointer to the destination image ROI.
- **dstStep**: Distance, in bytes, between the starting points of consecutive lines in the destination image.
- **scaleFactor**: Scale factor.
- **algType**: Bit-field mask for the algorithm type definition. Possible values are the results of composition of the IppAlgType, IppiROIShape, and IppiNormOp values.
- **pBuffer**: Pointer to the work buffer.

Description

This function operates with ROI. Depending on the IppiNormOp value set to the algType parameter, the function calculates the following results:

<table>
<thead>
<tr>
<th>IppiNormOp Value</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>ippiNormNone</td>
<td>Cross-correlation values $R_{tx}(r,c)$</td>
</tr>
<tr>
<td>ippiNorm</td>
<td>Normalized cross-correlation values $\rho_{tx}(r,c)$</td>
</tr>
<tr>
<td>ippiNormCoefficient</td>
<td>Normalized correlation coefficients $\gamma_{tx}(r,c)$</td>
</tr>
</tbody>
</table>

For more information about how each value is calculated, see Image Proximity Measures.

The size of the resulting matrix depends on the IppiROIShape value:

<table>
<thead>
<tr>
<th>IppiROIShape Value</th>
<th>Matrix Size</th>
</tr>
</thead>
<tbody>
<tr>
<td>ippiROIFull</td>
<td>$(W_s + W_t - 1) \times (H_s + H_t - 1)$</td>
</tr>
<tr>
<td>ippiROISame</td>
<td>$W_s \times H_s$</td>
</tr>
<tr>
<td>ippiROIValid</td>
<td>$(W_s - W_t + 1) \times (H_s \cdot H_t + 1)$</td>
</tr>
</tbody>
</table>

where

- $W_s$, $H_s$ is the width and height of the source image
• \( W_t, H_t \) is the width and height of the template image

Before using this function, you need to compute the size of the work buffer using the \texttt{ippiCrossCorrNormGetBufferSize} function.

Return Values

- \texttt{ippStsNoErr} Indicates no error.
- \texttt{ippStsNullPtrErr} Indicates an error when any of the specified pointers is NULL.
- \texttt{ippStsStepErr} Indicates an error when the value of \texttt{srcStep}, \texttt{tplStep}, or \texttt{dstStep} is negative, or equal to zero.
- \texttt{ippStsSizeErr} Indicates an error when:
  - the \texttt{srcRoiSize} or \texttt{tplRoiSize} is negative, or equal to zero.
  - the value of \texttt{srcRoiSize} is less than the corresponding value of the \texttt{tplRoiSize}.
- \texttt{ippStsAlgTypeErr} Indicates an error when:
  - the result of the bitwise AND operation between the \texttt{algType} and \texttt{ippAlgMask} differs from the \texttt{ippAlgAuto}, \texttt{ippAlgDirect}, or \texttt{ippAlgFFT} values.
  - the result of the bitwise AND operation between the \texttt{algType} and \texttt{ippiROIMask} differs from the \texttt{ippiROIFull}, \texttt{ippiROISame}, or \texttt{ippiROIValid} values.
  - the result of the bitwise AND operation between the \texttt{algType} and \texttt{ippiNormMask} differs from the \texttt{ippiNormNone}, \texttt{ippiNorm}, or \texttt{ippiNormCoefficient} values.

Example

The code example below demonstrates how to use the \texttt{ippiCrossCorrNormGetBufferSize} and \texttt{ippiCrossCorrNorm} functions.

```c
IppStatus CrossCorrNormExample()
{
    IppStatus status;
    IppiSize srcRoiSize = {5,4};
    IppiSize tplRoiSize = {3,3};
    IppiSize dstRoiSize = {5,4}; // same as src
    Ipp32f pSrc[5*4] = { 1.0f, 2.0f, 1.5f, 4.1f, 3.6f,
                        0.2f, 3.2f, 2.5f, 1.5f, 10.0f,
                        5.0f, 6.8f, 0.5f, 4.1f, 1.1f,
                        7.1f, 4.2f, 2.2f, 8.7f, 10.0f};
    Ipp32f pTpl[3*3] = { 2.1f, 3.5f, 7.7f,
                        0.4f, 2.3f, 5.5f,
                        1.4f, 2.8f, 3.1f};
    Ipp32f pDst[5*4];
    int srcStep = 5*sizeof(Ipp32f);
    int tplStep = 3*sizeof(Ipp32f);
    int dstStep = 5*sizeof(Ipp32f);
    IppEnum funCfg = (IppEnum)(ippAlgAuto|ippiROISame|ippiNorm);
    Ipp8u *pBuffer;
    int bufSize;
    status = ippiCrossCorrNormGetBufferSize(srcRoiSize, tplRoiSize, ipp32f, funCfg, &bufSize);
    // Use ippiCrossCorrNorm function
}
```
if ( status != ippStsNoErr ) return status;

pBuffer = ippsMalloc_8u( bufSize );

status = ippiCrossCorrNorm_32f_C1R(pSrc, srcStep, srcRoiSize, pTpl, tplStep, tplRoiSize, pDst, dstStep, funCfg, pBuffer);
printf_2D_32f("pDst", pDst, dstRoiSize);

ippsFree( pBuffer );
return status;
}

The result is as follows:

<table>
<thead>
<tr>
<th>pDst</th>
<th>-&gt;</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.53</td>
<td>0.54</td>
</tr>
<tr>
<td>0.68</td>
<td>0.62</td>
</tr>
<tr>
<td>0.77</td>
<td>0.55</td>
</tr>
<tr>
<td>0.81</td>
<td>0.46</td>
</tr>
</tbody>
</table>

See Also
Integer Result Scaling
Regions of Interest in Intel IPP
Structures and Enumerators
Image Proximity Measures

**SADGetBufferSize**

*Computes the size of the work buffer for the ippiSAD function.*

**Syntax**

IppStatus ippiSADGetBufferSize(IppiSize srcRoiSize, IppiSize tplRoiSize, IppDataType dataType, int numChannels, IppROIShape shape, int* pBufferSize);

**Include Files**

ippi.h

**Domain Dependencies**

Headers: ippcore.h, ippvm.h, ipps.h
Libraries: ippcore.lib, ippvm.lib, ipps.lib

**Parameters**

- *srcRoiSize*  
  Size of the source ROI in pixels.

- *tplRoiSize*  
  Size of the template ROI in pixels.

- *shape*  
  Enumeration that defines the shape of the result of the SAD operation (see Structures and Enumerators).

- *dataType*  
  Type of the input data.

- *numChannels*  
  Number of channels in the images.

- *pBufferSize*  
  Pointer to the computed value of the external buffer size.
Description

The \texttt{ippiSADGetBufferSize} function computes the size of the external work buffer (in bytes) needed for the \texttt{ippiSAD} function and stores the result in the \texttt{*pBufferSize} parameter.

Return Values

\begin{itemize}
  \item \texttt{ippStsNoErr} Indicates no error. Any other value indicates an error or a warning.
  \item \texttt{ippStsNullPtrErr} Indicates an error condition if the \texttt{pBufferSize} pointer is NULL.
  \item \texttt{ippStsSizeErr} Indicates an error condition if \texttt{srcRoiSize} or \texttt{tplRoiSize} has a field with a zero or negative value.
  \item \texttt{ippStsNotSupportedModeErr} Indicates an error condition if either \texttt{shape} does not equal \texttt{ippiROIValid} or \texttt{numChannels} does not equal 1.
\end{itemize}

SAD

\textit{Computes sums of absolute differences (SAD) for a template image and different locations within a source image where the template image can fit.}

Syntax

\begin{verbatim}
IppStatus ippiSAD_<mod>(const Ipp<srcDatatype>* pSrc, int srcStep, IppiSize srcRoiSize, const Ipp<srcDatatype>* pTpl, int tplStep, IppiSize tplRoiSize, Ipp<dstDatatype>* pDst, int dstStep, IppiROIShape shape, int scaleFactor, Ipp8u* pBuffer);
IppStatus ippiSAD_32f_C1R(const Ipp<srcDatatype>* pSrc, int srcStep, IppiSize srcRoiSize, const Ipp<srcDatatype>* pTpl, int tplStep, IppiSize tplRoiSize, Ipp<dstDatatype>* pDst, int dstStep, IppiROIShape shape, Ipp8u* pBuffer);
\end{verbatim}

Supported values for \texttt{mod}:

\begin{itemize}
  \item \texttt{8u32s_C1RSfs}
  \item \texttt{16u32s_C1RSfs}
  \item \texttt{16s32s_C1RSfs}
\end{itemize}

Include Files

\texttt{ippi.h}

Domain Dependencies

Headers: ippcore.h, ippvm.h, ipps.h

Libraries: ippcore.lib, ippvm.lib, ipps.lib

Parameters

\begin{itemize}
  \item \texttt{pSrc} Pointer to the source image ROI.
  \item \texttt{srcStep} Distance, in bytes, between the starts of consecutive lines in the source image.
  \item \texttt{srcRoiSize} Size of the source ROI in pixels.
  \item \texttt{pTpl} Pointer to the template image ROI.
  \item \texttt{tplStep} Distance, in bytes, between the starts of consecutive lines in the template image.
\end{itemize}
tplRoiSize
pDst
dstStep
shape
scaleFactor
pBuffer

Description
The `ippiSAD` function operates with ROI (see Regions of Interest in Intel IPP).

The function only supports the `ippiROIValid` value of `shape`, and so the sizes of the destination ROI are different from the sizes of both the source ROI and template ROI and depend on those sizes as follows:

\[
\text{dstRoiSize}.\text{width} = W = \text{srcRoiSize}.\text{width} - \text{tplRoiSize}.\text{width} + 1
\]
\[
\text{dstRoiSize}.\text{height} = H = \text{srcRoiSize}.\text{height} - \text{tplRoiSize}.\text{height} + 1
\]

So there are exactly \(W \times H\) unique locations within the source image where the template image can fit. The top-left pixel determines each of these locations. For each location, the function computes absolute differences between corresponding pixel values of the source and template images, sums these differences to make the SAD value of the top-left pixel, and assigns the SAD value to the appropriate pixel in the destination buffer.

For integer flavors, the resulting values are also scaled by `scaleFactor`.

Return Values

- **ippStsNoErr**: Indicates no error. Any other value indicates an error or a warning.
- **ippStsNullPtrErr**: Indicates an error condition if the `pSrc`, `pTpl`, `pDst`, or `pBuffer` pointer is NULL.
- **ippStsSizeErr**: Indicates an error condition if `srcRoiSize` or `tplRoiSize` has a field with a zero or negative value or if `srcRoiSize` in any direction is less than `tplRoiSize`.
- **ippStsStepErr**: Indicates an error condition if `srcStep`, `tplStep`, or `dstStep` has a zero or negative value or is not a multiple of the image data size (4 for floating-point images or 2 for short-integer images).
- **ippStsBadArgErr**: Indicates an error condition if `scaleFactor` <0.
- **ippStsNotSupportedModeErr**: Indicates an error condition if intersection of the source and destination ROI is detected or if `shape` differs from `ippiROIValid`.

Example

The code example below shows how to use the `ippiSAD_8u32s_C1RSfs` function.

```cpp
Ipp8u src[8*8] = {1, 2, 3, 4, 8, 8, 8, 8,
    1, 2, 3, 4, 8, 8, 8, 8,
    1, 2, 3, 4, 8, 8, 8, 8,
    1, 2, 3, 4, 8, 8, 8, 8,
    1, 2, 3, 4, 8, 8, 8, 8,
    1, 2, 3, 4, 8, 8, 8, 8,
...}```
IppSize srcRoi = { 7, 7 };

Ipp8u template[3*3] = {
10, 10, 10,
10, 10, 10,
10, 10, 10};

IppSize tplRoi = { 3, 3 };

Ipp32s dst[8*8] = {0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0};

IppSize srcRoi = { 3, 3 };

ippiSADGetBufferSize( srcRoi, tplRoi, ippiROIVaid , 1, &bufferSize);
Ipp8u* pBuffer = new Ipp8u [bufferSize];
ippiSAD_8u32s_C1RSfs(src, 8, srcRoi, template, 3, tplRoi, dst, 8, ippiROIVaid , 1, pBuffer);

Result:
1 2 3 4 8 8 8 8 10 10 10 36 32 22 15 9 0 0 0 dst
1 2 3 4 8 8 8 8 src 10 10 10 tpl 36 32 22 15 9 0 0 0 dst
1 2 3 4 8 8 8 8 10 10 10 36 32 22 15 9 0 0 0
1 2 3 4 8 8 8 8 36 32 22 15 9 0 0 0
1 2 3 4 8 8 8 8 36 32 22 15 9 0 0 0
1 2 3 4 8 8 8 8 0 0 0 0 0 0 0 0
1 2 3 4 8 8 8 8 0 0 0 0 0 0 0 0
1 2 3 4 8 8 8 8 0 0 0 0 0 0 0 0
1 2 3 4 8 8 8 8 0 0 0 0 0 0 0 0
1 2 3 4 8 8 8 8 0 0 0 0 0 0 0 0
This chapter describes the Intel® IPP image processing functions that perform geometric operations of resizing, rotating, warping and remapping an image.

Most functions performing geometric transform of an image use an interpolation algorithm to resample the image. The type of interpolation method to be used is passed to the function in the `interpolation` parameter for rotate, warp, and remap. For resize transform, the interpolation type is part of the function name.

The following interpolation algorithms are used:

- nearest neighbor
- linear interpolation
- cubic convolution
- supersampling
- interpolation using Lanczos window function
- interpolation with two-parameter cubic filters
- optional edge smoothing of the destination image.

The nearest neighbor algorithm is the fastest, while other methods yield higher quality results, but are slower.

Use one of the following constant identifiers for the applicable interpolation methods:

- `IPPI_INTER_NN` / `ippNearest`: Nearest neighbor interpolation.
- `IPPI_INTER_LINEAR` / `ippLinear`: Linear interpolation.
- `IPPI_INTER_CUBIC`: Cubic interpolation.
- `IPPI_INTER_LANCZOS` / `ippLanczos`: Interpolation using 3-lobed Lanczos window function.
- `IPPI_INTER_CUBIC2P_BSPLINE`: Interpolation using B-spline.
- `IPPI_INTER_CUBIC2P_CATMULLROM`: Interpolation using Catmull-Rom spline.
- `IPPI_INTER_CUBIC2P_B05C03`: Interpolation using special cubic filter.
- `ippSuper`: Supersampling interpolation.
- `ippCubic`: Interpolation with two-parameter cubic filters.

For certain functions, you can combine the above interpolation algorithms with additional smoothing (antialiasing) of edges to which the original image borders are transformed. To use this edge smoothing, set the parameter `interpolation` to the bitwise OR of `IPPI_SMOOTH_EDGE` or `IPPI_SUBPIXEL_EDGE` and the desired interpolation mode, or use the special function flags.

**Caution**

You can use interpolation with edge smoothing option only in those geometric transform functions where this option is explicitly listed in the parameters definition section.

See appendix B “Interpolation in Image Geometric Transform Functions” for more information on the interpolation algorithms that are used in the library.

Super Sampling mode of resize transform has several limitations. It can be used only:
ROI Processing in Geometric Transforms

All the transform functions described in this chapter operate in rectangular regions of interest (ROIs) that are defined in both the source and destination images. The procedures for handling ROIs in geometric transform functions differ from those used in other functions (see Regions of Interest in Intel IPP in chapter 2). The main difference is that operations take place in the intersection of the transformed source ROI and the destination ROI. More specifically, all geometric transform functions (except those which perform inverse warping operations) handle ROIs with the following sequence of operations (see figure below):

- transform the rectangular ROI of the source image to quadrangle in the destination image;
- find the intersection of this quadrangle and the rectangular ROI of the destination image;
- update the destination image in the intersection area.

The coordinates in the source and destination images must have the same origin.

When using functions with ROI, every scan line of a source image has a stride. It is padded with zeroes for alignment, so the actual size of a scan line in bytes is often greater than the image width. The size of each row of image data in bytes is determined by the value of srcStep parameter, which gives distance in bytes between the starts of consecutive lines of an image.

To fully describe a rectangular ROI, both its origin (coordinates of top left corner) and size must be referenced. For geometrical transform functions, the source image ROI is specified by srcRoi parameter of IppiRect type, meaning that all four values describing the rectangular ROI are given explicitly.

The destination image ROI for different functions can be specified either by dstRoi parameter of IppiRect type, or dstRoiSize parameter of IppiSize type. In the latter case, only the destination ROI size is passed, while its origin is referenced by pDst pointer.

The destination image origin ROI for different functions can be specified by dstOffset parameter of IppiPoint type and dstSize parameter of IppiSize type. In this case, the processed destination image coresponds to the processed ROI of the destination image origin.
Geometric Transform Functions

ResizeYCbCr422GetBufSize
*Computes the size of the external buffer for the NV12 resize transform.*

**Syntax**

```c
IppStatus ippiResizeYCbCr422GetBufSize(IppiRect srcROI, IppiSize dstRoiSize, int interpolation, int* pSize);
```

**Include Files**

ippi.h

**Domain Dependencies**

Headers: ippcore.h, ippvm.h, ipps.h

Libraries: ippcore.lib, ippvm.lib, ipps.lib

**Parameters**

- `srcROI`: Region of interest of the source image.
- `dstRoiSize`: Region of interest of the destination image.
- `interpolation`: Type of interpolation to apply to the source image:
  - IPP_INTER_NN: Nearest neighbor interpolation
  - IPP_INTER_LINEAR: Linear interpolation
  - IPP_INTER_CUBIC: Cubic interpolation
  - IPPI_INTER_CUBIC2P_C: Catmull-Rom cubic filter
  - ATMULLROM: Catmull-Rom cubic filter
  - IPP_INTER_LANCZOS: Lanczos filter with size 6x6
- `pSize`: Pointer to the size, in bytes, of the external buffer.

**Description**

This function computes the size of the external buffer for the YCbCr resize transform.

**Return Values**

- ippStsNoErr: Indicates no error.
- ippStsNullPtrErr: Indicates an error if `pSize` is NULL.
- ippStsSizeErr: Indicates an error in the following cases:
  - width of either source or destination ROI is less than 2
  - height of either source or destination ROI is less than 1
- ippStsDoubleSize: Indicates a warning if width of either source or destination ROI is not a multiple of 2.
Indicates an error if interpolation has an illegal value.

**ResizeYCbCr422**

*Performs resizing of a 4:2:2 two-channel image.*

**Syntax**

```c
IppStatus ippiResizeYCbCr422_8u_C2R(const Ipp8u* pSrc, IppiSize srcSize, int srcStep, IppiRect srcROI, Ipp8u* pDst, int dstStep, IppiSize dstRoiSize, int interpolation, Ipp8u* pBuffer);
```

**Include Files**

ippi.h

**Domain Dependencies**

- **Headers:** ippcore.h, ippvm.h, ipps.h
- **Libraries:** ippcore.lib, ippvm.lib, ipps.lib

**Parameters**

- **pSrc**
  - Pointer to the source image.

- **srcSize**
  - Size, in pixels, of the source image.

- **srcStep**
  - Distance, in bytes, between the starting points of consecutive lines in the source image buffer.

- **srcROI**
  - Region of interest in the source image.

- **pDst**
  - Pointer to the destination image ROI.

- **dstStep**
  - Distance, in bytes, between the starting points of consecutive lines in the destination image buffer.

- **dstRoiSize**
  - Size of the destination ROI in pixels.

- **interpolation**
  - Type of interpolation to apply to the source image:
    - IPP_INTER_NN: Nearest neighbor interpolation
    - IPP_INTER_LINEAR: Linear interpolation
    - IPP_INTER_CUBIC: Cubic interpolation
    - IPP_INTER_CUBIC2P_CATMULLROM: Catmull-Rom cubic filter
    - IPP_INTER_LANCZOS: Lanczos filter with size 6x6

- **pBuffer**
  - Pointer to the external buffer.

**Description**

This function operates with ROI (see ROI Processing in Geometric Transforms).

This function resizes the source image ROI to the destination image ROI origin. The function performs resampling of the result using the interpolation mode specified by the interpolation parameter, and stores it in the destination image ROI.
The source image is a two-channel image in the 4:2:2 sampling format in color spaces with decoupled luminance and chrominance components, for example, YUV422 or YCrCb422.

**Return Values**

- **ippStsNoErr**: Indicates no error.
- **ippStsNullPtrErr**: Indicates an error if `pSrc`, `pDst`, or `pBuffer` is NULL.
- **ippStsSizeErr**: Indicates an error in the following cases:
  - width of either source or destination ROI is less than 2
  - height of either source or destination ROI is less than 1
- **ippStsDoubleSize**: Indicates a warning if width wither source or destination ROI is not a multiple of 2.
- **ippStsInterpolationErr**: Indicates an error if `interpolation` has an illegal value.
- **ippStsWrongIntersectROI**: Indicates a warning if `srcROI` does not intersect with the source image; no operation is required.

**Resize Functions with Prior Initialization**

This section describes the Intel® IPP resize functions that use the specification structure in operation. Before using these functions, you need to initialize the structure.

**Using Intel® IPP Resize Functions with Prior Initialization**

You can use one of the following approaches to image resizing:

- **Resizing the whole image**
- **Resizing a tiled image with one prior initialization**
- **Resizing a tiled image with prior initialization for each tile**

Interpolation algorithms of the Lanczos, Linear, and Cubic types use edge pixels of the source image that are out of the image origin. When calling the `ippiResize<Filter>` function with one of these interpolation algorithms applied, you need to specify the appropriate border type. The following border types are supported:

- Replicated borders: border pixels are replicated from the source image boundary pixels;
- Borders in memory: the source image border pixels are obtained from the source image pixels in memory;
- Mixed borders: a combined approach is applied.

**NOTE**

If you want to resize an image with antialiasing, follow the same instructions as provided below, but use `ippiResizeAntialiasing<Filter>Init` instead of `ippiResize<Filter>Init` for initialization, and `ippiResizeAntialiasing<Filter>` instead of `ippiResize<Filter>`, as a processing function.

**Resizing the Whole Image**

You can apply the approach described below to resize when source and destination images are fully accessible in memory. However, this method only runs on a single thread.

To resize the whole image:

1. **Call the `ippiResizeGetSize` function with the appropriate interpolation type.** This function uses source and destination image sizes to calculate how much memory must be allocated for the `IppResizeSpec` structure and initialization work buffer.
2. Initialize the IppResizeSpec structure by calling the ippiResize<Filter>Init, where <Filter> can take one of the following values: Nearest, Linear, Cubic, Lanczos, and Super. These prerequisite steps allow resize to be called multiple times without recalculations.

3. Call the ippiResizeGetBufferSize function for the initialized IppResizeSpec structure. This function uses the destination image size to calculate how much memory must be allocated for the resize work buffer.

4. Call ippiResize<Filter> with the appropriate image type.

5. If you call the ippiResize<Filter> function with a ippBorderInMem border or any mixed border type, the applied interpolation algorithm uses weighted values from edge pixels of the source image when outside the image boundaries. To obtain the size of the border required for correct edge calculation, call the ippiResizeGetBorderSize function for the appropriate flavor. In case of mixed border type, out of image pixels are used only behind the non-replicated edge.

6. You can use mixed borders by using the bitwise OR operation between the ippBorderRepl type and the following border types: ippBorderInMemTop, ippBorderInMemBottom, ippBorderInMemLeft, ippBorderInMemRight.

Figure Simple Image Resize shows a simple image resizing example, in which image resolution is increased by 1.5x.

Simple Image Resize

a) source image

b) image after resize transform
Example

The code example below demonstrates whole image resizing with the Lanczos interpolation method:

```c
IppStatus resizeExample_C3R(Ipp8u* pSrc, IppiSize srcSize, Ipp32s srcStep, Ipp8u* pDst, IppiSize dstSize, Ipp32s dstStep)
{
    IppiResizeSpec_32f* pSpec = 0;
    int specSize = 0, initSize = 0, bufSize = 0;
    Ipp8u* pBuffer = 0;
    Ipp8u* pInitBuf = 0;
    Ipp32u numChannels = 3;
    IppiPoint dstOffset = {0, 0};
    IppStatus status = ippStsNoErr;
    IppiBorderType border = ippBorderRepl;

    /* Spec and init buffer sizes */
    status =ippiResizeGetSize_8u(srcSize, dstSize, ippLanczos, 0, &specSize, &initSize);
    if (status != ippStsNoErr) return status;

    /* Memory allocation */
    pInitBuf = ippsMalloc_8u(initSize);
    pSpec    = (IppiResizeSpec_32f*)ippsMalloc_8u(specSize);
    if (pInitBuf == NULL || pSpec == NULL)
    {
        ippsFree(pInitBuf);
        ippsFree(pSpec);
        return ippStsNoMemErr;
    }

    /* Filter initialization */
    status =ippiResizeLanczosInit_8u(srcSize, dstSize, 3, pSpec, pInitBuf);
    if (status != ippStsNoErr)
    {
        ippsFree(pSpec);
        return status;
    }

    /* work buffer size */
    status =ippiResizeGetBufferSize_8u(pSpec, dstSize, numChannels, &bufSize);
    if (status != ippStsNoErr)
    {
        ippsFree(pSpec);
        return status;
    }

    pBuffer = ippsMalloc_8u(bufSize);
    if (pBuffer == NULL)
    {
        ippsFree(pSpec);
        return ippStsNoMemErr;
    }

    /* Resize processing */
    status =ippiResizeLanczos_Bu_C3R(pSrc, srcStep, pDst, dstStep, dstOffset, dstSize, border,
```

Intel® Integrated Performance Primitives Developer Reference, Volume 2: Image Processing
Resizing a Tiled Image with One Prior Initialization

You can apply the approach described below to resize when source and destination images are not fully accessible in memory, or to improve the performance of resizing by external threading.

The main difference between this approach and whole image resizing is that the processing is split into sections of the image called tiles. Each call of the Resize<Filter> function works with the destination image origin region of interest (ROI) that is defined by dstOffset and dstSize parameters. The destination and source ROI must be fully accessible in memory.

To resize an image with the tiled approach:

1. Call theippiResizeGetSize function with the appropriate interpolation type. This function uses the source and destination image sizes to calculate how much memory must be allocated for the IppResizeSpec structure and initialization work buffer.
2. Initialize the IppResizeSpec structure by callingippiResize<Filter>Init, where <Filter> can take one of the following values: Nearest, Cubic, Linear, and Lanczos.
3. Determine an appropriate partitioning scheme to divide the destination image into tiles. Tiles can be sets of rows or a regular grid of subimages. A simple vertical subdivision into sets of lines is often sufficient.
4. Obtain the source ROI for the defined destination tile by calling theippiResizeGetSrcRoi function for the corresponding flavor. The algorithm uses edge pixels that are out of the source ROI to calculate edge pixels of the destination ROI. These out of the source ROI edge pixels must be accessible in memory.
5. If the source ROI is an interior field of the source image origin, obtain the border ROI size by calling theippiResizeGetBorderSize function for the corresponding flavor.
6. If the source ROI is an edge tile, the algorithm can interpolate pixels beyond the image boundary as in the previous method.
7. If the source and destination images are fully accessible in memory, you can use the source ROI offset for the pSrc calculation. To obtain the offset, call theippiResizeGetSrcOffset function for the corresponding flavor.
8. Call theippiResizeGetBufferSize function to obtain the size of the resize work buffer required for each tile processing. The dstSize parameter must be equal to the tile size.
9. CallippiResize<Filter> for each tile (ROI). The dstOffset parameter must specify the image ROI offset with respect to the destination image origin. The dstSize parameter must be equal to the ROI size. Parameters pSrc and pDst must point to the beginning of the source and destination ROI in memory respectively. The source and destination ROIs must be fully accessible in memory.

You can process tiles in any order. When using multiple threads you can process all tiles simultaneously.

NOTE
If you resize a tiled image with the Super Sampling algorithm, and the source image width to destination image width ratio is \( m/n \), you can reach better performance of resize operation if all destination tiles have width that is a multiple of \( n \).
Figure *Tiling Image Resize* shows the resize of the image divided into tiles.

**Tiling Image Resize**

Example

The code example below demonstrates a multithreading resize operation using OpenMP* with parallelization only in the y direction:

```c
#define MAX_NUM_THREADS 16

IppStatus tileResizeExample_C3R(Ipp8u* pSrc, IppiSize srcSize, Ipp32s srcStep, Ipp8u* pDst, IppiSize dstSize, Ipp32s dstStep)
{
    IppiResizeSpec_32f* pSpec = 0;
    int specSize = 0, initSize = 0, bufSize = 0;
    Ipp8u* pBuffer = 0;
    Ipp8u* pInitBuf = 0;
    Ipp32u numChannels = 3;
    IppiPoint dstOffset = {0, 0};
    IppiPoint srcOffset = {0, 0};
    IppStatus status = ippStsNoErr;
    IppiBorderSize borderSize = {0, 0, 0, 0};
    IppiBorderType border = ippBorderRepl;
    int numThreads, slice, tail;
    int bufSize1, bufSize2;
    IppiSize dstTileSize, dstLastTileSize;
    IppStatus pStatus[MAX_NUM_THREADS];

    /* Spec and init buffer sizes */
    status = ippiResizeGetSize_8u(srcSize, dstSize, ippLinear, 0, &specSize, &initSize);
    if (status != ippStsNoErr) return status;
```
/* Memory allocation */
pInitBuf = ippsMalloc_8u(initSize);
pSpec = (IppiResizeSpec_32f*)ippsMalloc_8u(specSize);

if (pInitBuf == NULL || pSpec == NULL)
{
    ippsFree(pInitBuf);
    ippsFree(pSpec);
    return ippStsNoMemErr;
}

/* Filter initialization */
status = ippiResizeLinearInit_8u(srcSize, dstSize, pSpec);
ippsFree(pInitBuf);

if (status != ippStsNoErr)
{
    ippsFree(pSpec);
    return status;
}

status = ippiResizeGetBorderSize_8u(pSpec, &borderSize);
if (status != ippStsNoErr)
{
    ippsFree(pSpec);
    return status;
}

/* General transform function */
/* Parallelized only by Y-direction here */
#pragma omp parallel num_threads(MAX_NUM_THREADS)
{
    #pragma omp master
    {
        numThreads = omp_get_num_threads();
        slice = dstSize.height / numThreads;
        tail = dstSize.height % numThreads;

        dstTileSize.width = dstLastTileSize.width = dstSize.width;
        dstTileSize.height = slice;
        dstLastTileSize.height = slice + tail;

        ippiResizeGetBufferSize_8u(pSpec, dstTileSize, ippC3, &bufSize1);
        ippiResizeGetBufferSize_8u(pSpec, dstLastTileSize, ippC3, &bufSize2);

        pBuffer = ippsMalloc_8u(bufSize1 * (numThreads - 1) + bufSize2);
    }

    #pragma omp barrier
    {
        if (pBuffer)
        {
            Ipp32u i;
            Ipp8u *pSrcT, *pDstT;
            Ipp8u *pOneBuf;
            IppiPoint srcOffset = {0, 0};
            IppiPoint dstOffset = {0, 0};
            IppiSize srcSizeT = srcSize;
IppSize dstSizeT = dstTileSize;

i = omp_get_thread_num();
dstSizeT.height = slice;
dstOffset.y += i * slice;

if (i == numThreads - 1) dstSizeT = dstLastTileSize;

pStatus[i] = ippiResizeGetSrcRoi_8u(pSpec, dstOffset, dstSizeT, &srcOffset, &srcSizeT);

if (pStatus[i] == ippStsNoErr)
{
    pSrcT = (Ipp8u*)((char*)pSrc + srcOffset.y * srcStep);
    pDstT = (Ipp8u*)((char*)pDst + dstOffset.y * dstStep);
    pOneBuf = pBuffer + i * bufSize1;

    pStatus[i] = ippiResizeLinear_8u_C3R (pSrcT, srcStep, pDstT, dstStep, 
dstOffset, dstSizeT, border, 0, pSpec, pOneBuf);
}

ippsFree(pSpec);

if (pBuffer == NULL) return ippStsNoMemErr;

ippsFree(pBuffer);

for (Ipp32u i = 0; i < numThreads; ++i)
{
    /* Return bad status */
    if(pStatus[i] != ippStsNoErr) return pStatus[i];
}

return status;

---

**Resizing a Tiled Image with Prior Initialization for Each Tile**

You can apply this approach only in cases when the destination image can be divided into tiles so that each destination tile corresponds to a source image tile that starts with an integer pixel value origin. For example, if the ratio of the source and destination images sizes is 2/3, the destination image can be divided into 3x3 tiles, each of which corresponds to the source image tile 2x2.

This approach is useful if there are restrictions on memory size when processing an image, or if the image size is large and `ippiResizeGetBufferSize` function returns `ippStsSizeErr` error. The initialization data for a tile is less than the same data for the whole image.

Each tile of the source image can be considered as an independent image that can be resized. For interior tile processing, the border must be always of the `ippBorderInMem` type. If you need to replicate any borders of the source image origin, you should combine the border type of the outer tiles so that interior tiles edges have border in memory and external tile borders are of the specified border type. This approach enables the right linking order of tiles.
Figure *Resize of the Image Divided into Subimages* shows the approach, when the source image is divided into several subimages that are resized independently.

**Resize of the Image Divided into Subimages**

![Source image divided into subimages](image1)

![Source image after resize transform](image2)

**Example**

The code example below divides the source image into tiles and resizes each image independently:

```c
IppStatus separateTileResizeExample_C3R(Ipp8u* pSrc, IppiSize srcTileSize, Ipp32s srcStep,
Ipp8u* pDst, IppiSize dstTileSize, Ipp32s dstStep, Ipp32s xNumTiles, Ipp32s yNumTiles)
{
    IppiResizeSpec_32f* pSpec = 0;
    int specSize = 0, initSize = 0, bufSize = 0;
    Ipp8u* pBuffer  = 0;
    Ipp8u* pInitBuf = 0;
    Ipp32u numChannels = 3;
    IppStatus status = ippStsNoErr;

    /* tiles cicle */
    for (Ipp32s j = 0; j < xNumTiles; j++)
    {
        for (Ipp32s i = 0; i < yNumTiles; i++)
        {
            /* calculation of the destination image ROI offset */
            IppiPoint dstOffset = {j * dstTileSize.width, i * dstTileSize.height};
            Ipp8u* pDstT = pDst + dstStep * dstOffset.y + dstOffset.x * numChannels * sizeof(Ipp8u);

            /* calculation of the source image ROI offset */
            IppiPoint srcOffset = {j * srcTileSize.width, i * srcTileSize.height};
            Ipp8u* pSrcT = pSrc + srcStep * srcOffset.y + srcOffset.x * numChannels * sizeof(Ipp8u);
        }
    }

    ... more code here ...
}
```

Image Geometry Transforms
IppiBorderType borderT = ippBorderRepl;
IppiPoint dstOffsetZero = {0, 0};

/* correction of the border type for the tile processing */
if (j > 0) /* the processed tile is not on the left image origin edge*/
{
    borderT = (IppiBorderType)((int)borderT | (int)ippBorderInMemLeft);
}
if (j < xNumTiles - 1) /* the processed tile is not on the right image origin edge*/
{
    borderT = (IppiBorderType)((int)borderT | (int)ippBorderInMemRight);
}
if (i > 0) /* the processed tile is not on the top image origin edge*/
{
    borderT = (IppiBorderType)((int)borderT | (int)ippBorderInMemTop);
}
if (i < yNumTiles - 1) /* the processed tile is not on the bottom image origin edge*/
{
    borderT = (IppiBorderType)((int)borderT | (int)ippBorderInMemBottom);
}

/* Spec and init buffer sizes */
status =ippiResizeGetSize_8u(srcTileSize, dstTileSize, ippLanczos, 0, &specSize,
&initSize);
if (status != ippStsNoErr) return status;

/* Memory allocation */
pInitBuf = ippsMalloc_8u(initSize);
pSpec = (IppiResizeSpec_32f*)ippsMalloc_8u(specSize);
if (pInitBuf == NULL || pSpec == NULL)
{
    ippsFree(pInitBuf);
    ippsFree(pSpec);
    return ippStsNoMemErr;
}

/* Filter initialization */
status =ippiResizeLanczosInit_8u(srcTileSize, dstTileSize, 3, pSpec, pInitBuf);
ippsFree(pInitBuf);
if (status != ippStsNoErr)
{
    ippsFree(pSpec);
    return status;
}

/* work buffer size */
status =ippiResizeGetBufferSize_8u(pSpec, dstTileSize, numChannels, &bufSize);
if (status != ippStsNoErr)
{
    ippsFree(pSpec);
    return status;
}
 pBuffer = ippsMalloc_8u(bufSize);
  if (pBuffer == NULL)
    {
      ippsFree(pSpec);
      return ippStsNoMemErr;
    }

  /* Resize processing */
  status = ippiResizeLanczos_8u_C3R(pSrcT, srcStep, pDstT, dstStep, dstOffsetZero,
  dstTileSize, borderT, 0, pSpec, pBuffer);

  ippsFree(pSpec);
  ippsFree(pBuffer);

  if (status != ippStsNoErr) return status;

  return ippStsNoErr;
}

See Also
User-defined Border Types

ResizeGetSize

Computes the size of the specification structure and the size of the external temporary buffer for the resize transform initialization.

Syntax
Case 1: Processing images of 32-bit sizes
IppStatus ippiResizeGetSize_<mod>(IppiSize srcSize, IppiSize dstSize,
IppiInterpolationType interpolation, Ipp32u antialiasing, int* pSpecSize, int* pInitBufSize);

Supported values for mod:

  8u    16u    32f    64f

IppStatus ippiResizeGetSize_16s(IppiSize srcSize, IppiSize dstSize,
IppiInterpolationType interpolation, Ipp32u antialiasing, int* pSpecSize, Ipp32s* pInitBufSize);

Case 2: Processing images with platform-aware functions
IppStatus ippiResizeGetSize_L(IppiSize srcSize, IppiSize dstSize, IppDataType
dataType, IppiInterpolationType interpolation, Ipp32u antialiasing, IppSizeL* pSpecSize, IppSizeL* pInitBufSize);

Case 3: Processing images with threading layer (TL) functions
IppStatus ippiResizeGetSize_LT(IppiSize srcSize, IppiSize dstSize, IppDataType
dataType, IppiInterpolationType interpolation, Ipp32u antialiasing, IppSizeL* pSpecSize, IppSizeL* pInitBufSize);

Include Files
ippi.h
Flavors with the _LT suffix: ippi_tl.h
Flavors with the _L suffix: ippi_l.h

Domain Dependencies
Flavors declared in ippi.h:
Headers: ippcore.h, ippvm.h, ipp.h
Libraries: ippcore.lib, ippvm.lib, ipp.lib

Flavors declared in ippi_tl.h:

Parameters

**srcSize**
Size, in pixels, of the source image.

**dstSize**
Size, in pixels, of the destination image.

**interpolation**
Interpolation method. Supported values: ippNearest, ippLinear, ippCubic, ippLanczos, and ippSuper.

**antialiasing**
Supported values: 1 - resizing with antialiasing, 0 - resizing without antialiasing.

**dataType**
Data type of the image. Supported values: ipp8u, ipp16u, ipp16s, ipp32f, ipp64f (only for linear interpolation).

**pSpecSize**
Pointer to the size, in bytes, of the specification structure.

**pInitBufSize**
Pointer to the size, in bytes, of the temporary buffer required for initialization of the specification structure.

Description
This function computes the size of the specification structure and the size of the external buffer for the following functions depending on the **interpolation** parameter value:

- **ippiResizeAntialiasingCubicInit, ippiResizeAntialiasingLanczosInit, or ippiResizeAntialiasingLinearInit** for resizing with antialiasing
- **ippiResizeNearestInit, ippiResizeLinearInit, ippiResizeCubicInit, ippiResizeLanczosInit, or ippiResizeSuperInit** for resizing without antialiasing

Interpolation algorithms have the following filter sizes:

<table>
<thead>
<tr>
<th>Method</th>
<th>Filter Size</th>
</tr>
</thead>
<tbody>
<tr>
<td>Nearest Neighbor</td>
<td>1x1</td>
</tr>
<tr>
<td>Linear</td>
<td>2x2</td>
</tr>
<tr>
<td>Cubic</td>
<td>4x4</td>
</tr>
<tr>
<td>2-lobed Lanczos</td>
<td>4x4</td>
</tr>
</tbody>
</table>

**NOTE** The **ippiResizeGetSize** function always returns non-zero value for the **pInitBufSize** parameter, even if the temporary buffer is not required for the specification structure initialization. The temporary buffer is only required when initializing the specification structure for the following functions: **ippiResizeAntialiasingCubic, ippiResizeAntialiasingLanczos, ippiResizeAntialiasingLinear, ippiResizeLanczos, and ippiResizeCubic.**
Return Values

- **ippStsNoErr**
  Indicates no error.
- **ippStsNullPtrErr**
  Indicates an error if one of the specified pointers is **NULL**.
- **ippStsResizeNoOperation**
  Indicates an error if width or height of the image is equal to zero.
- **ippStsSizeErr**
  Indicates an error in the following cases:
  - If the source image size is less than the filter size of the chosen interpolation method (except ippSuper).
  - If one of the specified dimensions of the source image is less than the corresponding dimension of the destination image (for ippSuper method only).
  - If the width or height of the source or destination image is negative.
- **ippStsExceededSizeErr**
  Indicates an error in the following cases:
  - If at least one of the computed values exceeds the maximum of the data type positive value pointed by pSpecSize or pInitBufSize correspondingly (the size of one of the processed images is too large).
  - If width or height of the destination image or the source image exceeds 33554431 (0x1FFFFFF) (only for platform-aware and TL functions).
- **ippStsInterpolationErr**
  Indicates an error if interpolation has an illegal value.
- **ippStsNoAntialiasing**
  Indicates a warning if the specified interpolation method does not support antialiasing.
- **ippStsNotSupportedModeErr**
  Indicates an error if the requested mode is not supported.
- **ippStsDataTypeErr**
  Indicates an error if dataType has an illegal value.

**ResizeGetBufferSize**
*Computes the size of the external buffer for image resizing.*

**Syntax**

**Case 1: Single precision**

```c
IppStatus ippiResizeGetBufferSize_<mod>(const IppiResizeSpec_32f* pSpec, IppSize dstSize, Ipp32u numChannels, int* pBufSize);
```

Supported values for mod:

- 8u
- 16u
- 16s
- 32f

**Case 2: Double precision**

```c
IppStatus ippiResizeGetBufferSize_64f(const IppiResizeSpec_64f* pSpec, IppSize dstSize, Ipp32u numChannels, int* pBufSize);
```

**Case 3: Processing images with platform-aware functions**

```c
IppStatus ippiResizeGetBufferSize_L(const IppiResizeSpec* pSpec, IppSizeL dstSize, Ipp32u numChannels, IppSizeL* pBufSize);
```
Case 4: Processing images with threading layer (TL) functions

IppStatus ippiResizeGetBufferSize_LT(const IppiResizeSpec_LT* pSpec, IppSizeL* pBufSize);

Include Files
ippi.h

Flavors with the _LT suffix: ippi_tl.h
Flavors with the _L suffix: ippi_l.h

Domain Dependencies
Flavors declared in ippi.h:
Headers: ippcore.h, ippvm.h, ipps.h
Libraries: ippcore.lib, ippvm.lib, ipps.lib
Flavors declared in ippi_tl.h:

Parameters
pSpec Pointer to the spec structure for the resize filter.
dstSize Size in pixels of the destination image.
numChannels Number of channels, possible values: 1, 3, or 4.
pBufSize Pointer to the size, in bytes, of the external buffer.

Description
This function computes the size of the external buffer for image resizing. The pSpec parameter defines the resize algorithm parameters. Prior to using the ippiResizeGetBufferSize function, you need to initialize the pSpec parameter by calling one of the following functions: ippiResizeNearestInit, ippiResizeLinearInit, ippiResizeCubicInit, ippiResizeLanczosInit, and ippiResizeSuperInit.

Return Values
ippStsNoErr Indicates no error.
ippStsNullPtrErr Indicates an error if pBufferSize pointer is NULL.
ippStsNoOperation Indicates a warning if width or height of the destination image is equal to zero.
ippStsContextMatchErr Indicates an error if pointer to the spec structure is invalid.
ippStsNumChannelErr Indicates an error if the value of numChannels is illegal.
ippStsSizeErr Indicates an error condition If width or height of the destination image is negative.
ippStsSizeWrn Indicates a warning if the destination image size is more than the destination image origin size.
ippStsExceededSizeErr Indicates an error If at least one of the computed values exceeds maximum of IppSizeL type positive value (the size of one of the processed images is too large).
ResizeGetBorderSize

Computes the size of possible borders for the resize transform.

Syntax

Case 1: Interpolation with single precision

IppStatus ippiResizeGetBorderSize_<mod>(const IppiResizeSpec_32f* pSpec,
    IppiBorderSize* borderSize);

Supported values for \( \text{mod} \):

\[ 8u \quad 16u \quad 16s \quad 32f \]

Case 2: Interpolation with double precision

IppStatus ippiResizeGetBorderSize_64f(const IppiResizeSpec_64f* pSpec,
    IppiBorderSize* borderSize);

Case 3: Processing images with platform-aware functions

IppStatus ippiResizeGetBorderSize_L(const IppiResizeSpec* pSpec,
    IppiBorderSize* borderSize);

Case 4: Processing images with threading layer (TL) functions

IppStatus ippiResizeGetBorderSize_LT(const IppiResizeSpec_LT* pSpec,
    IppiBorderSize* borderSize);

Include Files

ippi.h

Flavors with the _LT suffix: ippi_tl.h
Flavors with the _L suffix: ippi_l.h

Domain Dependencies

Flavors declared in ippi.h:

Headers: ippcore.h, ippvm.h, ipps.h
Libraries: ippcore.lib, ippvm.lib, ipps.lib

Flavors declared in ippi_t1.l.h:

Libraries: ippcore.lib, ippvm.lib, ipps.lib, ipp.lib, ippcore_tl.lib, ipp_t1.lib

Parameters

\( pSpec \)  \hspace{1cm} \text{Pointer to the spec structure for the resize filter.}

\( \text{borderSize} \)  \hspace{1cm} \text{Size in pixels of necessary borders.}

Description

This function computes the size of the source image ROI that is used by the corresponding resize transform and is out of the processing boundaries. The \( pSpec \) parameter defines the resize algorithm parameters. Prior to using the \( \text{ippiResizeGetBorderSize} \) function, you need to initialize the \( pSpec \) parameter by calling one of the following functions: \( \text{ippiResizeNearestInit}, \text{ippiResizeLinearInit}, \text{ippiResizeCubicInit}, \text{ippiResizeLanczosInit}, \text{and ippiResizeSuperInit}. \)
Return Values

- ippStsNoErr: Indicates no error.
- ippStsNullPtrErr: Indicates an error if one of the specified pointers is NULL.
- ippStsContextMatchErr: Indicates an error if pointer to the spec structure is invalid.
- ippStsBorderErr: Indicates an error if border has an illegal value.

ResizeGetSrcOffset

_Computes the offset of the source image for resizing by tile processing._

Syntax

**Single precision**

```
IppStatus ippiResizeGetSrcOffset_<mod>(const IppiResizeSpec_<mod>* pSpec, IppiPoint dstOffset, IppiPoint* srcOffset);
```

Supported values for `mod`:

- 8u
- 16u
- 16s
- 32f

**Double precision**

```
IppStatus ippiResizeGetSrcOffset_64f(const IppiResizeSpec_64f* pSpec, IppiPoint dstOffset, IppiPoint* srcOffset);
```

Processing images with platform-aware functions

```
IppStatus ippiResizeGetSrcOffset_L(const IppiResizeSpec* pSpec, IppiPointL dstOffset, IppiPointL* srcOffset);
```

Include Files

- ippi.h

Flavors with the `_L` suffix: ippi_l.h

Domain Dependencies

- Headers: ippcore.h, ippvm.h, ipps.h
- Libraries: ippcore.lib, ippvm.lib, ipps.lib

Parameters

- `pSpec`: Pointer to the spec structure for the resize filter.
- `dstOffset`: Offset of the tiled destination image with respect to the destination image origin.
- `srcOffset`: Offset of the source image.
**Description**

This function computes the offset of the processed source image ROI using the offset of the processed destination image ROI for the corresponding resize transform by tile processing. The *pSpec* parameter defines the resize algorithm parameters. Prior to using the `ippiResizeGetSrcOffset` function, you need to initialize the *pSpec* parameter by calling one of the following functions: `ippiResizeNearestInit`, `ippiResizeLinearInit`, `ippiResizeCubicInit`, `ippiResizeLanczosInit`, or `ippiResizeSuperInit`.

**Return Values**

- `ippStsNoErr` Indicates no error.
- `ippStsNullPtrErr` Indicates an error if one of the specified pointers is NULL.
- `ippStsContextMatchErr` Indicates an error if pointer to the spec structure is invalid.
- `ippStsOutOfRangeErr` Indicates an error if the destination image offset point is outside the destination image origin.

**ResizeGetSrcRoi**

*Computes the ROI of the source image for resizing by tile processing.*

**Syntax**

**Single precision**

```c
IppStatus ippiResizeGetSrcRoi_<mod>(const IppiResizeSpec_32f* pSpec, IppiPoint dstRoiOffset, IppiSize dstRoiSize, IppiPoint* srcRoiOffset, IppiSize* srcRoiSize);
```

Supported values for *mod*:

- `8u`
- `16u`
- `16s`
- `32f`

**Double precision**

```c
IppStatus ippiResizeGetSrcRoi_64f(const IppiResizeSpec_64f* pSpec, IppiPoint dstRoiOffset, IppiSize dstRoiSize, IppiPoint* srcRoiOffset, IppiSize* srcRoiSize);
```

**Processing images with platform-aware functions**

```c
IppStatus ippiResizeGetSrcRoi_L(const IppiResizeSpec* pSpec, IppiPointL dstRoiOffset, IppiSizeL dstRoiSize, IppiPointL* srcRoiOffset, IppiSizeL* srcRoiSize);
```

**Include Files**

`ippi.h`

**Flavors with the _L suffix**: `ippi_l.h`

**Domain Dependencies**

**Headers**: `ippcore.h`, `ippvm.h`, `ipps.h`

**Libraries**: `ippcore.lib`, `ippvm.lib`, `ipps.lib`

**Parameters**

- `pSpec` Pointer to the spec structure for the resize filter.
- `dstRoiOffset` Offset of the tiled destination image ROI.
**dstRoiSize**
Size of the tiled destination image ROI.

**srcRoiOffset**
Offset of the source image ROI.

**srcRoiSize**
Pointer to the size of the source image ROI.

**Description**
This function computes the ROI of the processed source image using the processed ROI of the destination image for the corresponding resize transform by tile processing. The `pSpec` parameter defines the resize algorithm parameters. Prior to using the `ippiResizeGetSrcRoi` function, you need to initialize the `pSpec` parameter by calling one of the following functions: `ippiResizeNearestInit`, `ippiResizeLinearInit`, `ippiResizeCubicInit`, `ippiResizeLanczosInit`, or `ippiResizeSuperInit`.

**NOTE**
If the destination ROI size exceeds the image origin, the source ROI will be obtained for an intersection of the destination ROI and image origin.

**Return Values**
- `ippStsNoErr` Indicates no error.
- `ippStsNullPtrErr` Indicates an error if one of the specified pointers is NULL.
- `ippStsContextMatchErr` Indicates an error if pointer to the spec structure is invalid.
- `ippStsOutOfRangeErr` Indicates an error if the destination image offset point is outside the destination image origin.
- `ippStsSizeWrn` Indicates a warning if the destination ROI exceeds the destination image origin.

**ResizeNearestInit**
*Initializes the specification structure for image resizing with the nearest neighbor interpolation method.*

**Syntax**

```
IppStatus ippiResizeNearestInit_<mod>(IppiSize srcSize, IppiSize dstSize, IppiResizeSpec_32f* pSpec);
```

Supported values for `mod`:

- 8u
- 16u
- 16s
- 32f

**Platform-aware function**

```
IppStatus ippiResizeNearestInit_L(IppiSizeL srcSize, IppiSizeL dstSize, IppDataType dataType, IppiResizeSpec* pSpec);
```

**Threading layer (TL) function**

```
IppStatus ippiResizeNearestInit_LT(IppiSizeL srcSize, IppiSizeL dstSize, IppDataType dataType, Ipp32u numChannels, IppiResizeSpec_LT* pSpec);
```

**Include Files**

`ippi.h`
Flavors with the _LT suffix:ippi_tl.h
Flavors with the _L suffix:ippi_l.h

Domain Dependencies
Flavors declared in ippi.h:
Headers: ippcore.h, ippvm.h, ipps.h
Libraries: ippcore.lib, ippvm.lib, ipps.lib
Flavors declared in ippi_tl.h:

Parameters
- srcSize: Size, in pixels, of the source image.
- dstSize: Size, in pixels, of the destination image.
- dataType: Data type of the image. Supported values: ipp8u, ipp16u, ipp16s, ipp32f, ipp64f.
- pSpec: Pointer to the specification structure for the resize filter.
- numChannels: Number of channels. Possible values are 1, 3, and 4.

Description
This function initializes the specification structure for the resize algorithm with the nearest neighbor interpolation method. To calculate the size of the specification structure object, call the ippiResizeGetSize function.

Return Values
- ippStsNoErr: Indicates no error.
- ippStsNullPtrErr: Indicates an error when one of the specified pointers is NULL.
- ippStsNoOperation: Indicates a warning when width or height of the image is equal to zero.
- ippStsSizeErr: Indicates an error when width or height of the source or destination image is negative.
- ippStsDataTypeErr: Indicates an error if dataType has an illegal value.
- ippStsExceededSizeErr: Indicates an error if width or height of the source or destination image exceeds 33554431 (0x1FFFFFF) (only for platform-aware and TL functions).

See Also
- ResizeGetSize Computes the size of the specification structure and the size of the external temporary buffer for the resize transform initialization.
- ResizeNearest Changes an image size using the nearest neighbor interpolation method.
Syntax

IppStatus ippiResizeNearest_<mod>(const Ipp<datatype>* pSrc, Ipp32s srcStep, Ipp<datatype>* pDst, Ipp32s dstStep, IppiPoint dstOffset, IppiSize dstSize, const IppiResizeSpec_32f* pSpec, Ipp8u* pBuffer);

Supported values for mod:

- 8u_C1R 16u_C1R 16s_C1R 32f_C1R
- 8u_C3R 16u_C3R 16s_C3R 32f_C3R
- 8u_C4R 16u_C4R 16s_C4R 32f_C4R

Platform-aware functions

IppStatus ippiResizeNearest_<mod>_L(const Ipp<datatype>* pSrc, IppSizeL srcStep, Ipp<datatype>* pDst, IppSizeL dstStep, IppiPointL dstOffset, IppiSizeL dstSize, const IppiResizeSpec* pSpec, Ipp8u* pBuffer);

Supported values for mod:

- 8u_C1R 16u_C1R 16s_C1R 32f_C1R
- 8u_C3R 16u_C3R 16s_C3R 32f_C3R
- 8u_C4R 16u_C4R 16s_C4R 32f_C4R

Threading layer (TL) functions

IppStatus ippiResizeNearest_<mod>_LT(const Ipp<datatype>* pSrc, IppSizeL srcStep, Ipp<datatype>* pDst, IppSizeL dstStep, const IppiResizeSpec_LT* pSpec, Ipp8u* pBuffer);

Supported values for mod:

- 8u_C1R 16u_C1R 16s_C1R 32f_C1R
- 8u_C3R 16u_C3R 16s_C3R 32f_C3R
- 8u_C4R 16u_C4R 16s_C4R 32f_C4R

Include Files

ippi.h

Flavors with the _LT suffix: ippi_tl.h
Flavors with the _L suffix: ippi_l.h

Domain Dependencies

Flavors declared in ippi.h:

Headers: ippcore.h, ippvm.h, ipps.h
Libraries: ippcore.lib, ippvm.lib, ipps.lib

Flavors declared in ippi_tl.h:

Parameters

*pSrc*  
Pointer to the source image.

*srcStep*  
Distance, in bytes, between the starting points of consecutive lines in the source image buffer.

*pDst*  
Pointer to the destination image.

*dstStep*  
Distance, in bytes, between the starting points of consecutive lines in the destination image buffer.

*dstOffset*  
Offset of the tiled destination image with respect to the destination image origin.

*dstSize*  
Size of the destination image in pixels.

*pSpec*  
Pointer to the specification structure for the resize filter.

*pBuffer*  
Pointer to the work buffer.

Description

This function changes an image size using the nearest neighbor interpolation method. The image size can be either reduced or increased in each direction, depending on the destination image size.

This function operates with ROI. It resizes the source image ROI origin to the destination image ROI origin. You need to define the destination image ROI origin by the following parameters: the offset of the tiled destination image with respect to the destination image origin and the destination image size. The source image ROI origin is defined automatically. To obtain the source image ROI, use the *ippiResizeGetSrcRoi* function with the corresponding *mod* value. To obtain the source image ROI origin offset, call the *ippiResizeGetSrcOffset* function with the corresponding *mod* value. Parameters *pSrc* and *pDst* must point to the processed source and destination image ROI origins, respectively.

Function flavors operating on images of 64-bit sizes (with the *L* suffix) can process only whole images.

The interpolation algorithm applied uses only pixels of the source image origin that are inside of the image boundaries.

Before using the *ippiResizeNearest* function, you need to initialize the resize specification structure using the *ippiResizeNearestInit* function and compute the size of the external buffer *pBuffer* using the *ippiResizeGetBufferSize* function for the corresponding flavor.

Return Values

*ippStsNoErr*  
Indicates no error.

*ippStsNullPtrErr*  
Indicates an error when one of the specified pointers is NULL.

*ippStsNoOperation*  
Indicates a warning when width or height of the destination image is equal to zero.

*ippStsContextMatchErr*  
Indicates an error when pointer to the spec structure is invalid.

*ippStsSizeErr*  
Indicates an error when width or height of the source or destination image is negative.

*ippStsStepErr*  
Indicates an error when the step value is not data type multiple.

*ippStsOutOfRangeErr*  
Indicates an error when the destination image offset point is outside the destination image origin.

*ippStsSizeWrn*  
Indicates a warning when the destination image size is more than the destination image origin size.
See Also
ROI Processing in Geometric Transforms
ResizeGetSrcRoi Computes the ROI of the source image for resizing by tile processing.
ResizeGetSrcOffset Computes the offset of the source image for resizing by tile processing.
ResizeNearestInit Initializes the specification structure for image resizing with the nearest
neighbor interpolation method.
ResizeGetBufferSize Computes the size of the external buffer for image resizing.

**ResizeLinearInit**
*Initializes the specification structure for image resizing with the linear interpolation method.*

**Syntax**

```c
IppStatus ippiResizeLinearInit_<mod>(IppiSize srcSize, IppiSize dstSize,
IppiResizeSpec_32f* pSpec);
```

**Supported values for mod:**

- 8u
- 16u
- 16s
- 32f

```c
IppStatus ippiResizeLinearInit_64f(IppiSize srcSize, IppiSize dstSize,
IppiResizeSpec_64f* pSpec);
```

**Platform-aware function**

```c
IppStatus ippiResizeLinearInit_L(IppiSize srcSize, IppiSize dstSize, IppDataType
dataType, IppiResizeSpec* pSpec);
```

```c
IppStatus ippiResizeLinearInit_8u_L(IppiSize srcSize, IppiSize dstSize,
IppHintAlgorithm hint, IppiResizeSpec* pSpec);
```

**Threading layer (TL) function**

```c
IppStatus ippiResizeLinearInit_LT(IppiSize srcSize, IppiSize dstSize, IppDataType
dataType, Ipp32u numChannels, IppiResizeSpec_LT* pSpec);
```

**Include Files**

- ippi.h

**Flavors with the _LT suffix:** ippi_tl.h

**Flavors with the _L suffix:** ippi_l.h

**Domain Dependencies**

- Flavors declared in ippi.h:
  - Headers: ippcore.h, ippvm.h, ipps.h
  - Libraries: ippcore.lib, ippvm.lib, ipps.lib
- Flavors declared in ippi_tl.h:

**Parameters**

- **srcSize**
  - Size, in pixels, of the source image.

- **dstSize**
  - Size, in pixels, of the destination image.
**Description**

This function initializes the specification structure for the resize algorithm with the linear interpolation method. To calculate the size of the specification structure, call the `ippiResizeGetSize` function.

**NOTE**

The function with the parameter `hint` allows users to choose between a faster but less accurate algorithm and a slower but more accurate one. Without the parameter `hint` this function initializes the specification structure for the faster but less accurate algorithm for `Ipp8u` data type.

**Return Values**

- `ippStsNoErr` Indicates no error.
- `ippStsNullPtrErr` Indicates an error when one of the specified pointers is `NULL`.
- `ippStsNoOperation` Indicates a warning when width or height of the image is equal to zero.
- `ippStsSizeErr` Indicates an error if:
  - width or height of the source or destination image is negative
  - the source image size is less than the size of a 2x2 linear filter
- `ippStsDataTypeErr` Indicates an error if `dataType` has an illegal value.
- `ippStsExceededSizeErr` Indicates an error if width or height of the source or destination image exceeds 33554431 (0x1FFFFFF) (only for platform-aware or TL functions).
- `ippStsNumChannelErr` Indicates an error if `numChannels` has an illegal value.

**See Also**

- `ResizeGetSize` Computes the size of the specification structure and the size of the external temporary buffer for the resize transform initialization.
- `ResizeLinear` Changes an image size using the linear interpolation method.
Syntax

Case 1: Single precision

IppStatus ippiResizeLinear_<mod>(const Ipp<datatype>* pSrc, Ipp32s srcStep, Ipp<datatype>* pDst, Ipp32s dstStep, IppiPoint dstOffset, IppSize dstSize, IppiBorderType border, const Ipp<datatype>* pBorderValue, const IppiResizeSpec_32f* pSpec, Ipp8u* pBuffer);

Supported values for mod:

<table>
<thead>
<tr>
<th>8u_C1R</th>
<th>16u_C1R</th>
<th>16s_C1R</th>
<th>32f_C1R</th>
</tr>
</thead>
<tbody>
<tr>
<td>8u_C3R</td>
<td>16u_C3R</td>
<td>16s_C3R</td>
<td></td>
</tr>
<tr>
<td>8u_C4R</td>
<td>16u_C4R</td>
<td>16s_C4R</td>
<td>32f_C4R</td>
</tr>
</tbody>
</table>

IppStatus ippiResizeLinear_32f_C3R(const Ipp32f* pSrc, const Ipp32s srcStep, Ipp32f* pDst, Ipp32s dstStep, IppiPoint dstOffset, IppiSize dstSize, IppiBorderType border, const Ipp32f* pBorderValue, const IppiResizeSpec_32f* pSpec, Ipp8u* pBuffer);

Case 2: Double precision

IppStatus ippiResizeLinear_<mod>(const Ipp<datatype>* pSrc, Ipp32s srcStep, Ipp<datatype>* pDst, Ipp32s dstStep, IppiPoint dstOffset, IppiSize dstSize, IppiBorderType border, const Ipp<datatype>* pBorderValue, const IppiResizeSpec_64f* pSpec, Ipp8u* pBuffer);

Supported values for mod:

<table>
<thead>
<tr>
<th>64f_C1R</th>
</tr>
</thead>
<tbody>
<tr>
<td>64f_C3R</td>
</tr>
<tr>
<td>64f_C4R</td>
</tr>
</tbody>
</table>

Case 3: Platform-aware functions

IppStatus ippiResizeLinear_<mod>_L(const Ipp<datatype>* pSrc, IppSizeL srcStep, Ipp<datatype>* pDst, IppSizeL dstStep, IppiPointL dstOffset, IppSizeL dstSize, IppiBorderType border, const Ipp<datatype>* pBorderValue, const IppiResizeSpec* pSpec, Ipp8u* pBuffer);

Supported values for mod:

<table>
<thead>
<tr>
<th>8u_C1R</th>
<th>16u_C1R</th>
<th>16s_C1R</th>
<th>32f_C1R</th>
<th>64f_C1R</th>
</tr>
</thead>
<tbody>
<tr>
<td>8u_C3R</td>
<td>16u_C3R</td>
<td>16s_C3R</td>
<td>32f_C3R</td>
<td>64f_C3R</td>
</tr>
<tr>
<td>8u_C4R</td>
<td>16u_C4R</td>
<td>16s_C4R</td>
<td>32f_C4R</td>
<td>64f_C4R</td>
</tr>
</tbody>
</table>
Case 3: Threading layer (TL) functions

IppStatus ippiResizeLinear_<mod>_LT(const Ipp<datatype>* pSrc, IppSizeL srcStep,
Ipp<datatype>* pDst, IppSizeL dstStep, IppiBorderType border, const Ipp8u* pBorderValue,
const IppiResizeSpec_LT* pSpec, Ipp8u* pBuffer);

Supported values for mod:

8u_C1R  16u_C1R  16s_C1R  32f_C1R  64f_C1R
8u_C3R  16u_C3R  16s_C3R  32f_C3R  64f_C3R
8u_C4R  16u_C4R  16s_C4R  32f_C4R  64f_C4R

Include Files

ippi.h

Flavors with the _LT suffix: ippi_tl.h
Flavors with the _L suffix: ippi_l.h

Domain Dependencies

Flavors declared in ippi.h:
Headers: ippcore.h, ippvm.h, ipps.h
Libraries: ippcore.lib, ippvm.lib, ipps.lib

Flavors declared in ippi_tl.h:

Parameters

pSrc     Pointer to the source image.
srcStep  Distance, in bytes, between the starting points of consecutive lines
          in the source image buffer.
pDst     Pointer to the destination image.
dstStep  Distance, in bytes, between the starting points of consecutive lines
          in the destination image buffer.
dstOffset Offset of the tiled destination image with respect to the destination
          image origin.
dstSize  Size of the destination image in pixels.
border   Type of border. Possible values are:
ippBorderRepl  Border is replicated from the edge pixels.
ippBorderInMem Border is obtained from the source image
          pixels in memory.

Mixed borders are also supported. They can be obtained by the bitwise
operation OR between the ippBorderRepl type and the
ippBorderInMemTop, ippBorderInMemBottom, ippBorderInMemLeft, ippBorderInMemRight types.
**pBorderValue**
Pointer to the constant value to assign to pixels of the constant border. This parameter is applicable only to the ippBorderConst border type.

**pSpec**
Pointer to the specification structure for the resize filter.

**pBuffer**
Pointer to the work buffer.

**Description**
This function changes the size of an image using the linear interpolation method. The image size can be either reduced or increased in each direction, depending on the destination image size.

This function operates with ROI. It resizes the source image ROI origin to the destination image ROI origin. You need to define the destination image ROI origin by the following parameters: the offset of the tiled destination image with respect to the destination image origin and the destination image size. The source image ROI origin is defined automatically. To obtain the source image ROI, use the ippiResizeGetSrcRoi function with the corresponding mod value. To obtain the source image ROI origin offset, call the ippiResizeGetSrcOffset function with the corresponding mod value. Parameters pSrc and pDst must point to the processed source and destination image ROI origins, respectively.

The interpolation algorithm applied uses edge pixels of the source image that are out of the image origin. The function ippiResizeLinear uses in calculation the weighted values of these outer pixels. To obtain the size of the out of the source image origin, call the ippiResizeGetBorderSize function for the corresponding flavor.

If the border type is equal to ippBorderRepl, the source image edge pixels are replicated out of the image origin. If the border type is equal to ippBorderInMem, the outer pixels are obtained from the out of the source image origin space. For the mixed border types, the combined approach is applied.

Before using the ippiResizeLinear function, you need to initialize the specification structure using the ippiResizeLinearInit function and compute the size of the external buffer pBuffer using the ippiResizeGetBufferSize function for the corresponding flavor.

**Return Values**
- ippStsNoErr: Indicates no error.
- ippStsNullPtrErr: Indicates an error when one of the specified pointers is NULL.
- ippStsNoOperation: Indicates a warning when width or height of the destination image is equal to zero.
- ippStsBorderErr: Indicates an error when the border value is illegal.
- ippStsContextMatchErr: Indicates an error when pointer to the specification structure is invalid.
- ippStsNotSupportedModeErr: Indicates an error when the requested mode is not supported.
- ippStsSizeErr: Indicates an error when width or height of the source or destination image is negative.
- ippStsStepErr: Indicates an error when the step value is not a multiple of data type.
- ippStsOutOfRangeErr: Indicates an error when the destination image offset point is outside the destination image origin.
- ippStsSizeWrn: Indicates a warning when the destination image size is more than the destination image origin size.

**See Also**
ROI Processing in Geometric Transforms
ResizeGetSrcRoi Computes the ROI of the source image for resizing by tile processing.
ResizeGetSrcOffset Computes the offset of the source image for resizing by tile processing.
User-defined Border Types
ResizeGetBorderSize Computes the size of possible borders for the resize transform.
ResizeLinearInit Initializes the specification structure for image resizing with the linear interpolation method.
ResizeGetBufferSize Computes the size of the external buffer for image resizing.

ResizeCubicInit
*Initializes the specification structure for image resizing using interpolation with two-parameter cubic filters.*

Syntax

IppStatus ippiResizeCubicInit_<mod>(IpplSize srcSize, IpplSize dstSize, Ipp32f valueB, Ipp32f valueC, IpplResizeSpec_32f* pSpec, Ipp8u* pInitBuf);

Supported values for mod:

<table>
<thead>
<tr>
<th>mod</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>8u</td>
<td></td>
</tr>
<tr>
<td>16u</td>
<td></td>
</tr>
<tr>
<td>16s</td>
<td></td>
</tr>
<tr>
<td>32f</td>
<td></td>
</tr>
</tbody>
</table>

Platform-aware functions

IppStatus ippiResizeCubicInit_L(IppSize srcSize, IppSize dstSize, IppDataType dataType, Ipp32f valueB, Ipp32f valueC, IpplResizeSpec* pSpec, Ipp8u* pInitBuf);

IppStatus ippiResizeCubicInit_8u_L(IppSize srcSize, IppSize dstSize, Ipp32f valueB, Ipp32f valueC, IppHintAlgorithm hint, IpplResizeSpec* pSpec, Ipp8u* pInitBuf);

Threading layer (TL) functions

IppStatus ippiResizeCubicInit_LT(IppSizeL srcSize, IppSizeL dstSize, IppDataType dataType, Ipp32u numChannels, Ipp32f valueB, Ipp32f valueC, IpplResizeSpec_LT* pSpec, Ipp8u* pInitBuf);

Include Files

ippi.h

Flavors with the _LT suffix: ippi_tl.h

Flavors with the _L suffix: ippi_l.h

Domain Dependencies

Flavors declared in ippi.h:

Headers: ippcore.h, ippvm.h, ipps.h

Libraries: ippcore.lib, ippvm.lib, ipps.lib

Flavors declared in ippi_tl.h:


Parameters

srcSize Size, in pixels, of the source image.
dstSize Size, in pixels, of the destination image.
**dataType**

Data type of the image. Supported values: ipp8u, ipp16u, ipp16s, ipp32f, ipp64f.

**hint**

Computation algorithm for processing ipp8u data. Possible values are:

- ippAlgHintNone: The hint is absent. Equivalent to ippAlgHintFast.
- ippAlgHintFast: A faster but less accurate algorithm.
- ippAlgHintAccurate: A slower but more accurate algorithm.

**numChannels**

Number of channels, possible values: 1, 3, or 4.

**valueB**

The first parameter for cubic filters.

**valueC**

The second parameter for cubic filters.

**pSpec**

Pointer to the specification structure for the resize filter.

**pInitBuf**

Pointer to the temporary buffer for the cubic filter initialization.

**Description**

This function initializes the specification structure for the resize algorithm with interpolation with two-parameter cubic filters. To calculate the size of the specification structure, call the ippiResizeGetSize function.

Before using this function, you need to calculate the size of the specification structure and the external buffer pInitBuf using the ippiResizeGetSize function for the corresponding flavor.

**NOTE**

The function with the parameter hint allows users to choose between a faster but less accurate algorithm and a slower but more accurate one. Without the parameter hint this function initializes the specification structure for the faster but less accurate algorithm for ipp8u data type.

**Return Values**

- ippStsNoErr: Indicates no error.
- ippStsNullPtrErr: Indicates an error when one of the specified pointers is NULL.
- ippStsNoOperation: Indicates a warning when width or height of the image is equal to zero.
- ippStsSizeErr: Indicates an error if:
  - width or height of the source or destination image is negative
  - the source image size is less than the size of a 2x2 linear filter
- ippStsDataTypeErr: Indicates an error if dataType has an illegal value.
- ippStsExceededSizeErr: Indicates an error if width or height of the source or destination image exceeds 33554431 (0x1FFFFFF) (only for platform-aware and TL functions).
- ippStsNumChannelErr: Indicates an error if numChannel has an illegal value.
See Also

ResizeGetSize Computes the size of the specification structure and the size of the external temporary buffer for the resize transform initialization.

ResizeCubic
Changes an image size using interpolation with two-parameter cubic filters.

Syntax

IppStatus ippiResizeCubic_<mod>(const Ipp<datatype>* pSrc, Ipp32s srcStep, Ipp<datatype>* pDst, Ipp32s dstStep, IpPiPoint dstOffset, IppSize dstSize, IppiBorderType border, const Ipp<datatype>* pBorderValue, const IppiResizeSpec_32f* pSpec, Ipp8u* pBuffer);

Supported values for mod:

8u_C1R  16u_C1R  16s_C1R  32f_C1R
8u_C3R  16u_C3R  16s_C3R  32f_C3R
8u_C4R  16u_C4R  16s_C4R  32f_C4R

Platform-aware functions

IppStatus ippiResizeCubic_<mod>_L(const Ipp<datatype>* pSrc, IppSizeL srcStep, Ipp<datatype>* pDst, IppSizeL dstStep, IpPiPointL dstOffset, IppSizeL dstSize, IppiBorderType border, const Ipp<datatype>* pBorderValue, const IppiResizeSpec* pSpec, Ipp8u* pBuffer);

Supported values for mod:

8u_C1R  16u_C1R  16s_C1R  32f_C1R
8u_C3R  16u_C3R  16s_C3R  32f_C3R
8u_C4R  16u_C4R  16s_C4R  32f_C4R

Threading layer (TL) functions

IppStatus ippiResizeCubic_<mod>_LT(const Ipp<datatype>* pSrc, IppSizeL srcStep, Ipp<datatype>* pDst, IppSizeL dstStep, IpPiPointL dstOffset, IppSizeL dstSize, IppiBorderType border, const Ipp<datatype>* pBorderValue, const IppiResizeSpec_LT* pSpec, Ipp8u* pBuffer);

Supported values for mod:

8u_C1R  16u_C1R  16s_C1R  32f_C1R
8u_C3R  16u_C3R  16s_C3R  32f_C3R
8u_C4R  16u_C4R  16s_C4R  32f_C4R

Include Files

ippi.h

Flavors with the _LT suffix: ippi_tl.h

Flavors with the _L suffix: ippi_l.h
Domain Dependencies

Flavors declared in ippi.h:
Headers: ippcore.h, ippvm.h, ipps.h
Libraries: ippcore.lib, ippvm.lib, ipps.lib

Flavors declared inippi_tl.h:

Parameters

pSrc
Pointer to the source image.

srcStep
Distance, in bytes, between the starting points of consecutive lines in the source image buffer.

pDst
Pointer to the destination image.

dstStep
Distance, in bytes, between the starting points of consecutive lines in the destination image buffer.

dstOffset
Offset of the tiled destination image with respect to the destination image origin.

dstSize
Size of the destination image in pixels.

border
Type of border. Possible values are:

ippBorderRepl Border is replicated from the edge pixels.
ippBorderInMem Border is obtained from the source image pixels in memory.

Mixed borders are also supported. They can be obtained by the bitwise operation OR between the ippBorderRepl type and the
ippBorderInMemTop, ippBorderInMemBottom, ippBorderInMemLeft, ippBorderInMemRight types.

pBorderValue
Pointer to the constant value to assign to pixels of the constant border. This parameter is applicable only to the ippBorderConst border type.

pSpec
Pointer to the spec structure for the resize filter.

pBuffer
Pointer to the work buffer.

Description

This function changes an image size using interpolation with two-parameter cubic filters. The image size can be either reduced or increased in each direction, depending on the destination image size.

This function operates with ROI. It resizes the source image ROI origin to the destination image ROI origin. You need to define the destination image ROI origin by the following parameters: the offset of the tiled destination image with respect to the destination image origin and the destination image size. The source image ROI origin is defined automatically. To obtain the source image ROI, use the ippiResizeGetSrcRoi function with the corresponding mod value. To obtain the source image ROI origin offset, call the ippiResizeGetSrcOffset function with the corresponding mod value. Parameters pSrc and pDst must point to the processed source and destination image ROI origins, respectively.
The interpolation algorithm applied uses edge pixels of the source image that are out of the image origin. The function `ippiResizeCubic` uses in calculation the weighted values of these outer pixels. To obtain the size of the out of the source image origin, call the `ippiResizeGetBorderSize` function for the corresponding flavor.

If the border type is equal to `ippBorderRepl`, the source image edge pixels are replicated out of the image origin. If the border type is equal to `ippBorderInMem`, the outer pixels are obtained from the out of the source image origin space. For the mixed border types, the combined approach is applied.

Before using the `ippiResizeLinear` function, you need to initialize the specification structure using the `ippiResizeCubicInit` function and compute the size of the external buffer `pBuffer` using the `ippiResizeGetBufferSize` function for the corresponding flavor.

**Return Values**

<table>
<thead>
<tr>
<th>Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ippStsNoErr</td>
<td>Indicates no error.</td>
</tr>
<tr>
<td>ippStsNullPtrErr</td>
<td>Indicates an error when one of the specified pointers is NULL.</td>
</tr>
<tr>
<td>ippStsNoOperation</td>
<td>Indicates a warning when width or height of the destination image is equal to zero.</td>
</tr>
<tr>
<td>ippStsBorderErr</td>
<td>Indicates an error when the border value is illegal.</td>
</tr>
<tr>
<td>ippStsContextMatchErr</td>
<td>Indicates an error when pointer to the spec structure is invalid.</td>
</tr>
<tr>
<td>ippStsNotSupportedModeErr</td>
<td>Indicates an error when the requested mode is not supported.</td>
</tr>
<tr>
<td>ippStsSizeErr</td>
<td>Indicates an error when width or height of the source or destination image is negative.</td>
</tr>
<tr>
<td>ippStsStepErr</td>
<td>Indicates an error when the step value is not data type multiple.</td>
</tr>
<tr>
<td>ippStsOutOfRangeErr</td>
<td>Indicates an error when the destination image offset point is outside the destination image origin.</td>
</tr>
<tr>
<td>ippStsSizeWrn</td>
<td>Indicates a warning when the destination image size is more than the destination image origin size.</td>
</tr>
</tbody>
</table>

**See Also**

ROI Processing in Geometric Transforms

ResizeGetSrcRoi Computes the ROI of the source image for resizing by tile processing.

ResizeGetSrcOffset Computes the offset of the source image for resizing by tile processing.

User-defined Border Types

ResizeGetBorderSize Computes the size of possible borders for the resize transform.

ResizeCubicInit Initializes the specification structure for image resizing using interpolation with two-parameter cubic filters.

ResizeGetBufferSize Computes the size of the external buffer for image resizing.

**ResizeLanczosInit**

*Initializes the specification structure for image resizing with the Lanczos interpolation method.*
Syntax
IppStatus ippiResizeLanczosInit_<mod>(IppiSize srcSize, IppiSize dstSize, Ipp32u numLobes, IppiResizeSpec_32f* pSpec, Ipp8u* pInitBuf);

Supported values for mod:
8u  16u  16s  32f

Platform-aware functions
IppStatus ippiResizeLanczosInit_L(IppiSizeL srcSize, IppiSizeL dstSize, IppDataType dataType, Ipp32u numLobes, IppiResizeSpec* pSpec, Ipp8u* pInitBuf);
IppStatus ippiResizeLanczosInit_8u_L(IppiSizeL srcSize, IppiSizeL dstSize, Ipp32u numLobes, IppHintAlgorithm hint, IppiResizeSpec* pSpec, Ipp8u* pInitBuf);

Threading layer (TL) functions
IppStatus ippiResizeLanczosInit_LT(IppiSizeL srcSize, IppiSizeL dstSize, IppDataType dataType, Ipp32u numChannels, Ipp32u numLobes, IppiResizeSpec_LT* pSpec, Ipp8u* pInitBuf);

Include Files
ippi.h

Flavors with the _LT suffix: ippi_tl.h
Flavors with the _L suffix: ippi_l.h

Domain Dependencies
Flavors declared in ippi.h:

Headers: ippcore.h, ippvm.h, ipps.h
Libraries: ippcore.lib, ippvm.lib, ipps.lib

Flavors declared in ippi_tl.h:


Parameters
srcSize
Size, in pixels, of the source image.
dstSize
Size, in pixels, of the destination image.
dataType
Data type of the image. Supported values: ipp8u, ipp16u, ipp16s, ipp32f, ipp64f.

hint
Computation algorithm for processing ipp8u data. Possible values are:
ippAlgHintNone The hint is absent. Equivalent to ippAlgHintFast.
ippAlgHintFast A faster but less accurate algorithm.
ippAlgHintAccurate A slower but more accurate algorithm.

numChannels
Number of image channels. Possible values are 1, 3, or 4.

numLobes
Parameter for Lanczos filters. Possible values are 2 or 3.
pSpec

Pointer to the specification structure for the resize filter.

pInitBuf

Pointer to the temporary buffer for the cubic filter initialization.

Description

This function initializes the specification structure for the resize algorithm with the Lanczos filter interpolation method. This method is based on the 2-lobed or 3-lobed Lanczos window function as an interpolation function, depending on the value of the numLobes parameter.

Before using this function, you need to calculate the size of the specification structure and the external buffer pInitBuf using theippiResizeGetSize function for the corresponding flavor.

NOTE

The function with the parameter hint allows users to choose between a faster but less accurate algorithm and a slower but more accurate one. Without the parameter hint this function initializes the specification structure for the faster but less accurate algorithm for Ipp8u data type.

Return Values

ippStsNoErr

Indicates no error.

ippStsNullPtrErr

Indicates an error when one of the specified pointers is NULL.

ippStsResizeNoOperation

Indicates an error when width or height of the image is equal to zero.

ippStsNotSupportedModeErr

Indicates an error when the requested mode is not supported.

ippStsDataTypeErr

Indicates an error if dataType has an illegal value.

ippStsExceededSizeErr

Indicates an error if width or height of the source or destination image exceeds 33554431 (0x1FFFFFF) (only for platform-aware and TL functions).

ippStsSizeErr

Indicates an error in the following cases:

• If width or height of the source or destination image is negative,
• If the source image size is less than the Lanczos interpolation filter size: 4x4 for the 2-lobed Lanczos function, or 6x6 for the 3-lobed Lanczos function.

See Also

ResizeGetSize Computes the size of the specification structure and the size of the external temporary buffer for the resize transform initialization.

ResizeLanczos

Changes an image size using interpolation with the Lanczos filter.
Syntax

IppStatusippiResizeLanczos_<mod>(const Ipp<datatype>* pSrc, Ipp32s srcStep, Ipp<datatype>* pDst, Ipp32s dstStep, Ipipoint dstOffset, IppiSize dstSize, IppiBorderType border, const Ipp<datatype>* pBorderValue, const IppiResizeSpec_32f* pSpec, Ipp8u* pBuffer);

Supported values for mod:

- 8u_C1R 16u_C1R 16s_C1R 32f_C1R
- 8u_C3R 16u_C3R 16s_C3R 32f_C3R
- 8u_C4R 16u_C4R 16s_C4R 32f_C4R

Platform-aware functions

IppStatusippiResizeLanczos_<mod>_L(const Ipp<datatype>* pSrc, IppSizeL srcStep, Ipp<datatype>* pDst, IppSizeL dstStep, IpipointL dstOffset, IppiSizeL dstSize, IppiBorderType border, const Ipp<datatype>* pBorderValue, const IppiResizeSpec* pSpec, Ipp8u* pBuffer);

Supported values for mod:

- 8u_C1R 16u_C1R 16s_C1R 32f_C1R
- 8u_C3R 16u_C3R 16s_C3R 32f_C3R
- 8u_C4R 16u_C4R 16s_C4R 32f_C4R

Threading layer (TL) functions

IppStatusippiResizeLanczos_<mod>_LT(const Ipp<datatype>* pSrc, IppSizeL srcStep, Ipp<datatype>* pDst, IppSizeL dstStep, IppiBorderType border, const Ipp<datatype>* pBorderValue, const IppiResizeSpec_LT* pSpec, Ipp8u* pBuffer);

Supported values for mod:

- 8u_C1R 16u_C1R 16s_C1R 32f_C1R
- 8u_C3R 16u_C3R 16s_C3R 32f_C3R
- 8u_C4R 16u_C4R 16s_C4R 32f_C4R

Include Files

ippi.h

Flavors with the _LT suffix: ippi_tl.h

Flavors with the _L suffix: ippi_l.h

Domain Dependencies

Flavors declared in ippi.h:

Headers: ippcore.h, ippvm.h, ipps.h

Libraries: ippcore.lib, ippvm.lib, ipps.lib

Flavors declared in ippi_tl.h:

Parameters

- **pSrc**: Pointer to the source image.
- **srcStep**: Distance, in bytes, between the starting points of consecutive lines in the source image buffer.
- **pDst**: Pointer to the destination image.
- **dstStep**: Distance, in bytes, between the starting points of consecutive lines in the destination image buffer.
- **dstOffset**: Offset of the tiled destination image with respect to the destination image origin.
- **dstSize**: Size of the destination image in pixels.
- **border**: Type of border. Possible values are:
  - ippBorderRepl: Border is replicated from the edge pixels.
  - ippBorderInMem: Border is obtained from the source image pixels in memory.
  - Mixed borders are also supported. They can be obtained by the bitwise operation OR between the ippBorderRepl type and the ippBorderInMemTop, ippBorderInMemBottom, ippBorderInMemLeft, ippBorderInMemRight types.
- **pBorderValue**: Pointer to the constant value to assign to pixels of the constant border. This parameter is applicable only to the ippBorderConst border type.
- **pSpec**: Pointer to the specification structure for the resize filter.
- **pBuffer**: Pointer to the work buffer.

Description

This function changes an image size using interpolation with the Lanczos filter. The image size can be either reduced or increased in each direction, depending on the destination image size.

This function operates with ROI. It resizes the source image ROI origin to the destination image ROI origin. You need to define the destination image ROI origin by the following parameters: the offset of the tiled destination image with respect to the destination image origin and the destination image size. The source image ROI origin is defined automatically. To obtain the source image ROI, use theippiResizeGetSrcRoi function with the corresponding mod value. To obtain the source image ROI origin offset, call theippiResizeGetSrcOffset function with the corresponding mod value. Parameters pSrc and pDst must point to the processed source and destination image ROI origins, respectively.

The interpolation algorithm applied uses edge pixels of the source image that are out of the image origin. The function ippiResizeLanczos uses in calculation the weighted values of these outer pixels. To obtain the size of the out of the source image origin, call theippiResizeGetBorderSize function for the corresponding flavor.

If the border type is equal to ippBorderRepl, the source image edge pixels are replicated out of the image origin. If the border type is equal to ippBorderInMem, the outer pixels are obtained from the out of the source image origin space. For the mixed border types, a combined approach is applied.
Before using the `ippiResizeLanczos` function, you need to initialize the specification structure using the `ippiResizeLanczosInit` function and compute the size of the external buffer `pBuffer` using the `ippiResizeGetBufferSize` function for the corresponding flavor.

### Return Values

- **ippStsNoErr**
  Indicates no error.
- **ippStsNullPtrErr**
  Indicates an error when one of the specified pointers is `NULL`.
- **ippStsNoOperation**
  Indicates a warning when width or height of the destination image is equal to zero.
- **ippStsBorderErr**
  Indicates an error when the `border` value is illegal.
- **ippStsContextMatchErr**
  Indicates an error when pointer to the specification structure is invalid.
- **ippStsNotSupportedModeErr**
  Indicates an error when the requested mode is not supported.
- **ippStsSizeErr**
  Indicates an error when width or height of the destination image is negative.
- **ippStsStepErr**
  Indicates an error when the step value is not data type multiple.
- **ippStsOutOfRangeErr**
  Indicates an error when the destination image offset point is outside the destination image origin.
- **ippStsSizeWrn**
  Indicates a warning when the destination image size is more than the destination image origin size.

### See Also

- **ROI Processing in Geometric Transforms**
- **ResizeGetSrcRoi** Computes the ROI of the source image for resizing by tile processing.
- **ResizeGetSrcOffset** Computes the offset of the source image for resizing by tile processing.
- **User-defined Border Types**
- **ResizeGetBorderSize** Computes the size of possible borders for the resize transform.
- **ResizeLanczosInit** Initializes the specification structure for image resizing with the Lanczos interpolation method.
- **ResizeGetBufferSize** Computes the size of the external buffer for image resizing.

### ResizeSuperInit

*Initializes the specification structure for image resizing with the super sampling interpolation method.*

#### Syntax

```c
IppStatus ippiResizeSuperInit_<mod>(IppiSize srcSize, IppiSize dstSize, IppiResizeSpec_32f* pSpec);
```

Supported values for `mod`:

- 8u
- 16u
- 16s
- 32f

#### Platform-aware function

```c
IppStatus ippiResizeSuperInit_L(IppiSizeL srcSize, IppiSizeL dstSize, IppDataType dataType, IppiResizeSpec* pSpec);
```
Threading layer (TL) function

IppStatus ippiResizeSuperInit_LT(IppSizeL srcSize, IppSizeL dstSize, IppDataType dataType, Ipp32u numChannels, IppiResizeSpec_LT* pSpec);

Include Files

ippi.h

Flavors with the _LT suffix: ippi_tl.h

Flavors with the _L suffix: ippi_l.h

Domain Dependencies

Flavors declared in ippi.h:

Headers: ippcore.h, ippvm.h, ipps.h

Libraries: ippcore.lib, ippvm.lib, ipps.lib

Flavors declared in ippi_tl.h:


Parameters

- srcSize: Size, in pixels, of the source image.
- dstSize: Size, in pixels, of the destination image.
- dataType: Data type of the image. Supported values: ipp8u, ipp16u, ipp16s, ipp32f, ipp64f.
- pSpec: Pointer to the specification structure for the resize filter.
- numChannels: Number of channels. Possible values are 1, 3, and 4.

Description

This function initializes the specification structure for the resize algorithm with the super sampling interpolation method. To calculate the size of the specification structure, call the ippiResizeGetSize function.

Return Values

- ippStsNoErr: Indicates no error.
- ippStsNullPtrErr: Indicates an error if one of the specified pointers is NULL.
- ippStsNoOperation: Indicates a warning if width or height of the image is equal to zero.
- ippStsSizeErr: Indicates an error in the following cases:
  - Indicates that one of the specified dimensions of the source image is less than the corresponding dimension of the destination image, or that the width or height of the source or destination image is negative.
  - If the width or height of the source or destination image is negative.
- ippStsExceededSizeErr: Indicates an error if width or height of the source or destination image exceeds 33554431 (0x1FFFFFF) (only for platform-aware and TL functions).
Indicates an error if `dataType` has an illegal value.

See Also

**ResizeGetSize** Computes the size of the specification structure and the size of the external temporary buffer for the resize transform initialization.

**ResizeSuper**

Changes an image size using the super sampling interpolation method.

Syntax

```c
IppStatus ippiResizeSuper_<mod>(const Ipp<datatype>* pSrc, Ipp32s srcStep, Ipp<datatype>* pDst, Ipp32s dstStep, IppiPoint dstOffset, IppiSize dstSize, const IppiResizeSpec_32f* pSpec, Ipp8u* pBuffer);
```

Supported values for `mod`:

- `8u_C1R`  `16u_C1R`  `16s_C1R`  `32f_C1R`
- `8u_C3R`  `16u_C3R`  `16s_C3R`  `32f_C3R`
- `8u_C4R`  `16u_C4R`  `16s_C4R`  `32f_C4R`

Platform-aware functions

```c
IppStatus ippiResizeSuper_<mod>_L(const Ipp<datatype>* pSrc, IppSizeL srcStep, Ipp<datatype>* pDst, IppSizeL dstStep, IppiPointL dstOffset, IppiSizeL dstSize, const IppiResizeSpec* pSpec, Ipp8u* pBuffer);
```

Supported values for `mod`:

- `8u_C1R`  `16u_C1R`  `16s_C1R`  `32f_C1R`
- `8u_C3R`  `16u_C3R`  `16s_C3R`  `32f_C3R`
- `8u_C4R`  `16u_C4R`  `16s_C4R`  `32f_C4R`

Threading layer (TL) functions

```c
IppStatus ippiResizeSuper_<mod>_LT(const Ipp<datatype>* pSrc, IppSizeL srcStep, Ipp<datatype>* pDst, IppSizeL dstStep, const IppiResizeSpec_LT* pSpec, Ipp8u* pBuffer);
```

Supported values for `mod`:

- `8u_C1R`  `16u_C1R`  `16s_C1R`  `32f_C1R`
- `8u_C3R`  `16u_C3R`  `16s_C3R`  `32f_C3R`
- `8u_C4R`  `16u_C4R`  `16s_C4R`  `32f_C4R`

Include Files

`ippi.h`

Flavors with the `_LT` suffix: `ippi_tl.h`

Flavors with the `_L` suffix: `ippi_l.h`
Domain Dependencies

Flavors declared in ippi.h:

Headers: ippcore.h, ippvm.h, ipps.h
Libraries: ippcore.lib, ippvm.lib, ipps.lib

Flavors declared in ippi_tl.h:


Parameters

- pSrc  
  Pointer to the source image.
- srcStep  
  Distance, in bytes, between the starting points of consecutive lines in the source image buffer.
- pDst  
  Pointer to the destination image.
- dstStep  
  Distance, in bytes, between the starting points of consecutive lines in the destination image buffer.
- dstOffset  
  Offset of the tiled destination image with respect to the destination image origin.
- dstSize  
  Size of the destination image in pixels.
- pSpec  
  Pointer to the specification structure for the resize filter.
- pBuffer  
  Pointer to the work buffer.

Description

This function changes an image size using the super sampling interpolation method. This method only reduces the image size.

This function operates with ROI. It resizes the source image ROI origin to the destination image ROI origin. You need to define the destination image ROI origin by the following parameters: the offset of the tiled destination image with respect to the destination image origin and the destination image size. The source image ROI origin is defined automatically. To obtain the source image ROI, use the ippiResizeGetSrcRoi function with the corresponding mod value. To obtain the source image ROI origin offset, call the ippiResizeGetSrcOffset function with the corresponding mod value. Parameters pSrc and pDst must point to the processed source and destination image ROI origins, respectively.

The interpolation algorithm applied uses only pixels of the source image origin that are inside of the image boundaries.

Before using the ippiResizeLinear function, you need to initialize the resize structure using the ippiResizeSuperInit function and compute the size of the external buffer pBuffer using the ippiResizeGetBufferSize function for the corresponding flavor.

NOTE

You can get better performance if you use the following scaling factors along the x and y axes: 1/2, 2/3, 3/4, 4/5, 5/6, 8/9, 1/3, 2/5, 3/5, 3/7, 4/9, 7/10, 1/4, 2/7, 3/8, 1/8.

Return Values

- ippStsNoErr  
  Indicates no error.
ippStsNullPtrErr Indicates an error when one of the specified pointers is NULL.
ippStsNoOperation Indicates a warning when width or height of the destination image is equal to zero.
ippStsContextMatchErr Indicates an error when pointer to the spec structure is invalid.
ippStsStepErr Indicates an error when the step value is not data type multiple.
ippStsOutOfRangeErr Indicates an error when the destination image offset point is outside the destination image origin.
ippStsSizeWrn Indicates a warning when the destination image size is more than the destination image origin size.

**See Also**
ROI Processing in Geometric Transforms
ResizeGetSrcRoi Computes the ROI of the source image for resizing by tile processing.
ResizeGetSrcOffset Computes the offset of the source image for resizing by tile processing.
ResizeSuperInit Initializes the specification structure for image resizing with the super sampling interpolation method.
ResizeGetBufferSize Computes the size of the external buffer for image resizing.

**ResizeAntialiasingLinearInit**
*Initializes the specification structure for image resizing with antialiasing using linear interpolation.*

**Syntax**
```
IppStatus ippiResizeAntialiasingLinearInit(IppiSize srcSize, IppiSize dstSize, IppiResizeSpec_32f* pSpec, Ipp8u* pInitBuf);
```

**Platform-aware function**
```
IppStatus ippiResizeAntialiasingLinearInit_L(IppiSizeL srcSize, IppiSizeL dstSize, IppDataType dataType, IppiResizeSpec* pSpec, Ipp8u* pInitBuf);
```

**Threading layer (TL) function**
```
IppStatus ippiResizeAntialiasingLinearInit_LT(IppiSizeL srcSize, IppiSizeL dstSize, IppDataType dataType, Ipp32u numChannels, IppiResizeSpec_LT* pSpec, Ipp8u* pInitBuf);
```

**Include Files**
ippi.h

Flavors with the _LT suffix: ippi_tl.h

Flavors with the _L suffix: ippi_l.h

**Domain Dependencies**
Flavors declared in ippi.h:

Headers: ippcore.h, ippvm.h, ipps.h
Libraries: ippcore.lib, ippvm.lib, ipps.lib

Flavors declared in ippi_tl.h:
Parameters

- **srcSize**: Size of the source image, in pixels.
- **dstSize**: Size of the destination image, in pixels.
- **dataType**: Data type of the image. Supported values: ipp8u, ipp16u, ipp16s, ipp32f, ipp64f.
- **numChannels**: Number of image channels. Possible values: 1, 3, or 4.
- **pSpec**: Pointer to the specification structure for the resize filter.
- **pInitBuf**: Pointer to the temporary buffer for initialization of the linear filter.

Description

This function initializes the IppiResizeSpec_32f structure for the resize operation with antialiasing using the linear interpolation method. Before using this function, calculate the size of the temporary buffer and specification structure using the ippiResizeGetSize function with the antialiasing parameter equal to 1.

Return Values

- **ippStsNoErr**: Indicates no error.
- **ippStsNullPtrErr**: Indicates an error when one of the specified pointers is NULL.
- **ippStsNoOperation**: Indicates a warning when width or height of the image is equal to zero.
- **ippStsSizeErr**: Indicates an error if width or height of the source or destination image is negative.
- **ippStsExceededSizeErr**: Indicates an error if width or height of the source or destination image exceeds 33554431 (0x1FFFFFF).
- **ippStsDataTypeErr**: Indicates an error if dataType has an illegal value.

See Also

- **ResizeGetSize**: Computes the size of the specification structure and the size of the external temporary buffer for the resize transform initialization.
- **Linear Interpolation**

**ResizeAntialiasingCubicInit**

Initializes the specification structure for image resizing with antialiasing using interpolation with the two-parameter cubic filters.

Syntax

```c
IppStatus ippiResizeAntialiasingCubicInit(IppiSize srcSize, IppiSize dstSize, Ipp32f valueB, Ipp32f valueC, IppiResizeSpec_32f* pSpec, Ipp8u* pInitBuf);
```

Platform-aware function

```c
IppStatus ippiResizeAntialiasingCubicInit_L(IppiSizeL srcSize, IppiSizeL dstSize, IppDataType dataType, Ipp32f valueB, Ipp32f valueC, IppiResizeSpec* pSpec, Ipp8u* pInitBuf);
```
Threading layer (TL) function

IppStatus ippiResizeAntialiasingCubicInit_LT(IppSizeL srcSize, IppSizeL dstSize, IppDataType dataType, Ipp32u numChannels, Ipp32f valueB, Ipp32f valueC, IppResizeSpec_LT* pSpec, Ipp8u* pInitBuf);

Include Files

ippi.h

Flavors with the _LT suffix: ippi_tl.h

Flavors with the _L suffix: ippi_l.h

Domain Dependencies

Flavors declared in ippi.h:

Headers: ippcore.h, ippvm.h, ipps.h
Libraries: ippcore.lib, ippvm.lib, ipps.lib

Flavors declared in ippi_tl.h:


Parameters

srcSize        Size of the source image, in pixels.
dstSize        Size of the destination image, in pixels.
valueB         The first parameter for cubic filters.
valueC         The second parameter for cubic filters.
numChannels    Number of image channels. Possible values: 1, 3, or 4.
dataType       Data type of the image. Supported values: ipp8u, ipp16u, ipp16s, ipp32f, ipp64f.
pSpec          Pointer to the specification structure for the resize filter.
pInitBuf       Pointer to the temporary buffer for initialization of the cubic filter.

Description

This function initializes the IppiResizeSpec_32f structure for the resize operation with antialiasing using interpolation with the two-parameter cubic filters.

Before using this function, calculate the size of the temporary buffer and specification structure using the ippiResizeGetSize function with the antialiasing parameter equal to 1.

Return Values

ippStsNoErr       Indicates no error.
ippStsNullPtrErr  Indicates an error when one of the specified pointers is NULL.
ippStsNoOperation Indicates a warning when width or height of the image is equal to zero.
ippStsSizeErr     Indicates an error if width or height of the source or destination image is negative.
Indicates an error if width or height of the source or destination image exceeds 33554431 (0x1FFFFFF) (only for platform-aware and TL functions).

Indicates an error if dataType has an illegal value.

See Also
ResizeGetSize Computes the size of the specification structure and the size of the external temporary buffer for the resize transform initialization.

Cubic Interpolation

**ResizeAntialiasingLanczosInit**

Initializes the specification structure for image resizing with antialiasing using interpolation with the Lanczos filter.

**Syntax**

```c
IppStatus ippiResizeAntialiasingLanczosInit(IppiSize srcSize, IppiSize dstSize, Ipp32u numLobes, IppiResizeSpec_32f* pSpec, Ipp8u* pInitBuf);
```

**Platform-aware function**

```c
IppStatus ippiResizeAntialiasingLanczosInit_L(IppiSizeL srcSize, IppiSizeL dstSize, IppDataType dataType, Ipp32u numLobes, IppiResizeSpec* pSpec, Ipp8u* pInitBuf);
```

**Threading layer (TL) function**

```c
IppStatus ippiResizeAntialiasingLanczosInit_LT(IppiSizeL srcSize, IppiSizeL dstSize, IppDataType dataType, Ipp32u numChannels, Ipp32u numLobes, IppiResizeSpec_LT* pSpec, Ipp8u* pInitBuf);
```

**Include Files**

- ippi.h

**Domain Dependencies**

- **Parameters**
  - `srcSize` Size of the source image, in pixels.
  - `dstSize` Size of the destination image, in pixels.
  - `numLobes` Number of lobes for the Lanczos window. Possible values: 2 or 3.
  - `numChannels` Number of image channels. Possible values: 1, 3, or 4.
  - `dataType` Data type of the image. Supported values: `ipp8u`, `ipp16u`, `ipp16s`, `ipp32f`, `ipp64f`. 
pSpec

Pointer to the specification structure for the resize filter.

pInitBuf

Pointer to the temporary buffer for initialization of the cubic filter.

Description

This function initializes the IppiResizeSpec_32f structure for the resize operation with antialiasing using interpolation with the Lanczos filter. The Lanczos interpolation method is based on the 2-lobed or 3-lobed Lanczos window function as an interpolation function depending on the value of the numLobes parameter.

Before using this function, calculate the size of the temporary buffer and specification structure using the ippiResizeGetSize function with the antialiasing parameter equal to 1.

Return Values

ippStsNoErr

Indicates no error.

ippStsNullPtrErr

Indicates an error when one of the specified pointers is NULL.

ippStsNoOperation

Indicates a warning when width or height of the image is equal to zero.

ippStsSizeErr

Indicates an error if width or height of the source or destination image is negative.

ippStsExceededSizeErr

Indicates an error if width or height of the source or destination image exceeds 33554431 (0x1FFFFFF) (only for platform-aware and TL functions).

ippStsDataTypeErr

Indicates an error if dataType has an illegal value.

See Also

ResizeGetSize Computes the size of the specification structure and the size of the external temporary buffer for the resize transform initialization.

Lanczos Interpolation

ResizeAntialiasing

Changes an image size using the chosen interpolation method with antialiasing.

Syntax

IppStatus ippiResizeAntialiasing_<data_type>_<chan>(const Ipp<data_type>* pSrc, Ipp32s srcStep, Ipp<data_type>* pDst, Ipp32s dstStep, IppiPoint dstOffset, IppiSize dstSize, IppiBorderType border, const Ipp<data_type>* pBorderValue, const IppiResizeSpec_32f* pSpec, Ipp8u* pBuffer);

Supported values for data_type:

8u  16u  16s  32f

Supported values for chan:

C1R  C3R  C4R
Platform-aware functions

IppStatus ippiResizeAntialiasing_<data_type>_<mod>_L(const Ipp<data_type>** pSrc, IppSizeL srcStep, Ipp<data_type>** pDst, IppSizeL dstStep, IppiPointL dstOffset, IppiSizeL dstSize, IppiBorderType border, const Ipp<data_type>** pBorderValue, const IppiResizeSpec* pSpec, Ipp8u* pBuffer);

Supported values for data_type:

\begin{align*}
8u & \quad 16u & \quad 16s & \quad 32f \\
\end{align*}

Supported values for mod:

\begin{align*}
C1R & \quad C3R & \quad C4R \\
\end{align*}

Threading layer (TL) functions

IppStatus ippiResizeAntialiasing_<data_type>_<mod>_LT(const Ipp<data_type>** pSrc, IppSizeL srcStep, Ipp<data_type>** pDst, IppSizeL dstStep, IppiBorderType border, const Ipp<data_type>** pBorderValue, const IppiResizeSpec_LT* pSpec, Ipp8u* pBuffer);

Supported values for data_type:

\begin{align*}
8u & \quad 16u & \quad 16s & \quad 32f \\
\end{align*}

Supported values for mod:

\begin{align*}
C1R & \quad C3R & \quad C4R \\
\end{align*}

Include Files

ippi.h

Flavors with the _LT suffix: ippi_tl.h

Flavors with the _L suffix: ippi_l.h

Domain Dependencies

Flavors declared in ippi.h:

Headers: ippcore.h, ippvm.h, ipps.h

Libraries: ippcore.lib, ippvm.lib, ipps.lib

Flavors declared in ippi_tl.h:


Parameters

\begin{align*}
pSrc \\
srcStep \\
pDst \\
\end{align*}

Pointer to the source image.

Distance, in bytes, between the starting points of consecutive lines in the source image buffer.

Pointer to the destination image.
**dstStep**
Distance, in bytes, between the starting points of consecutive lines in the destination image buffer.

**dstOffset**
Offset of the tiled destination image with respect to the destination image origin.

**dstSize**
Size of the destination image, in pixels.

**border**
Type of border. Possible values are:
- ippBorderRepl: Border is replicated from the edge pixels.
- ippBorderConst: Border pixels are set to constants.
- ippBorderInMem: Border is obtained from the source image pixels in memory.

Mixed borders are also supported. They can be obtained by the bitwise operation OR between the ippBorderRepl type and the ippBorderInMemTop, ippBorderInMemBottom, ippBorderInMemLeft, ippBorderInMemRight types.

**pBorderValue**
Pointer to the constant value to assign to pixels of the constant border. This parameter is applicable only to the ippBorderConst border type.

**pSpec**
Pointer to the specification structure for the resize filter.

**pBuffer**
Pointer to the external buffer.

**Description**

The `ippiResizeAntialiasing` function changes the size of an image using the chosen interpolation method with antialiasing. The interpolation method to be applied is defined by the function that you use for the resize filter initialization. Use this function to reduce the image size with minimization of moire artifacts. For more information about the implemented algorithm, refer to [SCHU92].

If you use `ippiResizeAntialiasing` to increase the image size, the function applies the same algorithm as one of the following resize functions, depending on the interpolation type chosen at the initialization stage: `ippiResizeLinear`, `ippiResizeCubic`, or `ippiResizeLanczos`.

This function operates with ROI. It resizes the source image ROI origin to the destination image ROI origin. Define the destination image ROI origin by the following parameters: the offset of the tiled destination image with respect to the destination image origin and the destination image size. The source image ROI origin is defined automatically. To obtain the source image ROI, use the corresponding flavor of the `ippiResizeGetSrcRoi` function. To obtain the source image ROI origin offset, call the corresponding flavor of the `ippiResizeGetSrcOffset` function. Parameters `pSrc` and `pDst` must point to the processed source and destination image ROI origins, respectively.

The interpolation algorithm applied uses edge pixels of the source image that are out of the image origin. The `ippiResizeAntialiasing` function uses the weighted values of these outer pixels. To obtain the size of the out of the source image origin, call the corresponding flavor of the `ippiResizeGetBorderSize` function.

Before using the `ippiResizeAntialiasing` function, you need to initialize the `IpipResizeSpec_32f` structure using one of the following functions, depending on the interpolation method to be applied: `ippiResizeAntialiasingLinearInit`, `ippiResizeAntialiasingCubicInit`, or `ippiResizeAntialiasingLanczosInit`, and compute the size of the external buffer `pBuffer` using the corresponding flavor of the `ippiResizeGetBufferSize` function.

For more information about the supported border types, see [User-defined Border Types](#).
Return Values

ippStsNoErr  Indicates no error.
ippStsNullPtrErr  Indicates an error when one of the specified pointers is NULL.
ippStsNoOperation  Indicates a warning when width or height of the destination image is equal to zero.
ippStsBorderErr  Indicates an error when the border value is illegal.
ippStsContextMatchErr  Indicates an error when a pointer to the specification structure is invalid.
ippStsNotSupportedModeErr  Indicates an error when the requested mode is not supported.
ippStsSizeErr  Indicates an error when width or height of the destination image is negative.
ippStsStepErr  Indicates an error when the step value is not data type multiple.
ippStsOutOfRangeErr  Indicates an error when the destination image offset point is outside the destination image origin.
ippStsSizeWrn  Indicates a warning when the destination image size is more than the destination image origin size.

Example

To better understand usage of the ippiResizeAntialiasing function, refer to the following example in the examples archive available for download from https://software.intel.com/en-us/ipp-manual-examples:

ResizeAntialiasing.c

The figure below demonstrates the results of reducing a 1751x1044 image by five times with antialiasing (a) and without (b).

See Also
ROI Processing in Geometric Transforms
User-defined Border Types
ResizeGetSrcRoi  Computes the ROI of the source image for resizing by tile processing.
ResizeGetSrcOffset  Computes the offset of the source image for resizing by tile processing.
ResizeGetBorderSize  Computes the size of possible borders for the resize transform.
ResizeGetBufferSize  Computes the size of the external buffer for image resizing.
ResizeAntialiasingLinearInit  Initializes the specification structure for image resizing with antialiasing using linear interpolation.
ResizeAntialiasingCubicInit  Initializes the specification structure for image resizing with antialiasing using interpolation with the two-parameter cubic filters.
ResizeAntialiasingLanczosInit  Initializes the specification structure for image resizing with antialiasing using interpolation with the Lanczos filter.
ResizeFilterGetSize  Calculates the size of the state structure for resizing filter.
Syntax
IppStatus ippiResizeFilterGetSize_8u_C1R(IppiSize srcRoiSize, IppiSize dstRoiSize, IppiResizeFilterType filter, Ipp32u* pSize);

Include Files
ippi.h

Domain Dependencies
Headers: ippcore.h, ippvm.h, ipps.h
Libraries: ippcore.lib, ippvm.lib, ipps.lib

Parameters
srcRoiSize
Size of the source image ROI in pixels.

dstRoiSize
Size of the destination image ROI in pixels.

filter
Type of filter used in resizing; possible values:
ippiResizeFilterHann, ippiResizeFilterLanczos.

pSize
Pointer to the size (in bytes) of the state structure.

Description
This function operates with ROI (see ROI Processing in Geometric Transforms).
This function calculates the size pSize of the state structure required for the function ippiResizeFilter.
The type of filter is specified by the parameter filter.

Return Values
ippStsNoErr
Indicates no error. Any other value indicates an error.

ippStsNullPtrErr
Indicates an error condition if the pSize pointer is NULL.

ippStsSizeErr
Indicates an error condition if srcRoiSize or dstRoiSize has a field with zero or negative value.

ippStsNotSupportedModeErr
Indicates an error condition if filter has an invalid value.

ResizeFilterInit
Initializes the state structure for the resize filter.

Syntax
IppStatus ippiResizeFilterInit_8u_C1R(IppiResizeFilterState* pState, IppiSize srcRoiSize, IppiSize dstRoiSize, IppiResizeFilterType filter);

Include Files
ippi.h

Domain Dependencies
Headers: ippcore.h, ippvm.h, ipps.h
Libraries: ippcore.lib, ippvm.lib, ipps.lib
Parameters

- pState: Pointer to the state structure for resize filter.
- srcRoiSize: Size of the source image ROI in pixels.
- dstRoiSize: Size of the destination image ROI in pixels.
- filter: Type of filter used in resizing; possible values: ippResizeFilterHann, ippResizeFilterLanczos

Description

This function operates with ROI (see ROI Processing in Geometric Transforms).

This function initializes the state structure pState for the resizing filter used by the function ippiResizeFilter. The size of the structure must be computed by the function ippiResizeFilterGetSize beforehand. The type of filter is specified by the parameter filter and it must be the same as in the function ippiResizeFilterGetSize.

Return Values

- ippStsNoErr: Indicates no error. Any other value indicates an error.
- ippStsNullPtrErr: Indicates an error condition if the pState pointer is NULL.
- ippStsSizeErr: Indicates an error condition if srcRoiSize or dstRoiSize has a field with zero or negative value.
- ippStsNotSupportedModeErr: Indicates an error condition if filter has an invalid value.

ResizeFilter

Changes the size of an image using a generic filter.

Syntax

IppStatus ippiResizeFilter_8u_C1R(const Ipp8u* pSrc, int srcStep, IppiSize srcRoiSize, Ipp8u* pDst, int dstStep, IppiSize dstRoiSize, IppiResizeFilterState* pState);

Include Files

ippi.h

Domain Dependencies

Headers: ippcore.h, ippvm.h, ipps.h
Libraries: ippcore.lib, ippvm.lib, ipps.lib

Parameters

- pSrc: Pointer to the source image ROI.
- srcStep: Distance in bytes between starts of consecutive lines in the source image buffer.
- srcRoiSize: Size of the source image ROI in pixels.
- pDst: Pointer to the destination image ROI.
- dstStep: Distance in bytes between starts of consecutive lines in the destination image buffer.
dstRoiSize  
Size of the destination image ROI in pixels.

pState  
Pointer to the state structure for the resize filter.

**Description**

This function operates with ROI (see ROI Processing in Geometric Transforms).

This function resizes the source image *pSrc* using the special generic filters. The state structure *pState* contains the parameters of filtering and must be initialized by the function `ippiResizeFilterInit` beforehand.

**Return Values**

- **ippStsNoErr**: Indicates no error. Any other value indicates an error.
- **ippStsNullPtrErr**: Indicates an error condition if one of the specified pointers is NULL.
- **ippStsStepErr**: Indicates an error condition if *srcStep* or *dstStep* has a zero or negative value.
- **ippStsSizeErr**: Indicates an error condition if *srcRoiSize* or *dstRoiSize* has a field with zero or negative value.
- **ippStsContextMatchErr**: Indicates an error condition if a pointer to an invalid state structure is passed.

**ResizeYUV420GetSize**

*Computes sizes of the spec structure and the external buffer for the NV12 resize transform initialization.*

**Syntax**

```c
IppStatus ippiResizeYUV420GetSize(IppiSize srcSize, IppiSize dstSize, IppiInterpolationType interpolation, Ipp32u antialiasing, Ipp32s* pSpecSize, Ipp32s* pInitBufSize);
```

**Include Files**

`ippi.h`

**Domain Dependencies**

*Headers:* `ippcore.h`, `ippvm.h`, `ipps.h`

*Libraries:* `ippcore.lib`, `ippvm.lib`, `ipps.lib`

**Parameters**

- **srcSize**: Size of the source image in pixels.
- **dstSize**: Size of the destination image in pixels.
- **interpolation**: Interpolation method. Supported values are `ippLanczos` and `ippSuper`.
- **antialiasing**: Antialiasing method.
- **pSpecSize**: Pointer to the size in bytes of the spec structure.
- **pInitBufSize**: Pointer to the size in bytes of the temporal buffer.
Description
This function computes sizes of the spec structure and the external buffer that are required for one of the following functions depending on the interpolation method parameter: ResizeYUV420LanczosInit and ResizeYUV420SuperInit.
The size of the 2-Lobed Lanczos filter is 8x8.

NOTE
Antialiasing is currently not supported. The value for the antialiasing parameter must be equal to zero.

Return Values

ippStsNoErr
Indicates no error. Any other value indicates an error.

ippStsNullPtrErr
Indicates an error condition if one of the specified pointers is NULL.

ippStsNoOperation
Indicates a warning if width or height of the image is equal to zero.

ippStsSizeErr
Indicates an error in the following cases:
- if width or height of the image is equal to 1,
- if the source image size is less than a filter size of the chosen interpolation method (except ippSuper),
- if one of the specified dimensions of the source image is less than the corresponding dimension of the destination image (for ippSuper method only),
- if width or height of the source or destination image is negative,
- if one of the calculated sizes exceeds maximum 32 bit signed integer positive value (the size of the one of the processed images is too large).

ippStsSizeWrn
Indicates a warning if width or height of the image is odd.

ippStsInterpolationErr
Indicates an error if interpolation has an illegal value.

ippStsNoAntialiasing
Indicates a warning if the specified interpolation method does not support antialiasing.

ippStsNotSupportedModeErr
Indicates an error if the requested mode is currently not supported.

ResizeYUV420GetSrcRoi
Computes the ROI of the source image for NV12 resizing by tile processing.

Syntax
Single precision
IppStatusippiResizeYUV420GetSrcRoi(const IppiResizeYUV420Spec*pSpec, IppiPoint dstRoiOffset, IppiSize dstRoiSize, IppiPoint* srcRoiOffset, IppiSize* srcRoiSize);
Include Files
ippi.h

Domain Dependencies
Headers: ippicore.h, ippvm.h, ipps.h
Libraries: ippcore.lib, ippvm.lib, ipps.lib

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>pSpec</td>
<td>Pointer to the spec structure for the resize filter.</td>
</tr>
<tr>
<td>dstRoiOffset</td>
<td>Offset of the tiled destination image ROI.</td>
</tr>
<tr>
<td>dstRoiSize</td>
<td>Size of the tiled destination image ROI.</td>
</tr>
<tr>
<td>srcRoiOffset</td>
<td>Offset of the source image ROI.</td>
</tr>
<tr>
<td>srcRoiSize</td>
<td>Pointer to the size of the source image ROI.</td>
</tr>
</tbody>
</table>

Description

This function computes the ROI of the processed source image using the processed ROI of the destination image for the corresponding resize transform by tile processing. The pSpec parameter defines the resize algorithm parameters. Prior to using the ippiResizeYUV420GetSrcRoi function, you need to initialize the pSpec parameter by calling one of the following functions: ippiResizeYUV420LanczosInit or ippiResizeYUV420SuperInit functions.

NOTE
If the destination ROI size exceeds the image origin, the source ROI will be obtained for an intersection of the destination ROI and image origin.

Return Values

<table>
<thead>
<tr>
<th>Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ippStsNoErr</td>
<td>Indicates no error.</td>
</tr>
<tr>
<td>ippStsNullPtrErr</td>
<td>Indicates an error if one of the specified pointers is NULL.</td>
</tr>
<tr>
<td>ippStsContextMatchErr</td>
<td>Indicates an error if pointer to the spec structure is invalid.</td>
</tr>
<tr>
<td>ippStsOutOfRangeErr</td>
<td>Indicates an error if the destination image offset point is outside the destination image origin.</td>
</tr>
<tr>
<td>ippStsMisalignedOffsetErr</td>
<td>Indicates an error if one of the fields of the dstRoiOffset parameter is odd.</td>
</tr>
<tr>
<td>ippStsSizeErr</td>
<td>Indicates an error if one of the fields of the dstRoiSize is less than 2.</td>
</tr>
<tr>
<td>ippStsSizeWrn</td>
<td>Indicates a warning if the destination ROI exceeds the destination image origin or contains odd values.</td>
</tr>
</tbody>
</table>

ResizeYUV420LanczosInit

Initializes the spec structure for the NV12 resize transform by interpolation with the Lanczos filter.
Syntax

IppStatus ippiResizeYUV420LanczosInit(IppiSize srcSize, IppiSize dstSize, Ipp32u numLobes, IppiResizeYUV420Spec* pSpec, Ipp8u* pInitBuf);

Include Files

ippi.h

Domain Dependencies

Headers: ippcore.h, ippvm.h, ipps.h
Libraries: ippcore.lib, ippvm.lib, ipps.lib

Parameters

srcSize  Size in pixels of the source image.
dstSize  Size in pixels of the destination image.
numLobes Parameter for Lanczos filters. Possible values are 2 or 3.
pSpec  Pointer to the spec structure for the resize filter.
pInitBuf  Pointer to the temporal buffer for the cubic filter initialization.

Description

This function initializes the IppiResizeYUV420Spec structure for the resize algorithm with the Lanczos filter interpolation method. This method is based on the 2-lobed or 3-lobed Lanczos window function as an interpolation function depending on the value of the numLobes parameter.

To calculate the size of the spec structure object, call the ippiResizeYUV420GetSize function.

The function ippiResizeYUV420LanczosInit requires the external buffer pInitBuf. Prior to using this function, you need to call ippiResizeYUV420GetSize for the corresponding flavors to compute the size of the buffer.

Return Values

ippStsNoErr  Indicates no error.
ippStsNullPtrErr  Indicates an error if one of the specified pointers is NULL.
ippStsResizeNoOperation  Indicates an error if width or height of the image is equal to zero.
ippStsSizeWrn  Indicates a warning if width or height of the image is odd.
ippStsSizeErr  Indicates an error in the following cases:
• if width or height of the source or destination image is equal to 1,
• if width or height of the source or destination image is negative,
• if the source image size is less than the Lanczos interpolation filter size: 8x8 for 2-lobed Lanczos function, or 12x12 for 3-lobed Lanczos function.
ippStsNotSupportedModeErr  Indicates an error if the requested mode is not supported.
**ResizeYUV420SuperInit**

_Initializes the spec structure for the NV12 resize transform by interpolation with the super sampling algorithm._

**Syntax**

```c
IppStatus ippiResizeYUV420SuperInit(IppiSize srcSize, IppiSize dstSize, IppiResizeYUV420Spec* pSpec);
```

**Include Files**

ippi.h

**Domain Dependencies**

Headers: ippcore.h, ippvm.h, ipps.h

Libraries: ippcore.lib, ippvm.lib, ipps.lib

**Parameters**

- `srcSize`: Size in pixels of the source image.
- `dstSize`: Size in pixels of the destination image.
- `pSpec`: Pointer to the spec structure for the resize filter.

**Description**

This function initializes the `IppiResizeYUV420Spec` structure for the resize algorithm using interpolation with the super sampling algorithm.

To calculate the size of the spec structure object, call the `ippiResizeYUV420GetSize` function.

**Return Values**

- `ippStsNoErr`: Indicates no error.
- `ippStsNullPtrErr`: Indicates an error if one of the specified pointers is `NULL`.
- `ippStsResizeNoOperation`: Indicates an error if width or height of the image is equal to zero.
- `ippStsSizeWrn`: Indicates a warning if width or height of the image is odd.
- `ippStsSizeErr`: Indicates an error in the following cases:
  - if width or height of the image is equal to 1,
  - if one of the specified dimensions of the source image is less than the corresponding dimension of the destination image,
  - if width or height of the source or destination image is negative.

**ResizeYUV420GetBorderSize**

_Computes the size of possible borders for the NV12 resize transform._

**Syntax**

```c
IppStatus ippiResizeYUV420GetBorderSize(const IppiResizeYUV420Spec* pSpec, IppiBorderSize* borderSize);
```
Include Files
ippi.h

Domain Dependencies
Headers: ippcore.h, ippvm.h, ipps.h
Libraries: ippcore.lib, ippvm.lib, ipps.lib

Parameters

pSpec Pointer to the spec structure for the resize filter.
borderSize Size in pixels of necessary borders.

Description
This function computes the size of the source image ROI that is used by the corresponding resize transform and is out of the processing boundaries for Luma and Chroma planes. The pSpec parameter defines the resize algorithm parameters. Prior to using theippiResizeYUV420GetBorderSize function, you need to initialize the pSpec parameter by calling one of the following functions:ippiResizeYUV420LanczosInit orippiResizeYUV420SuperInit.

NOTE
The returned border size is in Luma/Chroma plane pixels. This means that the chosen resize algorithm uses the returned outer size of the source image ROI for each plane.

Return Values

ippStsNoErr Indicates no error.
ippStsNullPtrErr Indicates an error if one of the specified pointers is NULL.
ippStsContextMatchErr Indicates an error if pointer to the spec structure is invalid.

ResizeYUV420GetSrcOffset
Computes the offset of the source image for the NV12 resize transform by tile processing.

Syntax
IppStatusippiResizeYUV420GetSrcOffset(const IppiResizeYUV420Spec* pSpec, IppiPoint dstOffset, IppiPoint* srcOffset);

Include Files
ippi.h

Domain Dependencies
Headers: ippcore.h, ippvm.h, ipps.h
Libraries: ippcore.lib, ippvm.lib, ipps.lib

Parameters

pSpec Pointer to the spec structure for the resize filter.
**Description**

This function computes the offset of the processed source image ROI using the offset of the processed destination image ROI for the corresponding resize transform by tile processing. The `pSpec` parameter defines the resize algorithm parameters. Prior to using the `ippiResizeGetSrcOffset` function, you need to initialize the `pSpec` parameter by calling one of the following functions: `ippiResizeYUV420LanczosInit` and `ippiResizeYUV420SuperInit`.

**Return Values**

- `ippStsNoErr`: Indicates no error.
- `ippStsNullPtrErr`: Indicates an error if one of the specified pointers is NULL.
- `ippStsContextMatchErr`: Indicates an error if pointer to the spec structure is invalid.
- `ippStsOutOfRangeErr`: Indicates an error if the destination image offset point is outside the destination image origin.
- `ippStsMisalignedOffsetErr`: Indicates an error if one of the fields of the `dstOffset` parameter is odd.

**ResizeYUV420GetBufferSize**

*Computes the size of the external buffer for the NV12 resize transform.*

**Syntax**

```c
IppStatus ippiResizeYUV420GetBufferSize(const IppiResizeYUV420Spec* pSpec, IppiSize dstSize, Ipp32s* pBufSize);
```

**Include Files**

`ippi.h`

**Domain Dependencies**

- **Headers**: `ippicore.h`, `ippvm.h`, `ipps.h`
- **Libraries**: `ippicore.lib`, `ippvm.lib`, `ipps.lib`

**Parameters**

- `pSpec`: Pointer to the spec structure for the resize filter.
- `dstSize`: Size in pixels of the destination image.
- `pBufSize`: Pointer to the size in bytes of the external buffer.

**Description**

This function computes the size of the external buffer for the NV12 resize transform. The `pSpec` parameter defines the resize algorithm parameters. Prior to using the `ippiResizeYUV420GetBufferSize` function, you need to initialize the `pSpec` parameter by calling one of the following functions: `ippiResizeYUV420LanczosInit` and `ippiResizeYUV420SuperInit`. 
Return Values

- **ippStsNoErr**: Indicates no error.
- **ippStsNullPtrErr**: Indicates an error if `pBufferSize` pointer is NULL.
- **ippStsNoOperation**: Indicates a warning if width or height of the destination image is equal to zero.
- **ippStsContextMatchErr**: Indicates an error if pointer to the spec structure is invalid.
- **ippStsSizeWrn**: Indicates a warning in the following cases:
  - if width or height of the image is odd,
  - if the destination image size is more than the destination image origin size.
- **ippStsSizeErr**: Indicates an error in the following cases:
  - if width or height of the image is equal to 1,
  - if width or height of the destination image is negative,
  - if the calculated buffer size exceeds maximum 32 bit signed integer positive value (the processed image size is too large).

**ResizeYUV420Lanczos**

*Changes the size of the NV12 image by interpolation with the Lanczos filter.*

**Syntax**

```c
IppStatus ippiResizeYUV420Lanczos_8u_P2R(const Ipp8u* pSrcY, Ipp32s srcYStep, const Ipp8u* pSrcUV, Ipp32s srcUVStep, Ipp8u* pDstY, Ipp32s dstYStep, Ipp8u* pDstUV, Ipp32s dstUVStep, IppiPoint dstOffset, IppiSize dstSize, IppiBorderType border, const Ipp8u* pBorderValue, const IppiResizeYUV420Spec* pSpec, Ipp8u* pBuffer);
```

**Include Files**

ippi.h

**Domain Dependencies**

- Headers: ippcore.h, ippvm.h, ipps.h
- Libraries: ippcore.lib, ippvm.lib, ipps.lib

**Parameters**

- **pSrcY**: Pointer to the source image Y plane.
- **srcYStep**: Distance in bytes between starts of consecutive lines in the source image Y plane.
- **pSrcUV**: Pointer to the source image UV plane.
- **srcUVStep**: Distance in bytes between starts of consecutive lines in the source image UV plane.
- **pDstY**: Pointer to the destination image Y plane.
- **dstYStep**: Distance in bytes between starts of consecutive lines in the destination image Y plane.
**pDstUV**  
Pointer to the destination image UV plane.

**dstUVStep**  
Distance in bytes between starts of consecutive lines in the destination image UV plane.

**dstOffset**  
Offset of the tiled destination image with respect to the destination image origin.

**dstSize**  
Size of the destination image in pixels.

**border**  
Type of the border.

**pBorderValue**  
Pointer to the constant value to assign to pixels of the constant border. This parameter is applicable only to the `ippBorderConst` border type.

**pSpec**  
Pointer to the spec structure for the resize filter.

**pBuffer**  
Pointer to the work buffer.

**Description**

This function changes an image size using interpolation with the Lanczos filter. The image size can be either reduced or increased in each direction, depending on the destination image size.

This function operates with ROI (see ROI Processing in Geometric Transforms). It resizes the source image ROI origin to the destination image ROI origin. The destination image ROI origin must be defined by the following parameters: the offset of the tiled destination image with respect to the destination image origin and the destination image size. The source image ROI origin is defined automatically. To obtain the source image ROI origin offset, call the `ippiResizeYUV420GetSrcOffset` function. Parameters `pSrcY`, `pSrcUV` and `pDstY`, `pDstUV` must point to the processed source and destination image ROI origins respectively.

The source and destination images are in the 4:2:0 two-plane image format (NV12): all Y samples (`pSrcY`) are found first in memory as an array of unsigned chars with an even number of lines memory alignment, followed by an array (`pSrcUV`) of unsigned chars containing interleaved U and V samples. Supported values for **border** are `ippBorderRepl` and `ippBorderInMem`.

Applied interpolation algorithm uses edge pixels of the source image that are out of the image origin. The function `ippiResizeYUV420Lanczos` uses in calculation the weighted values of these outer pixels. To obtain the size of the out of the source image origin, call the function `ippiResizeYUV420GetBorderSize` for the corresponding flavor.

If the border type is equal to `ippBorderRepl`, the source image edge pixels are replicated out of the image origin. If the border type is equal to `ippBorderInMem`, the outer pixels are obtained from the out of the source image origin space. For the mixed border types, the combined approach is applied.

Prior to using the `ippiResizeLinear` function, initialize the `IppiResizeYUV420Spec` structure by calling the `ippiResizeYUV420LanczosInit` and compute the size of the external buffer `pBuffer` by calling the `ippiResizeYUV420GetBufferSize` for the corresponding flavor.

**Return Values**

- `ippStsNoErr`  
  Indicates no error.

- `ippStsNullPtrErr`  
  Indicates an error if one of the specified pointers is **NULL**.

- `ippStsNoOperation`  
  Indicates a warning if width or height of the destination image is equal to zero.

- `ippStsBorderErr`  
  Indicates an error if the **border** value is illegal.

- `ippStsContextMatchErr`  
  Indicates an error if pointer to the spec structure is invalid.
ippStsMisalignedOffsetErr Indicates an error if one of the fields of the dstOffset parameter is odd.

ippStsSizeWrn Indicates a warning in the following cases:

- if width of the image is odd,
- if the destination image size is more than the destination image origin.

ippStsSizeErr Indicates an error if width of the destination image is equal to 1, or if width or height of the source or destination image is negative.

ippStsOutOfRangeErr Indicates an error if the destination image offset point is outside the destination image origin.

ippStsNotSupportedModeErr Indicates an error if the requested mode is not supported.

ResizeYUV420Super
Changes the size of the NV12 image by the super sampling interpolation method.

Syntax
IppStatus ippiResizeYUV420Super_8u_P2R(const Ipp8u* pSrcY, Ipp32s srcYStep, const Ipp8u* pSrcUV, Ipp32s srcUVStep, Ipp8u* pDstY, Ipp32s dstYStep, Ipp8u* pDstUV, Ipp32s dstUVStep, IppiPoint dstOffset, IppiSize dstSize, const IppiResizeYUV420Spec* pSpec, Ipp8u* pBuffer);

Include Files
ippi.h

Domain Dependencies
Headers: ippcore.h, ippvm.h, ipps.h
Libraries: ippcore.lib, ippvm.lib, ipps.lib

Parameters

- **pSrcY** Pointer to the source image Y plane.
- **srcYStep** Distance in bytes between starts of consecutive lines in the source image Y plane.
- **pSrcUV** Pointer to the source image UV plane.
- **srcUVStep** Distance in bytes between starts of consecutive lines in the source image UV plane.
- **pDstY** Pointer to the destination image Y plane.
- **dstYStep** Distance in bytes between starts of consecutive lines in the destination image Y plane.
- **pDstUV** Pointer to the destination image UV plane.
- **dstUVStep** Distance in bytes between starts of consecutive lines in the destination image UV plane.
\begin{description}
\item[dstOffset] Offset of the tiled destination image with respect to the destination image origin.
\item[dstSize] Size of the destination image in pixels.
\item[pSpec] Pointer to the spec structure for the resize filter.
\item[pBuffer] Pointer to the work buffer.
\end{description}

**Description**

This function changes an image size using interpolation with the super sampling algorithm. The image size can be either reduced or increased in each direction, depending on the destination image size.

This function operates with ROI (see ROI Processing in Geometric Transforms). It resizes the source image ROI origin to the destination image ROI origin. The destination image ROI origin must be defined by the following parameters: the offset of the tiled destination image with respect to the destination image origin and the destination image size. The source image ROI origin is defined automatically. To obtain the source image ROI origin offset, call the \texttt{ippiResizeYUV420GetSrcOffset} function. Parameters \texttt{pSrcY}, \texttt{pSrcUV} and \texttt{pDstY}, \texttt{pDstUV} must point to the processed source and destination image ROI origins respectively.

The interpolation algorithm applied uses only pixels of the source image origin that are inside of the image boundaries.

The source and destination images are in the 4:2:0 two-plane image format (NV12): all Y samples (\texttt{pSrcY}) are found first in memory as an array of unsigned chars with an even number of lines memory alignment, followed by an array (\texttt{pSrcY}) of unsigned chars containing interleaved U and V samples.

Prior to using the \texttt{ippiResizeLinear} function, initialize the \texttt{IppiResizeYUV420Spec} structure by calling the \texttt{ippiResizeYUV420LanczosInit} and compute the size of the external buffer \texttt{pBuffer} by calling the \texttt{ippiResizeYUV420GetBufferSize} for the corresponding flavor.

**NOTE**

This function provides optimized code paths for the following scaling factors along the \(x\) and \(y\) axes: \(1/2, 2/3, 3/4, 4/5, 5/6, 8/9, 1/3, 2/5, 3/5, 3/7, 4/9, 7/10, 1/4, 2/7, 3/8, 1/8\).

**Return Values**

\begin{description}
\item[ippStsNoErr] Indicates no error.
\item[ippStsNullPtrErr] Indicates an error if one of the specified pointers is \texttt{NULL}.
\item[ippStsNoOperation] Indicates a warning if width or height of the destination image is equal to zero.
\item[ippStsBorderErr] Indicates an error if the boundary value is illegal.
\item[ippStsContextMatchErr] Indicates an error if pointer to the spec structure is invalid.
\item[ippStsSizeWrn] Indicates a warning in the following cases:
  \begin{itemize}
  \item if width of the image is odd,
  \item if the destination image size is more than the destination image origin.
  \end{itemize}
\item[ippStsSizeErr] Indicates an error if width or height of the destination image is equal to 1; or if width or height of the source or destination image is negative.
\end{description}
iğiStsMisalignedOffsetErr  Indicates an error if one of the fields of the dstOffset parameter is odd.

ığıStsOutOfRangeErr  Indicates an error if the destination image offset point is outside the destination image origin.

**ResizeYUV422GetSize**

*Computes sizes of the spec structure and the external buffer for YUY2 resize transform initialization.*

**Syntax**

IppStatus ippiResizeYUV422GetSize(IppiSize srcSize, IppiSize dstSize, IppiInterpolationType interpolation, Ipp32u antialiasing, Ipp32s* pSpecSize, Ipp32s* pInitBufSize);

**Include Files**

ippi.h

**Domain Dependencies**

Headers: ippcore.h, ippvm.h, ipps.h
Libraries: ippcore.lib, ippvm.lib, ipps.lib

**Parameters**

- **srcSize**  Size of the source image in pixels.
- **dstSize**  Size of the destination image in pixels.
- **interpolation**  Interpolation method. Supported values are ippNearest and ippLinear.
- **antialiasing**  Antialiasing method.
- **pSpecSize**  Pointer to the size in bytes of the spec structure.
- **pInitBufSize**  Pointer to the size in bytes of the temporal buffer.

**Description**

This function computes sizes of the spec structure and the external buffer that are required for one of the following functions depending on the interpolation method parameter: ResizeYUV422NearestInit and ResizeYUV422LinearInit.

The filter sizes of the Nearest Neighbor and Linear interpolation algorithms are 2x1 and 4x2 respectively.

**NOTE**

Antialiasing is currently not supported. The value for the antialiasing parameter must be equal to zero.

**Return Values**

- **ippiStsNoErr**  Indicates no error. Any other value indicates an error.
** IPP Status Codes**

- **ippStsNullPtrErr**  
  Indicates an error condition if one of the specified pointers is NULL.

- **ippStsNoOperation**  
  Indicates a warning if width or height of the image is equal to zero.

- **ippStsSizeErr**  
  Indicates an error in the following cases:
  - if the source image size is less than the filter size for the chosen interpolation method,
  - if one of the calculated sizes exceeds maximum 32 bit signed integer positive value (the size of one of the processed images is too large).

- **ippStsSizeWrn**  
  Indicates a warning if width of the image is odd.

- **ippStsInterpolationErr**  
  Indicates an error if interpolation has an illegal value.

- **ippStsNoAntialiasing**  
  Indicates a warning if the specified interpolation method does not support antialiasing.

- **ippStsNotSupportedModeErr**  
  Indicates an error if the requested mode is currently not supported.

**ResizeYUV422GetBorderSize**

*Computes the size of possible borders for the YUY2 resize transform.*

**Syntax**

```c
IppStatus ippiResizeYUV422GetBorderSize(const IppiResizeYUV422Spec* pSpec, IppiBorderSize* borderSize);
```

**Include Files**

```c
ippi.h
```

**Domain Dependencies**

- **Headers:** ippcore.h, ippvm.h, ipps.h
- **Libraries:** ippcore.lib, ippvm.lib, ipps.lib

**Parameters**

- **pSpec**
  Pointer to the spec structure for the resize filter.

- **borderSize**
  Size in pixels of necessary borders.

**Description**

This function computes the size of the source image ROI that is used by the corresponding resize transform and is out of the processing boundaries. The `pSpec` parameter defines the resize algorithm parameters. Prior to using the `ippiResizeYUV422GetBorderSize` function, you need to initialize the `pSpec` parameter by calling one of the following functions: `ippiResizeYUV422NearestInit`, and `ippiResizeYUV422LinearInit`.

**Return Values**

- **ippStsNoErr**  
  Indicates no error.
Indicates an error if one of the specified pointers is NULL.

Indicates an error if pointer to the spec structure is invalid.

**ResizeYUV422GetSrcOffset**

*Computes the offset of the source image for the YUY2 resize transform by tile processing.*

**Syntax**

```c
IppStatus ippiResizeYUV422GetSrcOffset(const IppiResizeYUV422Spec* pSpec, IppiPoint dstOffset, IppiPoint* srcOffset);
```

**Include Files**

`ippi.h`

**Domain Dependencies**

**Headers:** `ippcore.h`, `ippvm.h`, `ipps.h`

**Libraries:** `ippcore.lib`, `ippvm.lib`, `ipps.lib`

**Parameters**

- `pSpec` Pointer to the spec structure for the resize filter.
- `dstOffset` Offset of the tiled destination image with respect to the destination image origin.
- `srcOffset` Offset of the source image.

**Description**

This function computes the offset of the processed source image ROI using the offset of the processed destination image ROI for the corresponding resize transform by tile processing. The `pSpec` parameter defines the resize algorithm parameters. Prior to using the `ippiResizeYUV422GetSrcOffset` function, you need to initialize the `pSpec` parameter by calling one of the following functions: `ippiResizeYUV422NearestInit` and `ippiResizeYUV422LinearInit`.

**Return Values**

- `ippStsNoErr` Indicates no error.
- `ippStsNullPtrErr` Indicates an error if one of the specified pointers is NULL.
- `ippStsMisalignedOffsetErr` Indicates an error if the `x` field of the `dstOffset` parameter is odd.
- `ippStsOutOfRangeErr` Indicates an error if the destination image offset point is outside the destination image origin.
- `ippStsContextMatchErr` Indicates an error if pointer to the spec structure is invalid.
Syntax
IppStatus ippiResizeYUV422GetBufSize(const IppiResizeYUV422Spec* pSpec, IppSize dstSize, Ipp32s* pBufSize);

Include Files
ippi.h

Domain Dependencies
Headers: ippcore.h, ippvm.h, ipps.h
Libraries: ippcore.lib, ippvm.lib, ipps.lib

Parameters
pSpec  
Pointer to the spec structure for the resize filter.

dstSize  
Size in pixels of the destination image.

pBufSize  
Pointer to the size in bytes of the external buffer.

Description
This function computes the size of the external buffer for the YUY2 resize transform. The pSpec parameter defines the resize algorithm parameters. Prior to using the ippiResizeYUV420GetBufferSize function, you need to initialize the pSpec parameter by calling one of the following functions: ippiResizeYUV422NearestInit and ippiResizeYUV422LinearInit.

Return Values
ippStsNoErr  
Indicates no error.

ippStsNullPtrErr  
Indicates an error if one of the specified pointers is NULL.

ippStsNoOperation  
Indicates a warning if width or height of the destination image is equal to zero.

ippStsContextMatchErr  
Indicates an error if pointer to the spec structure is invalid.

ippStsSizeWrn  
Indicates a warning in the following cases:
  
  • if width of the image is odd,
  • if the destination image size is more than the destination image origin size.

ippStsSizeErr  
Indicates an error in the following cases:
  
  • if width of the image is equal to 1,
  • if width or height of the destination image is negative,
  • if the calculated buffer size exceeds maximum 32 bit signed integer positive value (the processed image size is too large).

ResizeYUV422GetSrcRoi

Computes the ROI of the source image for YUV422 resizing by tile processing.

Syntax
IppStatus ippiResizeYUV422GetSrcRoi(const IppiResizeYUV422Spec* pSpec, IppiPoint dstRoiOffset, IppSize dstRoiSize, IppiPoint* srcRoiOffset, IppSize* srcRoiSize);
Include Files
ippi.h

Domain Dependencies
Headers: ippcore.h, ippvm.h, ipps.h
Libraries: ippcore.lib, ippvm.lib, ipps.lib

Parameters

pSpec   Pointer to the spec structure for the resize filter.
dstRoiOffset   Offset of the tiled destination image ROI.
dstRoiSize   Size of the tiled destination image ROI.
srcRoiOffset   Offset of the source image ROI.
srcRoiSize   Pointer to the size of the source image ROI.

Description
This function computes the ROI of the processed source image using the processed ROI of the destination image for the corresponding resize transform by tile processing. The pSpec parameter defines the resize algorithm parameters. Prior to using theippiResizeYUV422GetSrcRoi function, you need to initialize the pSpec parameter by calling one of the following functions: ippiResizeYUV422NearestInit or ippiResizeYUV422LinearInit.

NOTE
If the destination ROI size exceeds the image origin, the source ROI will be obtained for an intersection of the destination ROI and image origin.

Return Values

ippiStsNoErr   Indicates no error.
ippiStsNullPtrErr   Indicates an error if one of the specified pointers is NULL.
ippiStsContextMatchErr   Indicates an error if pointer to the specification structure is invalid.
ippiStsOutOfRangeErr   Indicates an error if the destination image offset point is outside the destination image origin.
ippiStsMisalignedOffsetErr   Indicates an error if x-value of the parameter dstRoiOffset is odd.
ippiStsSizeErr   Indicates an error in the following cases:
  • If the height of the destination ROI is zero or negative.
  • If the width of the destination ROI is less than 2.
ippiStsSizeWrn   Indicates a warning in the following cases:
  • If the width of the destination ROI is odd.
  • If the destination ROI exceeds the destination image origin.
**ResizeYUV422NearestInit**  
*Initializes the spec structure for the YUY2 resize transform by the nearest neighbor interpolation method.*

**Syntax**

```c
IppStatus ippiResizeYUV422NearestInit(IppiSize srcSize, IppiSize dstSize,
IppiResizeYUV422Spec* pSpec);
```

**Include Files**

ippi.h

**Domain Dependencies**

Headers: ippcore.h, ippvm.h, ipps.h  
Libraries: ippcore.lib, ippvm.lib, ipps.lib

**Parameters**

- `srcSize`  
  Size in pixels of the source image.
- `dstSize`  
  Size in pixels of the destination image.
- `pSpec`  
  Pointer to the spec structure for the resize filter.

**Description**

This function initializes the `IppiResizeYUV422Spec` structure for the resize algorithm with the nearest neighbor interpolation method. To calculate the size of the spec structure object, call the `ippiResizeYUV422GetSize` function.

**Return Values**

- `ippStsNoErr`  
  Indicates no error.
- `ippStsNullPtrErr`  
  Indicates an error if one of the specified pointers is **NULL**.
- `ippStsNoOperation`  
  Indicates a warning if width or height of the image is equal to zero.
- `ippStsSizeWrn`  
  Indicates a warning if width of the image is odd.
- `ippStsSizeErr`  
  Indicates an error if width of the image is equal to 1, or if width or height of the source or destination image is negative.

**ResizeYUV422LinearInit**  
*Initializes the spec structure for the YUY2 resize transform by the linear interpolation method.*

**Syntax**

```c
IppStatus ippiResizeYUV422LinearInit(IppiSize srcSize, IppiSize dstSize,
IppiResizeYUV422Spec* pSpec);
```

**Include Files**

ippi.h

---

**Intel® Integrated Performance Primitives Developer Reference, Volume 2: Image Processing**

800
Domain Dependencies
Headers: ippcore.h, ippvm.h, ipps.h
Libraries: ippcore.lib, ippvm.lib, ipps.lib

Parameters

srcSize  
Size in pixels of the source image.

dstSize  
Size in pixels of the destination image.

pSpec  
Pointer to the spec structure for the resize filter.

Description
This function initializes the IppiResizeYUV422Spec structure for the resize algorithm with the linear interpolation method. To calculate the size of the spec structure object, call the ippiResizeYUV422GetSize function.

Return Values

ippStsNoErr  
Indicates no error.

ippStsNullPtrErr  
Indicates an error if one of the specified pointers is NULL.

ippStsNoOperation  
Indicates a warning if width or height of the image is equal to zero.

ippStsSizeWrn  
Indicates a warning if width of the image is odd.

ippStsSizeErr  
Indicates an error in the following cases:

- if width of the image is equal to 1,
- if width or height of the source or destination image is negative,
- if the source image size is less than the linear filter size 4x2.

ResizeYUV422Nearest
Changes an YUY2 image size by the nearest neighbor interpolation method.

Syntax
IppStatus ippiResizeYUV422Nearest_8u_C2R(const Ipp8u* pSrc, Ipp32s srcStep, Ipp8u* pDst, Ipp32s dstStep, IppiPoint dstOffset, IppiSize dstSize, const IppiResizeYUV422Spec* pSpec, Ipp8u* pBuffer);

Include Files
ippi.h

Domain Dependencies
Headers: ippcore.h, ippvm.h, ipps.h
Libraries: ippcore.lib, ippvm.lib, ipps.lib

Parameters

pSrc  
Pointer to the source image.
srcStep
Destination in bytes between starts of consecutive lines in the source image buffer.
pDst
Pointer to the destination image.
dstStep
Distance in bytes between starts of consecutive lines in the destination image buffer.
dstOffset
Offset of the tiled destination image with respect to the destination image origin.
dstSize
Size of the destination image in pixels.
pSpec
Pointer to the spec structure for the resize filter.
pBuffer
Pointer to the work buffer.

Description
This function changes an image size using the nearest neighbor interpolation method. The image size can be either reduced or increased in each direction, depending on the destination image size.

This function operates with ROI (see ROI Processing in Geometric Transforms). It resizes the source image ROI origin to the destination image ROI origin. The destination image ROI origin must be defined by the following parameters: the offset of the tiled destination image with respect to the destination image origin and the destination image size. The source image ROI origin is defined automatically. To obtain the source image ROI origin offset, call the ippiResizeYUV422GetSrcOffset function. Parameters pSrc and pDst must point to the processed source and destination image ROI origins respectively.

The source and destination images are in the YUY2 pixel format (Y0U0Y1V0,Y2U1Y3V1,.. or Y0Cb0Y1Cr0,Y2Cb1Y3Cr1,..).

The interpolation algorithm applied uses only pixels of the source image origin that are inside of the image boundaries.

Prior to using the ippiResizeYUV422Nearest function, initialize the IppiResizeYUV422Spec structure by calling the ippiResizeYUV422NearestInit and compute the size of the external buffer pBuffer by calling the ippiResizeYUV422GetBufSize for the corresponding flavor.

Return Values

ippStsNoErr Indicates no error.
ippStsNullPtrErr Indicates an error if one of the specified pointers is NULL.
ippStsNoOperation Indicates a warning if width or height of the destination image is equal to zero.
ippStsContextMatchErr Indicates an error if pointer to the spec structure is invalid.
ippStsSizeWrn Indicates a warning in the following cases:
  • if width of the image is odd,
  • if the destination image size is more than the destination image origin.
ippStsMisalignedOffsetErr Indicates an error if the x field of the dstOffset parameter is odd.
ippStsSizeErr Indicates an error if width of the destination image is equal to 1, or if width or height of the source or destination image is negative.
ippStsOutOfRangeErr Indicates an error if the destination image offset point is outside the destination image origin.
ResizeYUV422Linear
Changes an YUY2 image size by the linear interpolation method.

Syntax
IppStatus ippiResizeYUV422Linear_8u_C2R(const Ipp8u* pSrc, Ipp32s srcStep, Ipp8u* pDst, Ipp32s dstStep, IppiPoint dstOffset, IppiSize dstSize, IppiBorderType border, const Ipp8u* pBorderValue, const IppiResizeYUV422Spec* pSpec, Ipp8u* pBuffer);

Include Files
ippi.h

Domain Dependencies
Headers: ippcore.h, ippvm.h, ipps.h
Libraries: ippcore.lib, ippvm.lib, ipps.lib

Parameters

pSrc
Pointer to the source image.
srcStep
Distance in bytes between starts of consecutive lines in the source image buffer.
pDst
Pointer to the destination image.
dstStep
Distance in bytes between starts of consecutive lines in the destination image buffer.
dstOffset
Offset of the tiled destination image with respect to the destination image origin.
dstSize
Size of the destination image in pixels.
border
Type of the border.
pBorderValue
Pointer to the constant value to assign to pixels of the constant border. This parameter is applicable only to the ippBorderConst border type.
pSpec
Pointer to the spec structure for the resize filter.
pBuffer
Pointer to the work buffer.

Description
This function changes an image size using the linear interpolation method. The image size can be either reduced or increased in each direction, depending on the destination image size.

This function operates with ROI (see ROI Processing in Geometric Transforms). It resizes the source image ROI origin to the destination image ROI origin. The destination image ROI origin must be defined by the following parameters: the offset of the tiled destination image with respect to the destination image origin and the destination image size. The source image ROI origin is defined automatically. To obtain the source image ROI origin offset, call the ippiResizeYUV422GetSrcOffset function. The source and destination images are in the YUY2 pixel format (Y0U0Y1V0,Y2U1Y3V1,.. or Y0Cb0Y1Cr0,Y2Cb1Y3Cr1,..).

Supported values for border are ippBorderRepl and ippBorderInMem.
The interpolation algorithm applied uses edge pixels of the source image that are out of the ROI. The function `ippiResizeYUV422Linear` uses the weighted values of these outer pixels in calculation. To obtain the size of the out of the ROI source image, call the function `ippiResizeYUV422GetBorderSize`.

If the border type is equal to `ippBorderRepl`, the source image edge pixels are replicated out of the ROI. If the border type is equal to `ippBorderInMem`, the outer pixels are obtained from the out of the ROI image space.

Prior to using the `ippiResizeYUV422Linear` function, initialize the `IppiResizeYUV422Spec` structure by calling the `ippiResizeYUV422LinearInit` and compute the size of the external buffer `pBuffer` by calling the `ippiResizeYUV422GetBufSize` for the corresponding flavor.

**Return Values**

- **ippStsNoErr**
  Indicates no error.

- **ippStsNullPtrErr**
  Indicates an error if one of the specified pointers is NULL.

- **ippStsNoOperation**
  Indicates a warning if width or height of the destination image is equal to zero.

- **ippStsBorderErr**
  Indicates an error if the border value is illegal.

- **ippStsContextMatchErr**
  Indicates an error if pointer to the spec structure is invalid.

- **ippStsSizeWrn**
  Indicates a warning in the following cases:
  - if width of the destination image is odd,
  - if the destination image size is more than the destination image origin.

- **ippStsMisalignedOffsetErr**
  Indicates an error if the x field of the `dstOffset` parameter is odd.

- **ippStsSizeErr**
  Indicates an error if width of the destination image is equal to 1, or if width or height of the source or destination image is negative.

- **ippStsOutOfRangeErr**
  Indicates an error if the destination image offset point is outside the destination image origin.

**Warp Functions with Prior Initialization**

This section describes the Intel® IPP warping functions that use the specification structure in operation. Before using these functions, you need to initialize the structure.

**Using Intel® IPP Warp Affine Functions with Prior Initialization**

You can use one of the following approaches to image warping:

- **Warping the whole image**
- **Warping a tiled image with one prior initialization**

Interpolation algorithms of the Nearest Neighbor, Linear, and Cubic types can use edge pixels of the source image that are out of the image origin. When calling the `ippiWarpAffine<Filter>` function with one of these interpolation algorithms applied, you need to specify the appropriate border type. The following border types are supported:

- Replicated borders: border pixels are replicated from the source image boundary pixels
- Constant border: values of all border pixels are set to a constant
- Transparent borders: destination pixels that have inverse transformed location out of the source image are not processed
- Borders in memory: the source image border pixels are obtained from the source image pixels in memory
• Mixed borders: combination of transparent borders and borders in memory is applied

**Warping the Whole Image**

You can follow the approach described below to apply affine transformation when source and destination images are fully accessible in memory. However, this method only runs on a single thread.

To transform the whole image:

1. Call the **WarpAffineGetSize** function with the appropriate interpolation type. This function uses source and destination image sizes to calculate how much memory must be allocated for the **IppWarpSpec** structure and work buffer.

2. Initialize the **IppWarpSpec** structure by calling the **ippiWarp<Filter>Init** function, where `<Filter>` can take one of the following values: Nearest, Linear, and Cubic. These prerequisite steps enable calling the warp functions multiple times without recalculations.

3. Call the **WarpGetBufferSize** function for the initialized **IppWarpSpec** structure. This function uses the destination image size to calculate how much memory must be allocated for the warp work buffer.

4. Call **ippiWarpAffine<Filter>** with the appropriate image type.

5. Specify the algorithm for borders processing by setting the `borderType` and `pBorderValue` parameters when initializing the **IppiWarpSpec** structure. The data type of `pBorderValue` is automatically converted from `Ipp64f` to the data type of the processed images. The function supports the following algorithms for borders processing:

   - If the border type is equal to **ippBorderRepl**, the source image outer pixels are replicated from the edge pixels.
   - If the border type is equal to **ippBorderConst**, the outer pixels are set to the constant value specified in `pBorderValue`.
   - If the border type is equal to **ippBorderTransp**, destination image pixels mapped to the outer source image pixels are not changed. The outer pixels are replicated from the edge pixels, if they are required by interpolation algorithm.
   - If the border type is equal to **ippBorderInMem**, destination image pixels mapped to the outer source image pixels are not changed. The outer pixels are obtained from the out of the source image origin space, if they are required by interpolation algorithm.
   - The mixed border types can be obtained by the bitwise operation OR between **ippBorderTransp** and the **ippBorderInMemTop**, **ippBorderInMemBottom**, **ippBorderInMemLeft**, **ippBorderInMemRight** values.
Figure *Whole Image Warping* shows a simple image affine transformation example. Transformation coefficients are \([\{1.0, 0.5, 0.0\}, \{0.5, 1.0, 0.0\}]\), border type is *ippBorderConst*, *pBorderValue* is a white color pixel. The size of the destination image is 1.2x of the source image size.

**Whole Image Warping**

![Image](image.png)

---

**Example**

The code example below demonstrates affine transformation of the whole image with the linear interpolation method:

```c
IppStatus warpAffineExample_8u_C3R(Ipp8u* pSrc, IppiSize srcSize, Ipp32s srcStep, Ipp8u* pDst, IppiSize dstSize,
    Ipp32s dstStep, const double coeffs[2][3])
{
    IppiWarpSpec* pSpec = 0;
    int specSize = 0, initSize = 0, bufSize = 0; Ipp8u* pBuffer = 0;
    const Ipp32u numChannels = 3;
    IppiPoint dstOffset = {0, 0};
    IppStatus status = ippStsNoErr;
    IppiBorderType borderType = ippBorderConst;
    IppiWarpDirection direction = ippWarpForward;
    Ipp64f pBorderValue[numChannels];

    for (int i = 0; i < numChannels; ++i) pBorderValue[i] = 255.0;

    /* Spec and init buffer sizes */
    status = ippiWarpAffineGetSize(srcSize, dstSize, ipp8u, coeffs, ippLinear, direction, borderType, &specSize, &initSize);

    if (status != ippStsNoErr) return status;

    /* Memory allocation */
    pSpec = (IppiWarpSpec*)ippsMalloc_8u(specSize);

    if (pSpec == NULL)
    {
```

---
Warping a Tiled Image with One Prior Initialization

You can follow the approach described below to apply affine transformation when the source image is fully accessible in memory and destination image is not fully accessible in memory, or to improve performance of warping by external threading.

The main difference between this approach and whole image warping is that the image is split into sections called tiles. Each call of the \texttt{WarpAffine<Filter>} function works with the destination image origin region of interest (ROI) that is defined by \texttt{dstRoiOffset} and \texttt{dstRoiSize} parameters. The destination ROI must be fully accessible in memory.

To resize an image with the tiled approach:

1. Call the \texttt{WarpAffineGetSize} function with the appropriate interpolation type. This function uses the size of the source and destination images and transformation parameters to calculate how much memory must be allocated for the \texttt{IppWarpSpec} structure and initialization work buffer.
2. Initialize the \texttt{IppWarpSpec} structure by calling \texttt{ippiWarpAffine<Filter>Init}, where \texttt{<Filter>} can take one of the following values: Nearest, Linear, and Cubic.
Determine an appropriate partitioning scheme to divide the destination image into tiles. Tiles can be sets of rows or a regular grid of subimages. A simple vertical subdivision into sets of lines is sufficient in most cases.

Call the `WarpGetBufferSize` function to obtain the size of the work buffer required for each tile processing. The `dstRoiSize` parameter must be equal to the tile size.

Call `ippiWarpAffine<Filter>` for each tile (ROI). The `dstRoiOffset` parameter must specify the image ROI offset with respect to the destination image origin. The `dstRoiSize` parameter must be equal to the ROI size. The `pDst` parameter must point to the beginning of the destination ROI in memory. The source and destination ROIs must be fully accessible in memory.

You can process tiles in any order. When using multiple threads you can process all tiles simultaneously.

Figure *Tiled Image Warping* shows the affine transformation of the image divided into tiles. Transformation coefficients are \(\{(1.0, 0.5, 0.0), (0.5, 1.0, 0.0)\}\), applied border type is `ippBorderConst`, `pBorderValue` is a white color pixel. The size of the destination image is 1.2x of the source image size.

**Tiled Image Warping**

---

**Example**

The code example below demonstrates a multithreading affine transformation using OpenMP* with parallelization only in y direction:

```c
IppStatus tileWarpAffineExample_C3R(Ipp8u* pSrc, IppiSize srcSize, Ipp32s srcStep, Ipp8u* pDst, IppiSize dstSize, Ipp32s dstStep, const double coeffs[2][3])
{
    IppiWarpSpec* pSpec = 0;
    int specSize = 0, initSize = 0, bufSize = 0; Ipp8u* pBuffer = 0;
    Ipp8u* pInitBuf = 0;
    const Ipp32u numChannels = 3;
    IppiPoint dstOffset = {0, 0};
    IppiPoint srcOffset = {0, 0};
    IppStatus status = ippStsNoErr;
    IppiBorderType borderType = ippBorderConst;
    IppiWarpDirection direction = ippWarpForward;
    int numThreads, slice, tail;
    int bufSize1, bufSize2;
    IppiSize dstTileSize, dstLastTileSize; IppStatus pStatus[MAX_NUM_THREADS];
    Ipp64f pBorderValue[numChannels];

    for (int i = 0; i < numChannels; ++i) pBorderValue[i] = 255.0;
```
/* Spec and init buffer sizes */
status =ippiWarpAffineGetSize(srcSize, dstSize, ipp8u, coeffs, ippLinear, direction,
borderType, &specSize, &initSize);
if (status != ippStsNoErr) return status;

/* Memory allocation */
pSpec = (IppiWarpSpec*)ippsMalloc_8u(specSize);
if (pSpec == NULL)
{
    return ippStsNoMemErr;
}

/* Filter initialization */
status =ippiWarpAffineLinearInit(srcSize, dstSize, ipp8u, coeffs, direction, numChannels,
borderType, pBorderValue, 0, pSpec);
if (status != ippStsNoErr)
{
    ippsFree(pSpec);
    return status;
}

/* General transform function */
/* Parallelized only by Y-direction here */
#pragma omp parallel num_threads(MAX_NUM_THREADS)
{
    #pragma omp master
    {
        numThreads = omp_get_num_threads();

        slice = dstSize.height / numThreads; tail = dstSize.height % numThreads;
        dstTileSize.width = dstLastTileSize.width = dstSize.width;
        dstTileSize.height = slice;
        dstLastTileSize.height = slice + tail;

        ippiWarpGetBufferSize(pSpec, dstTileSize, &bufSize1);
        ippiWarpGetBufferSize(pSpec, dstLastTileSize, &bufSize2);

        pBuffer = ippsMalloc_8u(bufSize1 * (numThreads - 1) + bufSize2);
    }
    #pragma omp barrier
    {
        if (pBuffer)
        {
            Ipp32u  i;
            Ipp8u  *pDstT; Ipp8u  *pOneBuf;
            IppiPoint srcOffset = {0, 0};
            IppiPoint dstOffset = {0, 0};
            IppiSize srcSizeT = srcSize; IppiSize dstSizeT = dstTileSize;

            i = omp_get_thread_num();

            dstSizeT.height = slice; dstOffset.y += i * slice;

            /* Parallelized by Y-direction */
            ...
if (i == numThreads - 1) dstSizeT = dstLastTileSize;

pDstT = (Ipp8u*)((char*)pDst + dstOffset.y * dstStep);
pOneBuf = pBuffer + i * bufSize1;

pStatus[i] =ippiWarpAffineLinear_8u_C3R (pSrc, srcStep, pDstT, dstStep, dstOffset, dstSizeT, pSpec, pOneBuf);
}

ippiFree(pSpec);
if (pBuffer == NULL) return ippStsNoMemErr;
ippiFree(pBuffer);

for (Ipp32u i = 0; i < numThreads; ++i)
{
    /* Return bad status */
    if(pStatus[i] != ippStsNoErr) return pStatus[i];
}

return status;

See Also

User-defined Border Types

WarpAffineGetSize Computes the size of the specification structure and the size of the external work buffer for the warp affine transform.

WarpGetBufferSize Computes the size of the work buffer for the warp transform.

Edge Smoothing

The Smooth Edge feature is an artificial method to reduce aliasing artifacts at the transformed source image edges. Aliasing artifacts may appear because the transformation algorithms skip a destination pixel if its source origin is out of the source image ROI. Thus, borders of the transformed source image can look stepped:
If the smoothEdge flag is set, destination pixels that are closest to the transformed source image edges are mixed with sampled source pixels by the following formula:

\[ \text{dstRes} = \text{srcSampled} * (1 - a) + \text{dstExist} * a \]

where

- \text{srcSampled} is the intensity of the source pixel after transformation.
- \text{dstExist} is the intensity of the destination pixel before transformation.
- \( a \) is the weight of the outer pixel; set by the function.
- \text{dstRes} is the intensity of the resulting destination pixel.

The edge smoothing method is not universal: in some cases it can improve the image, but in other cases it can be inefficient. For example, edge smoothing does not increase the quality of images with high contrast borders, and it is not recommended to apply edge smoothing to such images.

**NOTE**

Edge smoothing is a post-processing operation: it is performed after transformation. When warping a tiled image, artifacts may appear on tile borders. In this case, edges are not smoothed.

**GetAffineQuad**

*Computes vertex coordinates of the quadrangle, to which the source ROI rectangle is mapped by the affine transform.*

**Syntax**

\[
\text{IppStatus ippiGetAffineQuad (IppRect srcRoi, double quad[4][2], const double coeffs[2][3]);}
\]

**Include Files**

ippi.h

**Flavors with the _L suffix:** ippi_L.h

**Domain Dependencies**

*Headers: ippcore.h, ippvm.h, ipps.h*
Libraries: ippcore.lib, ippvm.lib, ipps.lib

**Parameters**

- **srcRoi**
  Region of interest in the source image (of the IppiRect type).

- **quad**
  Output array. Contains vertex coordinates of the quadrangle, to which the source ROI is mapped by the affine transform function. The given affine transform coefficients.

- **coeffs**

**Description**

This function operates with ROI (see ROI Processing in Geometric Transforms).

This function is used as a support function for WarpAffineNearest, WarpAffineLinear, and WarpAffineCubic functions. It computes vertex coordinates of the quadrangle, to which the source rectangular ROI is mapped by the affine transform function using the given coefficients `coeffs`.

The first dimension [4] of the array `quad[4][2]` is equal to the number of vertices, and the second dimension [2] means x and y coordinates of the vertex. Quadrangle vertices have the following meaning:

- `quad[0]` corresponds to the transformed top-left corner of the source ROI,
- `quad[1]` corresponds to the transformed top-right corner of the source ROI,
- `quad[2]` corresponds to the transformed bottom-right corner of the source ROI,
- `quad[3]` corresponds to the transformed bottom-left corner of the source ROI.

**Return Values**

- **ippStsNoErr**
  Indicates no error. Any other value indicates an error.

- **ippStsCoeffErr**
  Indicates an error condition if $c_{00}c_{11} - c_{01}c_{10} = 0$.

- **ippStsSizeErr**
  Indicates an error condition if `srcRoi` has a size field with zero or negative value.

**GetAffineBound**

*Computes the bounding rectangle for the source ROI transformed by the ippiWarpAffine function.*

**Syntax**

```c
IppStatus ippiGetAffineBound (IppiRect srcRoi, double bound[2][2], const double coeffs[2][3]);
```

**Platform-aware functions**

```c
IppStatus ippiGetAffineBound_L (IppiRectL srcRoi, double bound[2][2], const double coeffs[2][3]);
```

**Include Files**

`ippi.h`

**Flavors with the _L suffix:** `ippi_l.h`

**Domain Dependencies**

**Headers:** ippcore.h, ippvm.h, ipps.h

**Libraries:** ippcore.lib, ippvm.lib, ipps.lib
**Parameters**

*srcRoi*  
Region of interest in the source image (of the `IppiRect` type).

*bound*  
Output array. Contains vertex coordinates of the bounding rectangle for the transformed source ROI.

*coeffs*  
The given affine transform coefficients.

**Description**

This function is used as a support function for `WarpAffineNearest`, `WarpAffineLinear`, and `WarpAffineCubic` functions. It computes vertex coordinates of the smallest bounding rectangle for the quadrangle `quad`, to which the source ROI is mapped by the affine transform function using coefficients `coeffs`.

*bound[0]* specifies x, y coordinates of the top-left corner, *bound[1]* specifies x, y coordinates of the bottom-right corner.

**Return Values**

- **ippStsNoErr**: Indicates no error. Any other value indicates an error or a warning.
- **ippStsCoeffErr**: Indicates an error condition if \( c_{00} \cdot c_{11} - c_{01} \cdot c_{10} = 0 \).
- **ippStsSizeErr**: Indicates an error condition if *srcRoi* has a size field with zero or negative value.

**GetAffineSrcRoi**

*Computes ROI of an image for affine transform.*

**Syntax**

```c
IppStatus ippiGetAffineSrcRoi (IppiSize srcSize, const double coeffs[2][3], IppiWarpDirection direction, IppiPoint dstRoiOffset, IppiSize dstRoiSize, IppiRect *srcRoi);
```

**Platform-aware functions**

```c
IppStatus ippiGetAffineSrcRoi_L (IppiSizeL srcSize, const double coeffs[2][3], IppiWarpDirection direction, IppiPointL dstRoiOffset, IppiSizeL dstRoiSize, IppiRectL *srcRoi);
```

**Include Files**

`ippi.h`

Flavors with the `_L` suffix: `ippi_l.h`

**Domain Dependencies**

Headers: `ippcore.h`, `ippvm.h`, `ipps.h`

Libraries: `ippcore.lib`, `ippvm.lib`, `ipps.lib`

**Parameters**

*srcSize*  
Size of the source image, in pixels.

*coeffs*  
Coefficients for affine transform.

*direction*  
Transformation direction. Supported values:
ippWarpForward Forward transformation.
ippWarpBackward Backward transformation.

dstRoiSize Size of the ROI of destination image.
dstRoiOffset Offset of the destination image ROI.
srcRoi Pointer to the computed region of interest in the source image.

**Description**
This function operates with ROI (see **ROI Processing in Geometric Transforms**).

This function is used as a support function for the ippiWarpAffineLinear, WarpAffineNearest, and WarpAffineCubic functions. It computes ROI of the source image to perform affine transformation for a given destination ROI. To process the given destination ROI, the computed source ROI with borders must be accessible in memory. If the source ROI outside pixels are out of the source image origin, the border pixels are processed according to the border flag that is passed to the ippiWarpAffineLinear, WarpAffineNearest, and WarpAffineCubic functions.

**Return Values**

- ippStsNoErr Indicates no error. Any other value indicates an error or warning.
- ippStsRectErr Indicates an error condition if width or height of the srcRoi is less than or equal to 1.
- ippStsOutOfRangeErr Indicates an error if the destination image offset has a field with a negative value.
- ippStsSizeErr Indicates an error condition if width or height of the source or destination image is less than, or equal to zero.
- ippStsWrongIntersetQuad Indicates a warning that no operation is performed because the transformed source image has no intersection with the destination ROI.
- ippStsNullPtrErr Indicates an error if one of the specified pointers is NULL.
- ippStsWarpDirectionErr Indicates an error when the direction value is illegal.
- ippStsCoefErr Indicates an error condition, if affine transformation is singular.

**GetAffineTransform**
*Computes affine transform coefficients to map the source ROI to the quadrangle with the specified vertex coordinates*

**Syntax**

```c
IppStatus ippGetAffineTransform (IppiRect srcRoi, const double quad[4][2], double coeffs[2][3]);
```

**Platform-aware functions**

```c
IppStatus ippGetAffineTransform_L (IppiRectL srcRoi, const double quad[4][2], double coeffs[2][3]);
```

**Include Files**

ippi.h
Flavors with the \_L suffix: ippi_l.h

**Domain Dependencies**

**Headers:** ippcore.h, ippvm.h, ipps.h  
**Libraries:** ippcore.lib, ippvm.lib, ipps.lib

**Parameters**

- **srcRoi**: Region of interest in the source image (of the IppiRect type).
- **quad**: Vertex coordinates of the quadrangle, to which the source ROI is mapped by the affine transform function.  
- **coeffs**: Output array. Contains the target affine transform coefficients.

**Description**

This function operates with ROI (see ROI Processing in Geometric Transforms).

This function is used as a support function for the ippiWarpAffineLinear, WarpAffineNearest, and WarpAffineCubic functions. It computes the coefficients \textit{coeffs} of the affine transform that must be used by the warping function to map the source rectangular ROI to the quadrangle with the specified vertex coordinates \textit{quad}. The first dimension \([4]\) of the array \textit{quad}[4][2] is equal to the number of vertices, and the second dimension \([2]\) means \(x\) and \(y\) coordinates of the vertex. Quadrangle vertices have the following meaning:

- \textit{quad}[0] corresponds to the transformed top-left corner of the source ROI,
- \textit{quad}[1] corresponds to the transformed top-right corner of the source ROI,
- \textit{quad}[2] corresponds to the transformed bottom-right corner of the source ROI,
- \textit{quad}[3] corresponds to the transformed bottom-left corner of the source ROI.

The function computes the coordinates of the 4th vertex of the destination quadrangle that uniquely depends on the three other vertices. If the computed coordinates are not equal to the ones specified in \textit{quad}, the function returns the warning message and continues operation with the computed values.

**Return Values**

- **ippStsNoErr**: Indicates no error. Any other value indicates an error or warning.
- **ippStsRectErr**: Indicates an error condition if width or height of the \textit{srcRoi} is less than or equal to 1.
- **ippStsCoeffErr**: Indicates an error condition if \(c_{00}*c_{11} - c_{01}*c_{10} = 0\).
- **ippStsSizeErr**: Indicates an error condition if \textit{srcRoi} has a size field with zero or negative value.
- **ippStsAffineQuadChanged**: Indicates a warning that coordinates of the 4th vertex of the specified quadrangle \textit{quad} are not correct.

**GetRotateTransform**

\textit{Computes the affine coefficients for the rotation transform.}

**Syntax**

```c
IppStatus ippiGetRotateTransform(double \textit{angle}, double \textit{xShift}, double \textit{yShift}, double \textit{coeffs}[2][3]);
```
Platform-aware functions

IppStatus ippiGetRotateTransform_L(double angle, double xShift, double yShift, double coeffs[2][3]);

Include Files

ippi.h

Flavors with the _L suffix: ippi_l.h

Domain Dependencies

Headers: ippcore.h, ippvm.h, ipps.h
Libraries: ippcore.lib, ippvm.lib, ipps.lib

Parameters

angle  
Angle of rotation, in degrees. The source image is rotated counterclockwise around the origin (0, 0).

xShift, yShift  
Shift along horizontal (x) or vertical (y) axis that is performed after rotation.

coeffs  
Computed affine transform coefficients for the given rotation parameters.

Description

This function computes the coefficients for the affine transform that rotates an image by the specified angle around the origin (0, 0) and shifts the image after rotation. The result is stored in the coeffs parameter.

Return Values

ippStsNoErr  
Indicates no error. Any other value indicates an error.

ippStsSizeErr  
Indicates an error when one of the coeffs values is NULL.

ippStsOutOfRangeErr  
Indicates an error when angle is not-a-number (NaN) or infinity.

Example

To better understand usage of the ippiGetRotateTransform function, refer to the following example in the examples archive available for download from https://software.intel.com/en-us/ipp-manual-examples:

GetRotateTransform.c

GetRotateShift

Computes shift values for rotation of an image around the specified center.

Syntax

IppStatus ippiGetRotateShift (double xCenter, double yCenter, double angle, double* xShift, double* yShift);

Platform-aware functions

IppStatus ippiGetRotateShift_L (double xCenter, double yCenter, double angle, double* xShift, double* yShift);
Include Files
ippi.h

Flavors with the _L suffix: ippi_l.h

Domain Dependencies
Headers: ippcore.h, ippvm.h, ipps.h
Libraries: ippcore.lib, ippvm.lib, ipps.lib

Parameters

xCenter, yCenter

Coordinates of the required center of rotation.

angle

The angle in degrees to rotate the image clockwise around the point with coordinates (xCenter, yCenter).

xShift, yShift

Pointers to computed shift values along horizontal and vertical axes. These shift values should be passed to ippiRotate function to bring about the desired rotation around (xCenter, yCenter).

Description

Use this function if you need to rotate an image about an arbitrary center (xCenter, yCenter) rather than the origin (0,0). The function helps compute shift values xShift, yShift that should be passed to the warping function for the rotation around (xCenter, yCenter) to take place.

Example shows how to use the function ippiGetRotateShift.

Return Values

ippStsNoErr

Indicates no error. Any other value indicates an error.

ippStsNullPtrErr

Indicates an error condition if xShift or yShift pointer is NULL.

WarpAffineGetSize
Computes the size of the specification structure and the size of the external work buffer for the warp affine transform.

Syntax

IppStatus ippiWarpAffineGetSize(IppiSize srcSize, IppiSize dstSize, IppDataType dataType, const double coeffs[2][3], IppiInterpolationType interpolation, IppiWarpDirection direction, IppiBorderType borderType, int* pSpecSize, int* pInitBufSize);

Platform-aware functions

IppStatus ippiWarpAffineGetSize_L(IppSizeL srcSize, IppSizeL dstSize, IppDataType dataType, const double coeffs[2][3], IppInterpolationType interpolation, IppiWarpDirection direction, IppiBorderType borderType, int flags, IppSizeL* pSpecSize, IppSizeL* pInitBufSize);

Include Files
ippi.h

Flavors with the _L suffix: ippi_l.h
Domain Dependencies

Headers: ippcore.h, ippvm.h, ipps.h
Libraries: ippcore.lib, ippvm.lib, ipps.lib

Parameters

- **srcSize**: Size of the source image, in pixels.
- **dstSize**: Size of the destination image, in pixels.
- **dataType**: Data type of the source and destination images. Supported values: ipp8u, ipp16u, ipp16s, ipp32f, and ipp64f.
- **coeffs**: Coefficients for the affine transform.
- **interpolation**: Interpolation method. Supported values: ippNearest, ippLinear, and ippCubic.
- **direction**: Transformation direction. Supported values:
  - ippWarpForward: Forward transformation
  - ippWarpBackward: Backward transformation
- **borderType**: Type of border. Possible values are:
  - ippBorderConst: Values of all border pixels are set to constant.
  - ippBorderRepl: Border is replicated from the edge pixels.
  - ippBorderTransp: Outer pixels are not processed.
  - ippBorderInMem: Border is obtained from the source image pixels in memory.
  - Mixed borders are also supported. They can be obtained by the bitwise operation OR between ippBorderTransp and the ippBorderInMemTop, ippBorderInMemBottom, ippBorderInMemLeft, ippBorderInMemRight values.
- **flags**: Warp algorithm flags. Supported values:
  - 0 - transformation without edge smoothing.
  - 1 - transformation with edge smoothing. The feature is supported for the ippBorderTransp and ippBorderInMem border types.
- **pSpecSize**: Pointer to the size, in bytes, of the specification structure.
- **pInitBufSize**: Pointer to the size, in bytes, of the temporary buffer.

Description

This function computes the size of the specification structure and the external work buffer for the following functions, depending on the interpolation parameter: WarpAffineNearestInit, WarpAffineLinearInit, or WarpAffineCubicInit.

Return Values

- **ippStsNoErr**: Indicates no error.
- **ippStsNullPtrErr**: Indicates an error when one of the specified pointers is NULL.
ippStsNoOperation
Indicates a warning if width or height of any image is zero.

ippStsSizeErr
Indicates an error if the width or height of the source or destination image is less than, or equal to one.

ippStsDataTypeErr
Indicates an error when dataType has an illegal value.

ippStsWarpDirectionErr
Indicates an error when direction has an illegal value.

ippStsInterpolationErr
Indicates an error when interpolation has an illegal value.

ippStsNotSupportedModeErr
Indicates an error if the requested mode is not supported.

ippStsCoeffErr
Indicates an error when affine transformation is singular.

ippStsBorderErr
Indicates an error when borderType has an illegal value.

ippStsWrongIntersectQuad
Indicates a warning that no operation is performed if the transformed source image extended with borders has no intersection with the destination image.

ippStsExceededSizeErr
Indicates an error in the following cases:

- If one of the calculated sizes exceeds maximum of the pSpecSize variable data type positive value (the size of one of the processed images is too large).
- If width or height of the destination image or the source image exceeds 33554431 (0x1FFFFFFF).

See Also
WarpAffineNearestInit Initializes the specification structure for image affine warping with the nearest neighbor interpolation method.
WarpAffineLinearInit Initializes the specification structure for image affine warping with the linear interpolation method.
WarpAffineCubicInit Initializes the specification structure for image affine warping with the cubic interpolation method.

WarpQuadGetSize
*Computes the size of the specification structure and the size of the temporary buffer for warping an arbitrary quadrangle in the source image to the quadrangle in the destination image.*

Syntax
IppStatusippiWarpQuadGetSize(IppSize srcSize, const double srcQuad[4][2], IppSize dstSize, const double dstQuad[4][2], IppiWarpTransformType transform, IppDataType dataType, IppiInterpolationType interpolation, IppiBorderType borderType, int* pSpecSize, int* pInitBufSize);

Include Files
ippi.h

Domain Dependencies
Headers: ippcore.h, ippvm.h, ipps.h
Libraries: ippcore.lib, ippvm.lib, ipps.lib
Parameters

- **srcSize**: Size of the source image, in pixels.
- **srcQuad**: Quadrangle in the source image.
- **dstSize**: Size of the destination image, in pixels.
- **dstQuad**: Quadrangle in the destination image.
- **transform**: Type of the warp transform. Supported values:
  - ippWarpAffine: Affine warping
  - ippWarpPerspective: Perspective warping
- **dataType**: Data type of the source and destination images. Supported values: ipp8u, ipp16u, ipp16s, and ipp32f.
- **interpolation**: Interpolation method. Supported values: ippNearest, ippLinear, and ippCubic.
- **borderType**: Type of border. Supported values:
  - ippBorderTransp: Outer pixels are not processed.
  - ippBorderInMem: Border is obtained from the source image pixels in memory.
- **pSpecSize**: Pointer to the size, in bytes, of the specification structure.
- **pInitBufSize**: Pointer to the size, in bytes, of the temporary buffer.

Description

This function computes the size of the specification structure and the temporary buffer for the following functions, depending on the **interpolation** parameter: ippiWarpQuadNearestInit, ippiWarpQuadLinearInit, or ippiWarpQuadCubicInit.

Return Values

- **ippStsNoErr**: Indicates no error.
- **ippStsNullPtrErr**: Indicates an error when one of the specified pointers is NULL.
- **ippStsSizeErr**: Indicates an error in the following cases:
  - If the width or height of the source or destination image is less than, or equal to one.
  - If one of the calculated sizes exceeds the maximum positive 32-bit signed integer value. The size of the one of the processed images is too large.
- **ippStsDataTypeErr**: Indicates an error when **dataType** has an illegal value.
- **ippStsWarpTransformErr**: Indicates an error when **transform** has an illegal value.
- **ippStsInterpolationErr**: Indicates an error when **interpolation** has an illegal value.
- **ippStsQuadErr**: Indicates an error when any of the given quadrangles is non-convex or degenerates into a triangle, line, or point.
- **ippStsWrongIntersectQuad**: Indicates a warning that no operation is performed if the transformed source image extended with borders has no intersection with the destination image.
ippStsAffineQuadChanged

Indicates a warning when coordinates of the fourth vertex of dstQuad are changed by the function, if transform is set to ippWarpAffine.

ippStsBorderErr

Indicates an error when borderType has an illegal value.

See Also

WarpQuadNearestInit Initializes the specification structure for warping an arbitrary quadrangle in the source image to the quadrangle in the destination image with the nearest neighbor interpolation method.

WarpQuadLinearInit Initializes the specification structure for warping an arbitrary quadrangle in the source image to the quadrangle in the destination image with the linear interpolation method.

WarpQuadCubicInit Initializes the specification structure for warping an arbitrary quadrangle in the source image to the quadrangle in the destination image with the cubic interpolation method.

WarpGetBufferSize

Computes the size of the work buffer for the warp transform.

Syntax

IppStatus ippiWarpGetBufferSize(const IppiWarpSpec* pSpec, IppiSize dstRoiSize, int* pBufSize);

Platform-aware functions

IppStatus ippiWarpGetBufferSize_L(const IppiWarpSpec* pSpec, IppiSizeL dstRoiSize, IppSizeL* pBufSize);

Include Files

ippi.h

Flavors with the _L suffix: ippi_l.h

Domain Dependencies

Headers: ippcore.h, ippvm.h, ipps.h
Libraries: ippcore.lib, ippvm.lib, ipps.lib

Parameters

pSpec Pointer to the specification structure for the warp operation.
dstRoiSize Size of the processed destination image ROI, in pixels.
pBufSize Pointer to the size of the external buffer, in bytes.

Description

This function computes the size of the external buffer for the warp transform. The specification structure pointed by pSpec defines the warp algorithm parameters.

Before using this function, you need to initialize the specification structure using one of the following functions: WarpAffineNearestInit, WarpAffineLinearInit, or WarpAffineCubicInit.

Return Values

ippStsNoErr Indicates no error.
### ippStsNullPtrErr
Indicates an error when one of the specified pointers is NULL.

### ippStsContextMatchErr
Indicates an error when the specification structure is invalid.

### ippStsSizeErr
Indicates an error in the following cases:
- If width or height of the destination image is negative, or equal to zero.
- If the calculated buffer size exceeds the maximum positive `pBufSize` data type. The size of the processed image ROI is too large.

### ippStsSizeWrn
Indicates a warning when the size of the destination image is more than the size of the destination image origin.

### See Also
**WarpAffineNearestInit** Initializes the specification structure for image affine warping with the nearest neighbor interpolation method.
**WarpAffineLinearInit** Initializes the specification structure for image affine warping with the linear interpolation method.
**WarpAffineCubicInit** Initializes the specification structure for image affine warping with the cubic interpolation method.

### WarpAffineNearestInit
*Initializes the specification structure for image affine warping with the nearest neighbor interpolation method.*

#### Syntax
```c
IppStatus ippiWarpAffineNearestInit(IppiSize srcSize, IppiSize dstSize, IppDataType dataType, const double coeffs[2][3], IppiWarpDirection direction, int numChannels, IppiBorderType borderType, const Ipp64f* pBorderValue, int smoothEdge, IppiWarpSpec* pSpec);
```

#### Platform-aware functions
```c
IppStatus ippiWarpAffineNearestInit_L(IppiSizeL srcSize, IppiSizeL dstSize, IppDataType dataType, const double coeffs[2][3], IppiWarpDirection direction, int numChannels, IppiBorderType borderType, const Ipp64f* pBorderValue, int flags, IppiWarpSpec* pSpec);
```

#### Include Files
```c
ippi.h
```

Flavors with the `_L` suffix: `ippi_l.h`

#### Domain Dependencies
**Headers:** ippcore.h, ippvm.h, ipps.h  
**Libraries:** ippcore.lib, ippvm.lib, ipps.lib

#### Parameters
- `srcSize`: Size of the source image, in pixels.
- `dstSize`: Size of the destination image, in pixels.
- `dataType`: Data type of the source and destination images. Supported values: ipp8u, ipp16u, ipp16s, ipp32f, and ipp64f.
Coefficients for the affine transform.

Transformation direction. Supported values:
- ippWarpForward: Forward transformation
- ippWarpBackward: Backward transformation

Number of channels in the image. Supported values: 1, 3, or 4.

Type of border. Supported values:
- ippBorderConst: Values of all border pixels are set to a constant.
- ippBorderRepl: Border is replicated from the edge pixels.
- ippBorderTransp: Outer pixels are not processed.
- ippBorderInMem: Border is obtained from the source image pixels in memory.

Mixed borders are also supported. They can be obtained by the bitwise operation OR between ippBorderTransp and the ippBorderInMemTop, ippBorderInMemBottom, ippBorderInMemLeft, ippBorderInMemRight values.

Warp algorithm flags. Supported values:
- 0 - transformation without edge smoothing.
- 1 - transformation with edge smoothing. The feature is supported for the ippBorderTransp and ippBorderInMem border types.

Pointer to the constant value to assign to pixels of the constant border. This parameter is applicable only to the ippBorderConst border type.

Flag for edge smoothing. Supported values:
- 0 - transformation without edge smoothing.
- 1 - transformation with edge smoothing. This feature is supported only for the ippBorderTransp and ippBorderInMem border types.

Pointer to the specification structure.

Description
This function initializes the IppiWarpSpec structure for theippiWarpAffineNearest function that performs warp affine transformation with the nearest neighbor interpolation method. To compute the size of the specification structure, use the WarpAffineGetSize function.

Return Values
- ippStsNoErr: Indicates no error.
- ippStsNullPtrErr: Indicates an error when:
  - one of the specified pointers is NULL, excepting pBorderValue
  - pBorderValue is NULL when border type is set to ippBorderConst
- ippStsNoOperation: Indicates a warning if width or height of any image is zero.
Indicates an error when width or height of the source or destination image is less than, or equal to one.

Indicates an error when `dataType` has an illegal value.

Indicates an error when `direction` has an illegal value.

Indicates an error when affine transformation is singular.

Indicates a warning that no operation is performed if the transformed source image extended with borders has no intersection with the destination image. The edge smoothing feature is not supported for the `ippBorderRepl` and `ippBorderConst` border types.

Indicates an error when the requested mode is not supported.

Indicates an error when `borderType` has an illegal value.

Indicates an error when `numChannels` has an illegal value.

Indicates an error if width or height of the destination image or the source image exceeds 33554431 (0x1FFFFFF).

See Also

`WarpAffineGetSize` Computes the size of the specification structure and the size of the external work buffer for the warp affine transform.

`WarpQuadNearestInit` Initializes the specification structure for warping an arbitrary quadrangle in the source image to the quadrangle in the destination image with the nearest neighbor interpolation method.

Syntax

```c
IppStatus ippiWarpQuadNearestInit(IppiSize srcSize, const double srcQuad[4][2],
IppiSize dstSize, const double dstQuad[4][2], IppiWarpTransformType transform,
IppDataType dataType, int numChannels, IppiBorderType borderType, const Ipp64f*
pBorderValue, int smoothEdge, IppiWarpSpec* pSpec);
```

Include Files

`ippi.h`

Domain Dependencies

Headers: `ippcore.h`, `ippvm.h`, `ipps.h`

Libraries: `ippcore.lib`, `ippvm.lib`, `ipps.lib`

Parameters

- `srcSize` Size of the source image, in pixels.
- `srcQuad` Quadrangle in the source image.
- `dstSize` Size of the destination image, in pixels.
- `dstQuad` Quadrangle in the destination image.
- `transform` Type of the warp tranform. Supported values:
ippWarpAffine  Affine warping
ippWarpPerspective  Perspective warping

dataType
Data type of the source and destination images. Supported values: ipp8u, ipp16u, ipp16s, and ipp32f.

numChannels
Number of channels in the image. Supported values: 1, 3, or 4.

borderType
Type of border. Supported values:

- ippBorderTransp  Outer pixels are not processed.
- ippBorderInMem  Border is obtained from the source image pixels in memory.

pBorderValue
Pointer to the constant value to assign to pixels of the constant border. This parameter is applicable only to the ippBorderConst border type.

smoothEdge
Flag for edge smoothing. Supported values:

- 0  - transformation without edge smoothing.
- 1  - transformation with edge smoothing.

pSpec
Pointer to the specification structure.

Description
This function initializes the IppiWarpSpec structure for warping an arbitrary quadrangle in the source image to quadrangle in the destination image with the nearest neighbor interpolation method. To compute the size of the specification structure, use the WarpQuadGetSize function.

Transformation coefficients are computed internally, based on the mapping of the source quadrangle to the specified destination quadrangle dstQuad and transform type transform. In case of affine transform, the function computes the coordinates of the fourth vertex of the destination quadrangle that uniquely depends on the three other vertices. If the computed coordinates are not equal to the corresponding values specified in dstQuad, the function returns the warning message and continues initialization with the computed values.


You can apply the edge smoothing feature only if the source quadrangle entirely lies in the source image.

Return Values

ippStsNoErr  Indicates no error.
ippStsNullPtrErr  Indicates an error when:

- One of the specified pointers is NULL, excepting pBorderValue
- The value of pBorderValue is NULL when the border type is set to ippBorderConst
ippStsSizeErr  Indicates an error when width or height of the source or destination image is less than, or equal to one.
ippStsDataTypeErr  Indicates an error when dataType has an illegal value.
ippStsWarpTransformErr  Indicates an error when transform has an illegal value.
ippStsQuadErr  Indicates an error when any of the given quadrangles is non-convex or degenerates into a triangle, line, or point.
Indicates a warning that no operation is performed if the transformed source image extended with borders has no intersection with the destination image.

**ippStsAffineQuadChanged**
Indicates a warning when coordinates of the fourth vertex of $\text{dstQuad}$ are changed by the function, if $\text{transform}$ is set to ippWarpAffine.

**ippStsBorderErr**
Indicates an error when $\text{borderType}$ has an illegal value.

**ippStsNumChannelsErr**
Indicates an error when $\text{numChannels}$ has an illegal value.

### Example

To better understand usage of the ippWarpQuadNearest, ippWarpQuadLinear, and ippWarpQuadCubic functions, refer to the following example in the examples archive available for download from https://software.intel.com/en-us/ipp-manual-examples:

WarpQuadNearestInit.c

### See Also

**WarpQuadGetSize** Computes the size of the specification structure and the size of the temporary buffer for warping an arbitrary quadrangle in the source image to the quadrangle in the destination image.

**WarpAffineNearest**
Performs warp affine transformation of an image using the nearest neighbor interpolation method.

### Syntax

```
IppStatus ippiWarpAffineNearest_<mod>(const Ipp<datatype>* pSrc, int srcStep, Ipp<datatype> pDst, int dstStep, IppiPoint dstRoiOffset, IppiSize dstRoiSize, const IppiWarpSpec* pSpec, Ipp8u* pBuffer);
```

Supported values for $\text{mod}$:

- $8u\_C1R$, $16u\_C1R$, $16s\_C1R$, $32f\_C1R$, $64f\_C1R$
- $8u\_C3R$, $16u\_C3R$, $16s\_C3R$, $32f\_C3R$, $64f\_C3R$
- $8u\_C4R$, $16u\_C4R$, $16s\_C4R$, $32f\_C4R$, $64f\_C4R$

### Platform-aware functions

```
IppStatus ippiWarpAffineNearest_<mod>(const Ipp<datatype>* pSrc, IppSizeL srcStep, Ipp<datatype>* pDst, IppSizeL dstStep, IppiPointL dstRoiOffset, IppiSizeL dstRoiSize, const IppiWarpSpec* pSpec, Ipp8u* pBuffer);
```

Supported values for $\text{mod}$:

- $8u\_C1R\_L$, $16u\_C1R\_L$, $16s\_C1R\_L$, $32f\_C1R\_L$, $64f\_C1R\_L$
- $8u\_C3R\_L$, $16u\_C3R\_L$, $16s\_C3R\_L$, $32f\_C3R\_L$, $64f\_C3R\_L$
- $8u\_C4R\_L$, $16u\_C4R\_L$, $16s\_C4R\_L$, $32f\_C4R\_L$, $64f\_C4R\_L$
Include Files
ippi.h
Flavors with the _L suffix: ippi_l.h

Domain Dependencies
Headers: ippcore.h, ippvm.h, ipps.h
Libraries: ippcore.lib, ippvm.lib, ipps.lib

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>pSrc</td>
<td>Pointer to the source image.</td>
</tr>
<tr>
<td>srcStep</td>
<td>Distance, in bytes, between the starting points of consecutive lines in the source image buffer.</td>
</tr>
<tr>
<td>pDst</td>
<td>Pointer to the destination image ROI.</td>
</tr>
<tr>
<td>dstStep</td>
<td>Distance, in bytes, between the starting points of consecutive lines in the destination image buffer.</td>
</tr>
<tr>
<td>dstRoiOffset</td>
<td>Offset of the destination image ROI with respect to the destination image origin.</td>
</tr>
<tr>
<td>dstRoiSize</td>
<td>Size of the destination image ROI, in pixels.</td>
</tr>
<tr>
<td>pSpec</td>
<td>Pointer to the specification structure for the warp operation.</td>
</tr>
<tr>
<td>pBuffer</td>
<td>Pointer to the work buffer.</td>
</tr>
</tbody>
</table>

Description

This function transforms the source image pixel coordinates \((x, y)\) according to the following formulas:

\[
\begin{align*}
    x' &= c_{00}x + c_{01}y + c_{02} \\
    y' &= c_{10}x + c_{11}y + c_{12}
\end{align*}
\]

where

- \(x'\) and \(y'\) are the pixel coordinates in the transformed image
- \(c_{ij}\) are the affine transform coefficients passed to the \(coeffs\) array during initialization

The affine warping is a general linear transform that incorporates such elementary transformations as scaling, rotation, translation, stretching, and shearing. It always transforms parallel lines into parallel lines and preserves equal distances between points on a line.

The \texttt{WarpAffineNearest} function operates with ROI. The transformed part of the image is resampled with the nearest neighbor interpolation method and stored in the destination image ROI. You need to define the destination image ROI origin by the following parameters: the offset of the destination ROI with respect to the destination image origin and the destination image ROI size. The parameter \(pSrc\) must point to the source image origin. The parameter \(pDst\) must point to the processed destination image ROI.

If you initialize the specification structure using the \texttt{WarpQuadNearestInit} function, the operations take place only inside the specified source quadrangle \(srcQuad\) that is set in \texttt{WarpQuadNearestInit}.

To specify the algorithm for borders processing, set the \texttt{borderType} and \texttt{pBorderValue} parameters when initializing the \texttt{IpplWarpSpec} structure. The data type of \texttt{pBorderValue} is automatically converted from \texttt{Ipp64f} to the data type of the processed images. The function supports the following algorithms for borders processing:

- If the border type is equal to \texttt{ippBorderRepl}, the source image outer pixels are replicated from the edge pixels.
If the border type is equal to ippBorderConst, the outer pixels are set to the constant value specified in pBorderValue.
If the border type is equal to ippBorderTransp or ippBorderInMem, destination image pixels mapped to the outer source image pixels are not changed.

Before using the WarpAffineNearest function, you need to initialize the IppiWarpSpec structure using the WarpAffineNearestInit function and compute the size of the external buffer pBuffer using the WarpGetBufferSize function.

To compute the affine transform parameters, use the GetAffineQuad, GetAffineBound, and GetAffineTransform functions.

For an example on how to use this function, refer to the WarpQuadNearestInit function description.

Return Values

- ippStsNoErr: Indicates no error.
- ippStsNullPtrErr: Indicates an error when one of the specified pointers is NULL.
- ippStsNoOperation: Indicates a warning when width or height of the destination image is equal to zero.
- ippStsBorderErr: Indicates an error if border type has an illegal value.
- ippStsContextMatchErr: Indicates an error when context data is invalid.
- ippStsNotSupportedModeErr: Indicates an error when the requested mode is not supported.
- ippStsSizeErr: Indicates an error when width or height of the source or destination image ROI is negative.
- ippStsStepErr: Indicates an error when the step value is not a multiple of data type.
- ippStsOutOfRangeErr: Indicates an error when the destination image offset point is outside the destination image origin.
- ippStsSizeWrn: Indicates a warning when the destination image ROI size is more than the destination image origin size.
- ippStsWrongIntersectQuad: Indicates a warning that no operation is performed if the destination ROI has no intersection with the transformed source image origin.

See Also

ROI Processing in Geometric Transforms
WarpAffineNearestInit Initializes the specification structure for image affine warping with the nearest neighbor interpolation method.
WarpQuadNearestInit Initializes the specification structure for warping an arbitrary quadrangle in the source image to the quadrangle in the destination image with the nearest neighbor interpolation method.
WarpGetBufferSize Computes the size of the work buffer for the warp transform.
GetAffineBound Computes the bounding rectangle for the source ROI transformed by the ippiWarpAffine function.
GetAffineQuad Computes vertex coordinates of the quadrangle, to which the source ROI rectangle is mapped by the affine transform.
GetAffineTransform Computes affine transform coefficients to map the source ROI to the quadrangle with the specified vertex coordinates.
**WarpAffineLinearInit**

*Initializes the specification structure for image affine warping with the linear interpolation method.*

**Syntax**

```c
IppStatus ippiWarpAffineLinearInit(IppiSize srcSize, IppiSize dstSize, IppDataType dataType, const double coeffs[2][3], IppiWarpDirection direction, int numChannels, IppiBorderType borderType, const Ipp64f* pBorderValue, int smoothEdge, IppiWarpSpec* pSpec);
```

**Platform-aware functions**

```c
IppStatus ippiWarpAffineLinearInit_L(IppiSizeL srcSize, IppiSizeL dstSize, IppDataType dataType, const double coeffs[2][3], IppiWarpDirection direction, int numChannels, IppiBorderType borderType, const Ipp64f* pBorderValue, int flags, IppiWarpSpec* pSpec);
```

**Include Files**

`ippi.h`

Flavors with the `_L` suffix: `ippi_l.h`

**Domain Dependencies**

Headers: `ippcore.h`, `ippvm.h`, `ipps.h`

Libraries: `ippcore.lib`, `ippvm.lib`, `ipps.lib`

**Parameters**

- `srcSize` Size of the source image, in pixels.
- `dstSize` Size of the destination image, in pixels.
- `dataType` Data type of the source and destination images. Supported values: `ipp8u`, `ipp16u`, `ipp16s`, `ipp32f`, and `ipp64f`.
- `coeffs` Coefficients for the affine transform.
- `numChannels` Number of channels in the image. Supported values: 1, 3, or 4.
- `borderType` Type of border. Supported values: `ippBorderConst` Values of all border pixels are set to a constant, `ippBorderRepl` Border is replicated from the edge pixels, `ippBorderTransp` Outer pixels are not processed, `ippBorderInMem` Border is obtained from the source image pixels in memory.

Mixed borders are also supported. They can be obtained by the bitwise operation OR between `ippBorderTransp` and the `ippBorderInMemTop`, `ippBorderInMemBottom`, `ippBorderInMemLeft`, `ippBorderInMemRight` values.
flags

Warp algorithm flags. Supported values:
- 0 - transformation without edge smoothing.
- 1 - transformation with edge smoothing. The feature is supported for the ippBorderTransp and ippBorderInMem border types.

pBorderValue

Pointer to the constant value to assign to pixels of the constant border. This parameter is applicable only to the ippBorderConst border type.

smoothEdge

Flag for edge smoothing. Supported values:
- 0 - transformation without edge smoothing.
- 1 - transformation with edge smoothing. This feature is supported only for the ippBorderTransp and ippBorderInMem border types.

pSpec

Pointer to the specification structure.

Description

This function initializes the IppiWarpSpec structure for the ippiWarpAffineLinear function that performs warp affine transformation with the linear interpolation method. To compute the size of the specification structure, use the WarpAffineGetSize function.

Return Values

- ippStsNoErr: Indicates no error.
- ippStsNullPtrErr: Indicates an error when:
  - one of the specified pointers is NULL, excepting pBorderValue
  - pBorderValue is NULL when border type is set to ippBorderConst
- ippStsNoOperation: Indicates a warning if width or height of any image is zero.
- ippStsSizeErr: Indicates an error when width or height of the source or destination image is less than, or equal to one.
- ippStsDataTypeErr: Indicates an error when dataType has an illegal value.
- ippStsWarpDirectionErr: Indicates an error when direction has an illegal value.
- ippStsCoeffErr: Indicates an error when affine transformation is singular.
- ippStsWrongIntersectQuad: Indicates a warning that no operation is performed if the transformed source image extended with borders has no intersection with the destination image.
- ippStsNotSupportedModeErr: Indicates an error when the requested mode is not supported. The edge smoothing feature is not supported for the ippBorderRepl and ippBorderConst border types.
- ippStsBorderErr: Indicates an error when borderType has an illegal value.
- ippStsNumChannelsErr: Indicates an error when numChannels has an illegal value.
- ippStsExceededSizeErr: Indicates an error if width or height of the destination image or the source image exceeds 33554431 (0x1FFFFFF).

See Also

WarpAffineGetSize Computes the size of the specification structure and the size of the external work buffer for the warp affine transform.
WarpQuadLinearInit

Initializes the specification structure for warping an arbitrary quadrangle in the source image to the quadrangle in the destination image with the linear interpolation method.

Syntax

IppStatus ippiWarpQuadLinearInit(IppiSize srcSize, const double srcQuad[4][2], IppiSize dstSize, const double dstQuad[4][2], IppiWarpTransformType transform, IppDataType dataType, int numChannels, IppiBorderType borderType, const Ipp64f* pBorderValue, int smoothEdge, IppiWarpSpec* pSpec);

Include Files

ippi.h

Domain Dependencies

Headers: ippcore.h, ippvm.h, ipps.h
Libraries: ippcore.lib, ippvm.lib, ipps.lib

Parameters

srcSize
Size of the source image, in pixels.

srcQuad
Quadrangle in the source image.

dstSize
Size of the destination image, in pixels.

dstQuad
Quadrangle in the destination image.

transform
Type of the warp transform. Supported values:
ippWarpAffine    Affine warping
ippWarpPerspective Perspective warping

dataType
Data type of the source and destination images. Supported values: ipp8u, ipp16u, ipp16s, and ipp32f.

numChannels
Number of channels in the image. Supported values: 1, 3, or 4.

borderType
Type of border. Supported values:
ippBorderTransp    Outer pixels are not processed.
ippBorderInMem    Border is obtained from the source image pixels in memory.

pBorderValue
Pointer to the constant value to assign to pixels of the constant border. This parameter is applicable only to the ippBorderConst border type.

smoothEdge
Flag for edge smoothing. Supported values:
• 0 - transformation without edge smoothing.
• 1 - transformation with edge smoothing.

pSpec
Pointer to the specification structure.
Description

This function initializes the IppiWarpSpec structure for warping an arbitrary quadrangle in the source image to quadrangle in the destination image with the linear interpolation method. To compute the size of the specification structure, use the WarpQuadGetSize function.

Transformation coefficients are computed internally, based on the mapping of the source quadrangle to the specified destination quadrangle dstQuad and transform type transform. In case of affine transform, the function computes the coordinates of the fourth vertex of the destination quadrangle that uniquely depends on the three other vertices. If the computed coordinates are not equal to the corresponding values specified in dstQuad, the function returns the warning message and continues initialization with the computed values.


For an example on how to use this function, refer to the example provided with the WarpQuadNearestInit function description.

You can apply the edge smoothing feature only if the source quadrangle entirely lies in the source image.

Return Values

<table>
<thead>
<tr>
<th>Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ippStsNoErr</td>
<td>Indicates no error.</td>
</tr>
<tr>
<td>ippStsNullPtrErr</td>
<td>Indicates an error when:</td>
</tr>
<tr>
<td></td>
<td>• One of the specified pointers is NULL, excepting pBorderValue</td>
</tr>
<tr>
<td></td>
<td>• The value of pBorderValue is NULL when the border type is set to ippBorderConst</td>
</tr>
<tr>
<td>ippStsSizeErr</td>
<td>Indicates an error when width or height of the source or destination image is less than, or equal to one.</td>
</tr>
<tr>
<td>ippStsDataTypeErr</td>
<td>Indicates an error when dataType has an illegal value.</td>
</tr>
<tr>
<td>ippStsWarpTransformErr</td>
<td>Indicates an error when transform has an illegal value.</td>
</tr>
<tr>
<td>ippStsQuadErr</td>
<td>Indicates an error when any of the given quadrangles is non-convex or degenerates into a triangle, line, or point.</td>
</tr>
<tr>
<td>ippStsWrongIntersectQuad</td>
<td>Indicates a warning that no operation is performed if the transformed source image extended with borders has no intersection with the destination image.</td>
</tr>
<tr>
<td>ippStsAffineQuadChanged</td>
<td>Indicates a warning when coordinates of the fourth vertex of dstQuad are changed by the function, if transform is set to ippWarpAffine.</td>
</tr>
<tr>
<td>ippStsBorderErr</td>
<td>Indicates an error when borderType has an illegal value.</td>
</tr>
<tr>
<td>ippStsNumChannelsErr</td>
<td>Indicates an error when numChannels has an illegal value.</td>
</tr>
</tbody>
</table>

See Also

WarpQuadGetSize Computes the size of the specification structure and the size of the temporary buffer for warping an arbitrary quadrangle in the source image to the quadrangle in the destination image.

WarpQuadNearestInit Initializes the specification structure for warping an arbitrary quadrangle in the source image to the quadrangle in the destination image with the nearest neighbor interpolation method.
WarpAffineLinear
Performs warp affine transformation of an image using the linear interpolation method.

Syntax
IppStatus ippiWarpAffineLinear_<mod>(const Ipp<datatype>* pSrc, int srcStep, Ipp<datatype>* pDst, int dstStep, IppiPoint dstRoiOffset, IppiSize dstRoiSize, const IppiWarpSpec* pSpec, Ipp8u* pBuffer);

Supported values for mod:

- 8u_C1R 16u_C1R 16s_C1R 32f_C1R 64f_C1R
- 8u_C3R 16u_C3R 16s_C3R 32f_C3R 64f_C3R
- 8u_C4R 16u_C4R 16s_C4R 32f_C4R 64f_C4R

Platform-aware functions
IppStatus ippiWarpAffineLinear_<mod>(const Ipp<datatype>* pSrc, IppSizeL srcStep, Ipp<datatype>* pDst, IppSizeL dstStep, IppiPointL dstRoiOffset, IppiSizeL dstRoiSize, const IppiWarpSpec* pSpec, Ipp8u* pBuffer);

Supported values for mod:

- 8u_C1R_L 16u_C1R_L 16s_C1R_L 32f_C1R_L 64f_C1R_L
- 8u_C3R_L 16u_C3R_L 16s_C3R_L 32f_C3R_L 64f_C3R_L
- 8u_C4R_L 16u_C4R_L 16s_C4R_L 32f_C4R_L 64f_C4R_L

Include Files
ippi.h

Flavors with the _L suffix: ippi_l.h

Domain Dependencies
Headers: ippcore.h, ippvm.h, ipps.h
Libraries: ippcore.lib, ippvm.lib, ipps.lib

Parameters
- pSrc
  Pointer to the source image.
- srcStep
  Distance, in bytes, between the starting points of consecutive lines in the source image buffer.
- pDst
  Pointer to the destination image ROI.
- dstStep
  Distance, in bytes, between the starting points of consecutive lines in the destination image buffer.
- dstRoiOffset
  Offset of the destination image ROI with respect to the destination image origin.


\textbf{Description}

This function transforms the source image pixel coordinates \((x, y)\) according to the following formulas:

\[
\begin{align*}
    x' &= c_{00} \cdot x + c_{01} \cdot y + c_{02} \\
    y' &= c_{10} \cdot x + c_{11} \cdot y + c_{12}
\end{align*}
\]

where

- \(x'\) and \(y'\) are the pixel coordinates in the transformed image
- \(c_j\) are the affine transform coefficients passed to the \textit{coeffs} array during initialization

The affine warping is a general linear transform that incorporates such elementary transformations as scaling, rotation, translation, stretching, and shearing. It always transforms parallel lines into parallel lines and preserves equal distances between points on a line.

The \texttt{ippiWarpAffineLinear} function operates with ROI. The transformed part of the image is resampled with the linear interpolation method and stored in the destination image ROI. You need to define the destination image ROI origin by the following parameters: the offset of the destination ROI with respect to the destination image origin and the destination image ROI size. The parameter \(pSrc\) must point to the source image origin. The parameter \(pDst\) must point to the processed destination image ROI.

If you initialize the specification structure using the \texttt{ippiWarpQuadLinearInit} function, the operations take place only inside the specified source quadrangle \textit{srcQuad} that is set in \texttt{ippiWarpQuadLinearInit}.

To specify the algorithm for borders processing, set the \texttt{borderType} and \(pBorderValue\) parameters when initializing the \texttt{IppiWarpSpec} structure. The data type of \(pBorderValue\) is automatically converted from \texttt{Ipp64f} to the data type of the processed images. The function supports the following algorithms for borders processing:

- If the border type is equal to \texttt{ippBorderRepl}, the source image outer pixels are replicated from the edge pixels.
- If the border type is equal to \texttt{ippBorderConst}, the outer pixels are set to the constant value specified in \textit{pBorderValue}.
- If the border type is equal to \texttt{ippBorderTransp} or \texttt{ippBorderInMem}, destination image pixels mapped to the outer source image pixels are not changed.

Before using the \texttt{ippiWarpAffineLinear} function, you need to initialize the \texttt{IppiWarpSpec} structure using the \texttt{ippiWarpAffineLinearInit} function and compute the size of the external buffer \(pBuffer\) using the \texttt{WarpGetBufferSize} function.

To compute the affine transform parameters, use the \texttt{GetAffineQuad}, \texttt{GetAffineBound}, and \texttt{GetAffineTransform} functions.

For an example on how to use this function, refer to the \texttt{WarpQuadNearestInit} function description.

\textbf{Return Values}

\begin{itemize}
    \item \texttt{ippStsNoErr} Indicates no error.
    \item \texttt{ippStsNullPtrErr} Indicates an error when one of the specified pointers is \texttt{NULL}.
    \item \texttt{ippStsNoOperation} Indicates a warning when width or height of the destination image is equal to zero.
    \item \texttt{ippStsBorderErr} Indicates an error if border type has an illegal value.
\end{itemize}
See Also
ROI Processing in Geometric Transforms
WarpAffineLinearInit Initializes the specification structure for image affine warping with the linear interpolation method.
WarpQuadLinearInit Initializes the specification structure for warping an arbitrary quadrangle in the source image to the quadrangle in the destination image with the linear interpolation method.
WarpGetBufferSize Computes the size of the work buffer for the warp transform.
GetAffineBound Computes the bounding rectangle for the source ROI transformed by the ippiWarpAffine function.
GetAffineQuad Computes vertex coordinates of the quadrangle, to which the source ROI rectangle is mapped by the affine transform.
GetAffineTransform Computes affine transform coefficients to map the source ROI to the quadrangle with the specified vertex coordinates.
WarpQuadNearestInit Initializes the specification structure for warping an arbitrary quadrangle in the source image to the quadrangle in the destination image with the nearest neighbor interpolation method.

WarpAffineCubicInit
Initializes the specification structure for image affine warping with the cubic interpolation method.

Syntax
IppStatus ippiWarpAffineCubicInit(IppSize srcSize, IppSize dstSize, IppDataType dataType, const double coeffs[2][3], IppiWarpDirection direction, int numChannels, Ipp64f valueB, Ipp64f valueC, IppiBorderType borderType, const Ipp64f* pBorderValue, int smoothEdge, IppiWarpSpec* pSpec, Ipp8u* pInitBuf);

Platform-aware functions
IppStatus ippiWarpAffineCubicInit_L(IppSizeL srcSize, IppSizeL dstSize, IppDataType dataType, const double coeffs[2][3], IppiWarpDirection direction, int numChannels, Ipp64f valueB, Ipp64f valueC, IppiBorderType borderType, const Ipp64f* pBorderValue, int flags, IppiWarpSpec* pSpec, Ipp8u* pInitBuf);

Include Files
ippi.h

Flavors with the _L suffix: ippi_l.h
Domain Dependencies

Headers: ippcore.h, ippvm.h, ipps.h
Libraries: ippcore.lib, ippvm.lib, ipps.lib

Parameters

srcSize
Size of the source image, in pixels.
dstSize
Size of the destination image, in pixels.
dataType
Data type of the source and destination images. Supported values: ipp8u, ipp16u, ipp16s, ipp32f, and ipp64f.
coeffs
Coefficients for the affine transform.
direction
Transformation direction. Supported values:
ippWarpForward  Forward transformation
ippWarpBackward  Backward transformation

numChannels
Number of channels in the image. Supported values: 1, 3, or 4.
valueB, valueC
The first (B) and second (C) parameter for the cubic filter.

borderType
Type of border. Supported values:
ippBorderConst  Values of all border pixels are set to a constant.
ippBorderRepl  Border is replicated from the edge pixels.
ippBorderTransp  Outer pixels are not processed.
ippBorderInMem  Border is obtained from the source image pixels in memory.

Mixed borders are also supported. They can be obtained by the bitwise operation OR between ippBorderTransp and the ippBorderInMemTop, ippBorderInMemBottom, ippBorderInMemLeft, ippBorderInMemRight values.

pBorderValue
Pointer to the constant value to assign to pixels of the constant border. This parameter is applicable only to the ippBorderConst border type.

flags
Warp algorithm flags. Supported values:
• 0 - transformation without edge smoothing.
• 1 - transformation with edge smoothing. The feature is supported for the ippBorderTransp and ippBorderInMem border types.

smoothEdge
The smooth edge flag. The following values are supported: 0 - transform without edge smoothing, 1 - transform with edge smoothing. This feature is supported only for the ippBorderTransp and ippBorderInMem border types.
pSpec
Pointer to the specification structure.
pInitBuf
Pointer to the temporary buffer for the cubic filter initialization.
Description
This function initializes the IpplWarpSpec structure for the ipplWarpAffineCubic function that performs warp affine transformation with the cubic interpolation method. Before using this function, compute the size of the specification structure and the size of the external buffer pInitBuf using the WarpAffineGetSize function.

Application Notes
Intel IPP warping functions do not support the IPPI_INTER_CUBIC mode. You can use interpolation with two-parameter cubic filters instead. This approach provides the interpolation quality that is comparable with IPPI_INTER_CUBIC. For interpolation formulas refer to Interpolation with Two-Parameter Cubic Filters. You can vary B and C values to get a result that fits the required task.

Return Values
ippStsNoErr Indicates no error.
ippStsNullPtrErr Indicates an error when:
  • one of the specified pointers is NULL, excepting pBorderValue
  • pBorderValue is NULL when border type is set to ippBorderConst
ippStsNoOperation Indicates a warning if width or height of any image is zero.
ippStsSizeErr Indicates an error when width or height of the source or destination image is less than, or equal to one.
ippStsDataTypeErr Indicates an error when dataType has an illegal value.
ippStsWarpDirectionErr Indicates an error when direction has an illegal value.
ippStsCoeffErr Indicates an error when affine transformation is singular.
ippStsWrongIntersectQuad Indicates a warning that no operation is performed if the transformed source image extended with borders has no intersection with the destination image.
ippStsNotSupportedModeErr Indicates an error when the requested mode is not supported. The edge smoothing feature is not supported for the ippBorderRepl and ippBorderConst border types.
ippStsBorderErr Indicates an error when borderType has an illegal value.
ippStsNumChannelsErr Indicates an error when numChannels has an illegal value.
ippStsExceededSizeErr Indicates an error if width or height of the destination image or the source image exceeds 33554431 (0x1FFFFFF).

See Also
WarpAffineGetSize Computes the size of the specification structure and the size of the external work buffer for the warp affine transform.
Interpolation with Two-Parameter Cubic Filters

WarpQuadCubicInit
Initializes the specification structure for warping an arbitrary quadrangle in the source image to the quadrangle in the destination image with the cubic interpolation method.
Syntax
IppStatus ippiWarpQuadCubicInit(IppiSize srcSize, const double srcQuad[4][2], IppiSize dstSize, const double dstQuad[4][2], IppiWarpTransformType transform, IppDataType dataType, int numChannels, Ipp64f valueB, Ipp64f valueC, IppiBorderType borderType, const Ipp64f* pBorderValue, int smoothEdge, IppiWarpSpec* pSpec, Ipp8u* pInitBuf);

Include Files
ippi.h

Domain Dependencies
Headers: ippcore.h, ippvm.h, ipps.h
Libraries: ippcore.lib, ippvm.lib, ipps.lib

Parameters
srcSize Size of the source image, in pixels.
srcQuad Quadrangle in the source image.

dstSize Size of the destination image, in pixels.
dstQuad Quadrangle in the destination image.

transform Type of the warp tranform. Supported values:
ippWarpAffine Affine warping
ippWarpPerspective Perspective warping

dataType Data type of the source and destination images. Supported values: ipp8u, ipp16u, ipp16s, and ipp32f.

numChannels Number of channels in the image. Supported values: 1, 3, or 4.

valueB, valueC The first (B) and second (C) parameter for the cubic filter.

borderType Type of border. Supported values:
ippBorderTransp Outer pixels are not processed.
ippBorderInMem Border is obtained from the source image pixels in memory.

pBorderValue Pointer to the constant value to assign to pixels of the constant border. This parameter is applicable only to the ippBorderConst border type.

smoothEdge Flag for edge smoothing. Supported values:
  • 0 - transformation without edge smoothing.
  • 1 - transformation with edge smoothing.

pSpec Pointer to the specification structure.

pInitBuf Pointer to the temporary buffer for the cubic filter initialization.
Description
This function initializes the IppiWarpSpec structure for warping an arbitrary quadrangle in the source image to quadrangle in the destination image with the cubic interpolation method. Before using this function, compute the size of the specification structure and the size of the external buffer pInitBuf using the WarpQuadGetSize function.

Transformation coefficients are computed internally, based on the mapping of the source quadrangle to the specified destination quadrangle dstQuad and transform type transform. In case of affine transform, the function computes the coordinates of the fourth vertex of the destination quadrangle that uniquely depends on the three other vertices. If the computed coordinates are not equal to the corresponding values specified in dstQuad, the function returns the warning message and continues initialization with the computed values.


For an example on how to use this function, refer to the example provided with the WarpQuadNearestInit function description.

You can apply the edge smoothing feature only if the source quadrangle entirely lies in the source image.

Return Values

<table>
<thead>
<tr>
<th>Error Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ippStsNoErr</td>
<td>Indicates no error.</td>
</tr>
<tr>
<td>ippStsNullPtrErr</td>
<td>Indicates an error when:</td>
</tr>
<tr>
<td></td>
<td>• One of the specified pointers is NULL, excepting pBorderValue</td>
</tr>
<tr>
<td></td>
<td>• The value of pBorderValue is NULL when the border type is set to ippBorderConst</td>
</tr>
<tr>
<td>ippStsSizeErr</td>
<td>Indicates an error when width or height of the source or destination image is less than, or equal to one.</td>
</tr>
<tr>
<td>ippStsDataTypeErr</td>
<td>Indicates an error when dataType has an illegal value.</td>
</tr>
<tr>
<td>ippStsWarpTransformErr</td>
<td>Indicates an error when transform has an illegal value.</td>
</tr>
<tr>
<td>ippStsQuadErr</td>
<td>Indicates an error when any of the given quadrangles is non-convex or degenerates into a triangle, line, or point.</td>
</tr>
<tr>
<td>ippStsWrongIntersectQuad</td>
<td>Indicates a warning that no operation is performed if the transformed source image extended with borders has no intersection with the destination image.</td>
</tr>
<tr>
<td>ippStsAffineQuadChanged</td>
<td>Indicates a warning when coordinates of the fourth vertex of dstQuad are changed by the function, if transform is set to ippWarpAffine.</td>
</tr>
<tr>
<td>ippStsBorderErr</td>
<td>Indicates an error when borderType has an illegal value.</td>
</tr>
<tr>
<td>ippStsNumChannelsErr</td>
<td>Indicates an error when numChannels has an illegal value.</td>
</tr>
</tbody>
</table>

See Also
WarpQuadGetSize Computes the size of the specification structure and the size of the temporary buffer for warping an arbitrary quadrangle in the source image to the quadrangle in the destination image.
WarpQuadNearestInit Initializes the specification structure for warping an arbitrary quadrangle in the source image to the quadrangle in the destination image with the nearest neighbor interpolation method.
WarpAffineCubic
Performs warp affine transformation of an image using the cubic interpolation method.

Syntax
IppStatus ippiWarpAffineCubic_<mod>(const Ipp<datatype>* pSrc, int srcStep, Ipp<datatype>* pDst, int dstStep, IppIPoint dstRoiOffset, IppSize dstRoiSize, const IppiWarpSpec* pSpec, Ipp8u* pBuffer);

Supported values for mod:

- 8u_C1R
- 16u_C1R
- 16s_C1R
- 32f_C1R
- 64f_C1R

- 8u_C3R
- 16u_C3R
- 16s_C3R
- 32f_C3R
- 64f_C3R

- 8u_C4R
- 16u_C4R
- 16s_C4R
- 32f_C4R
- 64f_C4R

Platform-aware functions

Supported values for mod:

- 8u_C1R_L
- 16u_C1R_L
- 16s_C1R_L
- 32f_C1R_L
- 64f_C1R_L

- 8u_C3R_L
- 16u_C3R_L
- 16s_C3R_L
- 32f_C3R_L
- 64f_C3R_L

- 8u_C4R_L
- 16u_C4R_L
- 16s_C4R_L
- 32f_C4R_L
- 64f_C4R_L

Include Files
ippi.h

Flavors with the _L suffix: ippi_l.h

Domain Dependencies
Headers: ippcore.h, ippvm.h, ipps.h
Libraries: ippcore.lib, ippvm.lib, ipps.lib

Parameters

- pSrc
  Pointer to the source image.
- srcStep
  Distance, in bytes, between the starting points of consecutive lines in the source image buffer.
- pDst
  Pointer to the destination image ROI.
- dstStep
  Distance, in bytes, between the starting points of consecutive lines in the destination image buffer.
- dstRoiOffset
  Offset of the destination image ROI with respect to the destination image origin.
Description

This function transforms the source image pixel coordinates \((x, y)\) according to the following formulas:

\[
\begin{align*}
x' &= c_{00} \cdot x + c_{01} \cdot y + c_{02} \\
y' &= c_{10} \cdot x + c_{11} \cdot y + c_{12}
\end{align*}
\]

where

- \(x'\) and \(y'\) are the pixel coordinates in the transformed image
- \(c_{ij}\) are the affine transform coefficients passed to the \(coeffs\) array during initialization

The affine warping is a general linear transform that incorporates such elementary transformations as scaling, rotation, translation, stretching, and shearing. It always transforms parallel lines into parallel lines and preserves equal distances between points on a line.

The \(ippiWarpAffineCubic\) function operates with ROI. The transformed part of the image is resampled with the cubic interpolation method and stored in the destination image ROI. You need to define the destination image ROI origin by the following parameters: the offset of the destination ROI with respect to the destination image origin and the destination image ROI size. The parameter \(pSrc\) must point to the source image origin. The parameter \(pDst\) must point to the processed destination image ROI.

If you initialize the specification structure using the \(ippiWarpQuadCubicInit\) function, the operations take place only inside the specified source quadrangle \(srcQuad\) that is set in \(ippiWarpQuadCubicInit\).

To specify the algorithm for borders processing, set the \(borderType\) and \(pBorderValue\) parameters when initializing the \(IpplWarpSpec\) structure. The data type of \(pBorderValue\) is automatically converted from \(Ipp64f\) to the data type of the processed images. The function supports the following algorithms for borders processing:

- If the border type is equal to \(ippBorderRepl\), the source image outer pixels are replicated from the edge pixels.
- If the border type is equal to \(ippBorderConst\), the outer pixels are set to the constant value specified in \(pBorderValue\).
- If the border type is equal to \(ippBorderTransp\), destination image pixels mapped to the outer source image pixels are not changed. The outer pixels required for cubic interpolation are replicated from the edge pixels.
- If the border type is equal to \(ippBorderInMem\), destination image pixels mapped to the outer source image pixels are not changed. The outer pixels required for cubic interpolation are obtained from the out of the source image origin space. Cubic interpolation requires additional one-pixel edge from each source image side.
- The mixed border types can be obtained by the bitwise operation \(OR\) between \(ippBorderTransp\) and the \(ippBorderInMemTop\), \(ippBorderInMemBottom\), \(ippBorderInMemLeft\), \(ippBorderInMemRight\) values.

Before using the \(ippiWarpAffineCubic\) function, you need to initialize the \(IpplWarpSpec\) structure using the \(ippiWarpAffineCubicInit\) function and compute the size of the external buffer \(pBuffer\) using the \(WarpGetBufferSize\) function.

To compute the affine transform parameters, use the \(GetAffineQuad\), \(GetAffineBound\), and \(GetAffineTransform\) functions.

For an example on how to use this function, refer to the \(WarpQuadNearestInit\) function description.
Return Values

ippStsNoErr Indicates no error.
ippStsNullPtrErr Indicates an error when one of the specified pointers is NULL.
ippStsNoOperation Indicates a warning when width or height of the destination image is equal to zero.
ippStsBorderErr Indicates an error if border type has an illegal value.
ippStsContextMatchErr Indicates an error when context data is invalid.
ippStsNotSupportedModeErr Indicates an error when the requested mode is not supported.
ippStsSizeErr Indicates an error when width or height of the source or destination image ROI is negative.
ippStsStepErr Indicates an error when the step value is not a multiple of data type.
ippStsOutOfRangeErr Indicates an error when the destination image offset point is outside the destination image origin.
ippStsSizeWrn Indicates a warning when the destination image ROI size is more than the destination image origin size.
ippStsWrongIntersectQuad Indicates a warning that no operation is performed, if the transformed source image has no intersection with the destination image.

See Also
ROI Processing in Geometric Transforms
WarpAffineCubicInit Initializes the specification structure for image affine warping with the cubic interpolation method.
WarpQuadCubicInit Initializes the specification structure for warping an arbitrary quadrangle in the source image to the quadrangle in the destination image with the cubic interpolation method.
WarpGetBufferSize Computes the size of the work buffer for the warp transform.
GetAffineBound Computes the bounding rectangle for the source ROI transformed by the ippiWarpAffine function.
GetAffineQuad Computes vertex coordinates of the quadrangle, to which the source ROI rectangle is mapped by the affine transform.
GetAffineTransform Computes affine transform coefficients to map the source ROI to the quadrangle with the specified vertex coordinates.
WarpQuadNearestInit Initializes the specification structure for warping an arbitrary quadrangle in the source image to the quadrangle in the destination image with the nearest neighbor interpolation method.

GetPerspectiveQuad

Computes vertex coordinates of the quadrangle, to which the source ROI rectangle is mapped by the perspective transform.

Syntax

IppStatus ippiGetPerspectiveQuad(IppiRect srcRoi, double quad[4][2], const double coeffs[3][3]);

Include Files
ippi.h
Domain Dependencies
Headers: ippcore.h, ippvm.h, ipps.h
Libraries: ippcore.lib, ippvm.lib, ipps.lib

Parameters
srcRoi  
Region of interest in the source image (of the IppiRect type).
quad  
Output array. Contains vertex coordinates of the quadrangle, to which the source ROI is mapped by the perspective transform function.
coeffs  
The given perspective transform coefficients.

Description
This function operates with ROI (see ROI Processing in Geometric Transforms). This function is used as a support function for the WarpPerspectiveNearest, WarpPerspectiveLinear, and WarpPerspectiveCubic functions. It computes vertex coordinates of the quadrangle, to which the source rectangular ROI is mapped by the perspective transform function using the given coefficients coeffs.
The first dimension [4] of the array quad[4][2] is equal to the number of vertices, and the second dimension [2] means x and y coordinates of the vertex. Quadrangle vertices have the following meaning:
quad[0] corresponds to the transformed top-left corner of the source ROI,
quad[1] corresponds to the transformed top-right corner of the source ROI,
quad[2] corresponds to the transformed bottom-right corner of the source ROI,
quad[3] corresponds to the transformed bottom-left corner of the source ROI.

Return Values
ippStsNoErr  
Indicates no error. Any other value indicates an error.
ippStsSizeErr  
Indicates an error condition if srcRoi has a size field with zero or negative value.
ippStsCoeffErr  
Indicates an error condition if coefficient values are invalid.

GetPerspectiveBound
Computes the bounding rectangle for the source ROI transformed by the ippiWarpPerspective function.

Syntax
IppStatusippiGetPerspectiveBound(IppiRect srcRoi, double bound[2][2], const double coeffs[3][3]);

Include Files
ippi.h

Domain Dependencies
Headers: ippcore.h, ippvm.h, ipps.h
Libraries: ippcore.lib, ippvm.lib, ipps.lib
**Parameters**

*srcRoi*  
Region of interest in the source image (of the `IppiRect` type).

*bound*  
Output array. Contains vertex coordinates of the bounding rectangle for the transformed source ROI.

*coeffs*  
The given perspective transform coefficients.

**Description**

This function operates with ROI (see ROI Processing in Geometric Transforms).

This function is used as a support function for the `WarpPerspectiveNearest`, `WarpPerspectiveLinear`, and `WarpPerspectiveCubic` functions. It computes vertex coordinates of the smallest bounding rectangle for the quadrangle `quad`, to which the source ROI is mapped by the perspective transform function using the given coefficients `coeffs`.

`bound[0]` specifies \( x, y \) coordinates of the top-left corner, `bound[1]` specifies \( x, y \) coordinates of the bottom-right corner.

**Return Values**

*ippStsNoErr*  
Indicates no error. Any other value indicates an error or a warning.

*ippStsSizeErr*  
Indicates an error condition if `srcRoi` has a size field with zero or negative value.

*ippStsCoeffErr*  
Indicates an error condition if coefficient values are invalid.

**GetPerspectiveTransform**

*Computes the perspective transform coefficients to map the source ROI to the quadrangle with the specified vertex coordinates.*

**Syntax**

```c
IppStatus ippiGetPerspectiveTransform(IppiRect srcRoi, const double quad[4][2], double coeffs[3][3]);
```

**Include Files**

`ippi.h`

**Domain Dependencies**

*Headers:* `ippcore.h`, `ippvm.h`, `ipps.h`

*Libraries:* `ippcore.lib`, `ippvm.lib`, `ipps.lib`

**Parameters**

*srcRoi*  
Region of interest in the source image (of the `IppiRect` type).

*quad*  
Vertex coordinates of the quadrangle, to which the source ROI is mapped by the perspective transform function.

*coeffs*  
Output array. Contains the target perspective transform coefficients.

**Description**

This function operates with ROI (see ROI Processing in Geometric Transforms).
This function is used as a support function for the WarpPerspectiveNearest, WarpPerspectiveLinear, and WarpPerspectiveCubic functions. It computes the coefficients \( \text{coeffs} \) that should be used by the function to map the source rectangular ROI to the quadrangle with the given vertex coordinates \( \text{quad} \).

The first dimension \([4]\) of the array \( \text{quad}[4][2] \) is equal to the number of vertices, and the second dimension \([2]\) means \( x \) and \( y \) coordinates of the vertex. Quadrangle vertices have the following meaning:

- \( \text{quad}[0] \) corresponds to the transformed top-left corner of the source ROI,
- \( \text{quad}[1] \) corresponds to the transformed top-right corner of the source ROI,
- \( \text{quad}[2] \) corresponds to the transformed bottom-right corner of the source ROI,
- \( \text{quad}[3] \) corresponds to the transformed bottom-left corner of the source ROI.

### Return Values

- **ippStsNoErr**: Indicates no error. Any other value indicates an error.
- **ippStsSizeErr**: Indicates an error condition if \( \text{srcRoi} \) has a size field with zero or negative value.
- **ippStsCoeffErr**: Indicates an error condition if coefficient values are invalid.
- **ippStsRectErr**: Indicates an error condition if width or height of the \( \text{srcRoi} \) is less than or equal to 1.

### Example

To better understand usage of the \text{ippiGetPerspectiveTransform} function, refer to the following example in the examples archive available for download from https://software.intel.com/en-us/ipp-manual-examples:

GetPerspectiveTransform.c

#### WarpGetRectInfinite

*Returns an infinite rectangle.*

**Syntax**

\[
\text{IppiRect} \quad \text{ippiWarpGetRectInfinite} (\text{void});
\]

**Include Files**

\[
\text{ippi.h}
\]

**Domain Dependencies**

- **Headers**: ippcore.h, ippvm.h, ipps.h
- **Libraries**: ippcore.lib, ippvm.lib, ipps.lib

**Description**

The function returns a constant rectangle that is considered as an infinite rectangle by Intel IPP WarpPerspective functions. Use this rectangle in the following functions: WarpPerspectiveGetSize, WarpPerspectiveInitNearest, WarpPerspectiveInitLinear, and WarpPerspectiveCubic.

**NOTE**

The macro definition is: \#define ippRectInfinite ippiWarpGetRectInfinite().
**WarpPerspectiveGetSize**

Computes the size of the specification structure and the size of the external work buffer for the warp perspective transform.

**Syntax**

```c
IppStatus ippiWarpPerspectiveGetSize(IppiSize srcSize, IppiRect srcRoi, IppiSize dstSize, IppDataType dataType, const double coeffs[3][3], IppiInterpolationType interpolation, IppiWarpDirection direction, IppiBorderType borderType, int* pSpecSize, int* pInitBufSize);
```

**Include Files**

ippi.h

**Domain Dependencies**

**Headers:** ippcore.h, ippvm.h, ipps.h

**Libraries:** ippcore.lib, ippvm.lib, ipps.lib

**Parameters**

- `srcSize`  
  Size of the source image, in pixels.

- `srcRoi`  
  Source image ROI (of the IppRect type).

- `dstSize`  
  Size of the destination image, in pixels.

- `dataType`  
  Data type of the source and destination images. Supported values: ipp8u, ipp16u, ipp16s, and ipp32f.

- `coeffs`  
  Coefficients for the perspective transform.

- `interpolation`  
  Interpolation method. Supported values: ippNearest, ippLinear, and ippCubic.

- `direction`  
  Transformation direction. Supported values:
  - ippWarpForward  
    Forward transformation
  - ippWarpBackward  
    Backward transformation

- `borderType`  
  Type of border. Supported values:
  - ippBorderConst  
    Values of all border pixels are set to a constant.
  - ippBorderRepl  
    Border is replicated from the edge pixels.
  - ippBorderTransp  
    Outer pixels are not processed.
  - ippBorderInMem  
    Border is obtained from the source image pixels in memory.

  Mixed borders are also supported. They can be obtained by the bitwise operation **OR** between ippBorderTransp and the ippBorderInMemTop, ippBorderInMemBottom, ippBorderInMemLeft, ippBorderInMemRight values.

- `pSpecSize`  
  Pointer to the size, in bytes, of the specification structure.

- `pInitBufSize`  
  Pointer to the size, in bytes, of the temporary buffer.
Description
This function computes the size of the specification structure and the external work buffer for the following
functions, depending on the interpolation parameter: ippiWarpPerspectiveNearestInit,
ippiWarpPerspectiveLinearInit, or ippiWarpPerspectiveCubicInit.

You can set the value of the srcRoi parameter to ippRectInfinite, which means that the ROI is not
specified.

Return Values

ippiStsNoErr Indicates no error.
ippiStsNullPtrErr Indicates an error when one of the specified pointers is NULL.
ippiStsSizeErr Indicates an error in the following cases:
- If the width or height of the source or destination image is less
  than, or equal to one.
- If one of the calculated sizes exceeds the maximum positive 32-bit
  signed integer value. The size of the one of the processed images
  is too large.
ippiStsRectErr Indicates an error in the following cases, if the source image
ROI is not ippRectInfinite:
- If the source image ROI has no intersection with the image.
- Either x or y component of the source image ROI is negative.
- Width or height of the source image ROI is less than, or equal to
  zero.
ippiStsDataTypeErr Indicates an error when dataType has an illegal value.
ippiStsWarpDirectionErr Indicates an error when direction has an illegal value.
ippiStsInterpolationErr Indicates an error when interpolation has an illegal value.
ippiStsCoeffErr Indicates an error when perspective transformation is singular.
ippiStsWrongIntersectQuad Indicates a warning that no operation is performed if the
transformed source image extended with borders has no
intersection with the destination image.
ippiStsBorderErr Indicates an error when borderType has an illegal value.
ippiStsSizeWrn Indicates a warning when srcRoi exceeds the source image.

See Also
WarpPerspectiveNearestInit Initializes the specification structure for image perspective warping
with the nearest neighbor interpolation method.
WarpPerspectiveLinearInit Initializes the specification structure for image perspective warping with
the linear interpolation method.
WarpPerspectiveCubicInit Initializes the specification structure for image perspective warping with
the cubic interpolation method.

WarpPerspectiveNearestInit
Initializes the specification structure for image perspective warping with the nearest neighbor
interpolation method.
Syntax
IppStatus ippiWarpPerspectiveNearestInit(IppiSize srcSize, IppiRect srcRoi, IppiSize dstSize, IppDataType dataType, const double coeffs[3][3], IppiWarpDirection direction, int numChannels, IppiBorderType borderType, const Ipp64f* pBorderValue, int smoothEdge, IppiWarpSpec* pSpec);

Include Files
ippi.h

Domain Dependencies
Headers: ippcore.h, ippvm.h, ipps.h
Libraries: ippcore.lib, ippvm.lib, ipps.lib

Parameters

srcSize  
Size of the source image, in pixels.

srcRoi  
Source image ROI (of the IppRect type).

dstSize  
Size of the destination image, in pixels.

dataType  
Data type of the source and destination images. Supported values: ipp8u, ipp16u, ipp16s, and ipp32f.

coeffs  
Coefficients for the perspective transform.

direction  
Transformation direction. Supported values:
ippWarpForward  Forward transformation
ippWarpBackward  Backward transformation

numChannels  
Number of channels in the image. Supported values: 1, 3, or 4.

borderType  
Type of border. Supported values:
ippBorderConst  Values of all border pixels are set to a constant.
ippBorderRepl  Border is replicated from the edge pixels.
ippBorderTransp  Outer pixels are not processed.
ippBorderInMem  Border is obtained from the source image pixels in memory.

Mixed borders are also supported. They can be obtained by the bitwise operation OR between ippBorderTransp and the ippBorderInMemTop, ippBorderInMemBottom, ippBorderInMemLeft, ippBorderInMemRight values.

pBorderValue  
Pointer to the constant value to assign to pixels of the constant border. This parameter is applicable only to the ippBorderConst border type.

smoothEdge  
Flag for edge smoothing. Supported values:
• 0 - transformation without edge smoothing.
• 1 - transformation with edge smoothing. This feature is supported only for the ippBorderTransp and ippBorderInMem border types.
pSpec  

**Description**  
This function initializes the IppiWarpSpec structure for the WarpPerspectiveNearest function that performs warp perspective transformation with the nearest neighbor interpolation method. To compute the size of the specification structure, use the WarpPerspectiveGetSize function.

You can set the value of the srcRoi parameter to ippRectInfinite, which means that the ROI is not specified.

**Return Values**

- **ippStsNoErr** Indicates no error.
- **ippStsNullPtrErr** Indicates an error when:
  - pSpec is NULL
  - pBorderValue is NULL when the border type is set to ippBorderConst
- **ippStsSizeErr** Indicates an error when width or height of the source or destination image is less than, or equal to one.
- **ippStsRectErr** Indicates an error in the following cases, if the source image ROI is not ippRectInfinite:
  - If the source image ROI has no intersection with the image.
  - Either x or y component of the source image ROI is negative.
  - Width or height of the source image ROI is less than, or equal to zero.
- **ippStsDataTypeErr** Indicates an error when dataType has an illegal value.
- **ippStsWarpDirectionErr** Indicates an error when direction has an illegal value.
- **ippStsCoeffErr** Indicates an error when perspective transformation is singular.
- **ippStsWrongIntersectQuad** Indicates a warning that no operation is performed if the transformed source image extended with borders has no intersection with the destination image.
- **ippStsNotSupportedModeErr** Indicates an error when the requested mode is not supported. The edge smoothing feature is not supported for the ippBorderRepl and ippBorderConst border types.
- **ippStsBorderErr** Indicates an error when borderType has an illegal value.
- **ippStsNumChannelsErr** Indicates an error when numChannels has an illegal value.
- **ippStsSizeWrn** Indicates a warning when srcRoi exceeds the source image.

**See Also**

- **WarpPerspectiveGetSize** Computes the size of the specification structure and the size of the external work buffer for the warp perspective transform.
- **WarpPerspectiveNearest** Performs warp perspective transformation of an image using the nearest neighbor interpolation method.

**WarpPerspectiveNearest**

Performs warp perspective transformation of an image using the nearest neighbor interpolation method.
Syntax

IppStatusippiWarpPerspectiveNearest_<mod>(const Ipp<datatype>* pSrc, int srcStep, Ipp<datatype> pDst, int dstStep, IppiPoint dstRoiOffset, IppiSize dstRoiSize, const IppiWarpSpec* pSpec, Ipp8u* pBuffer);

Supported values for mod:

- 8u_C1R
- 16u_C1R
- 16s_C1R
- 32f_C1R
- 8u_C3R
- 16u_C3R
- 16s_C3R
- 32f_C3R
- 8u_C4R
- 16u_C4R
- 16s_C4R
- 32f_C4R

Include Files

ippi.h

Domain Dependencies

Headers: ippcore.h, ippvm.h, ipps.h
Libraries: ippcore.lib, ippvm.lib, ipps.lib

Parameters

- pSrc Pointer to the source image.
- srcStep Distance, in bytes, between the starting points of consecutive lines in the source image buffer.
- pDst Pointer to the destination image ROI.
- dstStep Distance, in bytes, between the starting points of consecutive lines in the destination image buffer.
- dstRoiOffset Offset of the destination image ROI with respect to the destination image origin.
- dstRoiSize Size of the destination image ROI, in pixels.
- pSpec Pointer to the specification structure for the warp operation.
- pBuffer Pointer to the work buffer.

Description

This function transforms the source image pixel coordinates \((x, y)\) according to the following formulas:

\[
\begin{align*}
x' &= \frac{(c_{00}*x + c_{01}*y + c_{02})}{(c_{20}*x + c_{21}*y + c_{22})} \\
y' &= \frac{(c_{10}*x + c_{11}*y + c_{12})}{(c_{20}*x + c_{21}*y + c_{22})}
\end{align*}
\]

where

- \(x'\) and \(y'\) are the pixel coordinates in the transformed image
- \(c_{ij}\) are the affine transform coefficients passed to the coeffs array during initialization

TheippiWarpPerspectiveNearest function operates with ROI (see Regions of Interest in Intel IPP). The transformed part of the source image is resampled with the nearest neighbor interpolation method and stored in the destination image ROI. You need to define the destination image ROI origin by the following parameters: the offset of the destination ROI with respect to the destination image origin and the destination image ROI size. The parameter pDst must point to the processed destination image ROI.
If you initialize the warp specification structure using the WarpPerspectiveNearestInit function, you can specify the source image ROI in the following ways:

- Set the srcRoi value to ippRectInfinite, which means that the ROI is not specified. In this case, pSrc must point to the processed source image. Pixels that are outside the source image boundaries are computed according to the specified border type.
- Set the srcRoi value to the part of the processed source image. In this case, pSrc must point to the processed source image ROI. The operations take place only inside the specified region of interest srcRoi. It means that the destination image pixels mapped to the outer pixels of the specified source image region are not changed.

If you initialize the warp specification structure using the WarpQuadNearestInit function, set the pSrc value to the processed source image. The operations take place only inside the source quadrangle srcQuad that is specified in the WarpQuadNearestInit function.

To specify the algorithm for borders processing, set the borderType and pBorderValue parameters when initializing the IppiWarpSpec structure. The data type of borderValue is automatically converted from Ipp64f to the data type of the processed images. The function supports the following algorithms for borders processing:

- If the border type is equal to ippBorderRepl, the source image outer pixels are replicated from the edge pixels.
- If the border type is equal to ippBorderConst, the outer pixels are set to the constant value specified in pBorderValue.
- If the border type is equal to ippBorderTransp or ippBorderInMem, destination image pixels mapped to the outer source image pixels are not changed.

Before using theippiWarpPerspectiveNearest function, you need to initialize the IppiWarpSpec structure using the WarpPerspectiveNearestInit function and compute the size of the external buffer pBuffer using the WarpGetBufferSize function.

To compute the perspective transform parameters, use the GetPerspectiveQuad, GetPerspectiveBound, and GetPerspectiveTransform functions.

**Return Values**

<table>
<thead>
<tr>
<th>Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ippStsNoErr</td>
<td>Indicates no error.</td>
</tr>
<tr>
<td>ippStsNullPtrErr</td>
<td>Indicates an error when one of the specified pointers is NULL.</td>
</tr>
<tr>
<td>ippStsNoOperation</td>
<td>Indicates a warning when width or height of the destination image is equal to zero.</td>
</tr>
<tr>
<td>ippStsContextMatchErr</td>
<td>Indicates an error when context data is invalid.</td>
</tr>
<tr>
<td>ippStsNotSupportedModeErr</td>
<td>Indicates an error when the requested mode is not supported.</td>
</tr>
<tr>
<td>ippStsSizeErr</td>
<td>Indicates an error when width or height of the source or destination image ROI is negative.</td>
</tr>
<tr>
<td>ippStsStepErr</td>
<td>Indicates an error when the step value is not a multiple of data type.</td>
</tr>
<tr>
<td>ippStsOutOfRangeErr</td>
<td>Indicates an error when the destination image offset point is outside the destination image origin.</td>
</tr>
<tr>
<td>ippStsSizeWrn</td>
<td>Indicates a warning when the destination image ROI size is more than the destination image origin size.</td>
</tr>
<tr>
<td>ippStsWrongIntersectQuad</td>
<td>Indicates a warning that no operation is performed if the transformed source image extended with borders has no intersection with the destination image.</td>
</tr>
</tbody>
</table>
Example
To better understand usage of Intel IPP functionality for perspective warping with Nearest Neighbor, Linear, and Cubic interpolation modes, refer to the following example in the examples archive available for download from https://software.intel.com/en-us/ipp-manual-examples:

WarpPerspectiveNearest.c

See Also
ROI Processing in Geometric Transforms

WarpPerspectiveNearestInit Initializes the specification structure for image perspective warping with the nearest neighbor interpolation method.

WarpQuadNearestInit Initializes the specification structure for warping an arbitrary quadrangle in the source image to the quadrangle in the destination image with the nearest neighbor interpolation method.

WarpGetBufferSize Computes the size of the work buffer for the warp transform.

GetPerspectiveBound Computes the bounding rectangle for the source ROI transformed by theippiWarpPerspective function.

GetPerspectiveQuad Computes vertex coordinates of the quadrangle, to which the source ROI rectangle is mapped by the perspective transform.

GetPerspectiveTransform Computes the perspective transform coefficients to map the source ROI to the quadrangle with the specified vertex coordinates.

WarpPerspectiveLinearInit
Initializes the specification structure for image perspective warping with the linear interpolation method.

Syntax

IppStatusippiWarpPerspectiveLinearInit(IppiSize srcSize, IppiRect srcRoi, IppiSize dstSize, IppDataType dataType, const double coeffs[3][3], IppiWarpDirection direction, int numChannels, IppiBorderType borderType, const Ipp64f*pBorderValue, int smoothEdge, IppiWarpSpec*pSpec);

Include Files

ippi.h

Domain Dependencies

Headers: ippcore.h, ippvm.h, ipps.h

Libraries: ippcore.lib, ippvm.lib, ipps.lib

Parameters

srcSize
Size of the source image, in pixels.

srcRoi
Source image ROI (of the IppiRect type).

dstSize
Size of the destination image, in pixels.

dataType
Data type of the source and destination images. Supported values: ipp8u, ipp16u, ipp16s, and ipp32f.

coeffs
Coefficients for the perspective transform.

direction
Transformation direction. Supported values:
ippWarpForward          Forward transformation
ippWarpBackward         Backward transformation

numChannels            Number of channels in the image. Supported values: 1, 3, or 4.

borderType             Type of border. Supported values:
                        ippBorderConst         Values of all border pixels are set to a constant.
                        ippBorderRepl          Border is replicated from the edge pixels.
                        ippBorderTransp        Outer pixels are not processed.
                        ippBorderInMem         Border is obtained from the source image pixels in memory.
                        
                        Mixed borders are also supported. They can be obtained by the bitwise operation OR between ippBorderTransp and the ippBorderInMemTop, ippBorderInMemBottom, ippBorderInMemLeft, ippBorderInMemRight values.

pBorderValue            Pointer to the constant value to assign to pixels of the constant border. This parameter is applicable only to the ippBorderConst border type.

smoothEdge             Flag for edge smoothing. Supported values:
                        • 0 - transformation without edge smoothing.
                        • 1 - transformation with edge smoothing. This feature is supported only for the ippBorderTransp and ippBorderInMem border types.

pSpec                   Pointer to the specification structure.

Description
This function initializes the IppiWarpSpec structure for theippiWarpPerspectiveLinear function that performs warp perspective transformation with the linear interpolation method. To compute the size of the specification structure, use the WarpPerspectiveGetSize function.

You can set the value of the srcRoi parameter to ippRectInfinite, which means that the ROI is not specified.

Return Values
ippStsNoErr             Indicates no error.
ippStsNullPtrErr        Indicates an error when:
                        • pSpec is NULL
                        • pBorderValue is NULL when the border type is set to ippBorderConst
ippStsSizeErr           Indicates an error when width or height of the source or destination image is less than, or equal to one.
ippStsRectErr           Indicates an error in the following cases, if the source image ROI is not ippRectInfinite:
                        • If the source image ROI has no intersection with the image.
                        • Either x or y component of the source image ROI is negative.
• Width or height of the source image ROI is less than, or equal to zero.

ippStsDataTypeErr
Indicates an error when dataType has an illegal value.

ippStsWarpDirectionErr
Indicates an error when direction has an illegal value.

ippStsCoeffErr
Indicates an error when perspective transformation is singular.

ippStsWrongIntersectQuad
Indicates a warning that no operation is performed if the transformed source image extended with borders has no intersection with the destination image.

ippStsNotSupportedModeErr
Indicates an error when the requested mode is not supported. The edge smoothing feature is not supported for the ippBorderRepl and ippBorderConst border types.

ippStsBorderErr
Indicates an error when borderType has an illegal value.

ippStsNumChannelsErr
Indicates an error when numChannels has an illegal value.

ippStsSizeWrn
Indicates a warning when srcRoi exceeds the source image.

See Also
WarpPerspectiveGetSize Computes the size of the specification structure and the size of the external work buffer for the warp perspective transform.

WarpPerspectiveLinear Performs warp perspective transformation of an image using the linear interpolation method.

WarpPerspectiveLinear
Performs warp perspective transformation of an image using the linear interpolation method.

Syntax
IppStatus ippiWarpPerspectiveLinear_<mod>(const Ipp<datatype>* pSrc, int srcStep, Ipp<datatype> pDst, int dstStep, IppiPoint dstRoiOffset, IppiSize dstRoiSize, const IppiWarpSpec* pSpec, Ipp8u* pBuffer);

Supported values for mod:

8u_C1R  16u_C1R  16s_C1R  32f_C1R
8u_C3R  16u_C3R  16s_C3R  32f_C3R
8u_C4R  16u_C4R  16s_C4R  32f_C4R

Include Files
ippi.h

Domain Dependencies
Headers: ippcore.h, ippvm.h, ipps.h
Libraries: ippcore.lib, ippvm.lib, ipps.lib
Parameters

- **pSrc**: Pointer to the source image.
- **srcStep**: Distance, in bytes, between the starting points of consecutive lines in the source image buffer.
- **pDst**: Pointer to the destination image ROI.
- **dstStep**: Distance, in bytes, between the starting points of consecutive lines in the destination image buffer.
- **dstRoiOffset**: Offset of the destination image ROI with respect to the destination image origin.
- **dstRoiSize**: Size of the destination image ROI, in pixels.
- **pSpec**: Pointer to the specification structure for the warp operation.
- **pBuffer**: Pointer to the work buffer.

Description

This function transforms the source image pixel coordinates \((x, y)\) according to the following formulas:

\[
\begin{align*}
x' &= (c_{00}x + c_{01}y + c_{02}) / (c_{20}x + c_{21}y + c_{22}) \\
y' &= (c_{10}x + c_{11}y + c_{12}) / (c_{20}x + c_{21}y + c_{22})
\end{align*}
\]

where

- \(x'\) and \(y'\) are the pixel coordinates in the transformed image
- \(c_{ij}\) are the affine transform coefficients passed to the `coeffs` array during initialization

The `ippiWarpPerspectiveLinear` function operates with ROI (see Regions of Interest in Intel IPP). The transformed part of the source image is resampled with the linear interpolation method and stored in the destination image ROI. You need to define the destination image ROI origin by the following parameters: the offset of the destination ROI with respect to the destination image origin and the destination image ROI size. The parameter `pDst` must point to the processed destination image ROI.

If you initialize the warp specification structure using the `ippiWarpPerspectiveLinearInit` function, you can specify the source image ROI in the following ways:

- Set the `srcRoi` value to `ippRectInfinite`, which means that the ROI is not specified. In this case, `pSrc` must point to the processed source image. Pixels that are outside the source image boundaries are computed according to the specified border type.
- Set the `srcRoi` value to the part of the processed source image. In this case, `pSrc` must point to the processed source image ROI. The operations take place only inside the specified region of interest `srcRoi`. It means that the destination image pixels mapped to the outer pixels of the specified source image region are not changed.

If you initialize the warp specification structure using the `ippiWarpQuadLinearInit` function, set the `pSrc` value to the processed source image. The operations take place only inside the source quadrangle `srcQuad` that is specified in the `ippiWarpQuadLinearInit` function.

To specify the algorithm for borders processing, set the `borderType` and `pBorderValue` parameters when initializing the `IppiWarpSpec` structure. The data type of `pBorderValue` is automatically converted from `Ipp64f` to the data type of the processed images. The function supports the following algorithms for borders processing:

- If the border type is equal to `ippBorderRepl`, the source image outer pixels are replicated from the edge pixels.
• If the border type is equal to ippBorderConst, the outer pixels are set to the constant value specified in pBorderValue.
• If the border type is equal to ippBorderTransp or ippBorderInMem, destination image pixels mapped to the outer source image pixels are not changed.

Before using theippiWarpPerspectiveLinear function, you need to initialize the IppiWarpSpec structure using theippiWarpPerspectiveLinearInit function and compute the size of the external buffer pBuffer using theippiWarpGetBufferSize function.

To compute the perspective transform parameters, use theippiGetPerspectiveQuad, ippiGetPerspectiveBound, andippiGetPerspectiveTransform functions.

For an example on how to use this functionality, refer to the example provided with theippiWarpPerspectiveNearest function description.

**Return Values**

- ippStsNoErr Indicates no error.
- ippStsNullPtrErr Indicates an error when one of the specified pointers is NULL.
- ippStsNoOperation Indicates a warning when width or height of the destination image is equal to zero.
- ippStsContextMatchErr Indicates an error when context data is invalid.
- ippStsNotSupportedModeErr Indicates an error when the requested mode is not supported.
- ippStsSizeErr Indicates an error when width or height of the source or destination image ROI is negative.
- ippStsStepErr Indicates an error when the step value is not a multiple of data type.
- ippStsOutOfRangeErr Indicates an error when the destination image offset point is outside the destination image origin.
- ippStsSizeWrn Indicates a warning when the destination image ROI size is more than the destination image origin size.
- ippStsWrongIntersectQuad Indicates a warning that no operation is performed if the transformed source image extended with borders has no intersection with the destination image.

**See Also**

ROI Processing in Geometric Transforms

- WarpPerspectiveLinearInit Initializes the specification structure for image perspective warping with the linear interpolation method.
- WarpQuadLinearInit Initializes the specification structure for warping an arbitrary quadrangle in the source image to the quadrangle in the destination image with the linear interpolation method.
- WarpGetBufferSize Computes the size of the work buffer for the warp transform.
- GetPerspectiveBound Computes the bounding rectangle for the source ROI transformed by theippiWarpPerspective function.
- GetPerspectiveQuad Computes vertex coordinates of the quadrangle, to which the source ROI rectangle is mapped by the perspective transform.
- GetPerspectiveTransform Computes the perspective transform coefficients to map the source ROI to the quadrangle with the specified vertex coordinates.
- WarpPerspectiveNearest Performs warp perspective transformation of an image using the nearest neighbor interpolation method.
**WarpPerspectiveCubicInit**

Initializes the specification structure for image perspective warping with the cubic interpolation method.

**Syntax**

```c
IppStatus ippiWarpPerspectiveCubicInit(IppiSize srcSize, IppiRect srcRoi, IppiSize dstSize, IppDataType dataType, const double coeffs[3][3], IppiWarpDirection direction, int numChannels, Ipp64f valueB, Ipp64f valueC, IppiBorderType borderType, const Ipp64f* pBorderValue, int smoothEdge, IppiWarpSpec* pSpec, Ipp8u* pInitBuf);
```

**Include Files**

ippi.h

**Domain Dependencies**

Headers: ippcore.h, ippvm.h, ipps.h

Libraries: ippcore.lib, ippvm.lib, ipps.lib

**Parameters**

- **srcSize**
  Size of the source image, in pixels.

- **srcRoi**
  Source image ROI (of the IppiRect type).

- **dstSize**
  Size of the destination image, in pixels.

- **dataType**
  Data type of the source and destination images. Supported values: ipp8u, ipp16u, ipp16s, and ipp32f.

- **coeffs**
  Coefficients for the perspective transform.

- **direction**
  Transformation direction. Supported values:
  - ippWarpForward  Forward transformation
  - ippWarpBackward  Backward transformation

- **numChannels**
  Number of channels in the image. Supported values: 1, 3, or 4.

- **valueB, valueC**
  The first (B) and second (C) parameter for the cubic filter.

- **borderType**
  Type of border. Supported values:
  - ippBorderConst  Values of all border pixels are set to a constant.
  - ippBorderRepl  Border is replicated from the edge pixels.
  - ippBorderTransp  Outer pixels are not processed.
  - ippBorderInMem  Border is obtained from the source image pixels in memory.

  Mixed borders are also supported. They can be obtained by the bitwise operation OR between ippBorderTransp and the ippBorderInMemTop, ippBorderInMemBottom, ippBorderInMemLeft, ippBorderInMemRight values.
pBorderValue

Pointer to the constant value to assign to pixels of the constant border. This parameter is applicable only to the ippBorderConst border type.

smoothEdge

Flag to enable/disable edge smoothing. Possible values: 0 - transform without edge smoothing, 1 - transform with edge smoothing. This feature is supported only for the ippBorderTransp and ippBorderInMem border types.

pSpec

Pointer to the specification structure.

pInitBuf

Pointer to the temporary buffer for the cubic filter initialization.

Description

This function initializes the IppiWarpSpec structure for the ippiWarpPerspectiveCubic function that performs warp perspective transformation with the cubic interpolation method. Before using this function, compute the size of the specification structure and the size of the external buffer pInitBuf using the WarpPerspectiveGetSize function.

You can set the value of the srcRoi parameter to ippRectInfinite, which means that the ROI is not specified.

Return Values

<table>
<thead>
<tr>
<th>Error Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ippStsNoErr</td>
<td>Indicates no error.</td>
</tr>
<tr>
<td>ippStsNullPtrErr</td>
<td>Indicates an error when:</td>
</tr>
<tr>
<td></td>
<td>• pSpec is NULL</td>
</tr>
<tr>
<td></td>
<td>• pBorderValue is NULL when the border type is set to ippBorderConst</td>
</tr>
<tr>
<td>ippStsSizeErr</td>
<td>Indicates an error when width or height of the source or destination image is less than, or equal to one.</td>
</tr>
<tr>
<td>ippStsRectErr</td>
<td>Indicates an error in the following cases, if the source image ROI is not ippRectInfinite:</td>
</tr>
<tr>
<td></td>
<td>• If the source image ROI has no intersection with the image.</td>
</tr>
<tr>
<td></td>
<td>• Either x or y component of the source image ROI is negative.</td>
</tr>
<tr>
<td></td>
<td>• Width or height of the source image ROI is less than, or equal to zero.</td>
</tr>
<tr>
<td>ippStsDataTypeErr</td>
<td>Indicates an error when dataType has an illegal value.</td>
</tr>
<tr>
<td>ippStsWarpDirectionErr</td>
<td>Indicates an error when direction has an illegal value.</td>
</tr>
<tr>
<td>ippStsCoeffErr</td>
<td>Indicates an error when perspective transformation is singular.</td>
</tr>
<tr>
<td>ippStsWrongIntersectQuad</td>
<td>Indicates a warning that no operation is performed if the transformed source image extended with borders has no intersection with the destination image.</td>
</tr>
<tr>
<td>ippStsNotSupportedModeErr</td>
<td>Indicates an error when the requested mode is not supported.</td>
</tr>
<tr>
<td></td>
<td>The edge smoothing feature is not supported for the ippBorderRepl and ippBorderConst border types.</td>
</tr>
<tr>
<td>ippStsBorderErr</td>
<td>Indicates an error when borderType has an illegal value.</td>
</tr>
<tr>
<td>ippStsNumChannelsErr</td>
<td>Indicates an error when numChannels has an illegal value.</td>
</tr>
<tr>
<td>ippStsSizeWrn</td>
<td>Indicates a warning when srcRoi exceeds the source image.</td>
</tr>
</tbody>
</table>
**See Also**

**WarpPerspectiveGetSize** Computes the size of the specification structure and the size of the external work buffer for the warp perspective transform.

**WarpPerspectiveCubic** Performs warp perspective transformation of an image using the cubic interpolation method.

**WarpPerspectiveCubic**

*Performs warp perspective transformation of an image using the cubic interpolation method.*

**Syntax**

```c
IppStatus ippiWarpPerspectiveCubic_<mod>(const Ipp<datatype>* pSrc, int srcStep, Ipp<datatype> pDst, int dstStep, IppiPoint dstRoiOffset, IppiSize dstRoiSize, const IppiWarpSpec* pSpec, Ipp8u* pBuffer);
```

Supported values for *mod*:

- 8u_C1R
- 16u_C1R
- 16s_C1R
- 32f_C1R
- 8u_C3R
- 16u_C3R
- 16s_C3R
- 32f_C3R
- 8u_C4R
- 16u_C4R
- 16s_C4R
- 32f_C4R

**Include Files**

ippi.h

**Domain Dependencies**

Headers: ippcore.h, ippvm.h, ipps.h

Libraries: ippcore.lib, ippvm.lib, ipps.lib

**Parameters**

- **pSrc**
  - Pointer to the source image.
- **srcStep**
  - Distance, in bytes, between the starting points of consecutive lines in the source image buffer.
- **pDst**
  - Pointer to the destination image ROI.
- **dstStep**
  - Distance, in bytes, between the starting points of consecutive lines in the destination image buffer.
- **dstRoiOffset**
  - Offset of the destination image ROI with respect to the destination image origin.
- **dstRoiSize**
  - Size of the destination image ROI, in pixels.
- **pSpec**
  - Pointer to the specification structure for the warp operation.
- **pBuffer**
  - Pointer to the work buffer.

**Description**

This function transforms the source image pixel coordinates \((x, y)\) according to the following formulas:

\[
x' = \frac{(c_{00} \cdot x + c_{01} \cdot y + c_{02})}{(c_{20} \cdot x + c_{21} \cdot y + c_{22})}
\]

\[
y' = \frac{(c_{10} \cdot x + c_{11} \cdot y + c_{12})}{(c_{20} \cdot x + c_{21} \cdot y + c_{22})}
\]
where

- \(x'\) and \(y'\) are the pixel coordinates in the transformed image
- \(c_{ij}\) are the affine transform coefficients passed to the \texttt{coeffs} array during initialization

The \texttt{ippiWarpPerspectiveCubic} function operates with ROI (see \textit{Regions of Interest in Intel IPP}). The transformed part of the source image is resampled with the cubic interpolation method and stored in the destination image ROI. You need to define the destination image ROI origin by the following parameters: the offset of the destination ROI with respect to the destination image origin and the destination image ROI size. The parameter \(pDst\) must point to the processed destination image ROI.

If you initialize the warp specification structure using the \texttt{ippiWarpPerspectiveCubicInit} function, you can specify the source image ROI in the following ways:

- Set the \texttt{srcRoi} value to \texttt{ippRectInfinite}, which means that the ROI is not specified. In this case, \(pSrc\) must point to the processed source image. Pixels that are outside the source image boundaries are computed according to the specified border type.
- Set the \texttt{srcRoi} value to the part of the processed source image. In this case, \(pSrc\) must point to the processed source image ROI. The operations take place only inside the specified region of interest \(srcRoi\). It means that the destination image pixels mapped to the outer pixels of the specified source image region are not changed.

If you initialize the warp specification structure using the \texttt{ippiWarpQuadCubicInit} function, set the \(pSrc\) value to the processed source image. The operations take place only inside the source quadrangle \(srcQuad\) that is specified in the \texttt{ippiWarpQuadCubicInit} function.

To specify the algorithm for borders processing, set the \texttt{borderType} and \(pBorderValue\) parameters when initializing the \texttt{IppiWarpSpec} structure. The data type of \(pBorderValue\) is automatically converted from \texttt{Ipp64f} to the data type of the processed images. The function supports the following algorithms for borders processing:

- If the border type is equal to \texttt{ippBorderRepl}, the source image outer pixels are replicated from the edge pixels.
- If the border type is equal to \texttt{ippBorderConst}, the outer pixels are set to the constant value specified in \(pBorderValue\).
- If the border type is equal to \texttt{ippBorderTransp}, destination image pixels mapped to the outer source image pixels are not changed. The outer pixels required for cubic interpolation are replicated from the edge pixels.
- If the border type is equal to \texttt{ippBorderInMem}, destination image pixels mapped to the outer source image pixels are not changed. The outer pixels required for cubic interpolation are obtained from the out of the source image origin space. Cubic interpolation requires additional one-pixel edge from each source image side.
- The mixed border types can be obtained by the bitwise operation \texttt{OR} between \texttt{ippBorderTransp} and the \texttt{ippBorderInMemTop}, \texttt{ippBorderInMemBottom}, \texttt{ippBorderInMemLeft}, \texttt{ippBorderInMemRight} values.

Before using the \texttt{ippiWarpPerspectiveCubic} function, you need to initialize the \texttt{IppiWarpSpec} structure using the \texttt{ippiWarpPerspectiveCubicInit} function and compute the size of the external buffer \(pBuffer\) using the \texttt{ippiWarpGetBufferSize} function.

To compute the perspective transform parameters, use the \texttt{ippiGetPerspectiveQuad}, \texttt{ippiGetPerspectiveBound}, and \texttt{ippiGetPerspectiveTransform} functions.

For an example on how to use this functionality, refer to the example provided with the \texttt{ippiWarpPerspectiveNearest} function description.

**Return Values**

\texttt{ippStsNoErr} Indicates no error.
Indicates an error when one of the specified pointers is NULL.

Indicates a warning when width or height of the destination image is equal to zero.

Indicates an error when context data is invalid.

Indicates an error when the requested mode is not supported.

Indicates an error when width or height of the source or destination image ROI is negative.

Indicates an error when the step value is not a multiple of data type.

Indicates an error when the destination image offset point is outside the destination image origin.

Indicates a warning when the destination image ROI size is more than the destination image origin size.

Indicates a warning that no operation is performed if the transformed source image extended with borders has no intersection with the destination image.

See Also
ROI Processing in Geometric Transforms
WarpPerspectiveCubicInit Initializes the specification structure for image perspective warping with the cubic interpolation method.
WarpQuadCubicInit Initializes the specification structure for warping an arbitrary quadrangle in the source image to the quadrangle in the destination image with the cubic interpolation method.
WarpGetBufferSize Computes the size of the work buffer for the warp transform.
GetPerspectiveBound Computes the bounding rectangle for the source ROI transformed by the ippiWarpPerspective function.
GetPerspectiveQuad Computes vertex coordinates of the quadrangle, to which the source ROI rectangle is mapped by the perspective transform.
GetPerspectiveTransform Computes the perspective transform coefficients to map the source ROI to the quadrangle with the specified vertex coordinates.
WarpPerspectiveNearest Performs warp perspective transformation of an image using the nearest neighbor interpolation method.

GetBilinearQuad
Computes the vertex coordinates of the quadrangle, to which the source rectangular ROI is mapped by the bilinear transform.

Syntax
IppStatus ippiGetBilinearQuad(IppiRect srcRoi, double quad[4][2], const double coeffs[2][4]);

Include Files
ippi.h

Domain Dependencies
Headers: ippcore.h, ippvm.h, ipps.h
Libraries: ippcore.lib, ippvm.lib, ipps.lib
Parameters

srcRoi  
Region of interest in the source image (of the IppiRect type).

quad  
Output array. Contains vertex coordinates of the quadrangle, to which the source ROI is mapped by the bilinear transform function. The given bilinear transform coefficients.

coeffs

Description

This function operates with ROI (see ROI Processing in Geometric Transforms).

This function is used as a support function for ippiWarpBilinear. It computes vertex coordinates of the quadrangle, to which the source rectangular ROI is mapped by the bilinear transform function ippiWarpBilinear using coefficients coeffs.

The first dimension [4] of the array quad[4][2] is equal to the number of vertices, and the second dimension [2] means x and y coordinates of the vertex. Quadrangle vertices have the following meaning:

quad[0] corresponds to the transformed top-left corner of the source ROI,
quad[1] corresponds to the transformed top-right corner of the source ROI,
quad[2] corresponds to the transformed bottom-right corner of the source ROI,
quad[3] corresponds to the transformed bottom-left corner of the source ROI.

Return Values

ippStsNoErr  
Indicates no error. Any other value indicates an error.

ippStsSizeErr  
Indicates an error condition if srcRoi has a size field with zero or negative value.

ippStsCoeffErr  
Indicates an error condition if coefficient values are invalid.

GetBilinearBound

Computes the bounding rectangle for the source ROI transformed by the ippiWarpBilinear function.

Syntax

IppStatus ippiGetBilinearBound(IppiRect srcRoi, double bound[2][2], const double coeffs[2][4]);

Include Files

ippi.h

Domain Dependencies

Headers: ippcore.h, ippvm.h, ipps.h
Libraries: ippcore.lib, ippvm.lib, ipps.lib

Parameters

srcRoi  
Region of interest in the source image (of the IppiRect type).

bound  
Output array. Contains vertex coordinates of the bounding rectangle for the transformed source ROI.
**coeffs**

The given bilinear transform coefficients.

**Description**

This function operates with ROI (see ROI Processing in Geometric Transforms).

This function is used as a support function for `ippiWarpBilinear`. It computes vertex coordinates of the smallest bounding rectangle for the quadrangle `quad`, to which the source ROI is mapped by the bilinear transform function `ippiWarpBilinear` using coefficients `coeffs`.

`bound[0]` specifies x, y coordinates of the top-left corner, `bound[1]` specifies x, y coordinates of the bottom-right corner.

**Return Values**

- `ippStsNoErr`: Indicates no error. Any other value indicates an error or a warning.
- `ippStsSizeErr`: Indicates an error condition if `srcRoi` has a size field with zero or negative value.
- `ippStsCoeffErr`: Indicates an error condition if coefficient values are invalid.

**GetBilinearTransform**

*Computes bilinear transform coefficients to map the source ROI to the quadrangle with the specified vertex coordinates.*

**Syntax**

```c
IppStatus ippiGetBilinearTransform(IppiRect srcRoi, const double quad[4][2], double coeffs[2][4]);
```

**Include Files**

`ippi.h`

**Domain Dependencies**

**Headers**: `ippcore.h`, `ippvm.h`, `ipps.h`

**Libraries**: `ippcore.lib`, `ippvm.lib`, `ipps.lib`

**Parameters**

- `srcRoi`: Region of interest in the source image (of the `IpplRect` type).
- `quad`: Vertex coordinates of the quadrangle, to which the source ROI is mapped by the bilinear transform function.
- `coeffs`: Output array. Contains the target bilinear transform coefficients.

**Description**

This function operates with ROI (see ROI Processing in Geometric Transforms).

This function is used as a support function for `ippiWarpBilinear`. It computes the coefficients `coeffs` of the bilinear transform that maps the source rectangular ROI to the quadrangle with the specified vertex coordinates `quad`.

The first dimension [4] of the array `quad[4][2]` is equal to the number of vertices, and the second dimension [2] means x and y coordinates of the vertex. Quadrangle vertices have the following meaning:
quad[0] corresponds to the transformed top-left corner of the source ROI,
quad[1] corresponds to the transformed top-right corner of the source ROI,
quad[2] corresponds to the transformed bottom-right corner of the source ROI,
quad[3] corresponds to the transformed bottom-left corner of the source ROI.

Return Values

ippStsNoErr Indicates no error. Any other value indicates an error.
ippStsSizeErr Indicates an error condition if srcRoi has a size field with zero
or negative value.
ippStsCoeffErr Indicates an error condition if coefficient values are invalid.
ippStsRectErr Indicates an error condition if width or height of the srcRoi is
less than or equal to 1.

Example

To better understand usage of the ippiGetBilinearTransform function, refer to the following example in
the examples archive available for download from https://software.intel.com/en-us/ipp-manual-examples :
GetBilinearTransform.c

WarpBilinearGetBufferSize

Computes the size of the work buffer for bilinear warping.

Syntax

IppStatus ippiWarpBilinearGetBufferSize(IppiSize srcSize, IppiRect srcRoi, IppiRect
dstRoi, IppiWarpDirection direction, const double coeffs[2][4], int interpolation, int*
pBufSize);

Include Files

ippi.h

Domain Dependencies

Headers: ippcore.h, ippvm.h, ipps.h
Libraries: ippcore.lib, ippvm.lib, ipps.lib

Parameters

srcSize Size of the source image, in pixels.
srcRoi Source image ROI (of the IppiRect type).
dstRoi Destination image ROI (of the IppiRect type).
direction Transformation direction. Supported values:
ippWarpForward Forward transformation
ippWarpBackward Backward transformation
coeffs Coefficients for the perspective transform.
Interpolation mode. Supported values:
- **IPPI_INTER_NN**: Nearest neighbor interpolation
- **IPPI_INTER_LINEAR**: Linear interpolation
- **IPPI_INTER_CUBIC**: Cubic interpolation
- **IPPI_INTER_EDGE**: Use edge smoothing in addition to one of the above modes

**pBufSize**
Pointer to the size, in bytes, of the external buffer.

**Description**
This function computes the size of the external work buffer required for bilinear warping of the source image ROI. The result is stored in the `pBufSize` parameter.

**Return Values**
- **ippStsNoErr**: Indicates no error.
- **ippStsNullPtrErr**: Indicates an error when `pBufSize` is NULL.
- **ippStsSizeErr**: Indicates an error when one of the image dimensions is less than, or equal to zero.
- **ippStsWarpDirectionErr**: Indicates an error when `direction` has an illegal value.
- **ippStsCoeffErr**: Indicates an error when bilinear transformation is singular.
- **ippStsInterpolationErr**: Indicates an error when `interpolation` has an illegal value.
- **ippStsWrongIntersectQuad**: Indicates a warning that no operation is performed if the transformed source image extended with borders has no intersection with the destination image.

**See Also**
- **WarpBilinear** MODIFIED API. Performs bilinear warping of the source image using the specified transform coefficients.

**WarpBilinear**
MODIFIED API. Performs bilinear warping of the source image using the specified transform coefficients.

**Syntax**
```c
IppStatus ippiWarpBilinear_<mod>(const Ipp<datatype>* pSrc, IppiSize srcSize, int srcStep, IppiRect srcRoi, Ipp<datatype>* pDst, int dstStep, IppiRect dstRoi, const double coeffs[2][4], int interpolation, Ipp8u* pBuffer);
```

Supported values for `mod`:
- 8u_C1R
- 16u_C1R
- 32f_C1R
- 8u_C3R
- 16u_C3R
- 32f_C3R
- 8u_C4R
- 16u_C4R
- 32f_C4R
Include Files
ippi.h

Domain Dependencies
Headers:ippicore.h, ippvm.h, ipps.h
Libraries:ippicore.lib, ippvm.lib, ipps.lib

Parameters
pSrc Pointer to the source image origin. An array of separate pointers to each plane in case of data in planar format.
srcSize Size in pixels of the source image.
srcStep Distance in bytes between starts of consecutive lines in the source image buffer.
srcRoi Region of interest in the source image (of the IppiRect type).
pDst Pointer to the destination image origin. An array of separate pointers to each plane in case of data in planar format.
dstStep Distance in bytes between starts of consecutive lines in the destination image buffer.
dstRoi Region of interest in the destination image (of the IppiRect type).
coeffs The bilinear transform coefficients.
interpolation Specifies the interpolation mode. Use one of the following values:
  IPPI_INTER_NN Nearest neighbor interpolation
  IPPI_INTER_LINEAR Linear interpolation
  IPPI_INTER_CUBIC Cubic interpolation
  IPPI_SMOOTH_EDGE Use edge smoothing in addition to one of the above modes.
pBuffer Pointer to the external work buffer.

Description

**Important** The API of this function has been modified in Intel IPP 9.0 release.

This function operates with ROI (see ROI Processing in Geometric Transforms).

This bilinear warp function transforms the source image pixel coordinates \((x, y)\) according to the following formulas:

\[
\begin{align*}
  x' &= c_{00} \cdot x \cdot y + c_{01} \cdot x + c_{02} \cdot y + c_{03} \\
  y' &= c_{10} \cdot x \cdot y + c_{11} \cdot x + c_{12} \cdot y + c_{13}
\end{align*}
\]

where \(x'\) and \(y'\) denote the pixel coordinates in the transformed image, and \(c_{ij}\) are the bilinear transform coefficients passed in the array **coeffs**.

The bilinear transform preserves equal distances between points on a line.
The transformed part of the source image is resampled using the interpolation mode specified by the `interpolation` parameter, and written to the destination image ROI.

Figure "Bilinear Transform of an Image" gives an example of applying the bilinear warping function `ippiWarpBilinear` to a sample image.

**Bilinear Transform of an Image**

To estimate how the source image ROI will be transformed by the `ippiWarpBilinear` function, use functions `ippiWarpBilinearQuad` and `ippiGetBilinearBound`. To calculate coefficients of the bilinear transform which maps source ROI to a given quadrangle, use `ippiGetBilinearTransform` function.

Before using this function, compute the size of the external work buffer `pBuffer` using the `WarpBilinearGetBufferSize` function.

**Example** shows how to use the `ippiWarpBilinear_32f_C1R` function.

**Return Values**

- `ippStsNoErr` Indicates no error.
- `ippStsNullPtrErr` Indicates an error condition if one of the specified pointers is NULL.
- `ippStsSizeErr` Indicates an error condition if any image dimension has zero or negative value.
ippStsStepErr
Indicates an error condition if srcStep or dstStep has a zero or negative value.

ippStsInterpolationErr
Indicates an error condition if interpolation has an illegal value.

ippStsRectErr
Indicates an error condition if width or height of the intersection of the srcRoi and source image is less than or equal to 1.

ippStsCoeffErr
Indicates an error condition if coefficient values are invalid.

ippStsWrongIntersectROIErr
Indicates an error condition if srcRoi has no intersection with the source image.

ippStsWrongIntersectQuad
Indicates a warning that no operation is performed if the transformed source ROI has no intersection with the destination ROI.

See Also
Regions of Interest in Intel IPP
WarpBilinearQuad MODIFIED API. Performs bilinear warping of the source image that transforms the given source quadrangle to the specified destination quadrangle.
GetBilinearBound Computes the bounding rectangle for the source ROI transformed by theippiWarpBilinear function.
GetBilinearTransform Computes bilinear transform coefficients to map the source ROI to the quadrangle with the specified vertex coordinates.
WarpBilinearGetBufferSize Computes the size of the work buffer for bilinear warping.

WarpBilinearBack
MODIFIED API. Performs an inverse bilinear warping of the source image.

Syntax
IppStatus ippiWarpBilinearBack_<mod>(const Ipp<datatype>* pSrc, IppSize srcSize, int srcStep, IppiRect srcRoi, Ipp<datatype>* pDst, int dstStep, IppiRect dstRoi, const double coeffs[2][4], int interpolation, Ipp8u* pBuffer);

Supported values for mod:
8u_C1R 16u_C1R 32f_C1R
8u_C3R 16u_C3R 32f_C3R
8u_C4R 16u_C4R 32f_C4R

Include Files
ippi.h

Domain Dependencies
Headers: ippcore.h, ippvm.h, ipps.h
Libraries: ippcore.lib, ippvm.lib, ipps.lib
Parameters

pSrc
Pointer to the source image origin. An array of separate pointers to each plane in case of data in planar format.

srcSize
Size in pixels of the source image.

srcStep
Distance in bytes between starts of consecutive lines in the source image buffer.

srcRoi
Region of interest in the source image (of the IppiRect type).

pDst
Pointer to the destination image origin. An array of separate pointers to each plane in case of data in planar format.

dstStep
Distance in bytes between starts of consecutive lines in the destination image buffer.

dstRoi
Region of interest in the destination image (of the IppiRect type).

coeffs
The bilinear transform coefficients.

interpolation
Specifies the interpolation mode. Use one of the following values:

- IPPI_INTER_NN nearest neighbor interpolation
- IPPI_INTER_LINEAR linear interpolation
- IPPI_INTER_CUBIC cubic interpolation.

pBuffer
Pointer to the external work buffer.

Description

**Important** The API of this function has been modified in Intel IPP 9.0 release.

This function operates with ROI (see ROI Processing in Geometric Transforms).

This function performs the inverse transform to that defined by `ippiWarpBilinear` function. Pixel coordinates \( x' \) and \( y' \) in the transformed image are obtained from the following equations

\[
\begin{align*}
    &c_{00} \cdot x' \cdot y' + c_{01} \cdot x' + c_{02} \cdot y' + c_{03} = x \\
    &c_{10} \cdot x' \cdot y' + c_{11} \cdot x' + c_{12} \cdot y' + c_{13} = y
\end{align*}
\]

where \( x \) and \( y \) denote the pixel coordinates in the source image, and coefficients \( c_{ij} \) are given in the array `coeffs`. Thus, you do not need to invert transform coefficients in your application program before calling `ippiWarpBilinearBack`.

Note that inverse transform functions handle source and destination ROI in a different way than other geometric transform functions. The implementation of the inverse transform functions has the following logic:

- Backward transform is applied to coordinates of each pixel in the destination ROI. The result is coordinates of some pixel in the source image.
- If the obtained source pixel is inside the source ROI, the corresponding pixel in the destination ROI is modified accordingly; otherwise, no changes are made.

Before using this function, compute the size of the external work buffer `pBuffer` using the `WarpBilinearGetBufferSize` function.

**Example** shows how to use the function `ippiWarpBilinearBack_32f_C1R`. 
## Return Values

<table>
<thead>
<tr>
<th>Status Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ippStsNoErr</td>
<td>Indicates no error. Any other value indicates an error or a warning.</td>
</tr>
<tr>
<td>ippStsNullPtrErr</td>
<td>Indicates an error condition if one of the specified pointers is NULL.</td>
</tr>
<tr>
<td>ippStsSizeErr</td>
<td>Indicates an error condition if any image dimension has zero or negative value.</td>
</tr>
<tr>
<td>ippStsStepErr</td>
<td>Indicates an error condition if srcStep or dstStep has a zero or negative value.</td>
</tr>
<tr>
<td>ippStsInterpolationErr</td>
<td>Indicates an error condition if interpolation has an illegal value.</td>
</tr>
<tr>
<td>ippStsCoeffErr</td>
<td>Indicates an error condition if coefficient values are invalid.</td>
</tr>
<tr>
<td>ippStsWrongIntersectROIErr</td>
<td>Indicates an error condition if srcRoi has no intersection with the source image.</td>
</tr>
<tr>
<td>ippStsWrongIntersectQuad</td>
<td>Indicates a warning that no operation is performed if the transformed source ROI has no intersection with the destination ROI.</td>
</tr>
</tbody>
</table>

## See Also

- **Regions of Interest in Intel IPP**
- **WarpBilinear** - MODIFIED API. Performs bilinear warping of the source image using the specified transform coefficients.
- **WarpBilinearGetBufferSize** - Computes the size of the work buffer for bilinear warping.

### WarpBilinearQuadGetBufferSize

*Computes the size of the work buffer for bilinear warping of an arbitrary quadrangle in the source image ROI to the quadrangle in the destination image.*

#### Syntax

```
IppStatus ippiWarpBilinearQuadGetBufferSize(IppiSize srcSize, IppiRect srcRoi, const double srcQuad[4][2], IppiRect dstRoi, const double dstQuad[4][2], int interpolation, int* pBufSize);
```

#### Include Files

`ippi.h`

#### Domain Dependencies

**Headers:** ippicore.h, ippvm.h, ipps.h  
**Libraries:** ippcore.lib, ippvm.lib, ipps.lib

#### Parameters

- **srcSize** - Size of the source image, in pixels.  
- **srcRoi** - Source image ROI (of the IppiRect type).  
- **srcQuad** - Quadrangle in the source image.  
- **dstRoi** - Destination image ROI (of the IppiRect type).
**dstQuad**

Quadrangle in the destination image.

**interpolation**

Interpolation mode. Supported values:

- **IPPI_INTER_NN** - Nearest neighbor interpolation
- **IPPI_INTER_LINEAR** - Linear interpolation
- **IPPI_INTER_CUBIC** - Cubic interpolation
- **IPPI_INTER_EDGE** - Use edge smoothing in addition to one of the above modes

**pBufSize**

Pointer to the size, in bytes, of the external buffer.

**Description**

This function computes the size of the external work buffer required for bilinear warping of an arbitrary quadrangle in the source image ROI to the quadrangle in the destination image. The result is stored in the `pBufSize` parameter.

**Return Values**

- **ippStsNoErr** Indicates no error.
- **ippStsNullPtrErr** Indicates an error when `pBufSize` is NULL.
- **ippStsSizeErr** Indicates an error:
  - If one of the ROI coordinates has a negative value.
  - If one of the ROI dimensions is less than, or equal to zero.
- **ippStsCoeffErr** Indicates an error when bilinear transformation is singular.
- **ippStsInterpolationErr** Indicates an error when `interpolation` has an illegal value.

**See Also**

- **WarpBilinearQuad** MODIFIED API. Performs bilinear warping of the source image that transforms the given source quadrangle to the specified destination quadrangle.

**Syntax**

```
IppStatus ippiWarpBilinearQuad_<mod>(const Ipp<datatype>* pSrc, IppiSize srcSize, int srcStep, IppiRect srcRoi, const double srcQuad[4][2], Ipp<datatype>* pDst, int dstStep, IppiRect dstRoi, const double dstQuad[4][2], int interpolation, Ipp8u* pBuffer);
```

Supported values for `mod`:

- 8u_C1R
- 16u_C1R
- 32f_C1R
- 8u_C3R
- 16u_C3R
- 32f_C3R
- 8u_C4R
- 16u_C4R
- 32f_C4R
Include Files
ippi.h

Domain Dependencies
Headers: ippicore.h, ippvm.h, ipps.h
Libraries: ippicore.lib, ippvm.lib, ipps.lib

Parameters

- **pSrc**: Pointer to the source image origin. An array of separate pointers to each plane in case of data in planar format.
- **srcSize**: Size in pixels of the source image.
- **srcStep**: Distance in bytes between starts of consecutive lines in the source image buffer.
- **srcRoi**: Region of interest in the source image (of the IppiRect type).
- **srcQuad**: A given quadrangle in the source image.
- **pDst**: Pointer to the destination image origin. An array of separate pointers to each plane in case of data in planar format.
- **dstStep**: Distance in bytes between starts of consecutive lines in the destination image buffer.
- **dstRoi**: Region of interest in the destination image (of the IppiRect type).
- **dstQuad**: A given quadrangle in the destination image.
- **interpolation**: Specifies the interpolation mode. Use one of the following values:
  - IPPI_INTER_NN: Nearest neighbor interpolation
  - IPPI_INTER_LINEAR: Linear interpolation
  - IPPI_INTER_CUBIC: Cubic interpolation
  - IPPI_SMOOTH_EDGE: Use edge smoothing in addition to one of the above modes.
- **pBuffer**: Pointer to the external work buffer.

Description

**Important** The API of this function has been modified in Intel IPP 9.0 release.

This function operates with ROI (see ROI Processing in Geometric Transforms).
This function applies a bilinear transform to an arbitrary quadrangle srcQuad in the source image pSrc. The operations take place only in the intersection of the source image ROI srcRoi and the source quadrangle srcQuad. The functionippiWarpBilinearQuad uses the same formulas for pixel mapping as in the case of the ippiWarpBilinear function. Transform coefficients are computed internally, based on the mapping of the source quadrangle to the quadrangle dstQuad specified in the destination image pDst. The dstQuad should have a non-empty intersection with the destination image ROI dstRoi.


Edge smoothing interpolation is applicable only if the source quadrangle lies in the source image ROI.
Before using this function, compute the size of the external work buffer `pBuffer` using the `WarpBilinearGetBufferSize` function.

**Return Values**

- **ippStsNoErr**: Indicates no error. Any other value indicates an error or a warning.
- **ippStsNullPtrErr**: Indicates an error condition if one of the specified pointers is NULL.
- **ippStsSizeErr**: Indicates an error condition if any image dimension has zero or negative value.
- **ippStsStepErr**: Indicates an error condition if `srcStep` or `dstStep` has a zero or negative value.
- **ippStsInterpolationErr**: Indicates an error condition if `interpolation` has an illegal value.
- **ippStsQuadErr**: Indicates an error condition if `srcQuad` or `dstQuad` degenerates into triangle.
- **ippStsWrongIntersectROIErr**: Indicates an error condition if `srcRoi` has no intersection with the source image.
- **ippStsWrongIntersectQuad**: Indicates a warning that no operation is performed if the `srcRoi` has no intersection with the `srcQuad`, or `dstRoi` has no intersection with the `dstQuad`.

**See Also**

- Regions of Interest in Intel IPP
- WarpBilinearMODIFIED API. Performs bilinear warping of the source image using the specified transform coefficients.
- WarpBilinearGetBufferSize Computes the size of the work buffer for bilinear warping.

**Mirror**

*Mirrors an image about the specified axis (axes).*

**Syntax**

**Case 1: Not-in-place operation**

```c
IppStatus ippiMirror_<mod>(const Ipp<datatype>* pSrc, int srcStep, Ipp<datatype>* pDst, int dstStep, IppSize roiSize, IppiAxis flip);
```

Supported values for `mod`:

- 8u_C1R  16u_C1R  16s_C1R  32s_C1R  32f_C1R
- 8u_C3R  16u_C3R  16s_C3R  32s_C3R  32f_C3R
- 8u_C4R  16u_C4R  16s_C4R  32s_C4R  32f_C4R
- 8u_AC4R 16u_AC4R  16s_AC4R  32s_AC4R  32f_AC4R
```
Case 2: In-place operation

IppStatus ippiMirror_<mod>(Ipp<datatype>* pSrcDst, int srcDstStep, IppSize roiSize, IppiAxis flip);

Supported values for mod:

<table>
<thead>
<tr>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>8u_C1IR</td>
</tr>
<tr>
<td>16u_C1IR</td>
</tr>
<tr>
<td>16s_C1IR</td>
</tr>
<tr>
<td>32s_C1IR</td>
</tr>
<tr>
<td>32f_C1IR</td>
</tr>
<tr>
<td>8u_C3IR</td>
</tr>
<tr>
<td>16u_C3IR</td>
</tr>
<tr>
<td>16s_C3IR</td>
</tr>
<tr>
<td>32s_C3IR</td>
</tr>
<tr>
<td>32f_C3IR</td>
</tr>
<tr>
<td>8u_C4IR</td>
</tr>
<tr>
<td>16u_C4IR</td>
</tr>
<tr>
<td>16s_C4IR</td>
</tr>
<tr>
<td>32s_C4IR</td>
</tr>
<tr>
<td>32f_C4IR</td>
</tr>
<tr>
<td>8u_AC4IR</td>
</tr>
<tr>
<td>16u_AC4IR</td>
</tr>
<tr>
<td>16s_AC4IR</td>
</tr>
<tr>
<td>32s_AC4IR</td>
</tr>
<tr>
<td>32f_AC4IR</td>
</tr>
</tbody>
</table>

Include Files

ippi.h

Domain Dependencies

Headers: ippcore.h, ippvm.h, ipps.h

Libraries: ippcore.lib, ippvm.lib, ipps.lib

Parameters

- **pSrc**: Pointer to the source buffer.
- **srcStep**: Distance, in bytes, between the starting points of consecutive lines in the source image buffer. Pointer to the destination buffer.
- **pDst**: Distance, in bytes, between the starting points of consecutive lines in the destination image buffer. Pointer to the source and destination buffer for the in-place operation.
- **dstStep**: Distance, in bytes, between the starting points of consecutive lines in the source and destination image buffer for the in-place operation.
- **roiSize**: Size of the destination ROI in pixels. Specifies the axis to mirror the image about. Use the following values to specify the axes:
  - ippAxsHorizontal for the horizontal axis.
  - ippAxsVertical for the vertical axis.
  - ippAxsBoth for both horizontal and vertical axes.
  - ippAxs45 for the 45-degree rotated axis.
  - ippAxs135 for the 135-degree rotated axis.

Description

The *ippiMirror* function operates with ROI (see Regions of Interest in Intel IPP). This function mirrors the source image *pSrc* about the axis (axes) specified by the value of the *flip* parameter and writes the result to the destination image *pDst*. Each function flavor can mirror an image about the horizontal or vertical axis or both.
The `ippiMirror_8u_C1R`, `ippiMirror_16u_C1R`, `ippiMirror_16s_C1R`, and `ippiMirror_32f_C1R` function flavors can also use the `ippAxs45` or `ippAxs135` value of the `flip` parameter to mirror the source image about an axis rotated counterclockwise by 45 degrees or 135 degrees, respectively. For mirroring with each of these values of the `flip` parameter, the sizes of the source and destination ROI are different, and

\[
\text{roiSize.height} = \text{srcRoiSize.width} \\
\text{roiSize.width} = \text{srcRoiSize.height}
\]

### Return Values

- **ippStsNoErr**: Indicates no error. Any other value indicates an error or a warning.
- **ippStsNullPtrErr**: Indicates an error condition if one of the specified pointers is NULL.
- **ippStsSizeErr**: Indicates an error condition if `roiSize` has a field with zero or negative value, or when one of the dimensions is equal to 1.
- **ippStsMirrorFlipErr**: Indicates an error condition if `flip` has an illegal value.
- **ippStsNotSupportedModeErr**: Indicates an error condition if intersection of the source and destination ROI is detected.
- **ippStsStepErr**: Indicates an error condition if `srcStep` or `dstStep` has a zero or negative value or is not a multiple of the image data size (4 for floating-point images or 2 for short-integer images).

### Examples

To better understand usage of the `ippiMirror` function, refer to the following examples in the examples archive available for download from https://software.intel.com/en-us/ipp-manual-examples:

- Mirror1.c
- Mirror2.c

### Remap

*Performs the look-up coordinate mapping of pixels of the source image.*

#### Syntax

**Case 1: Operation on pixel-order data**

```c
IppStatus ippiRemap_<mod>(const Ipp<datatype>* pSrc, IppiSize srcSize, int srcStep,
IppiRect srcRoi, const Ipp32f* pxMap, int xMapStep, const Ipp32f* pyMap, int yMapStep,
Ipp<datatype>* pDst, int dstStep, IppiSize dstRoiSize, int interpolation);
```

Supported values for `mod`:

- `8u_C1R`
- `16u_C1R`
- `16s_C1R`
- `32f_C1R`
- `8u_C3R`
- `16u_C3R`
- `16s_C3R`
- `32f_C3R`
- `8u_C4R`
- `16u_C4R`
- `16s_C4R`
- `32f_C4R`
- `16u_AC4R`
- `16s_AC4R`
- `32f_AC4R`
IppStatus ippiRemap_<mod>(const Ipp<datatype>* pSrc, IppSize srcSize, int srcStep, IppRect srcRoi, const Ipp64f* pxMap, int xMapStep, const Ipp64f* pyMap, int yMapStep, Ipp<datatype>* pDst, int dstStep, IppSize dstRoiSize, int interpolation);

Supported values for <mod>:

- 64f_C1R
- 64f_C3R
- 64f_C4R
- 64f_AC4R

IppStatus ippiRemap_8u_AC4R(const Ipp<datatype>* pSrc, IppSize srcSize, int srcStep, IppRect srcRoi, const Ipp32f* pxMap, int xMapStep, const Ipp32f* pyMap, int yMapStep, Ipp<datatype>* pDst, int dstStep, IppSize dstRoiSize, int interpolation);

Include Files
ippi.h

Domain Dependencies
Headers: ippcore.h, ippvm.h, ipps.h
Libraries: ippcore.lib, ippvm.lib, ipps.lib

Parameters

- **pSrc**: Pointer to the source image origin. An array of separate pointers to each plane in case of data in planar format.
- **srcSize**: Size, in pixels, of the source image.
- **srcStep**: Distance, in bytes, between the starting points of consecutive lines in the source image buffer.
- **srcRoi**: Region of interest in the source image (of the IppRect type).
- **pxMap, pyMap**: Pointers to the starts of 2D buffers, containing tables of the x- and y-coordinates.
- **xMapStep, yMapStep**: Steps, in bytes, through the buffers containing tables of the x- and y-coordinates.
- **pDst**: Pointer to the destination image ROI. An array of separate pointers to ROI in each plane for planar image.
- **dstStep**: Distance, in bytes, between the starting points of consecutive lines in the destination image buffer.
- **dstRoiSize**: Size of the destination ROI in pixels.
- **interpolation**: Specifies the interpolation mode. Possible values are:
  - IPPI_INTER_NN - nearest neighbor interpolation
  - IPPI_INTER_LINEAR - linear interpolation
  - IPPI_INTER_CUBIC - cubic interpolation
IPPI_INTER_LANCZOS - interpolation with Lanczos window
IPPI_INTER_CUBIC2P_CATMULLROM - Catmull-Rom spline

the following flag is used additionally to the above modes:
IPPI_SMOOTH_EDGE - edge smoothing

Description
This function operates with ROI (see ROI Processing in Geometric Transforms).
This function transforms the source image by remapping its pixels. Pixel remapping is performed using
pxMap and pyMap buffers to look-up the coordinates of the source image pixel that is written to the target
destination image pixel. The application has to supply these look-up tables. The remapping of the source
pixels to the destination pixels is made according to the following formula:

\[ dst\_pixel[i, j] = src\_pixel[pxMap[i, j], pyMap[i, j]] \]

where \( i, j \) are the \( x \)- and \( y \)-coordinates of the target destination image pixel \( dst\_pixel \);

\( pxMap[i, j] \) contains the \( x \)- coordinates of the source image pixels \( src\_pixel \) that are written to
\( dst\_pixel \);

\( pyMap[i, j] \) contains the \( y \)- coordinates of the source image pixels \( src\_pixel \) that are written to
\( dst\_pixel \).

If the referenced coordinates correspond to a pixel outside of the source ROI, and the flag
IPPI_SMOOTH_EDGE is not set, then no mapping of the source pixel is performed.

Figure "Remapping the Sample Image" gives an example of applying the functionippiRemap to a sample
image that is a square grid of alternating blue, red, and green lines.

Remapping the Sample Image

![Remapping the Sample Image](image)

The transformed part of the image is resampled using the interpolation method specified by the
interpolation parameter, and is written to the destination image ROI. The function can be used with or
without edge smoothing. The pseudo code below shows how it works.

The function works without edge smoothing - the flag IPPI_SMOOTH_EDGE is not set:
if (xMap < srcRoi.x || xMap > srcRoi.x + srcRoi.width -1 || yMap < srcRoi.y || yMap > srcRoi.y + srcRoi.height -1)
    not fill dst /* do not remap */
else
    fill dst with Interpolate(Src, xMap, yMap) /* remap */

The function works with edge smoothing - the flag IPPI_SMOOTH_EDGE is set:
if (xMap < srcRoi.x - 1 || xMap > srcRoi.x+srcRoi.width || yMap < srcRoi.y - 1 || yMap > srcRoi.y+srcRoi.height)
    not fill dst /* do not remap */
else if (xMap < srcRoi.x || xMap > srcRoi.x+srcRoi.width-1 || yMap < srcRoi.y || yMap > srcRoi.y+srcRoi.height-1)
    fill dst with Interpolate(Src, fillvalue, xMap, yMap) /* smoothing */
else
    fill dst with Interpolate(Src, xMap, yMap) /* remap */

Return Values

ippiStsNoErr Indicates no error. Any other value indicates an error or a warning.
ippiStsNullPtrErr Indicates an error condition if one of the specified pointers is NULL.
ippiStsSizeErr Indicates an error condition if srcSize or dstRoiSize has a field with zero or negative value.
ippiStsStepErr Indicates an error condition if one of the srcStep, dstStep, xMapStep, or yMapStep has a zero or negative value.
ippiStsInterpolationErr Indicates an error condition if interpolation has an illegal value.
ippiStsWrongIntersectROIErr Indicates an error condition if srcRoi has no intersection with the source image.

Example

To better understand usage of the ippiRemap function, refer to the following example in the examples archive available for download from https://software.intel.com/en-us/ipp-manual-examples:

Remap.c
This section describes the Intel® IPP image processing functions that perform adaptive thresholding of the image ROI.

### ThresholdAdaptiveBoxGetBufferSize

*Comes the size of the work buffer for adaptive thresholding with the Box method.*

**Syntax**

```c
IppStatus ippiThresholdAdaptiveBoxGetBufferSize(IppiSize roiSize, IppiSize maskSize, IppDataType dataType, int numChannels, int* pBufferSize);
```

**Include Files**

ippi.h

**Domain Dependencies**

- **Headers:** ippcore.h, ippvm.h, ipps.h
- **Libraries:** ippcore.lib, ippvm.lib, ipps.lib

**Parameters**

- `roiSize`: Size of the destination image ROI, in pixels.
- `maskSize`: Size of the kernel that is used to calculate a threshold level. Width and height of `maskSize` must be equal and odd.
- `dataType`: Data type of the source and destination images. Possible value is `ipp8u`.
- `numChannels`: Number of channels in the images. Possible value is 1.
- `pBufferSize`: Pointer to the size, in bytes, of the external work buffer.

**Description**

This function computes the size, in bytes, of the external work buffer needed for the `ThresholdAdaptiveBox` function. The result is stored in `pBufferSize`. For an example on how to use this function, refer to the example provided with the `ThresholdAdaptiveBox` function description.

**Return Values**

- `ippStsNoErr`: Indicates no error. Any other value indicates an error.
- `ippStsNullPtrErr`: Indicates an error when `pBufferSize` is NULL.
- `ippStsSizeErr`: Indicates an error when `roiSize` has a field with a zero or negative value.
- `ippStsMaskSizeErr`: Indicates an error when `maskSize` has a field with a zero or negative value, or fields of `maskSize` are not equal.
### ippStsDataTypeErr
Indicates an error when `dataType` has an illegal value.

### ippStsNumChannelsErr
Indicates an error when `numChannels` has an illegal value.

#### See Also
**ThresholdAdaptiveBox** Performs adaptive thresholding with the Box method.

# ThresholdAdaptiveBox
*Performs adaptive thresholding with the Box method.*

## Syntax

```c
IppStatus ippiThresholdAdaptiveBox_8u_C1R(const Ipp8u* pSrc, int srcStep, Ipp8u* pDst, int dstStep, IppiSize roiSize, IppiSize maskSize, Ipp32f delta, Ipp8u valGT, Ipp8u valLE, IppiBorderType borderType, Ipp8u borderValue, Ipp8u* pBuffer);
```

```c
IppStatus ippiThresholdAdaptiveBox_8u_C1IR(Ipp8u* pSrcDst, int srcDstStep, IppiSize roiSize, IppiSize maskSize, Ipp32f delta, Ipp8u valGT, Ipp8u valLE, IppiBorderType borderType, Ipp8u borderValue, Ipp8u* pBuffer);
```

## Include Files

`ippi.h`

## Domain Dependencies

**Headers:** ippcore.h, ippvm.h, ipps.h

**Libraries:** ippcore.lib, ippvm.lib, ipps.lib

### Parameters

- **pSrc**: Pointer to the source image ROI.
- **srcStep**: Distance, in bytes, between the starting points of consecutive lines in the source image.
- **pDst**: Pointer to the destination image ROI.
- **dstStep**: Distance, in bytes, between the starting points of consecutive lines in the destination image.
- **pSrcDst**: Pointer to the source and destination image ROI (for the in-place function).
- **srcDstStep**: Distance, in bytes, between the starting points of consecutive lines in the source and destination image (for the in-place function).
- **roiSize**: Size of the destination image ROI, in pixels.
- **maskSize**: Size of the kernel that is used to calculate a threshold level. Width and height of `maskSize` must be equal and odd.
- **delta**: Value for threshold calculation.
- **valGT**: Output pixel if the source pixel value is more than threshold.
- **valLE**: Output pixel if the source pixel value is less than, or equal to threshold.
- **borderType**: Type of border. Possible values are:
Values of all border pixels are set to a constant.
Border is replicated from the edge pixels.
Border is obtained from the source image pixels in memory.

Mixed borders are also supported. They can be obtained by the bitwise operation OR between ippBorderRepl or ippBorderConst and the following flags:

- ippBorderInMemTop
- ippBorderInMemBottom
- ippBorderInMemLeft
- ippBorderInMemRight

Each of these flags means that for the corresponding border the outside pixels of the source image are in memory.

Constant value(s) to assign to pixels of the constant border. This parameter is applicable only to the ippBorderConst border type.

Pointer to the work buffer. To calculate the size of the temporary buffer, use the ThresholdAdaptiveBoxGetBufferSize function.

This function performs adaptive thresholding of the source image ROI using the Box method. Output pixels are calculated according to the following formulas:

\[
p_{\text{Dst}}(x,y) = \text{valGT}, \text{ if } p_{\text{Src}}(x,y) > T(x,y) \\
p_{\text{Dst}}(x,y) = \text{valLE}, \text{ if } p_{\text{Src}}(x,y) \leq T(x,y)
\]

where

\[T(x,y)\] is a mean of the maskSize.width*maskSize.height neighborhood of a \((x,y)\) pixel minus \(\delta\).

Before using this function, compute the size of the external work buffer using the ThresholdAdaptiveBoxGetBufferSize function.

Indicates no error.
Indicates an error when \(p_{\text{Src}}, p_{\text{Dst}}, p_{\text{SrcDst}},\) or \(p_{\text{Buffer}}\) is NULL.
Indicates an error when roiSize has a field with a zero or negative value.
Indicates an error when maskSize has a field with a zero or negative value, or fields of maskSize are not equal.
Indicates an error when borderType has an illegal value.
Example

The code example below demonstrates how to use the `ippiThresholdAdaptiveBox_8u_C1R` and `ThresholdAdaptiveBoxGetBufferSize` functions.

```c
IppStatus threshold_adaptive_box_8u_c1( void ) {
    Ipp8u pSrc[8*8] = {
        0, 255, 1, 254, 2, 253, 3, 252,
        251, 4, 250, 5, 249, 6, 248, 7,
        8, 247, 9, 246, 10, 245, 11, 244,
        243, 12, 242, 13, 241, 14, 240, 15,
        16, 239, 17, 238, 18, 237, 19, 236,
        235, 20, 234, 21, 233, 22, 232, 23,
        24, 231, 25, 230, 26, 229, 26, 228,
        227, 27, 226, 28, 225, 29, 224, 30
    };
    Ipp8u pDst[8*8];
    IppiSize roiSize = {8, 8};
    IppiSize maskSize = {3, 3};
    IppiBorderType borderType = ippBorderConst;
    int srcStep = 8 * sizeof(Ipp8u);
    int dstStep = 8 * sizeof(Ipp8u);
    int bufferSize;
    IppStatus status;
    Ipp32f delta = 0.5f;
    Ipp8u valGT = 254;
    Ipp8u valLE = 1;
    Ipp8u *pBuffer;
    ippiThresholdAdaptiveBoxGetBufferSize(roiSize, maskSize, ipp8u, 1, &bufferSize);
    pBuffer = ippsMalloc_8u(bufferSize);
    ippiThresholdAdaptiveBox_8u_C1R(pSrc, srcStep, pDst, dstStep, roiSize, maskSize, delta,
    valGT, valLE, borderType, 33, pBuffer);
    ippsFree(pBuffer);
    return status;
}
```

*pDst after function execution:*

```
1 254 1 254 1 254 1 254
254 1 254 1 254 1 254 1
1 254 1 254 1 254 1 254
254 1 254 1 254 1 254 1
1 254 1 254 1 254 1 254
254 1 254 1 254 1 254 1
1 254 1 254 1 254 1 254
254 1 254 1 254 1 254 1
```

See Also

`ThresholdAdaptiveBoxGetBufferSize` Computes the size of the work buffer for adaptive thresholding with the Box method.

`ThresholdAdaptiveGaussGetBufferSize` Computes the size of the adaptive threshold specification structure and the size of the work buffer for adaptive thresholding with the Gaussian method.
Syntax
IppStatus ippiThresholdAdaptiveGaussGetBufferSize(IppSize roiSize, IppSize maskSize, IppDataType dataType, int numChannels, int* pSpecSize, int* pBufferSize);

Include Files
ippi.h

Domain Dependencies
Headers: ippcore.h, ippvm.h, ipps.h
Libraries: ippcore.lib, ippvm.lib, ipps.lib

Parameters
roiSize
Size of the destination image ROI, in pixels.

maskSize
Size of the kernel that is used to calculate a threshold level. Width and height of maskSize must be equal and odd.

dataType
Data type of the source and destination images. Possible value is ipp8u.

numChannels
Number of channels in the images. Possible value is 1.

pSpecSize
Pointer to the size of the adaptive threshold specification structure.

pBufferSize
Pointer to the size (in bytes) of the external work buffer.

Description
This function computes the size, in bytes, of the adaptive threshold specification structure and the size of the external work buffer needed for the ThresholdAdaptiveGauss function. The results are stored in pSpecSize and pBufferSize.

For an example on how to use this function, refer to the example provided with the ThresholdAdaptiveGauss function description.

Return Values
ippStsNoErr
Indicates no error. Any other value indicates an error.

ippStsNullPtrErr
Indicates an error when pSpecSize or pBufferSize is NULL.

ippStsSizeErr
Indicates an error when roiSize has a field with a zero or negative value.

ippStsMaskSizeErr
Indicates an error when maskSize has a field with a zero or negative value, or fields of maskSize are not equal.

ippStsDataTypeErr
Indicates an error when dataType has an illegal value.

ippStsNumChannelsErr
Indicates an error when numChannels has an illegal value.

See Also
ThresholdAdaptiveGauss Performs adaptive thresholding with the Gaussian method.
ThresholdAdaptiveGaussInit

Initializes the threshold adaptive specification structure for adaptive thresholding with the Gaussian method.

Syntax

IppStatus ippiThresholdAdaptiveGaussInit(IppiSize roiSize, IppiSize maskSize, IppDataType dataType, int numChannels, Ipp32f sigma, IppiThresholdAdaptiveSpec* pSpec);

Include Files

ippi.h

Domain Dependencies

Headers: ippcore.h, ippvm.h, ipps.h
Libraries: ippcore.lib, ippvm.lib, ipps.lib

Parameters

roiSize  
Size of the destination image ROI, in pixels.

maskSize  
Size of the kernel that is used to calculate a threshold value. Width and height of maskSize must be equal and odd.

dataType  
Data type of the source and destination images. Possible value is ipp8u.

numChannels  
Number of channels in the images. Possible value is 1.

sigma  
Value of sigma that is used to calculate a threshold value for the Gaussian method. If sigma value is less than, or equal to zero, sigma is set automatically in compliance with the kernel size.

pSpec  
Pointer to the adaptive threshold specification structure.

Description

This function initializes the adaptive threshold specification structure pSpec for adaptive thresholding with the Gaussian method. Before using this function, compute the size of the specification structure using the ThresholdAdaptiveGaussGetBufferSize function.

If sigma is less than, or equal to zero, it is set according to the following formula:

\[ \text{sigma} = 0.3 \times ((\text{maskSize}.\text{width} - 1) \times 0.5 - 1) + 0.8 \]

For an example on how to use this function, refer to the example provided with the ThresholdAdaptiveGauss function description.

Return Values

ippStsNoErr  
Indicates no error. Any other value indicates an error.

ippStsNullPtrErr  
Indicates an error when pSpecSize is NULL.

ippStsSizeErr  
Indicates an error when dstRoiSize has a field with a zero or negative value.

ippStsMaskSizeErr  
Indicates an error when maskSize has a field with a zero or negative value, or fields of maskSize are not equal.
Indicates an error when `dataType` has an illegal value.

Indicates an error when `numChannels` has an illegal value.

**See Also**

ThresholdAdaptiveGaussGetBufferSize Computes the size of the adaptive threshold specification structure and the size of the work buffer for adaptive thresholding with the Gaussian method.

ThresholdAdaptiveGauss Performs adaptive thresholding with the Gaussian method.

**ThresholdAdaptiveGauss**

Performs adaptive thresholding with the Gaussian method.

**Syntax**

```c
IppStatus ippiThresholdAdaptiveGauss_8u_C1R(const Ipp8u* pSrc, int srcStep, Ipp8u* pDst, int dstStep, IppSize roiSize, Ipp32f delta, Ipp8u valGT, Ipp8u valLE, IppiBorderType borderType, Ipp8u borderValue, IppiThresholdAdaptiveSpec* pSpec, Ipp8u* pBuffer);
```

```c
IppStatus ippiThresholdAdaptiveGauss_8u_C1IR(Ipp8u* pSrcDst, int srcDstStep, IppSize roiSize, Ipp32f delta, Ipp8u valGT, Ipp8u valLE, IppiBorderType borderType, Ipp8u borderValue, IppiThresholdAdaptiveSpec* pSpec, Ipp8u* pBuffer);
```

**Include Files**

`ippi.h`

**Domain Dependencies**

**Headers:** `ippcore.h`, `ippvm.h`, `ipps.h`

**Libraries:** `ippcore.lib`, `ippvm.lib`, `ipps.lib`

**Parameters**

- `pSrc`  
  Pointer to the source image ROI.

- `srcStep`  
  Distance, in bytes, between the starting points of consecutive lines in the source image.

- `pDst`  
  Pointer to the destination image ROI.

- `dstStep`  
  Distance, in bytes, between the starting points of consecutive lines in the destination image.

- `pSrcDst`  
  Pointer to the source and destination image ROI (for the in-place function).

- `srcDstStep`  
  Distance, in bytes, between the starting points of consecutive lines in the source and destination image (for the in-place function).

- `roiSize`  
  Size of the destination image ROI, in pixels.

- `delta`  
  Value for threshold calculation.

- `valGT`  
  Output pixel if the source pixel value is more than threshold.

- `valLE`  
  Output pixel if the source pixel value is less than, or equal to threshold.
**borderType**

Type of border. Possible values are:

- ippBorderConst: Values of all border pixels are set to a constant.
- ippBorderRepl: Border is replicated from the edge pixels.
- ippBorderInMem: Border is obtained from the source image pixels in memory.

Mixed borders are also supported. They can be obtained by the bitwise operation OR between ippBorderRepl or ippBorderConst and the following flags:

- ippBorderInMemTop
- ippBorderInMemBottom
- ippBorderInMemLeft
- ippBorderInMemRight

Each of these flags means that for the corresponding border the outside pixels of the source image are in memory.

**borderValue**

Constant value(s) to assign to pixels of the constant border. This parameter is applicable only to the ippBorderConst border type.

**pSpec**

Pointer to the adaptive threshold specification structure.

**pBuffer**

Pointer to the work buffer. To calculate the size of the temporary buffer, use the ThresholdAdaptiveGaussGetBufferSize function.

**Description**

This function performs adaptive thresholding of the source image ROI using the Gaussian method. Output pixels are calculated according to the following formulas:

\[
\begin{align*}
    pDst(x,y) &= \text{valGT if } pSrc(x,y) > T(x,y) \\
    pDst(x,y) &= \text{valLE if } pSrc(x,y) \leq T(x,y)
\end{align*}
\]

where

\[ T(x,y) \text{ is a weighted sum (cross-correlation with a Gaussian window) of the } \text{maskSize.width}\times\text{maskSize.height} \text{ neighborhood of a } (x, y) \text{ pixel minus delta.} \]

The function uses a separable Gaussian filter. Filter coefficients are computed according to the following formula:

\[
G_i = A \times \exp\left(-\left(i - \frac{\text{maskSize.width}-1}{2}\right)^2/(0.5 \times \sigma^2)\right)
\]

where

\[ A \text{ is a scale factor for } \sum_i G_i = 1 \ (i = 0, ..., \text{maskSize.width}-1) \]

Before using this function, compute the size of the external work buffer and specification structure using the ThresholdAdaptiveGaussGetBufferSize function, and initialize the structure using the ThresholdAdaptiveGaussInit function.

**Return Values**

- ippStsNoErr: Indicates no error.
- ippStsNullPtrErr: Indicates an error when \( pSrc, pDst, pSrcDst, pSpec, \) or \( pBuffer \) is NULL.
Indicates an error when `roiSize` has a field with a zero or negative value.

Indicates an error when `pSpec` does not match.

Indicates an error when `borderType` has an illegal value.

Example

The code example below demonstrates how to use the `ippiThresholdAdaptiveGauss_8u_C1R`, `ThresholdAdaptiveGaussGetBufferSize`, and `ThresholdAdaptiveGaussInit` functions.

```c
IppStatus threshold_adaptive_gauss_8u_c1( void ) {
    Ipp8u pSrc[8*8] = {
        0, 255, 1, 254, 2, 253, 3, 252,
        251, 4, 250, 5, 249, 6, 248, 7,
        8, 247, 9, 246, 10, 245, 11, 244,
        243, 12, 242, 13, 241, 14, 240, 15,
        16, 239, 17, 238, 18, 237, 19, 236,
        235, 20, 234, 21, 233, 22, 232, 23,
        24, 231, 25, 230, 26, 229, 26, 228,
        227, 27, 226, 28, 225, 29, 224, 30
    };
    Ipp8u    pDst[8*8];
    IppiSize roiSize = {8, 8};
    IppiSize maskSize = {3, 3};
    IppiBorderType borderType = ippBorderConst;
    int     srcStep = 8 * sizeof(Ipp8u);
    int     dstStep = 8 * sizeof(Ipp8u);
    int     bufferSize;
    int     specSize;
    IppStatus status;
    Ipp32f  sigma = 10.0f;
    Ipp32f  delta = 0.5f;
    Ipp8u   valGT = 254;
    Ipp8u   valLE = 1;
    IppiThresholdAdaptiveSpec *pSpec;
    Ipp8u   *pBuffer;

    ippiThresholdAdaptiveGaussGetBufferSize(roiSize, maskSize, ipp8u, 1, &specSize, &bufferSize);
    pSpec = (IppiThresholdAdaptiveSpec *)ippsMalloc_8u(specSize);
    pBuffer = ippsMalloc_8u(bufferSize);
    ippiThresholdAdaptiveGaussInit(roiSize, maskSize, ipp8u, 1, sigma, pSpec);
    ippiThresholdAdaptiveGauss_8u_C1R(pSrc, srcStep, pDst, dstStep, roiSize, delta, valGT,
                                      valLE, borderType, 33, pSpec, pBuffer);
    ippsFree(pBuffer);
    ippsFree(pSpec);

    return status;
}

pDst after function execution:

1 254 1 254 1 254 1 254
254 1 254 1 254 1 254 1
1 254 1 254 1 254 1 254
254 1 254 1 254 1 254 1
1 254 1 254 1 254 1 254
```
<table>
<thead>
<tr>
<th>254</th>
<th>1 254</th>
<th>1 254</th>
<th>1 254</th>
<th>1 254</th>
<th>1 254</th>
<th>1 254</th>
<th>1 254</th>
<th>1 254</th>
<th>1 254</th>
<th>1 254</th>
<th>1 254</th>
<th>1 254</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 254</td>
<td>1 254</td>
<td>1 254</td>
<td>1 254</td>
<td>1 254</td>
<td>1 254</td>
<td>1 254</td>
<td>1 254</td>
<td>1 254</td>
<td>1 254</td>
<td>1 254</td>
<td>1 254</td>
<td>1 254</td>
</tr>
<tr>
<td>1 254</td>
<td>1 254</td>
<td>1 254</td>
<td>1 254</td>
<td>1 254</td>
<td>1 254</td>
<td>1 254</td>
<td>1 254</td>
<td>1 254</td>
<td>1 254</td>
<td>1 254</td>
<td>1 254</td>
<td>1 254</td>
</tr>
</tbody>
</table>

**See Also**

`ThresholdAdaptiveGaussGetSize()` Computes the size of the adaptive threshold specification structure and the size of the work buffer for adaptive thresholding with the Gaussian method.

`ThresholdAdaptiveGaussInit()` Initializes the threshold adaptive specification structure for adaptive thresholding with the Gaussian method.
This chapter describes the Intel® IPP image processing functions that perform two-dimensional discrete wavelet transform (DWT).

In many applications the multiresolution analysis by discrete wavelet transforms is a better alternative to windowing and discrete Fourier analysis techniques. On the one hand, the forward two-dimensional wavelet transform may be considered as a decomposition of an image on the base of functions bounded or localized in space; and on the other, the wavelet transforms are related to subband filtering and resampling.

Intel IPP for image processing contains one-level discrete wavelet decomposition and reconstruction functions. It also provides the necessary interface for initialization and deallocation of the transform context structure.

The wavelet transform type can be set by specifying the appropriate filter taps in the initialization function. Note that Intel IPP supports only one-dimensional finite impulse response filters for separable convolution.

The Intel IPP functions for wavelet decomposition and reconstruction use fast polyphase algorithm, which is equivalent to traditional application of separable convolution and dyadic resampling in different order. Figure “Equivalent Scheme of Wavelet Decomposition Algorithm” shows the equivalent algorithm of wavelet-based image decomposition:

**Equivalent Scheme of Wavelet Decomposition Algorithm**

Decomposition operation applied to a source image produces four output images of equal size: approximation image, horizontal detail image, vertical detail image, and diagonal detail image.
These decomposition components have the following meaning:

- The 'approximation' image is obtained by vertical and horizontal lowpass filtering.
- The 'horizontal detail' image is obtained by vertical highpass and horizontal lowpass filtering.
- The 'vertical detail' image is obtained by vertical lowpass and horizontal highpass filtering.
- The 'diagonal detail' image is obtained by vertical and horizontal highpass filtering.

The above image names are used in this document for identification convenience only.

The wavelet-based image reconstruction can be represented by a sequence of separate convolution and dyadic upsampling.

The reconstruction function uses four input images that are the same as those resulting from the decomposition operation.

Figure "Equivalent Scheme of Wavelet Reconstruction Algorithm" shows the equivalent algorithm of wavelet reconstruction of an image.
Wavelet transform functions support regions of interest (ROI, see Regions of Interest in Intel IPP in chapter 2) in the images. However, these functions do not perform internally any border extensions of image ROI data. It means that source images must already contain all border data that are necessary for convolution operations. See descriptions of the functions `ippiWTfwd` and `ippiWTInv` for detailed information on how to calculate extended image border sizes.

**Equivalent Scheme of Wavelet Reconstruction Algorithm**

---

**WTFwdGetSize**

*Calculates the size of the specification structure and work buffer for a forward wavelet transform.*
Syntax

IppStatus ippiWTFwdGetSize_32f(int numChannels, int lenLow, int anchorLow, int lenHigh, int anchorHigh, int* pSpecSize, int* pBufSize);

Include Files

ippi.h

Domain Dependencies

Headers: ippcore.h, ippvm.h, ipps.h
Libraries: ippcore.lib, ippvm.lib, ipps.lib

Parameters

numChannels
Number of channels in the image. Possible values are 1 or 3.

lenLow
Length of the lowpass filter.

anchorLow
Anchor position of the lowpass filter.

lenHigh
Length of the highpass filter.

anchorHigh
Anchor position of the highpass filter.

pSpecSize
Pointer to the computed size of the ippiWTFwd specification structure, in bytes.

pBufSize
Pointer to the computed size of the work buffer, in bytes.

Description

This function computes the size, in bytes, of the specification structure and work buffer required for the
forward wavelet transform function ippiWTFwd.

For an example on how to use this function, refer to the example provided with the ippiWTInv function
description.

Return Values

ippStsNoErr
Indicates no error. Any other value indicates an error or a warning.

ippStsNullPtrErr
Indicates an error when any of the specified pointers is NULL.

ippStsNumChannlesErr
Indicates an error when numChannels has an illegal value.

ippStsNumChannlesErr
Indicates an error when lenLow or lenHigh is less than 2.

ippStsAnchorErr
Indicates an error when anchorLow or anchorHigh is less than zero.

See Also

WTFwd Performs one-level wavelet decomposition of an image.
WTInv Performs one-level wavelet reconstruction of an image.

WTFwdInit

Initializes the forward wavelet transform context structure.
Syntax

IppStatus ippiWTFwdInit_32f_C1R(IppiWTFwdSpec_32f_C1R* pSpec, const Ipp32f* pTapsLow, int lenLow, int anchorLow, const Ipp32f* pTapsHigh, int lenHigh, int anchorHigh);

IppStatus ippiWTFwdInit_32f_C3R(IppiWTFwdSpec_32f_C3R* pSpec, const Ipp32f* pTapsLow, int lenLow, int anchorLow, const Ipp32f* pTapsHigh, int lenHigh, int anchorHigh);

Include Files

ippi.h

Domain Dependencies

Headers: ippcore.h, ippvm.h, ipps.h
Libraries: ippcore.lib, ippvm.lib, ipps.lib

Parameters

pSpec
Double pointer to the forward wavelet transform specification structure.

pTapsLow
Pointer to lowpass filter taps.

lenLow
Length of the lowpass filter.

anchorLow
Anchor position of the lowpass filter.

pTapsHigh
Pointer to highpass filter taps.

lenHigh
Length of the highpass filter.

anchorHigh
Anchor position of the highpass filter.

Description

This function initializes the specification structure pSpec for a one-level wavelet decomposition.

The forward wavelet transform specification structure contains parameters of a wavelet filter bank used for image decomposition. The filter bank consists of two analysis filters and includes the lowpass decomposition filter (or coarse filter) and the highpass decomposition filter (or detail filter).

The parameters pTapsLow and pTapsHigh specify coefficients, and anchorLow and anchorHigh - anchor positions for two synthesis filters. The anchor value sets the initial leftmost filter position relative to image row or column as shown in the figure below:

Anchor Value and Initial Filter Position for Wavelet Decomposition

![Anchor Value and Initial Filter Position for Wavelet Decomposition](image-url)
Here a stands for anchor value, N is filter length, x0 is the starting pixel of the processed row or column, and x-1, x-2, ... are the additional border pixels that are needed for calculations. The anchor value and filter length determine right, left, top, and bottom border sizes for the source image used in decomposition. The corresponding C-language expressions to calculate border sizes are given in the description of ippiWTFwd function.

For an example on how to use this function, refer to the example provided with the ippiWTInv function description.

**Return Values**

- **ippStsNoErr**: Indicates no error. Any other value indicates an error or a warning.
- **ippStsNullPtrErr**: Indicates an error when any of the specified pointers is NULL.
- **ippStsNumChannelsErr**: Indicates an error when numChannels has an illegal value.
- **ippStsSizeErr**: Indicates an error when filter length lenLow or lenHigh is less than 2.
- **ippStsAnchorErr**: Indicates an error when anchor position anchorLow or anchorHigh is less than zero.

**See Also**

- **WTFwdGetSize**: Calculates the size of the specification structure and work buffer for a forward wavelet transform.
- **WTFwd**: Performs one-level wavelet decomposition of an image.
- **WTInv**: Performs one-level wavelet reconstruction of an image.

**WTFwd**

*Performs one-level wavelet decomposition of an image.*

**Syntax**

```c
IppStatus ippiWTFwd_32f_C1R (const Ipp32f* pSrc, int srcStep, Ipp32f* pApproxDst, int approxStep, Ipp32f* pDetailXDst, int detailXStep, Ipp32f* pDetailYDst, int detailYStep, Ipp32f* pDetailXYDst, int detailXYStep, IppiSize dstRoiSize, const IppiWTFwdSpec_32f_C1R* pSpec, Ipp8u* pBuffer);
IppStatus ippiWTFwd_32f_C3R (const Ipp32f* pSrc, int srcStep, Ipp32f* pApproxDst, int approxStep, Ipp32f* pDetailXDst, int detailXStep, Ipp32f* pDetailYDst, int detailYStep, Ipp32f* pDetailXYDst, int detailXYStep, IppiSize dstRoiSize, const IppiWTFwdSpec_32f_C3R* pSpec, Ipp8u* pBuffer);
```

**Include Files**

ippi.h

**Parameters**

- **pSrc**: Pointer to the source image ROI.
- **srcStep**: Distance, in bytes, between the starting points of consecutive lines in the source image buffer.
- **pApproxDst**: Pointer to ROI of the destination approximation image.
**approxStep**
Distance, in bytes, between the starting points of consecutive lines in the approximation image buffer.

**pDetailXDst**
Pointer to ROI of the destination horizontal detail image.

**detailXStep**
Distance, in bytes, between the starting points of consecutive lines in the horizontal detail image buffer.

**pDetailYDst**
Pointer to ROI of the destination vertical detail image.

**detailYStep**
Distance, in bytes, between the starting points of consecutive lines in the vertical detail image buffer.

**pDetailXYDst**
Pointer to ROI of the destination diagonal detail image.

**detailXYStep**
Distance, in bytes, between the starting points of consecutive lines in the diagonal detail image buffer.

**dstRoiSize**
Size of the ROI in pixels for all destination images.

**pSpec**
Pointer to the allocated and initialized forward DWT specification structure.

**pBuffer**
Pointer to the allocated buffer for intermediate operations.

**Description**
This function operates with ROI (see Regions of Interest in Intel IPP). This function performs one-level wavelet decomposition of a source image pointed to by **pSrc** into four destination subimages pointed to by **pApproxDst**, **pDetailXDst**, **pDetailYDst**, and **pDetailXYDst**. See Figure “Equivalent Scheme of Wavelet Decomposition Algorithm” for the equivalent algorithm of **ippiWTFwd** function operation.

Wavelet parameters are contained in the forward transform specification structure **pSpec**. Before using this function, compute the size of the structure and work buffer using the **WTFwdGetSize** function and initialize the structure using **WTFwdInit**.

---

**Optimization Notice**

Intel's compilers may or may not optimize to the same degree for non-Intel microprocessors for optimizations that are not unique to Intel microprocessors. These optimizations include SSE2, SSE3, and SSSE3 instruction sets and other optimizations. Intel does not guarantee the availability, functionality, or effectiveness of any optimization on microprocessors not manufactured by Intel. Microprocessor-dependent optimizations in this product are intended for use with Intel microprocessors. Certain optimizations not specific to Intel microarchitecture are reserved for Intel microprocessors. Please refer to the applicable product User and Reference Guides for more information regarding the specific instruction sets covered by this notice.

Notice revision #20110804

The pointer **pSrc** points to memory location of the source image rectangular ROI of size **srcWidth** by **srcHeight** which is uniquely determined by the size of destination ROI as:

\[
\text{srcWidth} = 2 \times \text{dstRoiSize.width} \\
\text{srcHeight} = 2 \times \text{dstRoiSize.height}
\]

The source image ROI size always has even dimensions, as it is computed from the **dstRoiSize** parameter as follows:

\[
\text{srcRoiSize.width} = 2 \times \text{dstRoiSize.width} \\
\text{srcRoiSize.height} = 2 \times \text{dstRoiSize.height}
\]
To use this function for images with uneven width or height, you should truncate the last column/row or extend an image to even dimensions.

**NOTE**
The source image ROI does not include border pixels necessary to compute some destination pixels. It means that prior to using ippiWTFwd function the application program must apply some border extension model (symmetrical, wraparound or another) to the source image ROI through filling of neighboring memory locations. As a result, the size of memory block allocated for the source image must be extended to accommodate for added border pixels outside ROI formal boundaries.

Figure "Extended Source Image for Wavelet Decomposition" schematically shows the source image ROI and extended image area.

Extended Source Image for Wavelet Decomposition

Use the following C-language expressions to calculate extended image border sizes:

```c
int leftBorderLow   = lenLow   - 1 - anchorLow;
int leftBorderHigh  = lenHigh  - 1 - anchorHigh;
int rightBorderLow  = anchorLow;
int rightBorderHigh = anchorHigh;
int leftTopBorder   = IPP_MAX(leftBorderLow, leftBorderHigh);
int rightBottomBorder = IPP_MAX(rightBorderLow, rightBorderHigh);
```

See the description of the function WTFwdInit for the explanation of the parameters.

Note that the left and top borders have equal size. The same holds for the right and bottom borders which have equal size too.

The size of the source image area extended by border pixels can be defined as

```c
srcWidthWithBorders = srcWidth  + leftTopBorder + rightBottomBorder;
srcHeightWithBorders = srcHeight + leftTopBorder + rightBottomBorder;
```
All destination images have equal size specified by the parameter $dstRoiSize$.

Conversely, to perform a wavelet reconstruction of the full size source image from the component images obtained by decomposition, use extended component images for the reconstruction pass. See the description of the function ippiWTInv for more details.

The ROI concept used in wavelet transform functions can be applied to processing large images by blocks, or 'tiles'. To accomplish this, the source image should be subdivided into overlapping blocks in the following way:

- Main part (ROI) of each block is adjacent to neighboring blocks and has no intersection with neighbor's ROIs;
- Extended borders of each block overlap with ROIs of neighboring blocks.

This subdivision scheme is illustrated in Figure "Image Division into Blocks with Overlapping Borders".

### Image Division into Blocks with Overlapping Borders

For an example on how to use this function, refer to the example provided with the WTInv function description.

**Return Values**

- ippStsNoErr: Indicates no error. Any other value indicates an error or a warning.
- ippStsNullPtrErr: Indicates an error condition if any of the specified pointers is NULL.
- ippStsSizeErr: Indicates an error condition if $dstRoiSize$ has a field with zero or negative value.
- ippStsStepErr: Indicates an error condition if step through any buffer is less than or equal to zero.
- ippStsContextMatchErr: Indicates an error condition if a pointer to an invalid specification structure is passed.

**See Also**

- WTFwdGetSize: Calculates the size of the specification structure and work buffer for a forward wavelet transform.
- WTFwdInit: Initializes the forward wavelet transform context structure.
- WTInv: Performs one-level wavelet reconstruction of an image.
WTFwdGetSize

Calculates the size of the specification structure and work buffer for an inverse wavelet transform.

Syntax

IppStatus ippiWTInvGetSize_32f(int numChannels, int lenLow, int anchorLow, int lenHigh, int anchorHigh, int* pSpecSize, int* pBufSize);

Include Files

ippi.h

Domain Dependencies

Headers: ippcore.h, ippvm.h, ipps.h
Libraries: ippcore.lib, ippvm.lib, ipps.lib

Parameters

numChannels

Number of channels in the image. Possible values are 1 or 3.

lenLow

Length of the lowpass filter.

anchorLow

Anchor position of the lowpass filter.

lenHigh

Length of the highpass filter.

anchorHigh

Anchor position of the highpass filter.

pSpecSize

Pointer to the computed size of the ippiWTInv specification structure, in bytes.

pBufSize

Pointer to the computed size of the work buffer, in bytes.

Description

This function computes the size, in bytes, of the specification structure and work buffer required for the inverse wavelet transform function ippiWTInv.

For an example on how to use this function, refer to the example provided with the ippiWTInv function description.

Return Values

ippStsNoErr

Indicates no error. Any other value indicates an error or a warning.

ippStsNullPtrErr

Indicates an error when any of the specified pointers is NULL.

ippStsNumChannelsErr

Indicates an error when numChannels has an illegal value.

ippStsNumChannelsErr

Indicates an error when lenLow or lenHigh is less than 2.

ippStsAnchorErr

Indicates an error when anchorLow or anchorHigh is less than zero.

See Also

WTInv Performs one-level wavelet reconstruction of an image.
WTInvInit

Initializes the inverse wavelet transform specification structure.

Syntax

IppStatus ippiWTInvInit_32f_C1R (IppiWTInvSpec_32f_C1R* pSpec, const Ipp32f* pTapsLow, int lenLow, int anchorLow, const Ipp32f* pTapsHigh, int lenHigh, int anchorHigh);
IppStatus ippiWTInvInit_32f_C3R (IppiWTInvSpec_32f_C3R* pSpec, const Ipp32f* pTapsLow, int lenLow, int anchorLow, const Ipp32f* pTapsHigh, int lenHigh, int anchorHigh);

Include Files

ippi.h

Domain Dependencies

Headers: ippcore.h, ippvm.h, ipps.h
Libraries: ippcore.lib, ippvm.lib, ipps.lib

Parameters

pSpec
  Double pointer to a new allocated and initialized inverse DWT context structure.
pTapsLow
  Pointer to lowpass filter taps.
lenLow
  Length of the lowpass filter.
anchorLow
  Anchor position of the lowpass filter.
pTapsHigh
  Pointer to highpass filter taps.
lenHigh
  Length of the highpass filter.
anchorHigh
  Anchor position of the highpass filter.

Description

This function initializes the specification structure pSpec for a one-level wavelet reconstruction.

The inverse wavelet transform specification structure contains parameters of a wavelet filter bank used for image reconstruction. The filter bank consists of two synthesis filters and includes the lowpass reconstruction filter (or coarse filter) and the highpass reconstruction filter (or detail filter).
The parameters \( p_{\text{TapsLow}} \) and \( p_{\text{TapsHigh}} \) specify coefficients, and \( \text{anchorLow} \) and \( \text{anchorHigh} \) - anchor positions for two synthesis filters. The anchor value sets the initial leftmost filter position relative to image row or column as shown in the figure below:

**Anchor Value and Initial Filter Position for Wavelet Reconstruction**

Here \( a \) stands for anchor value, \( N \) is filter length, \( x_0 \) is the starting pixel of the processed row or column, and \( x_{-1}, x_{-2}, \ldots \) are the additional border pixels that are needed for calculations. As seen from this figure, anchor position is specified relative to upsampled source data. The anchor value and filter length completely determine right, left, top, and bottom border sizes for source images used in reconstruction. The corresponding C-language expressions to calculate border sizes are given in the description of the \texttt{ippWTInv} function.

For an example on how to use this function, refer to the example provided with the \texttt{ippiWTInv} function description.

**Return Values**

- \texttt{ippStsNoErr} Indicates no error. Any other value indicates an error or a warning.
- \texttt{ippStsNullPtrErr} Indicates an error when any of the specified pointers is \texttt{NULL}.
- \texttt{ippStsNumChannelsErr} Indicates an error when \texttt{numChannels} has an illegal value.
- \texttt{ippStsSizeErr} Indicates an error when filter length \texttt{lenLow} or \texttt{lenHigh} is less than 2.
- \texttt{ippStsAnchorErr} Indicates an error when anchor position \texttt{anchorLow} or \texttt{anchorHigh} is less than zero.

**See Also**

- \texttt{WTInvGetSize} Calculates the size of the specification structure and work buffer for an inverse wavelet transform.
- \texttt{WTInv} Performs one-level wavelet reconstruction of an image.

**WTInv**

Performs one-level wavelet reconstruction of an image.
Syntax

IppStatus ippiWTInv_32f_C1R(const Ipp32f* pApproxSrc, int approxStep, const Ipp32f* pDetailXSrc, int detailXStep, const Ipp32f* pDetailYSrc, int detailYStep, const Ipp32f* pDetailXYSrc, int detailXYStep, IppiSize srcRoiSize, Ipp32f* pDst, int dstStep, const IppiWTInvSpec_32f_C1R* pSpec, Ipp8u* pBuffer);

IppStatus ippiWTInv_32f_C3R(const Ipp32f* pApproxSrc, int approxStep, const Ipp32f* pDetailXSrc, int detailXStep, const Ipp32f* pDetailYSrc, int detailYStep, const Ipp32f* pDetailXYSrc, int detailXYStep, IppiSize srcRoiSize, Ipp32f* pDst, int dstStep, const IppiWTInvSpec_32f_C3R* pSpec, Ipp8u* pBuffer);

Include Files

ippi.h

Domain Dependencies

Headers: ippcore.h, ippvm.h, ipps.h
Libraries: ippcore.lib, ippvm.lib, ipps.lib

Parameters

pApproxSrc Pointer to ROI of the source approximation image.
approxStep Distance, in bytes, between the starting points of consecutive lines in the approximation image buffer.
pDetailXSrc Pointer to ROI of the source horizontal detail image.
detailXStep Distance, in bytes, between the starting points of consecutive lines in the horizontal detail image buffer.
pDetailYSrc Pointer to ROI of the source vertical detail image.
detailYStep Distance, in bytes, between the starting points of consecutive lines in the vertical detail image buffer.
pDetailXYSrc Pointer to ROI of the source diagonal detail image.
detailXYStep Distance, in bytes, between the starting points of consecutive lines in the diagonal detail image buffer.
srcRoiSize Size of ROI in pixels for all source images.
pDst Pointer to the destination image ROI.
dstStep Distance, in bytes, between the starting points of consecutive lines in the destination image buffer.
pSpec Pointer to the allocated and initialized inverse DWT specification structure.
pBuffer Pointer to the allocated buffer for intermediate operations.

Description

This function operates with ROI (see Regions of Interest in Intel IPP ). This function performs wavelet reconstruction of the output image pDst from the four component images. See Figure "Image Division into Blocks with Overlapping Borders" for the equivalent algorithm of ippiWTInv function operation. Wavelet parameters are contained in the inverse transform specification structure pSpec. Before using this function, compute the size of the structure and work buffer using the ippiWTInvGetSize function and initialize the structure using ippiWTInvInit.
Optimization Notice

Intel’s compilers may or may not optimize to the same degree for non-Intel microprocessors for optimizations that are not unique to Intel microprocessors. These optimizations include SSE2, SSE3, and SSSE3 instruction sets and other optimizations. Intel does not guarantee the availability, functionality, or effectiveness of any optimization on microprocessors not manufactured by Intel. Microprocessor-dependent optimizations in this product are intended for use with Intel microprocessors. Certain optimizations not specific to Intel microarchitecture are reserved for Intel microprocessors. Please refer to the applicable product User and Reference Guides for more information regarding the specific instruction sets covered by this notice.

Notice revision #20110804

The pointers \texttt{pApproxSrc}, \texttt{pDetailXSrc}, \texttt{pDetailYSrc}, and \texttt{pDetailXYSrc} point to ROIs of source images excluding borders. All source ROIs have the same size \texttt{srcRoiSize}, while the destination image size is uniquely determined from the following relations:

\[
dstWidth = 2 * \texttt{srcRoiSize.width}; \quad dstHeight = 2 * \texttt{srcRoiSize.height};
\]

As source ROIs do not include border pixels required to computations, the application program have to apply a border extension model (symmetrical, wraparound or another) to ROIs of all source images filling the neighboring memory locations. Note the border sizes may be different for different source images. The following C-language expressions can be used to calculate extended image border sizes:

\[
\begin{align*}
    \text{leftBorderLow} & = (\text{lenLow} - 1 - \text{anchorLow}) / 2; \\
    \text{leftBorderHigh} & = (\text{lenHigh} - 1 - \text{anchorHigh}) / 2; \\
    \text{rightBorderLow} & = (\text{anchorLow} + 1) / 2; \\
    \text{rightBorderHigh} & = (\text{anchorHigh} + 1) / 2; \\
    \text{apprLeftBorder} & = \text{leftBorderLow}; \\
    \text{apprRightBorder} & = \text{rightBorderLow}; \\
    \text{apprTopBorder} & = \text{leftBorderLow}; \\
    \text{apprBottomBorder} & = \text{rightBorderLow}; \\
    \text{detxLeftBorder} & = \text{leftBorderLow}; \\
    \text{detxRightBorder} & = \text{rightBorderLow}; \\
    \text{detxTopBorder} & = \text{leftBorderHigh}; \\
    \text{detxBottomBorder} & = \text{rightBorderHigh}; \\
    \text{detyLeftBorder} & = \text{leftBorderHigh}; \\
    \text{detyRightBorder} & = \text{rightBorderHigh}; \\
    \text{detyTopBorder} & = \text{leftBorderLow}; \\
    \text{detyBottomBorder} & = \text{rightBorderLow}; \\
    \text{detxyLeftBorder} & = \text{leftBorderHigh}; \\
    \text{detxyRightBorder} & = \text{rightBorderHigh}; \\
    \text{detxyTopBorder} & = \text{leftBorderHigh}; \\
    \text{detxyBottomBorder} & = \text{rightBorderHigh}; \\
\end{align*}
\]

See the description of the function \texttt{ippiWTInvInit} for the explanation of the used parameters.

The above relations show that left and top borders always have equal size only for approximation and diagonal detail images. Right and bottom borders also have equal size only for approximation and diagonal detail images. Thus, the size of memory block allocated for each source image must be extended to accommodate for added border pixels outside ROI.
Figure "Extended Horizontal Detail Source Image for Wavelet Reconstruction" shows ROI and extended image area for the horizontal detail source image.

**Extended Horizontal Detail Source Image for Wavelet Reconstruction**

Sizes of source images extended by border pixels can be calculated as follows:

```plaintext
apprWidthWithBorders  = srcWidth  + apprLeftBorder + apprRightBorder;
apprHeightWithBorders = srcHeight + apprTopBorder  + apprBottomBorder;
dextWidthWithBorders  = srcWidth  + dextLeftBorder + dextRightBorder;
dextHeightWithBorders = srcHeight + dextTopBorder  + dextBottomBorder;
detyWidthWithBorders  = srcWidth  + detyLeftBorder + detyRightBorder;
detyHeightWithBorders = srcHeight + detyTopBorder  + detyBottomBorder;
detxyWidthWithBorders = srcWidth  + detxyLeftBorder + detxyRightBorder;
detxyHeightWithBorders = srcHeight + detxyTopBorder  + detxyBottomBorder;
```

The ROI concept can be used to reconstruct large images by blocks or 'tiles'.

To accomplish this, each the source images into blocks with overlapping borders, similar to what is considered in the description of the function `ippiWTFwd`. Each component must be subdivided into the same pattern of rectangular blocks.

**Return Values**

- **ippStsNoErr** indicates no error. Any other value indicates an error or a warning.
- **ippStsNullPtrErr** indicates an error condition if any of the specified pointers is NULL.
- **ippStsSizeErr** indicates an error condition if `srcRoiSize` has a field with zero or negative value.
- **ippStsStepErr** indicates an error condition if step through any buffer is less than or equal to zero.
- **ippStsContextMatchErr** indicates an error condition if a pointer to an invalid specification structure is passed.
Example

The example below shows how to use the function `ippiWTInv_32f_C1R`.

```c
void func_wavelet()
{
    IppiWTfwdSpec_32f_C1R* pSpecFwd;
    IppiWTInvSpec_32f_C1R* pSpecInv;
    int specSizeFwd, specSizeInv;
    Ipp32f pTapsLow[3] = {0.25, 0.5, 0.25};
    int lenLow = 3;
    int anchorLow = 1;
    Ipp32f pTapsHigh[3] = {0.75, -0.25, -0.125};
    int lenHigh = 3;
    int anchorHigh = 1;
    Ipp32f pSrc[8*8] = {
        0.0, 0.0, 0.0, 11.1, 11.1, 0.0, 0.0, 0.0,
        0.0, 0.0, 0.0, 11.1, 11.1, 0.0, 0.0, 0.0,
        0.0, 0.0, 0.0, 11.1, 11.1, 0.0, 0.0, 0.0,
        11.1, 11.1, 11.1, 11.1, 11.1, 11.1, 11.1, 11.1,
        11.1, 11.1, 11.1, 11.1, 11.1, 11.1, 11.1, 11.1,
        0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0,
        0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0,
        0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0};
    Ipp32f pSrcB[9*9];
    int srcStepB = 9*sizeof(Ipp32f);
    Ipp32f srcSizeB = {9, 9};
    int srcStep = 8*sizeof(Ipp32f);
    Ipp32f dstSize = {8, 8};
    Ipp32f pNodeDetailXDst[4*4];
    Ipp32f pNodeDetailYDst[4*4];
    Ipp32f pNodeDetailXYDst[4*4];
    Ipp32f pNodeApproxDst[4*4];
    Ipp32f dstSize = {4, 4};
    int bufSizeFwd, bufSizeInv;
    Ipp8u* pBufferFwd;
    Ipp8u* pBufferInv;
    Ipp32f pDestInv[8*8];
    Ipp32f pAppB[5*5];
    Ipp32f pYB[5*5];
    Ipp32f pXYB[5*5];
    int StepB = 5*sizeof(Ipp32f);
    Ipp32f roiInvSize = {4, 4};
    Ipp32f roiInvSizeB = {5, 5};
    int stepDestInv = 8*sizeof(Ipp32f);
    int approxStep, detailXStep, detailYStep, detailXYStep;
    approxStep = detailXStep = detailYStep = detailXYStep = 4*sizeof(Ipp32f);
    //adds border to the source image
    ipp1CopyWrapBorder_32s_C1R((Ipp32s*)pSrc, srcStep, roiSize, (Ipp32s*)pSrcB, srcStepB,
        roiSizeB, 1, 1);
    //performs forward wavelet transform
    ipp1WTfwdGetSize_32f(1, lenLow, anchorLow, lenHigh, anchorHigh, &specSizeFwd, &bufSizeFwd);
    pSpecFwd = (IppiWTfwdSpec_32f_C1R*) ippMalloc(specSizeFwd);
    pBufferFwd = (Ipp8u*) ippMalloc(bufSizeFwd);
    ipp1WTfwdInit_32f_C1R(pSpecFwd, pTapsLow, lenLow, anchorLow, pTapsHigh, lenHigh,
        anchorHigh);
    ipp1WTfwd_32f_C1R(pSrcB + roiSizeB.width + 1, srcStepB, pApproxDst, approxStep,
            pNodeDetailXDst, detailXStep, pNodeDetailYDst, detailYStep, pNodeDetailXYDst, detailXYStep,
            dstRoiSize, pSpecFwd, pBufferFwd);
}
```
After WTFwd ->
pApproxDst
0.0  2.8   8.3  0.0
2.8  4.9   9.0  2.8
8.3  9.0  10.4 8.3
0.0  2.8   8.3  0.0
pDetailXDst
0.0   1.0  3.1   0.0
8.3   7.3  5.2   8.3
-4.2  -2.1  2.1  -4.2
0.0   1.0  3.1   0.0
pDetailYDst
0.0   8.3  -4.2   0.0
1.0   7.3  -2.1  1.0
3.1   5.2  2.1   3.1
0.0   8.3  -4.2   0.0
pDetailXYDst
0.0   3.1   -1.6   0.0
3.1   0.0   4.7  3.1
-1.6  4.7   -4.7  -1.6
0.0   3.1   -1.6  0.0
After WTFinv ->
pDstInv
0.0  2.8  -0.3  -0.6  2.1  1.1  0.0  0.0
<table>
<thead>
<tr>
<th>2.8</th>
<th>5.9</th>
<th>2.7</th>
<th>4.9</th>
<th>3.4</th>
<th>5.4</th>
<th>2.8</th>
<th>5.1</th>
</tr>
</thead>
<tbody>
<tr>
<td>-0.3</td>
<td>2.7</td>
<td>-0.6</td>
<td>-1.2</td>
<td>2.2</td>
<td>0.7</td>
<td>-0.3</td>
<td>-0.5</td>
</tr>
<tr>
<td>-0.6</td>
<td>4.9</td>
<td>-1.2</td>
<td>-2.2</td>
<td>3.9</td>
<td>1.2</td>
<td>-0.6</td>
<td>-1.0</td>
</tr>
<tr>
<td>2.1</td>
<td>3.4</td>
<td>2.2</td>
<td>3.9</td>
<td>1.8</td>
<td>3.7</td>
<td>2.1</td>
<td>3.8</td>
</tr>
<tr>
<td>1.1</td>
<td>5.4</td>
<td>0.7</td>
<td>1.2</td>
<td>3.7</td>
<td>3.2</td>
<td>1.1</td>
<td>1.9</td>
</tr>
<tr>
<td>0.0</td>
<td>2.8</td>
<td>-0.3</td>
<td>-0.6</td>
<td>2.1</td>
<td>1.1</td>
<td>0.0</td>
<td>0.0</td>
</tr>
<tr>
<td>0.0</td>
<td>5.1</td>
<td>-0.5</td>
<td>-1.0</td>
<td>3.8</td>
<td>1.9</td>
<td>0.0</td>
<td>0.0</td>
</tr>
</tbody>
</table>

See Also

WTInvGetSize Calculates the size of the specification structure and work buffer for an inverse wavelet transform.

WTInvInit Initializes the inverse wavelet transform specification structure.

WTFwd Performs one-level wavelet decomposition of an image.
This chapter provides some background for the computer vision concepts used in the Intel® IPP library as well as detailed descriptions of the Intel IPP image processing functions for computer vision. These functions are combined in groups by their functionality.

**Using ippiAdd for Background Differencing**

This section describes functions that help build a statistical model of a background. This model can be used to subtract the background from an image.

Here, the term "background" stands for a set of motionless image pixels – that is, pixels that do not belong to any object, moving in front of the camera. The definition of background can vary if considered in other techniques of object extraction. For example, if the depth map of the scene can be obtained, for example, with the help of stereo, background can be determined as static parts of the scene that are located far enough from the camera.

The simplest background model assumes that every background pixel brightness varies independently, according to normal distribution. To calculate the characteristics of the background, several dozens of frames, as well as their squares, can be accumulated. That is, for every pixel location we find the sum of pixel values in this location \( S(x, y) \), using the function \( \text{ippiAdd} \), and the sum of squares of the values \( Sq(x, y) \), using the function \( \text{ippiAddSquare} \). Then mean is calculated as

\[
m(x, y) = \frac{S(x, y)}{N},
\]

where \( N \) is the number of collected frames, and standard deviation as

\[
\text{stddev}(x, y) = \sqrt{\frac{Sq(x, y)}{N} - \left(\frac{S(x, y)}{N}\right)^2}
\]

After that, the pixel in a certain pixel location within a certain frame is considered as belonging to a moving object, if the condition \( \text{abs}(p(x, y) - m(x, y)) < C \times \text{stddev}(x, y) \), where \( C \) is a constant, is met. If \( C \) is equal to 3, it satisfies the "three sigmas" rule. To obtain such background model, objects should be put away from the camera for a few seconds, so that the whole image from the camera represents the subsequent background observation.

Adapting the background differencing model to changes in lighting conditions and background scenes, for example, when the camera moves or an object passes behind the front object, can improve the described technique.

The mean brightness can be calculated through replacing the simple average with the running average found by using the function \( \text{ippiAddWeighted} \). Also, several techniques can be used to identify moving scene parts and exclude them while accumulating background information. These techniques include change detection (see the functions \( \text{ippiAbsDiff} \) in chapter 5 and \( \text{ippiThreshold} \) in chapter 7), optical flow, and some other operations.

Relevant addition functions used for background differencing include:
- \text{Add\_8u32f\_C1IR}, \text{Add\_8s32f\_C1IR}, \text{Add\_32f\_C1IR},
- \text{Add\_8u32f\_C1IMR}, \text{Add\_8s32f\_C1IMR}, \text{Add\_32f\_C1IMR} (see \text{Add}),
- and also all flavors of \text{AddSquare}, \text{AddProduct}, and \text{AddWeighted}.

**Feature Detection Functions**

This section describes feature detection functions.
The set of the Sobel derivative filters is generally used to find edges, ridges, and blobs, especially in case of scale-space images, for example, pyramids.

The following naming conventions are used in the equations described below:

- $D_x$ and $D_y$ are the first $x$ and $y$ derivatives, respectively.
- $D_{xx}$ and $D_{yy}$ are the second $x$ and $y$ derivatives, respectively.
- $D_{xy}$ is the partial $x$ and $y$ derivative.
- $D_{xxx}$ and $D_{yyy}$ are the third $x$ and $y$ derivatives, respectively.
- $D_{xxy}$ and $D_{xyy}$ are the third partial $x$ and $y$ derivatives.

Corner Detection

The Sobel and Scharr first derivative operators are to be used to take the $x$ and $y$ derivatives of an image. Then a small region of interest (ROI) is to be defined to detect corners in.

A 2x2 matrix of sums of the $x$ and $y$ derivatives is created as follows:

$$
A = \sum_{u,v} w(u, v) \begin{bmatrix} I_x^2 & I_x I_y \\ I_x I_y & I_y^2 \end{bmatrix} = \begin{bmatrix} \langle I_x^2 \rangle & \langle I_x I_y \rangle \\ \langle I_x I_y \rangle & \langle I_y^2 \rangle \end{bmatrix}
$$

Solving

$$\det(C - \lambda I) = 0,$$

where $\lambda$ is a column vector of the eigen values and $I$ is the identity matrix, gives the eigen values. For the 2x2 matrix of the equation above, the solutions may be written in a closed form:

$$\lambda = \frac{\sum D_x^2 + \sum D_y^2 \pm \sqrt{\left(\sum D_x^2 + \sum D_y^2\right)^2 - 4 \cdot \left(\sum D_x^2 \sum D_y^2 - (\sum D_x D_y)^2\right)}}{2}.$$

If $\lambda_1, \lambda_2 > t$, where $t$ is some threshold, then a corner is considered to be found at that location. This can be very useful for object or shape recognition.

FastNGetSize

*Computes the size of the FastN context structure.*

**Syntax**

```c
IppStatus ippiFastNGetSize(IppiSize srcSize, int circleRadius, int N, int orientationBins, int option, IppDataType dataType, int numChannels, int* pSpecSize);
```

**Include Files**

ippcv.h

**Domain Dependencies**

- *Headers:* ippcore.h, ippvm.h, ipps.h,ippi.h
- *Libraries:* ippcore.lib, ippvm.lib, ipps.lib,ippi.lib

**Parameters**

- `srcSize`: Size of the source ROI, in pixels.
circleRadius

Radius of the pixel circle. The radius value equal to 1 corresponds to the distance between the closest pixels from common row or column. Supported values are 1, 2, 3, 5, 7, 9.

N

Critical number of serial pixels on circle for defining a corner. The ranges are as follows:

<table>
<thead>
<tr>
<th>circleRadius</th>
<th>N</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>5 ≤ N ≤ 8</td>
</tr>
<tr>
<td>2</td>
<td>7 ≤ N ≤ 12</td>
</tr>
<tr>
<td>3</td>
<td>9 ≤ N ≤ 16</td>
</tr>
<tr>
<td>5</td>
<td>15 ≤ N ≤ 28</td>
</tr>
<tr>
<td>7</td>
<td>21 ≤ N ≤ 40</td>
</tr>
<tr>
<td>9</td>
<td>27 ≤ N ≤ 52</td>
</tr>
</tbody>
</table>

orientationBins

The number of bins to define direction. Supported values are from 2 to 64.

option

Mode of processing. Supported values:

- IPP_FASTN_CIRCLE
- (IPP_FASTN_CIRCLE | IPP_FASTN_ORIENTATION)
- (IPP_FASTN_CIRCLE | IPP_FASTN_SCORE_MODE0)
- (IPP_FASTN_CIRCLE | IPP_FASTN_ORIENTATION | IPP_FASTN_SCORE_MODE0)
- (IPP_FASTN_CIRCLE | IPP_FASTN_SCORE_MODE0 | IPP_FASTN_NMS)
- (IPP_FASTN_CIRCLE | IPP_FASTN_ORIENTATION | IPP_FASTN_SCORE_MODE0 | IPP_FASTN_NMS)

dataType

Data type of the source and destination images. Supported value is ipp8u.

numChannels

Number of channels in the images. Supported value is 1.

pSpecSize

Pointer to the computed size of the specification structure.

Description

This function computes the size of the FastN context structure for the FastN function. The result is stored in pSpecSize.

Use the computed pSpecSize value to allocate memory using the ippMalloc or ippsMalloc functions. The allocated memory can be freed only by the ippFree or ippsFree functions, respectively. For more information about the memory allocation functions, refer to the Support Functions section of the Intel IPP Developer Reference, vol. 1.

For an example on how to use this function, refer to the example provided with the FastN function description.

Return Values

- ippStsNoErr: Indicates no error. Any other value indicates an error.
- ippStsNullPtrErr: Indicates an error when pSpecSize is NULL.
- ippStsSizeErr: Indicates an error when srcSize is less than, or equal to zero.
- ippDataTypeErr: Indicates an error when dataType has an illegal value.
- ippNumChannelsErr: Indicates an error when numChannels has an illegal value.
ippOutOfRangeErr Indicates an error when `orientationBins` or `N` has an illegal value.

ippBadArgErr Indicates an error when `option` or `circleRadius` has an illegal value.

**See Also**

**FastN** Detects corners in an image using the FastN algorithm.

**FastNGetBufferSize**

*Computes the size of the work buffer for the FastN function.*

**Syntax**

```c
IppStatus ippiFastNGetBufferSize(IppiFastNSpec* pSpec, IppiSize dstRoiSize, int* pBufSize);
```

**Include Files**

ippcv.h

**Domain Dependencies**

*Headers:* ippcore.h, ippvm.h, ipps.h,ippi.h

*Libraries:* ippcore.lib, ippvm.lib, ipps.lib,ippi.lib

**Parameters**

- `pSpec` Pointer to the FastN specification structure.
- `dstRoiSize` Size of the destination ROI, in pixels.
- `pBufSize` Pointer to the computed size of the work buffer.

**Description**

This function computes the size of the work buffer for the `FastN` function. The result is stored in `pBufSize`. Use the computed `pBufSize` value to allocate memory using the `ippMalloc` or `ippsMalloc` functions. The allocated memory can be freed only by the `ippFree` or `ippsFree` functions, respectively. For more information about the memory allocation functions, refer to the **Support Functions** section of the *Intel IPP Developer Reference, vol. 1*.

For an example on how to use this function, refer to the example provided with the `FastN` function description.

**Return Values**

- `ippStsNoErr` Indicates no error. Any other value indicates an error.
- `ippStsNullPtrErr` Indicates an error when `pSpecSize` or `pBufSize` is `NULL`.
- `ippStsSizeErr` Indicates an error when `dstRoiSize` is less than, or equal to zero.

**See Also**

**FastN** Detects corners in an image using the FastN algorithm.
**FastNInit**

*Initializes the FastN context structure.*

**Syntax**

```c
IppStatus ippiFastNInit(IppiSize srcSize, int circleRadius, int N, int orientationBins, int option, Ipp32f threshold, IppDataType dataType, int numChannels, IppiFastNSpec* pSpec);
```

**Include Files**

`ippcv.h`

**Domain Dependencies**

**Headers:** `ippcore.h`, `ippvm.h`, `ipps.h`, `ippi.h`

**Libraries:** `ippcore.lib`, `ippvm.lib`, `ipps.lib`, `ippi.lib`

**Parameters**

- `srcSize`  
  Size of the source ROI, in pixels.

- `circleRadius`  
  Radius of the pixel circle. The radius value equal to 1 corresponds to the distance between the closest pixels from common row or column. Supported values are 1, 2, 3, 5, 7, 9.

- `N`  
  Critical number of serial pixels on circle for defining a corner. The ranges are as follows:

<table>
<thead>
<tr>
<th><code>circleRadius</code></th>
<th><code>N</code></th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>5 ≤ N ≤ 8</td>
</tr>
<tr>
<td>2</td>
<td>7 ≤ N ≤ 12</td>
</tr>
<tr>
<td>3</td>
<td>9 ≤ N ≤ 16</td>
</tr>
<tr>
<td>5</td>
<td>15 ≤ N ≤ 28</td>
</tr>
<tr>
<td>7</td>
<td>21 ≤ N ≤ 40</td>
</tr>
<tr>
<td>9</td>
<td>27 ≤ N ≤ 52</td>
</tr>
</tbody>
</table>

- `orientationBins`  
  The number of bins to define direction. Supported values are from 2 to 64.

- `option`  
  Mode of processing. Supported values:
  - `IPP_FASTN_CIRCLE`
  - `(IPP_FASTN_CIRCLE | IPP_FASTN_ORIENTATION)`
  - `(IPP_FASTN_CIRCLE | IPP_FASTN_SCORE_MODE0)`
  - `(IPP_FASTN_CIRCLE | IPP_FASTN_ORIENTATION | IPP_FASTN_SCORE_MODE0)`
  - `(IPP_FASTN_CIRCLE | IPP_FASTN_SCORE_MODE0 | IPP_FASTN_NMS)`
  - `(IPP_FASTN_CIRCLE | IPP_FASTN_ORIENTATION | IPP_FASTN_SCORE_MODE0 | IPP_FASTN_NMS)`

- `threshold`  
  Threshold value to detect critical pixels. The value must be more than, or equal to zero.

- `dataType`  
  Data type of the source and destination images. Supported value is `ipp8u`.

- `numChannels`  
  Number of channels in the images. Supported value is 1.


**Description**
This function initializes the FastN context structure for the `FastN` function. For an example on how to use this function, refer to the example provided with the `FastN` function description.

**Return Values**
- `ippStsNoErr`: Indicates no error. Any other value indicates an error.
- `ippStsNullPtrErr`: Indicates an error when `pSpec` is `NULL`.
- `ippStsSizeErr`: Indicates an error when `srcSize` is less than, or equal to zero.
- `ippDataTypeErr`: Indicates an error when `dataType` has an illegal value.
- `ippNumChannelsErr`: Indicates an error when `numChannels` has an illegal value.
- `ippOutOfRangeErr`: Indicates an error when `orientationBins` or `N` has an illegal value.
- `ippBadArgErr`: Indicates an error when `option` or `circleRadius` has an illegal value.
- `ippThresholdErr`: Indicates an error when `threshold` is negative.

**See Also**
- `FastN` Detects corners in an image using the FastN algorithm.

**Syntax**

```c
IppStatus ippiFastN_8u_C1R(const Ipp8u* pSrc, int srcStep, Ipp8u* pDstCorner, int dstCornerStep, Ipp8u* pDstScore, int dstScoreStep, int* pNumCorner, IppiPoint srcRoiOffset, IppiSize dstRoiSize, IppiFastNSpec* pSpec, Ipp8u* pBuffer);
```

**Include Files**

- `ippcv.h`

**Domain Dependencies**

- **Headers**: `ippcore.h, ippvm.h, ipps.h,ippi.h`
- **Libraries**: `ippcore.lib, ippvm.lib, ipps.lib,ippi.lib`

**Parameters**
- `pSpec`: Pointer to the specification structure.
- `pSrc`: Pointer to the source image.
- `srcStep`: Distance, in bytes, between the starting points of consecutive lines in the source image.
- `pDstCorner`: Pointer to the destination image with corners.
The `ippiFastN` function implements the FastN corner detection algorithm. This function detects corners in the source image, calculates orientation and score of corners.

The figures below show pixels location for different radius values.

**Radius = 1**

```
Px3  Px4  Px5
Px2  pxc  Px6
Px1  Px0  Px7
```

**Radius = 2**

```
   Px5  Px6  Px7
 Px4           Px8
 Px3  pxc  Px9
 Px2
 Px1  Px0  Px11
```

**Radius = 3**

```
Px5  Px6  Px7
 Px4  Px8
 Px3  pxc  Px9
 Px2  Px10
 Px1  Px0  Px11
```
Radius = 5

<table>
<thead>
<tr>
<th></th>
<th>Px7</th>
<th>Px8</th>
<th>Px9</th>
</tr>
</thead>
<tbody>
<tr>
<td>Px6</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Px5</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Px4</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Px3</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Px2</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Px1</td>
<td>Px0</td>
<td>Px15</td>
<td></td>
</tr>
</tbody>
</table>

Radius = 7

<table>
<thead>
<tr>
<th></th>
<th>Px12</th>
<th>Px13</th>
<th>Px14</th>
<th>Px15</th>
<th>Px16</th>
</tr>
</thead>
<tbody>
<tr>
<td>Px11</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Px17</td>
</tr>
<tr>
<td>Px10</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Px18</td>
</tr>
<tr>
<td>Px9</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Px19</td>
</tr>
<tr>
<td>Px8</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Px20</td>
</tr>
<tr>
<td>Px7</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Px21</td>
</tr>
<tr>
<td>Px6</td>
<td></td>
<td>pxc</td>
<td></td>
<td></td>
<td>Px22</td>
</tr>
<tr>
<td>Px5</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Px23</td>
</tr>
<tr>
<td>Px4</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Px24</td>
</tr>
<tr>
<td>Px3</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Px25</td>
</tr>
<tr>
<td>Px2</td>
<td>Px1</td>
<td>Px0</td>
<td>Px27</td>
<td>Px26</td>
<td></td>
</tr>
</tbody>
</table>

Radius = 7
The function defines a pixel as a corner if the value of $N$ consecutive pixels located on the circle with the center at the current pixel is greater (less) than the value of the current - central pixel. Differences between the values of pixels on the circle and the central pixel value must be greater than the threshold value. In this case, the corresponding pixel of the $pDSTCorner$ image is set to the following values in binary format:

- If greater - 01xxxxxx
- If less - 10xxxxxx

If the IPP\_FASTN\_ORIENTATION mode is not set, xxxxxx = 000000, otherwise, xxxxxx = the number of sector (bin) to which the corner directs. The orientationBin parameter defines the number of sectors that is computed from the bottom clockwise. If the source pixel is not a corner, the corresponding pixel of $pDSTCorner$ is set to zero.
If the IPP_FASTN_SCORE_MODE0 mode is set, the corresponding pixel of \texttt{pDstCorner} is set to the corner score. Score is a maximum among minimal differences with \texttt{threshold} calculated for every \texttt{N} consecutive pixels. If the source pixel is not a corner, the corresponding pixel of \texttt{pDstScore} is set to zero.

If the IPP_FASTN_NMS mode is set, the corners that have a corner with the greater score among neighboring pixels are cancelled.

Return Values

\begin{itemize}
\item \texttt{ippStsNoErr} Indicates no error. Any other value indicates an error.
\item \texttt{ippStsNullPtrErr} Indicates an error when:
\begin{itemize}
\item \texttt{pSrc}, \texttt{pDstCorner}, \texttt{pNumCorner}, \texttt{pSpec}, or \texttt{pBuffer} is NULL
\item \texttt{pDstScore} is NULL if \texttt{option} is not equal to IPP_FASTN_SCORE_MODE0
\end{itemize}
\item \texttt{ippStsSizeErr} Indicates an error when \texttt{dstRoiSize} is less than, or equal to zero.
\item \texttt{ippDataTypeErr} Indicates an error when \texttt{dataType} has an illegal value.
\item \texttt{ippNumChannelsErr} Indicates an error when \texttt{numChannels} has an illegal value.
\item \texttt{ippOutOfRangeErr} Indicates an error when \texttt{orientationBins} or \texttt{N} has an illegal value.
\item \texttt{ippBadArgsErr} Indicates an error when \texttt{option} or \texttt{circleRadius} has an illegal value.
\item \texttt{ippThresholdErr} Indicates an error when \texttt{threshold} is negative.
\end{itemize}

Example

To better understand usage of the \texttt{ippiFastN} function, refer to the following example in the examples archive available for download from https://software.intel.com/en-us/ipp-manual-examples:

\begin{verbatim}
FastN.c
\end{verbatim}

See Also

\texttt{FastNGetSize} Computes the size of the FastN context structure.
\texttt{FastNInit} Initializes the FastN context structure.
\texttt{FastNGetSize} Computes the size of the work buffer for the \texttt{FastN} function.
\texttt{FastN2DToVec} Converts corners from two-dimensional image to an array of structures.

\textbf{FastN2DToVec}

\emph{Converts corners from two-dimensional image to an array of structures.}

\textbf{Syntax}

\begin{verbatim}
IppStatus ippiFastN2DToVec_8u(const Ipp8u* pSrcCorner, int srcCornerStep, const Ipp8u* pSrcScore, int srcScoreStep, IppiCornerFastN* pDst, IppiSize srcRoiSize, int maxLen, int* pNumCorners, IppiFastNSpec* pSpec);
\end{verbatim}

\textbf{Include Files}

\texttt{ippcv.h}

\textbf{Domain Dependencies}

\texttt{Headers: ippcore.h, ippvm.h, ipps.h, ippi.h}

\texttt{Libraries: ippcore.lib, ippvm.lib, ipps.lib, ippi.lib}
Parameters

\textbf{pSrcCorner} 
Pointer to the source image with corners.

\textbf{srcCornerStep} 
Distance, in bytes, between the starting points of consecutive lines in the source image with corners.

\textbf{pSrcScore} 
Pointer to the source image with scores.

\textbf{srcScoreStep} 
Distance, in bytes, between the starting points of consecutive lines in the source image with score.

\textbf{pDst} 
Pointer to the destination vector of structures.

\textbf{srcRoiSize} 
Size of the source ROI, in pixels.

\textbf{maxLen} 
Length of the array of structures.

\textbf{pNumCorners} 
Pointer to the number of corners in the source image.

\textbf{pSpec} 
Pointer to the specification structure.

Description

This function converts two-dimensional image with corners to an array of structures. The result is stored in \texttt{pDst}.

For an example on how to use this function, refer to the example provided with the \texttt{FastN} function description.

Return Values

\textbf{ippStsNoErr} 
Indicates no error. Any other value indicates an error.

\textbf{ippStsNullPtrErr} 
Indicates an error when:
\begin{itemize}
  \item \texttt{pSrcCorner}, \texttt{pDst}, \texttt{pNumCorners}, or \texttt{pSpec} is NULL
  \item \texttt{pSrcScore} is NULL if \texttt{option} is not equal to IPP\_FASTN\_SCORE\_MODE0
\end{itemize}

\textbf{ippStsSizeErr} 
Indicates an error when \texttt{srcRoiSize} is less than, or equal to zero.

\textbf{ippContextMatchErr} 
Indicates an error when the specification structure does not match the operation.

\textbf{ippDataTypeErr} 
Indicates an error when \texttt{dataType} has an illegal value.

\textbf{ippNumChannelsErr} 
Indicates an error when \texttt{numChannels} has an illegal value.

\textbf{ippOutOfRangeErr} 
Indicates an error when \texttt{orientationBins} or \texttt{N} has an illegal value.

\textbf{ippBadArgsErr} 
Indicates an error when \texttt{option} or \texttt{circleRadius} has an illegal value.

\textbf{ippThresholdErr} 
Indicates an error when \texttt{threshold} is negative.

See Also

\texttt{FastN} Detects corners in an image using the FastN algorithm.

\texttt{HarrisCornerGetSizeBuffer} 
Calculates the size of the temporary buffer for the \texttt{ippiHarrisCorner} function.
**Syntax**

```c
IppStatus ippiHarrisCornerGetBufferSize(IppSize roiSize, IppiMaskSize filterMask,
Ipp32u avgWndSize, IppDataType dataType, int numChannels, int* pBufferSize);
```

**Include Files**

ippcv.h

**Domain Dependencies**

*Headers:* ippcore.h, ippvm.h, ipps.h, ippi.h

*Libraries:* ippcore.lib, ippvm.lib, ipps.lib, ippi.lib

**Parameters**

- **roiSize**
  Size of the source and destination image ROI in pixels.

- **filterMask**
  Size of the derivative filter aperture. Possible values are `ippMskSize3x3` and `ippMskSize5x5`.

- **avgWndSize**
  Linear size of a neighborhood block for averaging.

- **dataType**
  Data type of the source image. Possible values are `ipp8u` and `ipp32f`.

- **numChannels**
  Number of channels in the images. Possible value is 1.

- **pBufferSize**
  Pointer to the size (in bytes) of the external work buffer.

**Description**

This function calculates the size of the temporary buffer needed for the `HarrisCorner` function.

**Return Values**

- **ippStsNoErr**
  Indicates no error. Any other value indicates an error or a warning.

- **ippStsNullPtrErr**
  Indicates an error when `pBufferSize` is `NULL`.

- **ippStsSizeErr**
  Indicates an error in the following cases:

  - when `roiSize` is less than, or equal to zero
  - when `avgWndSize` is equal to zero

- **ippStsMaskSizeErr**
  Indicates an error when `filterMask` has an illegal value.

- **ippStsDataTypeErr**
  Indicates an error when `dataType` has an illegal value.

- **ippStsNumChannelsErr**
  Indicates an error when `numChannels` has an illegal value.

**See Also**

- **HarrisCorner** Implements Harris corner detection algorithm.

**HarrisCorner**

*Implements Harris corner detection algorithm.*
Syntax

IppStatus ippiHarrisCorner_8u32f_C1R(const Ipp8u* pSrc, int srcStep, Ipp32f* pDst, int dstStep, IppSize roiSize, IppiDifferentialKernel filterType, IppiMaskSize filterMask, Ipp32u avgWndSize, float k, float scale, IppiBorderType borderType, Ipp8u borderValue, Ipp8u* pBuffer);

IppStatus ippiHarrisCorner_32f_C1R(const Ipp32f* pSrc, int srcStep, Ipp32f* pDst, int dstStep, IppSize roiSize, IppiDifferentialKernel filterType, IppiMaskSize filterMask, Ipp32u avgWndSize, float k, float scale, IppiBorderType borderType, Ipp32f borderValue, Ipp8u* pBuffer);

Include Files

ippcv.h

Domain Dependencies

Headers: ippcore.h, ippvm.h, ipps.h,ippi.h
Libraries: ippcore.lib, ippvm.lib, ipps.lib,ippi.lib

Parameters

pSrc  
Pointer to the source image.

dstStep  
Distance, in bytes, between the starting points of consecutive lines in the source image.

pDst  
Pointer to the destination image.

dstStep  
Distance, in bytes, between the starting points of consecutive lines in the destination image.

roiSize  
Size of the source and destination image ROI in pixels.

filterType  
Type of the derivative operator. Possible values are:

ippKernelSobel Sobel filter

ippKernelSchar Scharr filter

ippKernelCentralDiff Central differences operator

filterMask  
Size of the derivative filter aperture. The list of possible values depends on the filterType value:

<table>
<thead>
<tr>
<th>Filter Type</th>
<th>Filter Mask</th>
</tr>
</thead>
<tbody>
<tr>
<td>ippKernelSobel</td>
<td>ippMskSize3x3, ippMskSize5x5</td>
</tr>
<tr>
<td>ippKernelSchar</td>
<td>ippMskSize3x3</td>
</tr>
<tr>
<td>ippKernelCentralDiff</td>
<td>ippMskSize3x3</td>
</tr>
</tbody>
</table>

avgWndSize  
Linear size of a neighborhood block for averaging.

k  
Harris detector free coefficient.

scale  
Destination image scale factor.

borderType  
Type of border. Possible values are:

ippBorderConst Values of all border pixels are set to constant.
ippBorderRepl  Border is replicated from the edge pixels.
ippBorderInMem  Border is obtained from the source image pixels in memory.
ippBorderMirror  Border pixels are mirrored from the source image boundary pixels.

Mixed borders are also supported. They can be obtained by the bitwise operation OR between ippBorderRepl or ippBorderConst and the following flags:
- ippBorderInMemTop
- ippBorderInMemBottom
- ippBorderInMemLeft
- ippBorderInMemRight

Each of these flags means that for the corresponding border the outside pixels of the source image are in memory.

borderValue

Constant value(s) to assign to pixels of the constant border. This parameter is applicable only to the ippBorderConst border type. TheippiHarrisCorner function uses the specified constant value only at the first stage of the algorithm. At the third stage, the function uses zero constant border.

pBuffer

Pointer to the pre-allocated temporary buffer. To calculate the size of the temporary buffer, use the HarrisCornerGetBufferSize function.

Description

This function goes through the following stages to implement the Harris corner detection algorithm:

1. Computes $I_x(x, y)$ and $I_y(x, y)$ gradients for each $(x, y)$ pixel of the image. The function computes gradients using the derivative operator specified by the filterType and filterMask parameters.

2. Computes products of the gradients at each $(x, y)$ pixel of the image:

   \[
   I_{xx}^{(x,y)} = I_x(x, y) \ast I_x(x, y), \quad I_{xy}^{(x,y)} = I_x(x, y) \ast I_y(x, y), \quad I_{yx}^{(x,y)} = I_y(x, y) \ast I_x(x, y), \quad I_{yy}^{(x,y)} = I_y(x, y) \ast I_y(x, y)
   \]

3. Performs averaging of the products of gradients over a rectangular neighborhood block at each pixel of the image. The block size is specified by the avgWndSize value.

   \[
   S_{xx}^{(x,y)} = \sum_{y'} \sum_{x'} I_{xx}^{(x', y')}, \quad S_{xy}^{(x,y)} = \sum_{y'} \sum_{x'} I_{xy}^{(x', y')}, \quad S_{yx}^{(x,y)} = \sum_{y'} \sum_{x'} I_{yx}^{(x', y')}, \quad S_{yy}^{(x,y)} = \sum_{y'} \sum_{x'} I_{yy}^{(x', y')}
   \]

4. Defines 2x2 gradient covariance matrix $H_x(x, y)$ over a rectangular neighborhood block for each $(x, y)$ pixel of the image.

   \[
   H_x(x, y) = \begin{pmatrix}
   S_{xx}^{(x,y)} & S_{xy}^{(x,y)} \\
   S_{yx}^{(x,y)} & S_{yy}^{(x,y)}
   \end{pmatrix}
   \]

5. Computes the corner response at each pixel of the image:
\[ dst(x, y) = \det H^{(x,y)} - k \ast (\text{tr} H^{(x,y)})^2 \]

where

\( k \) is the Harris detector free parameter

The first and third stages of the function algorithm are filtering operations; they use border processing approach that is specified by the \texttt{borderType} parameter.

The \texttt{scale} parameter is applied to the output image.

Before using this function, compute the size of the temporary work buffer using the \texttt{HarrisCornerGetBufferSize} function.

### Return Values

- \texttt{ippStsNoErr} Indicates no error. Any other value indicates an error or a warning.
- \texttt{ippStsNullPtrErr} Indicates an error when \texttt{pSrc}, \texttt{pDst}, or \texttt{pBufferSize} is NULL.
- \texttt{ippStsSizeErr} Indicates an error in the following cases:
  - when \texttt{roiSize} is less than, or equal to zero
  - when \texttt{avgWndSize} is equal to zero
- \texttt{ippStsNotEvenStepErr} Indicates an error when one of the step values is not divisible by 4 for floating point images.
- \texttt{ippStsFilterTypeErr} Indicates an error when \texttt{filterType} has an illegal value.
- \texttt{ippStsMaskSizeErr} Indicates an error when \texttt{filterMask} has an illegal value.
- \texttt{ippStsBorderErr} Indicates an error when \texttt{borderType} has an illegal value.
- \texttt{ippStsStepErr} Indicates an error when \texttt{srcStep} or \texttt{dstStep} has a negative value.
- \texttt{ippStsInplaceModeNotSupportedErr} Indicates an error when \texttt{pSrc} and \texttt{pDst} point to the same image.

### Example

The code example below demonstrates how to use the \texttt{ippiHarrisCorner_8u32f_C1R} and \texttt{ippiHarrisCornerGetBufferSize} functions.

```c
int    bufSize = 0;
Ipp8u* pBuffer = 0;
Ipp32u numChannels = 1;
IppStatus status = ippStsNoErr;
IppiBorderType borderType = ippBorderRepl;
IppiDifferentialKernel filterType = ippFilterSobel;
IppiMaskSize filterMask = ippMskSize5x5;
Ipp32u avgWndSize = 3;
Ipp32f scale = 1.0f;

/* Computes the temporary work buffer size */
status = ippiHarrisCornerGetBufferSize(roiSize, filterMask, avgWndSize, ipp8u, numChannels, &bufSize);

/* Memory allocation */
```
if (status != ippStsNoErr) pBuffer = ippsMalloc_8u(bufSize);

if (pBuffer != NULL)
{
    /* Harris Corner processing */
    status = ippiHarrisCorner_8u32f_C1R(pSrc, srcStep, pDst, dstStep, roiSize, filterType,
           filterMask, avgWndSize, 0.04f,
           scale, borderType, 0, pBuffer);
    ippsFree(pBuffer);
}
...

See Also

HarrisCornerBufferSize Calculates the size of the temporary buffer for the ippiHarrisCorner function.

Canny Edge Detector

This subsection describes a classic edge detector proposed by J.Canny, see [Canny86 ]. The detector uses a grayscale image as an input and outputs a black-and-white image, where non-zero pixels mark detected edges. The algorithm consists of three stages described below.

Stage 1: Differentiation

The image data is differentiated in $x$ and $y$ directions. From the computed $x$ and $y$ gradient components, Canny edge detector functions calculate the magnitude and angle of the gradient vector.

NOTE

The ippiSobel functions perform the first stage and Canny edge detector functions use their output.

Stage 2: Non-Maximum Suppression

With the rate of intensity change found at each point in the image, edges must be placed at the points of maximum values of gradient magnitude. It is preferable to suppress non-maximum points that are perpendicular to the edge direction, rather than parallel to the edge direction, as the edge is strong along an extended contour.

The algorithm starts off with sorting the direction of gradient to one of four sectors shown in the figure below.

Gradient Sectors
The algorithm passes a 3x3 neighborhood across the magnitude array. At each point, the center element of the neighborhood is compared with its two neighbors along the line of the gradient given by the sector value. If the central value is not greater than the neighbors, it is suppressed.

### Stage 3: Edge Thresholding

The Canny operator uses the so-called "hysteresis" thresholding. Most thresholders use a single threshold limit, which means that if the edge values fluctuate above and below this value, the line appears broken. This phenomenon is commonly referred to as "streaking". Hysteresis counters streaking by setting an upper and lower edge value limit. Considering a line segment, if a value lies above the upper threshold limit, it is immediately accepted. If the value lies below the low threshold, it is rejected. Points which lie between the two limits are accepted if they are connected to pixels which are also accepted. The likelihood of streaking is small, since the line segment points must fluctuate above the upper limit and below the lower limit for streaking to occur.

J. Canny recommends the ratio of high to low limit be in the range two or three to one, based on predicted signal-to-noise ratios.

**Example** shows how to use the Intel IPP functions for the Canny edge detector.

### CannyBorderGetSize

*Calculates the size of the temporary buffer for the ippiCannyBorder function.*

**Syntax**

```c
IppStatus ippiCannyBorderGetSize(IppiSize roiSize, IppiDifferentialKernel filterType, IppiMaskSize mask, IppDataType dataType, int* pBufferSize);
```

**Include Files**

`ippcv.h`

**Domain Dependencies**

Headers: `ippcv.h`, `ippvm.h`, `ipps.h`, `ippi.h`

Libraries: `ippcore.lib`, `ippvm.lib`, `ipps.lib`, `ippi.lib`

**Parameters**

- `roiSize`:
  Size of the image ROI in pixels.

- `filterType`:
  Type of the filter to be applied. Possible values are `ippFilterSobel` and `ippFilterScharr`.

- `mask`:
  The size of the mask. Possible values are `ippMskSize3x3` and `ippMskSize5x5`.

- `dataType`:
  Data type of the image. Possible value is `ipp8u`.

- `pBufferSize`:
  Pointer to the variable that returns the size of the temporary buffer.

**Description**

This function calculates the size of the temporary buffer needed for the `CannyBorder` function.
Return Values

- **ippStsNoErr**: Indicates no error. Any other value indicates an error or a warning.
- **ippStsNullPtrErr**: Indicates an error when the `pBufferSize` pointer is `NULL`.
- **ippStsMaskSizeErr**: Indicates an error when `mask` has an illegal value.
- **ippStsDataTypeErr**: Indicates an error when `dataType` has an illegal value.
- **ippStsSizeErr**: Indicates an error when `roiSize` has a field with a zero or negative value.

See Also

**CannyBorder** Implements Canny algorithm for edge detection.

### Syntax

```c
IppStatus ippiCannyBorder_8u_C1R(const Ipp8u* pSrc, int srcStep, Ipp8u* pDst, int dstStep, IppiSize roiSize, IppiDifferentialKernel filterType, IppiMaskSize mask, IppiBorderType borderType, Ipp8u borderValue, Ipp32f lowThresh, Ipp32f highThresh, IppNormType norm, Ipp8u* pBuffer);
```

### Include Files

- `ippcv.h`

### Domain Dependencies

- **Headers**: `ippcore.h`, `ippvm.h`, `ipps.h`, `ippi.h`
- **Libraries**: `ippcore.lib`, `ippvm.lib`, `ipps.lib`, `ippi.lib`

### Parameters

- **pSrc**: Pointer to the source image.
- **srcStep**: Distance, in bytes, between the starting points of consecutive lines in the source image.
- **pDst**: Pointer to the destination image.
- **dstStep**: Distance, in bytes, between the starting points of consecutive lines in the destination image.
- **roiSize**: Size of the source image ROI in pixels.
- **filterType**: Type of the filter to be applied. Possible values are `ippFilterSobel` and `ippFilterScharr`.
- **mask**: The size of the mask. Possible values are `ippMskSize3x3` and `ippMskSize5x5` for `ippFilterSobel`, and `ippMskSize3x3` for `ippFilterScharr`.
- **borderType**: Type of border. Possible values are:
  - `ippBorderConst`: Values of all border pixels are set to constant.
  - `ippBorderRepl`: Border is replicated from the edge pixels.
ippiCannyBorder

Border is obtained from the source image pixels in memory.

Mixed borders are also supported. They can be obtained by the bitwise operation OR between ippBorderRepl and ippBorderInMemTop, ippBorderInMemBottom, ippBorderInMemLeft, ippBorderInMemRight.

borderValue

Constant value to assign to pixels of the constant border. This parameter is applicable only to the ippBorderConst border type.

lowThresh

Lower threshold for edges detection.

highThresh

Upper threshold for edges detection.

norm

Normalization mode. Possible values are ippNormL1 and ippNormL2.

pBuffer

Pointer to the pre-allocated temporary buffer. To calculate the size of the temporary buffer, use the CannyBorderGetSize function.

Description

This function operates with ROI (see Regions of Interest in Intel IPP). This function finds edges in the ROI of the source image with the user-defined border types using the Canny edge detector algorithm. The output image is stored in pDst.

Before using this function, compute the size of the temporary work buffer using the CannyBorderGetSize function.

Return Values

ippStsNoErr

Indicates no error. Any other value indicates an error or a warning.

ippStsNullPtrErr

Indicates an error when one of the specified pointers is NULL.

ippStsMaskSizeErr

Indicates an error when mask has an illegal value.

ippStsSizeErr

Indicates an error when roiSize has a field with a zero or negative value.

ippStsStepErr

Indicates an error when srcStep or dstStep is less than roi.width*<pixelSize>.

ippStsBadArgErr

Indicates an error when lowThresh is negative, or highThresh is less than lowThresh.

ippStsNotEvenStepErr

Indicates an error when one of the step values is not divisible by 2 for 16s images, and by 4 for 32f images.

Example

To better understand usage of the ippiCannyBorder function, refer to the following example in the examples archive available for download from https://software.intel.com/en-us/ipp-manual-examples:

CannyBorder.c

See Also

CannyBorderGetSize Calculates the size of the temporary buffer for the ippiCannyBorder function.

CannyGetSize

Calculates size of temporary buffer for the ippiCanny function.
Syntax
IppStatus ippiCannyGetSize(IppiSize roiSize, int* pBufferSize);
IppStatus ippiCannyGetSize_L(IppiSizeL roi, IppSizeL* bufferSize);

Include Files
ippcv.h

Flavors with the _L suffix: ippcv_l.h

Domain Dependencies
Headers: ippcore.h, ippvm.h, ipps.h,ippi.h
Libraries: ippcore.lib, ippvm.lib, ipps.lib,ippi.lib

Parameters
roiSize, roi Size of the image ROI, in pixels.
pBufferSize, bufferSize Pointer to the computed size of the temporary buffer.

Description
This function calculates the size of a temporary buffer for the ippiCanny function.

Return Values
ippStsNoErr Indicates no error. Any other value indicates an error or a warning.
ippStsNullPtrErr Indicates an error condition if pBufferSize pointer is NULL.
ippStsSizeErr Indicates an error condition if pRoiSize has a field with zero or negative value.

Canny
*Implements Canny algorithm for edge detection.*

Syntax
IppStatus ippiCanny_16s8u_C1R(Ipp16s* pSrcDx, int srcDxStep, Ipp16s* pSrcDy, int srcDyStep, Ipp8u* pDstEdges, int dstEdgeStep, IppiSize roiSize, Ipp32f lowThreshold, Ipp32f highThreshold, Ipp8u* pBuffer);
IppStatus ippiCanny_32f8u_C1R(Ipp32f* pSrcDx, int srcDxStep, Ipp32f* pSrcDy, int srcDyStep, Ipp8u* pDstEdges, int dstEdgeStep, IppiSize roiSize, Ipp32f lowThreshold, Ipp32f highThreshold, Ipp8u* pBuffer);

Platform-aware functions
IppStatus ippiCanny_16s8u_C1R_L(Ipp16s* pSrcDx, IppSizeL srcDxStep, Ipp16s* pSrcDy, IppSizeL srcDyStep, Ipp8u* pDstEdges, IppSizeL dstEdgeStep, IppiSizeL roiSize, Ipp32f lowThreshold, Ipp32f highThreshold, IppNormType norm, Ipp8u* pBuffer);
IppStatus ippiCanny_32f8u_C1R_L(Ipp32f* pSrcDx, IppSizeL srcDxStep, Ipp32f* pSrcDy, IppSizeL srcDyStep, Ipp8u* pDstEdges, IppSizeL dstEdgeStep, IppiSizeL roiSize, Ipp32f lowThreshold, Ipp32f highThreshold, IppNormType norm, Ipp8u* pBuffer);

Include Files
ippcv.h
Flavors with the _L suffix: ippcv_l.h

**Domain Dependencies**

**Headers:** ippcore.h, ippvm.h, ipps.h,ippi.h  
**Libraries:** ippcore.lib, ippvm.lib, ipps.lib,ippi.lib

**Parameters**

- **pSrcDx**
  Pointer to the source image ROI x-derivative.
- **srcDxStep**
  Distance in bytes between starts of consecutive lines in the source image pSrcDx.
- **pSrcDy**
  Pointer to the source image ROI y-derivative.
- **srcDyStep**
  Distance, in bytes, between the starting points of consecutive lines in the source image pSrcDy.
- **pDstEdges**
  Pointer to the output array of the detected edges.
- **dstEdgeStep**
  Distance, in bytes, between the starting points of consecutive lines in the output image.
- **roiSize**
  Size of the source image ROI in pixels.
- **lowThreshold**
  Lower threshold for edges detection.
- **highThreshold**
  Upper threshold for edges detection.
- **norm**
  Normalization type; supported values: ippNormL1 and ippNormL2.
- **pBuffer**
  Pointer to the pre-allocated temporary buffer.

**Description**

This function operates with ROI (see Regions of Interest in Intel IPP). This function finds edges in the source image ROI and stores them into the output image pDstEdges using the Canny algorithm. The function requires a temporary working buffer; its size should be computed previously by calling the function ippiCannyGetSize.

**Return Values**

- **ippStsNoErr**
  Indicates no error. Any other value indicates an error or a warning.
- **ippStsNullPtrErr**
  Indicates an error condition if one of the specified pointers is NULL.
- **ippStsSizeErr**
  Indicates an error condition if pRoiSize has a field with zero or negative value.
- **ippStsStepErr**
  Indicates an error condition if srcDxStep, srcDyStep or dstEdgeStep is less than roi.width*<pixelSize>.
- **ippStsBadArgErr**
  Indicates an error when lowThresh is negative or highThresh is less than lowThresh.
- **ippStsNotEvenStepErr**
  Indicates an error condition if one of the step values is not divisible by 2 for 16s images, and by 4 for 32f images.

**Example**

To better understand usage of the ippiCanny function, refer to the following example in the examples archive available for download from https://software.intel.com/en-us/ipp-manual-examples:
Canny.c

**EigenValsVecsGetBufferSize**

*Calculates size of temporary buffer for the functionippiEigenValsVecs.*

**Syntax**

IppStatus ippiEigenValsVecsGetBufferSize_32f_C1R(IppiSize roiSize, int apertureSize, int avgWindow, int* pBufferSize);

IppStatus ippiEigenValsVecsGetBufferSize_8u32f_C1R(IppiSize roiSize, int apertureSize, int avgWindow, int* pBufferSize);

**Include Files**

ippcv.h

**Domain Dependencies**

**Headers:** ippcore.h, ippvm.h, ipps.h,ippi.h

**Libraries:** ippcore.lib, ippvm.lib, ipps.lib,ippi.lib

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>roiSize</td>
<td>Size of the source image ROI in pixels.</td>
</tr>
<tr>
<td>apertureSize</td>
<td>Size (pixels) of the derivative operator used by the function, possible values are 3 or 5.</td>
</tr>
<tr>
<td>avgWindow</td>
<td>Size of the blurring window in pixels, possible values are 3 or 5.</td>
</tr>
<tr>
<td>pBufferSize</td>
<td>Pointer to the variable that returns the size of the temporary buffer.</td>
</tr>
</tbody>
</table>

**Description**

This function operates with ROI (see Regions of Interest in Intel IPP).

This function calculates the size of a temporary buffer to be used by the functionippiEigenValsVecs.

**Caution**

The parameters apertureSize and avgWindow must be the same for both functionsippiEigenValsVecsGetBufferSize andippiEigenValsVecs.

**Example 14-2** shows how to use the functionippiEigenValsVecsGetBufferSizee.

**Return Values**

<table>
<thead>
<tr>
<th>Return Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ippStsNoErr</td>
<td>Indicates no error. Any other value indicates an error or a warning.</td>
</tr>
<tr>
<td>ippStsNullPtrErr</td>
<td>Indicates an error condition if one of the specified pointers is NULL.</td>
</tr>
<tr>
<td>ippStsSizeErr</td>
<td>Indicates an error condition if roiWidth has zero or negative value, or if apertureSize or avgWindow has an illegal value.</td>
</tr>
</tbody>
</table>
**EigenValsVecsBorder**

*Calculates eigen values and eigen vectors of image blocks for corner detection.*

**Syntax**

```c
IppStatus ippiEigenValsVecsBorder_8u32f_C1R(const Ipp8u* pSrc, int srcStep, Ipp32f* pEigenVV, int eigStep, IppiSize roiSize, IppiKernelType kernType, int apertureSize, int avgWindow, IppiBorderType borderType, Ipp8u borderValue, Ipp8u* pBuffer);

IppStatus ippiEigenValsVecsBorder_32f_C1R(const Ipp32f* pSrc, int srcStep, Ipp32f* pEigenVV, int eigStep, IppiSize roiSize, IppiKernelType kernType, int apertureSize, int avgWindow, IppiBorderType borderType, Ipp32f borderValue, Ipp8u* pBuffer);
```

**Include Files**

ippcv.h

**Domain Dependencies**

*Headers:* ippcore.h, ippvm.h, ipps.h,ippi.h

*Libraries:* ippcore.lib, ippvm.lib, ipps.lib,ippi.lib

**Parameters**

- **pSrc**: Pointer to the source image ROI.
- **srcStep**: Distance, in bytes, between the starting points of consecutive lines in the source image.
- **pEigenVV**: Image to store the results.
- **eigStep**: Distance, in bytes, between the starting points of consecutive lines in the output image.
- **roiSize**: Size of the source image ROI, in pixels.
- **kernType**: Specifies the type of kernel used to compute derivatives, possible values are:
  - ippKernelSobel: Sobel kernel 3x3 or 5x5
  - ippKernelSobelNeg: Negative Sobel kernel 3x3 or 5x5
  - ippKernelScharr: Scharr kernel 3x3
- **apertureSize**: Size of the derivative operator in pixels, possible values are 3 or 5.
- **border**: Type of image border. Possible values:
  - ippBorderConst: Values of all border pixels are set to a constant.
  - ippBorderRepl: Border is replicated from the edge pixels.
- **borderValue**: Constant value to assign to pixels of the constant border. This parameter is applicable only to the ippBorderConst border type.
- **avgWindow**: Size of the blurring window in pixels, possible values are 3 or 5.
- **pBuffer**: Pointer to the temporary buffer.
Description
This function operates with ROI (see Regions of Interest in Intel IPP).

This function takes a block around the pixel and computes the first derivatives $D_x$ and $D_y$. This operation is performed for every pixel of the image using either Sobel or Scharr kernel in accordance with the `kernType` parameter. The `apertureSize` parameter specifies the size of the Sobel kernel. If this parameter is set to 3, the function used 3x3 kernel, if it is set to 5, the function uses 5x5 kernel. Only 3x3 size is available for the Scharr kernel, therefore the parameter `apertureSize` must be set to 3 if the Scharr kernel is used.

**Caution**
If the parameter `apertureSize` is set to 5 for operation with the Scharr kernel, the function returns error status.

The function computes eigen values and vectors of the following matrix:

$$
\begin{bmatrix}
\sum D_x^2 & \sum D_x D_y \\
\sum D_x D_y & \sum D_y^2 
\end{bmatrix}
$$

The summation is performed over the full block with averaging over the blurring window with size `avgWindow`.

The image `eigenVV` has the following format. For every pixel of the source image it contains six floating-point values - $\lambda_1$, $\lambda_2$, $x_1$, $y_1$, $x_2$, $y_2$. These values are defined as follows:

- $\lambda_1$, $\lambda_2$ - Eigen values of the above matrix ($\lambda_1 \geq \lambda_2 \geq 0$).
- $x_1$, $y_1$ - Coordinates of the normalized eigen vector corresponding to $\lambda_1$.
- $x_2$, $y_2$ - Coordinates of the normalized eigen vector corresponding to $\lambda_2$.

In case of a singular matrix or when one eigen value is much smaller than the second one, all these six values are set to 0.

The function requires a temporary work buffer. Before using this function, compute the size of the buffer using the `ippiEigenValsVecsGetBufferSize` function.

**Caution**
The parameters `apertureSize` and `avgWindow` must be the same for both functions `ippiEigenValsVecsGetBufferSize` and `ippiEigenValsVecsBorder`.

Return Values
- `ippStsNoErr` - Indicates no error. Any other value indicates an error or a warning.
- `ippStsNullPtrErr` - Indicates an error condition if one of the specified pointers is NULL.
- `ippStsSizeErr` - Indicates an error condition if `pRoiSize` has a field with zero or negative value, or if `apertureSize` or `avgWindow` has an illegal value; or if `kernType` has a wrong value.
Indicates an error condition if `srcStep` is less than `roiSize.width*pixelSize`, or `eigStep` is less than `roiSize.width*sizeof(Ipp32f)*6`.

Indicates an error condition if steps for floating-point images are not divisible by 4.

Indicates an error if `borderType` has an illegal value.

See Also

Regions of Interest in Intel IPP

EigenValsVecsGetBufferSize Calculates size of temporary buffer for the function ippiEigenValsVecs.

**EigenValsVecs**

*Calculates eigen values and eigen vectors of image blocks for corner detection.*

**Syntax**

```c
IppStatus ippiEigenValsVecs_8u32f_C1R(const Ipp8u* pSrc, int srcStep, Ipp32f* pEigenVV, int eigStep, IppSize roiSize, IppiKernelType kernType, int apertureSize, int avgWindow, Ipp8u* pBuffer);
```

```c
IppStatus ippiEigenValsVecs_32f_C1R(const Ipp32f* pSrc, int srcStep, Ipp32f* pEigenVV, int eigStep, IppSize roiSize, IppiKernelType kernType, int apertureSize, int avgWindow, Ipp8u* pBuffer);
```

**Include Files**

ippcv.h

**Domain Dependencies**

Headers: ippcore.h, ippvm.h, ipps.h,ippi.h

Libraries: ippcore.lib, ippvm.lib, ipps.lib,ippi.lib

**Parameters**

- **pSrc**
  - Pointer to the source image ROI.

- **srcStep**
  - Distance, in bytes, between the starting points of consecutive lines in the source image.
  - Image to store the results.

- **pEigenVV**
  - Distance, in bytes, between the starting points of consecutive lines in the output image.

- **eigStep**
  - Size of the source image ROI, in pixels.

- **roiSize**
  - Specifies the type of kernel used to compute derivatives, possible values are:
    - `ippKernelSobel` Sobel kernel 3x3 or 5x5
    - `ippKernelSobelNeg` Negative Sobel kernel 3x3 or 5x5
    - `ippKernelScharr` Scharr kernel 3x3

- **kernType**
  - Size of the derivative operator in pixels, possible values are 3 or 5.
avgWindow
Size of the blurring window in pixels, possible values are 3 or 5.

pBuffer
Pointer to the temporary buffer.

Description
This function operates with ROI (see Regions of Interest in Intel IPP).
This function takes a block around the pixel and computes the first derivatives \( D_x \) and \( D_y \). This operation is performed for every pixel of the image using either Sobel or Scharr kernel in accordance with the kernType parameter. The apertureSize parameter specifies the size of the Sobel kernel. If this parameter is set to 3, the function used 3x3 kernel, if it is set to 5, the function uses 5x5 kernel. Only 3x3 size is available for the Scharr kernel, therefore the parameter apertureSize must be set to 3 if the Scharr kernel is used.

Caution
If the parameter apertureSize is set to 5 for operation with the Scharr kernel, the function returns error status.

The function computes eigen values and vectors of the following matrix:

\[
\begin{bmatrix}
\sum D_x^2 & \sum D_x D_y \\
\sum D_x D_y & \sum D_y^2 \\
\end{bmatrix}
\]

The summation is performed over the full block with averaging over the blurring window with size avgWindow.

The image eigenVV has the following format. For every pixel of the source image it contains six floating-point values - \( \lambda_1, \lambda_2, x_1, y_1, x_2, y_2 \). These values are defined as follows:

\( \lambda_1, \lambda_2 \) Eigen values of the above matrix (\( \lambda_1 \geq \lambda_2 \geq 0 \)).

\( x_1, y_1 \) Coordinates of the normalized eigen vector corresponding to \( \lambda_1 \).

\( x_2, y_2 \) Coordinates of the normalized eigen vector corresponding to \( \lambda_2 \).

In case of a singular matrix or when one eigen value is much smaller than the second one, all these six values are set to 0.

The function requires a temporary working buffer; its size should be computed previously by calling the function ippiEigenValsVecsGetBufferSize.

Caution
The parameters apertureSize and avgWindow must be the same for both functions ippiEigenValsVecsGetBufferSize and ippiEigenValsVecs.

Return Values

ippiStsNoErr
Indicates no error. Any other value indicates an error or a warning.

ippiStsNullPtrErr
Indicates an error condition if one of the specified pointers is NULL.
Indicates an error condition if \textit{pRoiSize} has a field with zero or negative value, or if \textit{apertureSize} or \textit{avgWindow} has an illegal value; or if \textit{kernType} has wrong value.

Indicates an error condition if \textit{srcStep} is less than \textit{roiSize.width*<pixelSize>}, or \textit{eigStep} is less than \textit{roiSize.width*sizeof(Ipp32f)}*6.

Indicates an error condition if steps for floating-point images are not divisible by 4.

Example
To better understand usage of the \texttt{ippiEigenValsVecs} function, refer to the following example in the examples archive available for download from https://software.intel.com/en-us/ipp-manual-examples:

\texttt{EigenValsVecs.c}

\textbf{MinEigenValGetBufferSize}

\textit{Calculates size of temporary buffer for the function \texttt{ippiMinEigenVal}.}

\textbf{Syntax}

\begin{verbatim}
IppStatus ippiMinEigenValGetBufferSize_32f_C1R(IppSize roiSize, int apertureSize, int avgWindow, int* pBufferSize);
IppStatus ippiMinEigenValGetBufferSize_8u32f_C1R(IppSize roiSize, int apertureSize, int avgWindow, int* pBufferSize);
\end{verbatim}

\textbf{Include Files}

\texttt{ippcv.h}

\textbf{Domain Dependencies}

\textbf{Headers:} ippcore.h, ippm.h, ipps.h,ippi.h
\textbf{Libraries:} ippcore.lib, ippm.lib, ipps.lib,ippi.lib

\textbf{Parameters}

\texttt{roiSize} \quad Size of the source image ROI in pixels.

\texttt{apertureSize} \quad Size (in pixels) of the derivative operator used by the function, possible values are 3 or 5.

\texttt{avgWindow} \quad Size of the blurring window in pixels, possible values are 3 or 5.

\texttt{pBufferSize} \quad Pointer to the variable that returns the size of the temporary buffer.

\textbf{Description}

This function operates with ROI (see Regions of Interest in Intel IPP). This function calculates the size of a temporary buffer to be used by the function \texttt{ippiMinEigenVal}.

\textbf{Caution}

The parameters \texttt{apertureSize} and \texttt{avgWindow} must be the same for both functions \texttt{ippiMinEigenValGetBufferSize} and \texttt{ippiMinEigenVal}. 
Return Values

ippStsNoErr  
Indicates no error. Any other value indicates an error or a warning.

ippStsNullPtrErr  
Indicates an error condition if one of the specified pointers is NULL.

ippStsSizeErr  
Indicates an error condition if roiWidth has a zero or negative value, or if apertureSize or avgWindow has an illegal value.

MinEigenValBorder
Calculates the minimal eigen value of image blocks for corner detection.

Syntax

IppStatus ippiMinEigenValBorder_8u32f_C1R(const Ipp8u* pSrc, int srcStep, Ipp32f* pMinEigenVal, int minValStep, IppiSize roiSize, IppKernelType kernType, int apertureSize, int avgWindow, IppiBorderType borderType, Ipp8u borderValue, Ipp8u* pBuffer);

IppStatus ippiMinEigenValBorder_32f_C1R(const Ipp32f* pSrc, int srcStep, Ipp32f* pMinEigenVal, int minValStep, IppiSize roiSize, IppKernelType kernType, int apertureSize, int avgWindow, IppiBorderType borderType, Ipp32f borderValue, Ipp8u* pBuffer);

Include Files

ippcv.h

Domain Dependencies

Headers: ippcore.h, ippvm.h, ipps.h,ippi.h
Libraries: ippcore.lib, ippvm.lib, ipps.lib,ippi.lib

Parameters

pSrc  
Pointer to the source image ROI.

srcStep  
Distance, in bytes, between the starting points of consecutive lines in the source image.

pMinEigenVal  
Pointer to the image that is filled with the minimal eigen values.

minValStep  
Distance, in bytes, between the starting points of consecutive lines in the output image.

roiSize  
Size of the source image ROI in pixels.

kernType  
Specifies the type of kernel used to compute derivatives, possible values are:

ippKernelSobel  
Sobel kernel 3x3 or 5x5

ippKernelSobelNeg  
Negative Sobel kernel 3x3 or 5x5

ippKernelSchar  
Scharr kernel 3x3

apertureSize  
Size of the derivative operator, in pixels; possible values are 3 or 5.

avgWindow  
Size of the averaging window, in pixels; possible values are 3 or 5.
**borderType**
Type of image border. Possible values:
- ippBorderConst: Values of all border pixels are set to a constant.
- ippBorderRepl: Border is replicated from the edge pixels.
- ippBorderMirror: Border pixels are mirrored from the source image boundary pixels.

**borderValue**
Constant value to assign to pixels of the constant border. This parameter is applicable only to the ippBorderConst border type.

**pBuffer**
Pointer to the temporary buffer.

**Description**
This function operates with ROI (see Regions of Interest in Intel IPP). This function takes a block around the pixel and computes the first derivatives $D_x$ and $D_y$. This operation is performed for every pixel of the image using either Sobel or Scharr kernel in accordance with the kernType parameter. The apertureSize parameter specifies the size of the Sobel kernel. If apertureSize is set to 3, the function uses 3x3 kernel, if it is set to 5, the function uses 5x5 kernel. Only 3x3 size is available for the Scharr kernel, therefore the parameter apertureSize must be set to 3 if the Scharr kernel is used.

**Caution**
If the parameter apertureSize is set to 5 for operation with the Scharr kernel, the function returns error status.

The function computes the minimal eigen value $\lambda$ ($\lambda \geq 0$) of the following matrix:

$$\begin{bmatrix}
\sum D_x^2 & \sum D_x D_y \\
\sum D_x D_y & \sum D_y^2
\end{bmatrix}$$

The summation is performed over the full block with averaging over the blurring window with size avgWindow.

The function requires a temporary work buffer. Before using this function, compute the size of the work buffer using theippiMinEigenValGetBufferSize function.

**Caution**
The parameters apertureSize and avgWindow must be the same for both functions ippiMinEigenValGetBufferSize and ippiMinEigenValBorder.

**Return Values**
- ippStsNo Err: Indicates no error. Any other value indicates an error or a warning.
- ippStsNullPtrErr: Indicates an error condition if one of the specified pointers is NULL.
- ippStsSizeErr: Indicates an error condition if pRoiSize has a field with zero or negative value, or if apertureSize or avgWindow has an illegal value; or if kernType has wrong value.
ippiMinEigenVal

**Calculates the minimal eigen value of image blocks for corner detection.**

**Syntax**

```c
IppStatus ippiMinEigenVal_8u32f_C1R(const Ipp8u* pSrc, int srcStep, Ipp32f* pMinEigenVal, int minValStep, IppiSize roiSize, IppiKernelType kernType, int apertureSize, int avgWindow, Ipp8u* pBuffer);
IppStatus ippiMinEigenVal_32f_C1R(const Ipp32f* pSrc, int srcStep, Ipp32f* pMinEigenVal, int minValStep, IppiSize roiSize, IppiKernelType kernType, int apertureSize, int avgWindow, Ipp8u* pBuffer);
```

**Include Files**

ippcv.h

**Domain Dependencies**

Headers: ippcore.h, ippvm.h, ipps.h, ippi.h
Libraries: ippcore.lib, ippvm.lib, ipps.lib, ippi.lib

**Parameters**

- **pSrc**: Pointer to the source image ROI.
- **srcStep**: Distance in bytes between starts of consecutive lines in the source image.
- **pMinEigenVal**: Pointer to the image that is filled with the minimal eigen values.
- **minValStep**: Distance in bytes between starts of consecutive lines in the output image.
- **roiSize**: Size of the source image ROI in pixels.
- **kernType**: Specifies the type of kernel used to compute derivatives, possible values are:
  - ippKernelSobel: Sobel kernel 3x3 or 5x5
  - ippKernelSobelNeg: Negative Sobel kernel 3x3 or 5x5
  - ippKernelScharr: Scharr kernel 3x3
- **apertureSize**: Size of the derivative operator in pixels, possible values are 3 or 5.
- **avgWindow**: Size of the averaging window in pixels, possible values are 3 or 5.
**pBuffer**

Pointer to the temporary buffer.

**Description**

This function operates with ROI (see Regions of Interest in Intel IPP). This function takes a block around the pixel and computes the first derivatives $D_x$ and $D_y$. This operation is performed for every pixel of the image using either Sobel or Scharr kernel in accordance with the `kernType` parameter. The size of the Sobel kernel may be specified the parameter `apertureSize`. If this parameter is set to 3, the function used 3x3 kernel, if it is set to 5, the function uses 5x5 kernel. Only 3x3 size is available for the Scharr kernel, therefore the parameter `apertureSize` must be set to 3 if the Scharr kernel is used.

**Caution**

If the parameter `apertureSize` is set to 5 for operation with the Scharr kernel, the function returns error status.

Then, the function computes the minimal eigen value $\lambda$ ($\lambda \geq 0$) of the following matrix:

$$
\begin{bmatrix}
\sum D_x^2 & \sum D_x D_y \\
\sum D_x D_y & \sum D_y^2
\end{bmatrix}
$$

The summation is performed over the full block with averaging over the blurring window with size `avgWindow`.

The function requires a temporary working buffer; its size should be computed previously by calling the function `ippiMinEigenValGetBufferSize`.

**Caution**

The parameters `apertureSize` and `avgWindow` must be the same for both functions `ippiMinEigenValGetBufferSize` and `ippiMinEigenVal`.

**Return Values**

- `ippiStsNoErr` Indicates no error. Any other value indicates an error or a warning.
- `ippiStsNullPtrErr` Indicates an error condition if one of the specified pointers is NULL.
- `ippiStsSizeErr` Indicates an error condition if `pRoiSize` has a field with zero or negative value, or if `apertureSize` or `avgWindow` has an illegal value; or if `kernType` has wrong value.
- `ippiStsStepErr` Indicates an error condition if `srcStep` is less than `roiSize.width*<pixelSize>`, or `eigenvvStep` is less than `roiSize.width*sizeof(Ipp32f)`.
- `ippiStsNotEvenStepErr` Indicates an error condition if steps for floating-point images are not divisible by 4.

**Histogram of Oriented Gradients (HOG) Descriptor**

Histogram of oriented gradients (HOG) is a feature descriptor used to detect objects in computer vision and image processing. The HOG descriptor technique counts occurrences of gradient orientation in localized portions of an image - detection window, or region of interest (ROI).
Implementation of the HOG descriptor algorithm is as follows:

1. Divide the image into small connected regions called cells, and for each cell compute a histogram of gradient directions or edge orientations for the pixels within the cell.
2. Discretize each cell into angular bins according to the gradient orientation.
3. Each cell's pixel contributes weighted gradient to its corresponding angular bin.
4. Groups of adjacent cells are considered as spatial regions called blocks. The grouping of cells into a block is the basis for grouping and normalization of histograms.
5. Normalized group of histograms represents the block histogram. The set of these block histograms represents the descriptor.

The following figure demonstrates the algorithm implementation scheme:

Computation of the HOG descriptor requires the following basic configuration parameters:

- Masks to compute derivatives and gradients
- Geometry of splitting an image into cells and grouping cells into a block
- Block overlapping
- Normalization parameters

According to [Dalal05] the recommended values for the HOG parameters are:

- 1D centered derivative mask [-1, 0, +1]
- Detection window size is 64x128
- Cell size is 8x8
- Block size is 16x16 (2x2 cells)

Intel® IPP implementation does not assume any default fixed set of parameters values. The `IppiHOGConfig` structure defines HOG parameters used in Intel IPP functions.

There are some limitations to the values of basic configuration parameters:

```c
#define IPP_HOG_MAX_CELL   (16) /* max size of cell */
#define IPP_HOG_MAX_BLOCK  (64) /* max size of block */
#define IPP_HOG_MAX_BINS   (16) /* max number of bins */
```

See Also

- Structures and Enumerators
HOGGetSize
*Computes the size of the HOG context structure.*

**Syntax**

IppStatus ippiHOGGetSize(const IppiHOGConfig* pConfig, int* pHOGSpecSize);

**Include Files**

ippi.h

**Domain Dependencies**

Headers: ippcore.h, ippvm.h, ipps.h
Libraries: ippcore.lib, ippvm.lib, ipps.lib

**Parameters**

- `pConfig`  
  Pointer to the HOG context structure.
- `pHOGSpecSize`  
  Pointer to the size of the HOG context structure, in bytes.

**Description**

This function checks the parameters of the HOG configuration and computes the size, in bytes, of the HOG context structure `pHOGSpecSize`.

For an example on how to use this function, refer to the example provided with the HOG function description.

**Return Values**

- `ippStsNoErr`  
  Indicates no error. Any other value indicates an error.
- `ippStsNullPtrErr`  
  Indicates an error when one of the specified pointers is `NULL`.
- `ippStsSizeErr`  
  Indicates an error when one of the `winSize` fields in the `pConfig` parameter has a zero or negative value.
- `ippStsNotSupportedModeErr`  
  Indicates an error in HOG configuration:
  - `cellSize` is less than 2, or more than `IPP_HOG_MAX_CELL`
  - `cellSize` is more than `blockSize`, or `blockSize` is more than `IPP_HOG_MAX_BLOCK`
  - `blockSize` is not a multiple of `cellSize`
  - Block does not have 2x2 cell geometry
  - `blockStride` is not a multiple of `cellSize`
  - Detection window size is not a multiple of `blockStride`
  - `nbins` is less than 2, or more than `IPP_HOG_MAX_BINS`
  - `sigma` or `threshold` value is less than, or equal to zero

**See Also**

HOG  Computes the HOG descriptor.

HOGInit  
*Initializes the HOG context structure.*

**Syntax**

IppStatus ippiHOGInit(const IppiHOGConfig* pConfig, IppiHOGSpec* pHOGSpec);
Include Files
ippi.h

Domain Dependencies
Headers: ippicore.h, ippvm.h, ipps.h
Libraries: ippcore.lib, ippvm.lib, ipps.lib

Parameters

\textit{pConfig} \hspace{2cm} \text{Pointer to the HOG context structure.}
\textit{pHOGSpec} \hspace{2cm} \text{Pointer to the HOG context structure.}

Description
This function checks the parameters of the HOG configuration and initializes the HOG context structure.
For an example on how to use this function, refer to the example provided with the HOG function description.

Return Values

\textit{ippStsNoErr} \hspace{2cm} \text{Indicates no error. Any other value indicates an error or a warning.}
\textit{ippStsNullPtrErr} \hspace{2cm} \text{Indicates an error when one of the specified pointers is NULL.}
\textit{ippStsSizeErr} \hspace{2cm} \text{Indicates an error when one of the \textit{winSize} fields in the \textit{pConfig} parameter has a zero or negative value.}
\textit{ippStsNotSupportedModErr} \hspace{2cm} \text{Indicates an error in HOG configuration:}
- \textit{cellSize} is less than 2, or more than IPP_HOG_MAX_CELL
- \textit{cellSize} is more than \textit{blockSize}, or \textit{blockSize} is more than IPP_HOG_MAX_BLOCK
- \textit{blockSize} is not a multiple of \textit{cellSize}
- Block does not have 2x2 cell geometry
- \textit{blockStride} is not a multiple of \textit{cellSize}
- Detection window size is not a multiple of \textit{blockStride}
- \textit{nbins} is less than 2, or more than IPP_HOG_MAX_BINS
- \textit{sigma} or \textit{threshold} value is less than, or equal to zero

See Also
\textbf{HOG} Computes the HOG descriptor.

\textbf{HOGGetBufferSize}
\textit{Computed the size of the work buffer for the ippiHOG function.}

Syntax
IppStatus ippiHOGGetBufferSize(const IppiHOGSpec* \textit{pHOGSpec}, IppiSize \textit{roiSize}, int* \textit{pBufferSize});

Include Files
ippi.h
**Domain Dependencies**

**Headers:** ippcore.h, ippvm.h, ipps.h  
**Libraries:** ippcore.lib, ippvm.lib, ipps.lib

**Parameters**

- `pHOGSpec`  
  Pointer to the HOG context structure.
- `roiSize`  
  Maximum size of the source image ROI, in pixels.
- `pBufferSize`  
  Pointer to the size of the work buffer, in bytes.

**Description**

This function computes the size of the work buffer for the HOG function. For an example on how to use this function, refer to the example provided with the HOG function description.

**Return Values**

- `ippStsNoErr`  
  Indicates no error. Any other value indicates an error.
- `ippStsNullPtrErr`  
  Indicates an error when one of the specified pointers is NULL.
- `ippStsContextmatchErr`  
  Indicates an error when the context parameter does not match the operation.

**See Also**

- **HOG** Computes the HOG descriptor.  
- **HOGGetDescriptorSize**  
  Computes the size of the HOG descriptor.

**Syntax**

```c
IppStatus ippiHOGGetDescriptorSize(const IppiHOGSpec* pHOGSpec, int* pBufferSize);
```

**Include Files**

ippi.h

**Domain Dependencies**

**Headers:** ippcore.h, ippvm.h, ipps.h  
**Libraries:** ippcore.lib, ippvm.lib, ipps.lib

**Parameters**

- `pHOGSpec`  
  Pointer to the HOG context structure.
- `pWinDescriptorSize`  
  Pointer to the size of the descriptor window, in bytes.

**Description**

This function computes the size of the buffer for a single detection window. If the subsequent call(s) of the HOG function target processing of multiple detection windows, the size of the buffer must be increased proportionally.

For an example on how to use this function, refer to the example provided with the HOG function description.
Return Values

ippStsNoErr Indicates no error. Any other value indicates an error.
ippStsNullPtrErr Indicates an error when one of the specified pointers is NULL.
ippStsContextmatchErr Indicates an error when the context parameter does not match the operation.

See Also
HOG Computes the HOG descriptor.

HOG
Computes the HOG descriptor.

Syntax

IppStatus ippiHOG_<mod>(const Ipp<srcDatatype>* pSrc, int srcStep, IippiSize roiSize, const IppiPoint* pLocation, int nLocations, Ipp32f* pDst, const IppiHOGSpec* pHOGSpec, IppiBorderType borderID, Ipp<srcDatatype> borderValue, Ipp8u* pBuffer);

Supported values for mod:

  8u32f_C1R  16u32f_C1R  16s32f_C1R  32f_C1R

IppStatus ippiHOG_<mod>(const Ipp<srcDatatype>* pSrc, int srcStep, IippiSize roiSize, const IppiPoint* pLocation, int nLocations, Ipp32f* pDst, const IppiHOGSpec* pHOGCtx, IppiBorderType borderID, Ipp<srcDatatype> borderValue[3], Ipp8u* pBuffer);

Supported values for mod:

  8u32f_C3R  16u32f_C3R  16s32f_C3R  32f_C3R

Include Files

ippi.h

Domain Dependencies

Headers: ippcore.h, ippvm.h, ipps.h
Libraries: ippcore.lib, ippvm.lib, ipps.lib

Parameters

pSrc Pointer to the source image ROI.
srcStep Distance, in bytes, between the starting points of consecutive lines in the source image.
roiSize Size of the source image ROI, in pixels.
pLocation Pointer to the array with detection window locations.
nLocations Number of locations.
pDst Pointer to the HOG descriptor.
pHOGCtx, pHOGSpec Pointer to the HOG context/specification structure.
borderID Type of border. Possible values are:
Values of all border pixels are set to a constant.

Border is replicated from the edge pixels.

Border is obtained from the source image pixels in memory.

Border pixels are mirrored from the source image boundary pixels.

Mixed borders are also supported. They can be obtained by the bitwise operation OR between ippBorderRepl and ippBorderInMemTop, ippBorderInMemBottom, ippBorderInMemLeft, ippBorderInMemRight.

Constant value to assign to pixels of the constant border. This parameter is applicable only to the ippBorderConst border type.

Pointer to the work buffer.

Description

This function computes the HOG descriptor over defined locations of the detection window. Flavors with the C1 suffix operate on one-channel (gray) images, and C3 flavors operate on color images.

Before using this function, compute the size of the context structure, work buffer, and descriptor using the HOGGetSize, HOGGetBufferSize, and HOGGetDescriptorSize functions, respectively. To initialize the HOG context structure, use the HOGInit function.

Return Values

Indicates no error. Any other value indicates an error or a warning.

Indicates an error when one of the specified pointers is NULL.

Indicates an error when the context parameter does not match the operation.

Indicates an error when srcStep is less than, or equal to zero.

Indicates an error when srcStep is not divisible by input data type size.

Indicates an error when borderID has an illegal value.

Indicates an error when roiSize is less than, or equal to zero.

Example

To better understand usage of the ippiHOG function, refer to the following example in the examples archive available for download from https://software.intel.com/en-us/ipp-manual-examples:

HOG.c

See Also

HOGGetSize Computes the size of the HOG context structure.

HOGGetBufferSize Computes the size of the work buffer for the ippiHOG function.

HOGGetDescriptorSize Computes the size of the HOG descriptor.

HOGInit Initializes the HOG context structure.
Hough Transform

The Hough transform is a general technique that allows to detect the flat curves in binary images [Gon93]. The current version of Intel IPP implements the following:

- Detection of the straight lines that are defined by the parametric equation:
  \[ r = x^*\cos(\theta) + y^*\sin(\theta), \] where \( r \) and \( \theta \) are the length and angle from the origin of a normal to the line, respectively.
- Detection of lines using the probabilistic Hough transform algorithm [Matas00].

HoughLineGetSize

Computes the size of the working buffer for the straight lines detection.

Syntax

\[
IppStatus ippiHoughLineGetSize_8u_C1R(IppiSize roiSize, IppPointPolar delta, int maxLineCount, int* pBufSize);
\]

Include Files

ippcv.h

Domain Dependencies

Headers: ippcore.h, ippvm.h, ipps.h,ippi.h
Libraries: ippcore.lib, ippvm.lib, ipps.lib,ippi.lib

Parameters

roiSize
  Size of the source image ROI, in pixels.

delta
  Step of discretization (delta.rho - radial increment, delta.theta - angular increment).

maxLineCount
  Number of elements of the destination buffer.

pBufSize
  Pointer to the size of the working buffer.

Description

This function operates with ROI (see Regions of Interest in Intel IPP).

This function computes the size of the temporary working buffer that is required for the function ippiHoughLine.

Return Values

ippStsNoErr
  Indicates no error.

ippStsNullPtrErr
  Indicates an error if pBufSize is NULL.

ippStsSizeErr
  Indicates an error condition if roiSize or delta has a field with zero or negative value.
HoughLine
Detects straight lines in the source image.

Syntax
IppStatus ippiHoughLine_8u32f_C1R(const Ipp8u* pSrc, int srcStep, IppiSize roiSize, IppPointPolar delta, int threshold, IppPointPolar* pLine, int maxLineCount, int* pLineCount, Ipp8u* pBuffer);

Include Files
ippcv.h

Domain Dependencies
Headers: ippcore.h, ippvm.h, ipps.h, ippi.h
Libraries: ippcore.lib, ippvm.lib, ipps.lib, ippi.lib

Parameters
pSrc Pointer to the source image ROI.
srcStep Distance in bytes between starts of consecutive lines in the source image.
roiSize Size of source image ROI in pixels.
delta Step of discretization (delta.rho - radial increment, delta.theta - angular increment).
threshold Minimum number of points that are required to detect the line.
pLine Pointer to the destination buffer for lines.
pLineCount Number of detected lines. If the value is more than maxLineCount, the function returns the ippStsDstSizeLessExpected status.
maxLineCount Number of elements of the destination buffer.
pBuffer Pointer to the working buffer.

Description
This function operates with ROI (see Regions of Interest in Intel IPP).
In the binarised source image pSrc, the function performs detection of the straight line defined by the equation given at the beginning of section Hough Transform. The level of discretization delta is specified as the input parameters. The performance and effectiveness of the function is strongly depends on this parameter. The function requires the external working buffer pBuffer, which size should be computed previously using the function ippiHoughLineGetSize.

Caution
The value of the parameter delta must not be greater than the value of the parameter delta set when the size of the working buffer is computed.
Return Values

- ippStsNoErr: Indicates no error.
- ippStsNullPtrErr: Indicates an error if one of the specified pointers is NULL.
- ippStsSizeErr: Indicates an error condition if roiSize has a field with zero or negative value; or if maxLineCount is less than or equal to 0.
- ippStsStepErr: Indicates an error condition if srcStep has an illegal value.
- ippStsBadArgErr: Indicates an error condition if threshold is less than or equal to 0; or delta.rho is less than 0, or greater than sum of the width and height of the ROI; or delta.theta is less than 0, or greater than p.
- ippStsDstSizeLessExpected: Indicates a warning if number of the detected lines is greater than the size of the destination buffer maxLineCount.

HoughLine_Region
Detects straight lines with the specified range of parameters in the source image.

Syntax

IppStatus ippiHoughLine_Region_8u32f_C1R(const Ipp8u* pSrc, int srcStep, IppiSize roiSize, IppPointPolar* pLine, IppPointPolar dstRoi[2], int maxLineCount, int* pLineCount, IppPointPolar delta, int threshold, Ipp8u* pBuffer);

Include Files

ippcv.h

Domain Dependencies

Headers: ippcore.h, ippvm.h, ipps.h,ippi.h
Libraries: ippcore.lib, ippvm.lib, ipps.lib,ippi.lib

Parameters

- pSrc: Pointer to the source image ROI.
- srcStep: Distance in bytes between starts of consecutive lines in the source image.
- roiSize: Size of source image ROI, in pixels.
- pLine: Pointer to the destination array of detected lines.
- dstRoi: Specifies the range of parameters of straight lines to be detected.
- pLineCount: Pointer to the number of detected lines.
- delta: Step of discretization (delta.rho - radial increment, delta.theta - angular increment).
- threshold: Minimum number of points that are required to detect the line.
- maxLineCount: Maximum number of lines in the destination buffer.
- pBuffer: Pointer to the working buffer.
Description
This function operates with ROI (see Regions of Interest in Intel IPP).

In the binary source image \( pSrc \), this function performs detection of the straight line defined by the equation given at the beginning of section Hough Transform. Only lines \( line[n] \) with the parameters satisfying the following conditions are detected:

\[
\begin{align*}
\text{dstRoi}[0].\rho & \leq \text{line}[n].\rho \leq \text{dstRoi}[1].\rho; \\
\text{dstRoi}[0].\theta & \leq \text{line}[n].\theta \leq \text{dstRoi}[1].\theta;
\end{align*}
\]

where \( n = 0 \). \( pLineCount \).

The level of discretization \( \delta \) is specified as the input parameters. The performance and effectiveness of the function is strongly depends on this parameter. The function requires the external working buffer \( pBuffer \), which size should be computed previously using the function \( \text{ippiHoughLineGetSize} \).

Caution
The value of the parameter \( \delta \) must not be greater than the value of the parameter \( \delta \) set when the size of the working buffer is computed.

Return Values

- ippStsNoErr: Indicates no error.
- ippStsNullPtrErr: Indicates an error if one of the specified pointers is NULL.
- ippStsSizeErr: Indicates an error condition if \( \text{roiSize} \) has a field with zero or negative value; or if \( \text{maxLineCount} \) is less than or equal to 0.
- ippStsStepErr: Indicates an error condition if \( \text{srcStep} \) has an illegal value.
- ippStsBadArgErr: Indicates an error condition if \( \text{threshold} \) is less than or equal to 0; or \( \delta.rho \) is less than 0, or greater than sum of the width and height of the ROI; or \( \delta.theta \) is less than 0, or greater than \( p \); or some filed of \( \text{dstRoi}[0] \) is greater than the corresponding filed of \( \text{dstRoi}[1] \).
- ippStsDstSizeLessExpected: Indicates a warning if number of the detected lines is greater than the size of the destination buffer \( \text{maxLineCount} \).

HoughProbLineGetSize

Computes the size of the working buffer and spec structure for line detection with the probabilistic Hough transform algorithm.

Syntax

\[
\text{IppStatus ippiHoughProbLineGetSize}_8u_\text{C1R}(\text{IppSize roiSize}, \text{IppPointPolar delta}, \text{int}* pSpecSize, \text{int}* pBufSize);
\]

Include Files

ippcv.h

Domain Dependencies

Headers: ippcore.h, ippvm.h, ipps.h,ippi.h

Libraries: ippcore.lib, ippvm.lib, ipps.lib,ippi.lib
Parameters

**roiSize**
Size of the source image ROI, in pixels.

**delta**
Step of discretization (\( \text{delta.rho} \) - radial increment, \( \text{delta.theta} \) - angular increment).

**pSpecSize**
Size of the IppiHoughProbSpec structure.

**pBufSize**
Pointer to the size of the working buffer.

Description
This function operates with ROI.
This function computes the size of the temporary working buffer and the IppiHoughProbSpec specification structure for the ippiHoughProbLine function.
For an example on how to use this function, see the example provided with the ippiHoughProbLine function description.

Return Values

- **ippStsNoErr**
  Indicates no error.
- **ippStsNullPtrErr**
  Indicates an error if \( pBufSize \) is NULL.
- **ippStsSizeErr**
  Indicates an error condition if \( roiSize \) or \( delta \) has a field with a zero or negative value.

See Also
Regions of Interest in Intel IPP
HoughProbLine Detects lines in the source image using the probabilistic Hough transform.

HoughProbLinelnit
*Initializes the specification structure for line detection with the probabilistic Hough transform algorithm.*

Syntax

```c
IppStatus ippiHoughProbLineInit_8u32f_C1R(IppiSize roiSize, IppPointPolar delta, IppHintAlgorithm hint, IppiHoughProbSpec* pSpec);
```

Include Files
ippcv.h

Domain Dependencies
Headers: ippcore.h, ippvm.h, ipps.h,ippi.h
Libraries: ippcore.lib, ippvm.lib, ipps.lib,ippi.lib

Parameters

**roiSize**
Size of the source image ROI, in pixels.

**delta**
Step of discretization (\( \text{delta.rho} \) - radial increment, \( \text{delta.theta} \) - angular increment).
Suggests using specific code for calculations.

*Hint*

*pSpec*

Pointer to the *IppiHoughProbSpec* structure.

**Description**

This function operates with ROI.

This function initializes the *IppiHoughProbSpec* specification structure that is required for the *ippiHoughProbLine* function.

For an example on how to use this function, see the example provided with the *ippiHoughProbLine* function description.

**Return Values**

- *ippStsNoErr*: Indicates no error.
- *ippStsNullPtrErr*: Indicates an error if *pBufSize* is **NULL**.
- *ippStsSizeErr*: Indicates an error condition if *roiSize* or *delta* has a field with zero or negative value.

**See Also**

- Regions of Interest in Intel IPP
- **HoughProbLine** Detects lines in the source image using the probabilistic Hough transform.

**HoughProbLine**

| Detects lines in the source image using the probabilistic Hough transform. |

**Syntax**

```c
IppStatus ippiHoughProbLine_8u32f_C1R(const Ipp8u* pSrc, int srcStep, IppiSize roiSize,
int threshold, int lineLength, int lineGap, IppiPoint* pLine, int maxLineCount, int*
pNextCount, Ipp8u* pBuffer, const IppiHoughProbSpec* pSpec);
```

**Include Files**

- ippcv.h

**Domain Dependencies**

- **Headers**: ippcore.h, ippvm.h, ipps.h,ippi.h
- **Libraries**: ippcore.lib, ippvm.lib, ipps.lib,ippi.lib

**Parameters**

- **pSrc**: Pointer to the source image ROI.
- **srcStep**: Distance, in bytes, between the starting points of consecutive lines in the source image.
- **roiSize**: Size of the source image ROI in pixels.
- **threshold**: Minimum number of points that are required to detect the line.
- **lineLength**: Minimum length of the line.
- **lineGap**: Maximum length of the gap between lines.
Description

This function operates with ROI.

This function detects line segments of the binary pSrc image using the probabilistic Hough transform. Before using this function, compute the size of the working buffer and specification structure using theippiHoughProbGetSize function and initialize the structure using theippiHoughProbLineInit function.

This function implements the probabilistic Hough transform algorithm for line detection, described in [Matas00].

Return Values

ippStsNoErr Indicates no error.
ippStsNullPtrErr Indicates an error if one of the specified pointers is NULL.
ippStsSizeErr Indicates an error condition if roiSize has a field with zero or negative value; or if maxLineCount is less than or equal to 0.
ippStsStepErr Indicates an error condition if srcStep has an illegal value.
ippStsBadArgErr Indicates an error condition if threshold is less than or equal to 0.
ippStsDstSizeLessExpected Indicates a warning if number of the detected lines is greater than the size of the destination buffer maxLineCount.

Example

To better understand usage of theippiHoughProbLine function, refer to the following example in theexamples archive available for download from https://software.intel.com/en-us/ipp-manual-examples:

HoughProbLine.c

See Also

Regions of Interest in Intel IPP
HoughProbLineGetSize Computes the size of the working buffer and spec structure for line detection with the probabilistic Hough transform algorithm.
HoughProbLineInit Initializes the specification structure for line detection with the probabilistic Hough transform algorithm.

LineSuppressionGetBufferSize

*Computes the size of the external buffer for theippiLineSuppression function.*

Syntax

IppStatus ippiLineSuppressionGetBufferSize(IppSize roiSize, IppMaskSize filterMask, Ipp32u avgWndSize, IppDataType dataType, int numChannels, int* pBufferSize);
Include Files
ippcv.h

Domain Dependencies
Headers: ippcore.h, ippvm.h, ipps.h,ippi.h
Libraries: ippcore.lib, ippvm.lib, ipps.lib,ippi.lib

Parameters
roiSize  Size of the source and destination image ROI, in pixels.
filterMask  Size of the derivative filter aperture. Possible values are ippMskSize3x3 and ippMskSize5x5.
avgWndSize  Linear size of a neighborhood block for averaging.
dataType  Data type of the source image.
numChannels  Number of channels in the images. Possible value is 1.
pBufferSize  Pointer to the size, in bytes, of the external work buffer.

Description
This function calculates the size of the temporary buffer for the ippiLineSuppression function.

Return Values
ippStsNoErr  Indicates no error. Any other value indicates an error or a warning.
ippStsNullPtrErr  Indicates an error when pBufferSize is NULL.
ippStsSizeErr  Indicates an error in the following cases:
  • when roiSize is less than, or equal to zero
  • when avgWndSize is equal to zero
ippStsMaskSizeErr  Indicates an error when filterMask has an illegal value.
ippStsDataTypeErr  Indicates an error when dataType has an illegal value.
ippStsNumChannelsErr  Indicates an error when numChannels has an illegal value.

See Also
LineSuppression Implements the line suppression algorithm.

LineSuppression
Implements the line suppression algorithm.

Syntax
IppStatus ippiLineSuppression_8u_C1R(const Ipp8u* pSrc, int srcStep, const Ipp8u* pFeature, int featureStep, Ipp8u* pDst, int dstStep, IppiSize roiSize, IppiDifferentialKernel filterType, IppiMaskSize filterMask, Ipp32u avgWndSize, float threshold, IppiBorderType borderType, Ipp8u borderValue, Ipp8u* pBuffer);
Include Files
ippcv.h

Domain Dependencies
Headers: ippcore.h, ippvm.h, ipps.h,ippi.h
Libraries: ippcore.lib, ippvm.lib, ipps.lib,ippi.lib

Parameters

cs
srcStep
pf
featureStep
pD

Distance, in bytes, between the starting points of consecutive lines in the source image.

Distance, in bytes, between the starting points of consecutive lines in the feature points image.

Distance, in bytes, between the starting points of consecutive lines in the destination image.

Size of the source and destination image ROI, in pixels.

Type of the derivative operator. Possible values are:

Sobel filter
Scharr filter
Central differences operator

Size of the derivative filter aperture. The list of possible values depends on the filterType value:

<table>
<thead>
<tr>
<th>Filter Type</th>
<th>Filter Mask</th>
</tr>
</thead>
<tbody>
<tr>
<td>ippKernelSobel</td>
<td>ippMskSize3x3, ippMskSize5x5</td>
</tr>
<tr>
<td>ippKernelScharr</td>
<td>ippMskSize3x3</td>
</tr>
<tr>
<td>ippKernelCentralDiff</td>
<td>ippMskSize3x3</td>
</tr>
</tbody>
</table>

Linear size of a neighborhood block for averaging.

Line suppression threshold.

Type of border. Possible values are:

Values of all border pixels are set to a constant.

Border is replicated from the edge pixels.

Border is obtained from the source image pixels in memory.

Mixed borders are also supported. They can be obtained by the bitwise operation OR between ippBorderRepl or ippBorderConst and the following flags:

• ippBorderInMemTop
Each of these flags means that for the corresponding border the outside pixels of the source image are in memory.

**borderValue**

Constant value(s) to assign to pixels of the constant border. This parameter is applicable only to the **ippBorderConst** border type.

**pBuffer**

Pointer to the work buffer. To calculate the size of the temporary buffer, use the **Line Suppression Get Buffer Size** function.

**Description**

The **ippiLine Suppression** function implements the line suppression algorithm. This function uses two input images: the original image and feature points image containing the lines (edges) and corners. This function performs the following steps:

1. Computes \( I(x, y)_x \) and \( I(x, y)_y \) gradients for each \((x, y)\) pixel of the image. The function computes gradients using the derivative operator specified by the **filterType** and **filterMask** parameters.

2. Computes products of the gradients at each \((x, y)\) pixel of the image:

\[
I_{xx}(x, y) = I_x(x, y) \ast I_x(x, y), \quad I_{xy}(x, y) = I_x(x, y) \ast I_y(x, y), \quad I_{yy}(x, y) = I_y(x, y) \ast I_y(x, y)
\]

3. Performs averaging of the products of gradients over a rectangular neighborhood block at each pixel of the image. The block size is specified by the **avgWndSize** value.

\[
S_{xx}(x, y) = \sum_{y'} \sum_{x'} I_{xx}(x', y'), \quad S_{xy}(x, y) = \sum_{y'} \sum_{x'} I_{xy}(x', y'), \quad S_{yy}(x, y) = \sum_{y'} \sum_{x'} I_{yy}(x', y')
\]

4. Defines 2x2 gradient covariance matrix \( H(x, y) \) over a rectangular neighborhood block for each \((x, y)\) pixel of the image.

\[
H(x, y) = \begin{pmatrix}
S_{xx}(x, y) & S_{xy}(x, y) \\
S_{xy}(x, y) & S_{yy}(x, y)
\end{pmatrix}
\]

5. For each \((x, y)\) pixel of the image checks the condition below. If the condition is true, the considered point is not a feature point.

\[
\frac{(\lambda_1 + \lambda_2)^2}{\lambda_1 \ast \lambda_2} = \frac{(trH(x, y))^2}{\det H(x, y)} > \text{threshold}
\]

where

- **threshold** is the line suppression threshold value passed to the function
- \(\lambda_1 \ast \lambda_2\) are eigenvalues of the matrix \(H(x, y)\). If both eigenvalues have large positive values, the point belongs to a corner. If \(\lambda_1\) is much bigger than \(\lambda_2\), the function clears out the candidate point.

The first and third stages of the function algorithm are filtering operations; they use border processing approach that is specified by the **borderType** parameter.
Before using this function, compute the size of the external work buffer using the
LineSuppressionGetBufferSize function.

**Return Values**

<table>
<thead>
<tr>
<th>Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ippStsNoErr</td>
<td>Indicates no error.</td>
</tr>
<tr>
<td>ippStsNullPtrErr</td>
<td>Indicates an error when <code>pSrc</code>, <code>pFeature</code>, <code>pDst</code>, or <code>pBufferSize</code> is NULL.</td>
</tr>
<tr>
<td>ippStsSizeErr</td>
<td>Indicates an error in the following cases:</td>
</tr>
<tr>
<td></td>
<td>• when <code>roiSize</code> is less than, or equal to zero</td>
</tr>
<tr>
<td></td>
<td>• when <code>avgWndSize</code> is equal to zero</td>
</tr>
<tr>
<td>ippStsNotEvenStepErr</td>
<td>Indicates an error when one of the step values is not divisible by 4 for floating point images.</td>
</tr>
<tr>
<td>ippStsFilterTypeErr</td>
<td>Indicates an error when <code>filterType</code> has an illegal value.</td>
</tr>
<tr>
<td>ippStsMaskSizeErr</td>
<td>Indicates an error when <code>filterMask</code> has an illegal value.</td>
</tr>
<tr>
<td>ippStsBorderErr</td>
<td>Indicates an error when <code>borderType</code> has an illegal value.</td>
</tr>
<tr>
<td>ippStsStepErr</td>
<td>Indicates an error when <code>srcStep</code> or <code>dstStep</code> has a negative value.</td>
</tr>
<tr>
<td>ippStsInplaceModeNotSupportedErr</td>
<td>Indicates an error when <code>pFeature</code> and <code>pDst</code> point to the same image.</td>
</tr>
</tbody>
</table>

**Example**

To better understand usage of the `ippiLineSuppression` function, refer to the following example in the examples archive available for download from https://software.intel.com/en-us/ipp-manual-examples:

LineSuppression.c

**See Also**

LineSuppressionGetBufferSize Computes the size of the external buffer for the `ippiLineSuppression` function.

---

**Distance Transform Functions**

This section describes the distance transform functions.

Distance transform is used for calculating the distance to an object. The input is an image with feature and non-feature pixels. The function labels every non-feature pixel in the output image with a distance to the closest feature pixel. Feature pixels are marked with zero.

Distance transform is used for a wide variety of subjects including skeleton finding and shape analysis.

**DistanceTransform**

*Calculates distance to the closest zero pixel for all non-zero pixels of source image.*

**Syntax**

**Case 1: Not-in-place operations**

```c
IppStatus ippiDistanceTransform_3x3_<mod>(const Ipp8u* pSrc, int srcStep, Ipp<dstDatatype>* pDst, int dstStep, IppiSize roiSize, Ipp32s* pMetrics);
```
IppStatus ippiDistanceTransform_5x5_<mod>(const Ipp8u* pSrc, int srcStep, Ipp<dstDatatype>* pDst, int dstStep, IppiSize roiSize, Ipp32s* pMetrics);

Supported values for mod:

8u_C1R 8u16u_C1R

IppStatus ippiDistanceTransform_3x3_8u32f_C1R(const Ipp8u* pSrc, int srcStep, Ipp32f* pDst, int dstStep, IppiSize roiSize, Ipp32f* pMetrics);
IppStatus ippiDistanceTransform_5x5_8u32f_C1R(const Ipp8u* pSrc, int srcStep, Ipp32f* pDst, int dstStep, IppiSize roiSize, Ipp32f* pMetrics);

Case 2: In-place operations
IppStatus ippiDistanceTransform_3x3_8u_C1IR(Ipp8u* pSrcDst, int srcDstStep, IppiSize roiSize, Ipp32s* pMetrics);
IppStatus ippiDistanceTransform_5x5_8u_C1IR(Ipp8u* pSrcDst, int srcDstStep, IppiSize roiSize, Ipp32s* pMetrics);

Include Files
ippcv.h

Domain Dependencies
Headers: ippcore.h, ippvm.h, ipps.h,ippi.h
Libraries: ippcore.lib, ippvm.lib, ipps.lib,ippi.lib

Parameters

pSrc  Pointer to the source image ROI.
srcStep  Distance in bytes between starts of consecutive lines in the source image.
pDst  Pointer to the ROI in the destination distance image.
dstStep  Distance in bytes between starts of consecutive lines in the destination image.
pSrcDst  Pointer to the source and destination image ROI for in-place operation.
srcDstStep  Distance in bytes between starts of consecutive lines in the source and destination image for in-place operation.
roiSize  Size of the image ROI in pixels.
pMetrics  Pointer to the array that specifies used metrics.

Description
This function operates with ROI (see Regions of Interest in Intel IPP).
This function approximates the actual distance from the closest zero pixel to each certain pixel with the sum of atomic distances from the fixed set. The set consists of two values for a 3x3 mask and three values for a 5x5 mask.
Figure “3x3 Mask” shows the result of the distance transform of a 7x7 image with zero point in the center. This example corresponds to a 3x3 mask. Two numbers specify metrics in case of the 3x3 mask:

- distance between two pixels that share an edge,
• distance between the pixels that share a corner.

In this case the values are 1 and 1.5 correspondingly.

<table>
<thead>
<tr>
<th>3x3 Mask</th>
</tr>
</thead>
<tbody>
<tr>
<td>4.5 4 3.5 3 3.5 4 4.5</td>
</tr>
<tr>
<td>4 3 2.5 2 2.5 3 4</td>
</tr>
<tr>
<td>3.5 2.5 1.5 1 1.5 2.5 3.5</td>
</tr>
<tr>
<td>3 2 1 0 1 2 3</td>
</tr>
<tr>
<td>3.5 2.5 1.5 1 1.5 2.5 3.5</td>
</tr>
<tr>
<td>4 3 2.5 2 2.5 3 4</td>
</tr>
<tr>
<td>4.5 4 3.5 3 3.5 4 4.5</td>
</tr>
</tbody>
</table>

Figure “5x5 Mask” shows the distance transform for the same image, but for a 5x5 mask.

For this mask yet another number is added to specify metrics - the additional distance, i.e., the distance between pixels corresponding to the chess knight move.

In this example, the additional distance is equal to 2.

<table>
<thead>
<tr>
<th>5x5 Mask</th>
</tr>
</thead>
<tbody>
<tr>
<td>4 3.5 3 3 3 3.5 4</td>
</tr>
<tr>
<td>3.5 3 2 2 2 3 3.5</td>
</tr>
<tr>
<td>3 2 1.5 1 1.5 2 3</td>
</tr>
<tr>
<td>3 2 1 0 1 2 3</td>
</tr>
<tr>
<td>3.5 3 2 2 2 3 3.5</td>
</tr>
<tr>
<td>4 3.5 3 3 3 3.5 4</td>
</tr>
</tbody>
</table>

Return Values

ippStsNoErr
- Indicates no error. Any other value indicates an error or a warning.

ippStsNullPtrErr
- Indicates an error condition if one of the specified pointers is NULL.

ippStsSizeErr
- Indicates an error condition if roiSize has a field with zero or negative value.

ippStsStepErr
- Indicates an error condition if srcStep or dstStep is less than roiSize.width*<pixelSize>.

ippStsNotEvenStepErr
- Indicates an error condition if step value is not divisible by 2 for 16u images, and by 4 for 32f images.

ippStsCoeffErr
- Indicates an error condition if at least one element of pMetrics array has zero or negative value.

Example

To better understand usage of the ippiDistanceTransform function, refer to the following example in the examples archive available for download from https://software.intel.com/en-us/ipp-manual-examples:

DistanceTransform.c

Result:

```
1 2 3 4 5 6 7
1 0 3 4 5 6 7
1 2 3 4 5 6 7
```
GetDistanceTransformMask

Returns an optimal mask for a given type of metrics and given mask size.

Syntax

IppStatus ippiGetDistanceTransformMask_\(mod\)(int kerSize, IppiNorm norm, Ipp<datatype>* pMetrics);

Supported values for \(mod\):

\[32s\quad 32f\]

IppStatus ippiGetDistanceTransformMask(int maskType, Ipp32f* pMetrics);

Include Files

ippcv.h

Domain Dependencies

Headers: ippcore.h, ippvm.h, ipps.h, ippi.h
Libraries: ippcore.lib, ippvm.lib, ipps.lib, ippi.lib

Parameters

\(kerSize\)

Specifies the mask size as follows: 3 for 3x3 mask, 5 for 5x5 mask.

\(norm\)

Specifies the type of metrics. Possible values are:

ippiNormInf(0) \(L_{\infty}\), \(\Delta = \max(|x_1 - x_2|, |y_1 - y_2|)\)\nippiNormL1(1) \(L_1\), \(\Delta = |x_1 - x_2| + |y_1 - y_2|\)\nippiNormL2(2) \(L_2\), \(\Delta = \sqrt{((x_1 - x_2)^2 + (y_1 - y_2)^2)}\)

\(maskType\)

Distance type.

\(pMetrics\)

Pointer to the output array to store metrics parameters. The array contains the following number of elements:

2 for 3x3 mask,

3 for 5x5 mask.
Description
This function fills up the output array with metrics parameters for the given type of metrics and size of mask. The function returns the following results:

(1, 1)  \(L_\infty\), 3x3 mask,
(1, 2)  \(L_1\), 3x3 mask,
(2, 3)  \(L_2\), 3x3 mask, 32s data type,
(0.955, 1.3693)  \(L_2\), 3x3 mask, 32f data type,
(1, 1, 2)  \(L_\infty\), 5x5 mask,
(1, 2, 3)  \(L_1\), 5x5 mask,
(4, 6, 9)  \(L_2\), 5x5 mask, 32s data type,
(1.0, 1.4, 2.1969)  \(L_2\), 5x5 mask, 32f data type.

For more information, see [Bor86].

NOTE
For compatibility with the previous versions of the library the earlier function ippiGetDistanceTransformMask replaced by the function ippiGetDistanceTransformMask_32f in the current version is also supported.

Return Values

<table>
<thead>
<tr>
<th>Return Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ippStsNoErr</td>
<td>Indicates no error. Any other value indicates an error or a warning.</td>
</tr>
<tr>
<td>ippStsNullPtrErr</td>
<td>Indicates an error condition if pMetrics pointer is NULL.</td>
</tr>
<tr>
<td>ippStsSizeErr</td>
<td>Indicates an error condition if kerSize has a wrong value.</td>
</tr>
<tr>
<td>ippStsBadArgErr</td>
<td>Indicates an error condition if kerSize or norm has a wrong value.</td>
</tr>
</tbody>
</table>

FastMarchingGetBufferSize

*Computes the size of the working buffer for the peak search.*

Syntax

```c
IppStatus ippiFastMarchingGetBufferSize_8u32f_C1R(IppiSize roiSize, int* pBufferSize);
```

Include Files

ippcv.h

Domain Dependencies

Headers: ippcore.h, ippvm.h, ipps.h,ippi.h

Libraries: ippcore.lib, ippvm.lib, ipps.lib,ippi.lib
Parameters

\textbf{roiSize} \hspace{1cm} \text{Maximum image size (in pixels).}

\textbf{pBufferSize} \hspace{1cm} \text{Pointer to the computed size of the buffer.}

Description

This function operates with ROI (see Regions of Interest in Intel IPP).

This function computes the size of the working buffer required for the \texttt{ippiFastMarching} function. The buffer with the length \texttt{pBufferSize[0]} can be used to filter images with width that is equal to or less than the parameter \texttt{roiSize} specified for the function \texttt{ippiFastMarching}.

Return Values

- \texttt{ippStsNoErr} \hspace{1cm} \text{Indicates no error. Any other value indicates an error or a warning.}
- \texttt{ippStsNullPtrErr} \hspace{1cm} \text{Indicates an error condition if \texttt{pBufferSize} pointer is NULL.}
- \texttt{ippStsSizeErr} \hspace{1cm} \text{Indicates an error condition if \texttt{roiSize} is less than 1.}

FastMarching

*Calculates distance transform to closest zero pixel for all non-zero pixels of source image using fast marching method.*

Syntax

\begin{verbatim}
IppStatus ippiFastMarching_8u32f_C1R(const Ipp8u* pSrc, int srcStep, Ipp32f* pDst, int dstStep, IppiSize roiSize, Ipp32f radius, Ipp8u* pBuffer);
\end{verbatim}

Include Files

ippcv.h

Domain Dependencies

Headers: ippcore.h, ippvm.h, ipps.h,ippi.h

Libraries: ippcore.lib, ippvm.lib, ipps.lib,ippi.lib

Parameters

- \textbf{pSrc} \hspace{1cm} \text{Pointer to the source image ROI.}
- \textbf{srcStep} \hspace{1cm} \text{Distance in bytes between starts of consecutive lines in the source image.}
- \textbf{pDst} \hspace{1cm} \text{Pointer to the destination image ROI.}
- \textbf{dstStep} \hspace{1cm} \text{Distance in bytes between starts of consecutive lines in the destination image.}
- \textbf{roiSize} \hspace{1cm} \text{Size of the source and destination image ROI.}
- \textbf{radius} \hspace{1cm} \text{Radius of the neighborhood of the marked area.}
- \textbf{pBuffer} \hspace{1cm} \text{Pointer to the working buffer.}
Description

This function operates with ROI (see Regions of Interest in Intel IPP).

This function computes the distance from the closest zero pixel to each image pixel according to the Fast Marching Method (FMM) [Telea04]. The FMM distance for area Ω with the border ∂Ω is a solution of the equations:

\[ |\nabla T(x, y)| = 1, \quad \{x, y\} \in \Omega \]
\[ T(x, y) = 0, \quad \{x, y\} \in \partial \Omega \]

The resulting distance complies with the equation

\[ T(x, y) = \min \left( \frac{T(u_1, v_1) + T(u_2, v_2) + \sqrt{\frac{2}{2} - (T(u_1, v_1) - T(u_2, v_2))^2}}{\min(T(u_1, v_1), T(u_2, v_2)) + 1} \right) \]

Here \( \{u_1, v_1\} \) and \( \{u_2, v_2\} \) are coordinates for pair of pixels adjacent to the pixel with \( \{x, y\} \) coordinates.

The area \( \Omega \) is defined by the non-zero pixel of the image \( pSrc \). If \( radius \) is positive, then the FMM distance with the negative sign is calculated in Euclidean \( radius \)-neighborhood of \( \Omega \).

The function requires the working buffer \( pBuffer \) whose size should be computed by the function \( FastMarchingBufferSize \) beforehand.

Figure "Result of the FFM Method" shows the result of the fast marching method for the 7x9 image with centered 3x5 non-zero mask and \( radius=1 \).

Result of the FFM Method

\[
\begin{array}{cccccccccccc}
0.0000 & 0.0000 & -1.0000 & -1.0000 & -1.0000 & -1.0000 & -1.0000 & 0.0000 & 0.0000 \\
0.0000 & 0.7071 & -1.e-10 & -1.e-10 & -1.e-10 & -1.e-10 & -1.e-10 & 0.7071 & -1.0000 \\
-1.0000 & -1.e-10 & 0.7071 & 0.9659 & 0.9994 & 0.9659 & 0.7071 & -1.e-10 & -1.0000 \\
-1.0000 & -1.e-10 & 0.9659 & 1.6730 & 1.9579 & 1.6730 & 0.9659 & -1.e-10 & -1.0000 \\
-1.0000 & -1.e-10 & 0.7071 & 0.9659 & 0.9994 & 0.9659 & 0.7071 & -1.e-10 & -1.0000 \\
0.0000 & 0.7071 & -1.e-10 & -1.e-10 & -1.e-10 & -1.e-10 & -1.e-10 & 0.7071 & -1.0000 \\
0.0000 & -1.0000 & -1.0000 & -1.0000 & -1.0000 & -1.0000 & -1.0000 & 0.0000 & 0.0000 \\
\end{array}
\]

Return Values

ippStsNoErr Indicates no error. Any other value indicates an error or a warning.

ippStsNullPtrErr Indicates an error condition if one of the specified pointers is NULL.

ippStsSizeErr Indicates an error condition if roiSize has a field with zero or negative value.

ippStsStepErr Indicates an error condition if srcStep or dstStep is less than roiSize.width *< pixelSize >.

ippStsNotEvenStepErr Indicates an error condition if the step value is not divisible by 4 for floating-point images.

ippStsBadArgErr Indicates an error condition if radius is negative.
**TrueDistanceTransformGetBufSize**  
Calculates the size of the temporary working buffer for the function ippiTrueDistanceTransform.

**Syntax**

```c
IppStatus ippiTrueDistanceTransformGetBufferSize_8u32f_C1R(IppiSize roiSize, int* pBufferSize);
```

**Include Files**

ippcv.h

**Domain Dependencies**

Headers: ippcore.h, ippvm.h, ipps.h,ippi.h  
Libraries: ippcore.lib, ippvm.lib, ipps.lib,ippi.lib

**Parameters**

- `roiSize`: Size of the image ROI in pixels.  
- `pBufferSize`: Pointer to the computed size of the buffer.

**Description**

This function operates with ROI (see Regions of Interest in Intel IPP).  
This function calculates the size of the work buffer required for the TrueDistanceTransform function.

**Return Values**

- `ippStsNoErr`: Indicates no error. Any other value indicates an error or a warning.  
- `ippStsNullPtrErr`: Indicates an error condition if one of the specified pointers is NULL.  
- `ippStsSizeErr`: Indicates an error condition if `roiSize` has a field with zero or negative value.  
- `ippStsStepErr`: Indicates an error condition if `srcStep` or `dstStep` is less than `roiSize.width*<pixelSize>`.  
- `ippStsNotEvenStepErr`: Indicates an error condition if step value is not divisible by 2 for 16u images, and by 4 for 32f images.

**TrueDistanceTransform**  
Calculates the Euclidean distance to the closest zero pixel for all non-zero pixels of the source image.

**Syntax**

```c
IppStatus ippiTrueDistanceTransform_8u32f_C1R(const Ipp8u* pSrc, int srcStep, Ipp32f* pDst, int dstStep, IppiSize roiSize, Ipp8u* pBuffer);
```

**Include Files**

ippcv.h
Domain Dependencies

Headers: ippcore.h, ippvm.h, ipps.h,ippi.h
Libraries: ippcore.lib, ippvm.lib, ipps.lib,ippi.lib

Parameters

pSrc  
Pointer to the source image ROI.

srcStep  
Distance in bytes between starts of consecutive lines in the source image.

pDst  
Pointer to the ROI in the destination distance image.

dstStep  
Distance in bytes between starts of consecutive lines in the destination image.

roiSize  
Size of the image ROI in pixels.

pBuffer  
Pointer to the temporary working buffer.

Description

This function operates with ROI (see Regions of Interest in Intel IPP).

This function calculates the Euclidean distance to the closest zero pixel for all non-zero pixels of the source image [Felz04].

The figure below shows the result of the integer version of the true distance transform of a 7x7 image with zero point in the center and with the scale factor -5.

<p>| | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>136</td>
<td>115</td>
<td>101</td>
<td>96</td>
<td>101</td>
<td>115</td>
</tr>
<tr>
<td>115</td>
<td>91</td>
<td>72</td>
<td>64</td>
<td>72</td>
<td>91</td>
</tr>
<tr>
<td>101</td>
<td>72</td>
<td>45</td>
<td>36</td>
<td>45</td>
<td>72</td>
</tr>
<tr>
<td>96</td>
<td>64</td>
<td>36</td>
<td>0</td>
<td>36</td>
<td>64</td>
</tr>
<tr>
<td>101</td>
<td>72</td>
<td>45</td>
<td>36</td>
<td>45</td>
<td>72</td>
</tr>
<tr>
<td>115</td>
<td>91</td>
<td>72</td>
<td>64</td>
<td>72</td>
<td>91</td>
</tr>
<tr>
<td>136</td>
<td>115</td>
<td>101</td>
<td>96</td>
<td>101</td>
<td>115</td>
</tr>
</tbody>
</table>

Return Values

ippStsNoErr  
Indicates no error. Any other value indicates an error or a warning.

ippStsNullPtrErr  
Indicates an error condition if one of the specified pointers is NULL.

ippStsSizeErr  
Indicates an error condition if roiSize has a field with zero or negative value.

ippStsStepErr  
Indicates an error condition if srcStep or dstStep is less than roiSize.width*pixelSize.

ippStsNotEvenStepErr  
Indicates an error condition if step value is not divisible by 2 for 16u images, and by 4 for 32f images.

Image Gradients
**GradientColorToGray**

*Converts a color gradient image to grayscale.*

**Syntax**

```c
IppStatus ippiGradientColorToGray_<mod>(const Ipp<datatype>* pSrc, int srcStep, Ipp<datatype>* pDst, int dstStep, IppiSize roiSize, IppiNorm norm);
```

Supported values for `mod`:

- 8u_C3C1R
- 16u_C3C1R
- 32f_C3C1R

**Include Files**

ippcv.h

**Domain Dependencies**

Headers: ippcore.h, ippvm.h, ipps.h, ippi.h
Libraries: ippcore.lib, ippvm.lib, ipps.lib, ippi.lib

**Parameters**

- `pSrc` Pointer to the source image ROI.
- `srcStep` Distance in bytes between starts of consecutive lines in the source image.
- `pDst` Pointer to the destination image ROI.
- `dstStep` Distance in bytes between starts of consecutive lines in the destination image.
- `roiSize` Size of the source and destination image ROI.
- `norm` Type of norm to form the mask for dilation; following values are possible:
  - ippNormInf Infinity norm.
  - ippNormL1 L1 norm.
  - ippNormL2 L2 norm.

**Description**

This function operates with ROI (see Regions of Interest in Intel IPP).

This function creates the grayscale gradient image `pDst` from the source three-channel gradient image `pSrc`. The type of norm is specified by the parameter. Pixel values for destination image are computed for different type of norm in accordance with the following formula:

\[
dst_{i,j} = \begin{cases} 
\max\{|src_{i,j,0}|,|src_{i,j,1}|,|src_{i,j,2}|\} & \text{norm} = \text{ippiNormInf} \\
|src_{i,j,0}| + |src_{i,j,1}| + |src_{i,j,2}| & \text{norm} = \text{ippiNormL1} \\
\sqrt{src_{i,j,0}^2 + src_{i,j,1}^2 + src_{i,j,2}^2} & \text{norm} = \text{ippiNormL2}
\end{cases}
\]

For integer flavors the result is scaled to the full range of the destination data type.
Return Values

ippStsNoErr
Indicates no error. Any other value indicates an error or a warning.

ippStsNullPtrErr
Indicates an error condition if one of the specified pointers is NULL.

ippStsSizeErr
Indicates an error condition if roiSize has a field with zero or negative value.

ippStsStepErr
Indicates an error condition if srcStep or dstStep is less than roiSize.width * pixelSize.

ippStsNotEvenStepErr
Indicates an error condition if one of the step values is not divisible by 2 for integer images, or by 4 for floating-point images.

ippStsBadArgErr
Indicates an error condition if norm has an illegal value.

GradientVectorGetBufferSize
Computes the size of the work buffer for the ippiGradientVector{Sobel|Scharr|Prewitt} functions.

Syntax
IppStatus ippiGradientVectorGetBufferSize (IppiSize roiSize, IppiMaskSize mask, IppDataType dataType, int numChannels, int* pBufferSize);

Include Files
ippi.h

Domain Dependencies
Headers: ippcore.h, ippvm.h, ipps.h
Libraries: ippcore.lib, ippvm.lib, ipps.lib

Parameters
roiSize
Size of the destination ROI in pixels.

mask
Predefined mask of IppiMaskSize type.

dataType
Data type of the source image.

numChannels
Number of channels in the image.

pBufferSize
Pointer to the computed size of the external work buffer.

Description
The ippiGradientVectorGetBufferSize function computes the size (in bytes) of the external work buffer needed for the ippiGradientVector{Sobel|Scharr|Prewitt} functions and stores the result in the pBufferSize parameter.
Return Values

<table>
<thead>
<tr>
<th></th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ippStsNoErr</td>
<td>Indicates no error.</td>
</tr>
<tr>
<td>ippStsNullPtrErr</td>
<td>Indicates an error when pBufferSize is NULL.</td>
</tr>
<tr>
<td>ippStsSizeErr</td>
<td>Indicates an error when one of the fields of roiSize has a zero or negative value.</td>
</tr>
<tr>
<td>ippStsMaskSizeErr</td>
<td>Indicates an error when mask has an illegal value.</td>
</tr>
<tr>
<td>ippStsDataTypeErr</td>
<td>Indicates an error when dataType has an illegal value.</td>
</tr>
<tr>
<td>ippStsNumChannelsErr</td>
<td>Indicates an error when numChannels has an illegal value.</td>
</tr>
</tbody>
</table>

See Also

Structures and Enumerators

GradientVectorPrewitt Computes gradient vectors of an image using the Prewitt operator.
GradientVectorScharr Computes gradient vectors of an image using the Scharr operator.
GradientVectorSobel Computes gradient vectors of an image using the Sobel operator.

GradientVectorPrewitt

Computes gradient vectors of an image using the Prewitt operator.

Syntax

IppStatus ippiGradientVectorPrewitt_<mod> (const Ipp<srcDatatype>* pSrc, int srcStep, Ipp<dstDatatype>* pGx, int gxStep, Ipp<dstDatatype>* pGy, int gyStep, Ipp<dstDatatype>* pMag, int magStep, Ipp32f* pAngle, int angleStep, IppiSize dstRoiSize, IppiMaskSize maskSize, IppNormType normType, IppiBorderType borderType, Ipp<srcDatatype> borderValue, Ipp8u* pBuffer);

Supported values for mod:

8u16s_C1R 16s32f_C1R 16u32f_C1R 32f_C1R

Include Files

ippi.h

Domain Dependencies

Headers: ippcore.h, ippvm.h, ipps.h
Libraries: ippcore.lib, ippvm.lib, ipps.lib
**Parameters**

- **pSrc**: Pointer to the source image ROI.
- **srcStep**: Distance, in bytes, between the starting points of consecutive lines in the source image.
- **pGx**: Pointer to the destination image ROI for the X component of the gradient vector (in Cartesian form).
- **gxStep**: Distance, in bytes, between the starting points of consecutive lines in the X-component destination image.
- **pGy**: Pointer to the destination image ROI for the Y component of the gradient vector (in Cartesian form).
- **gyStep**: Distance, in bytes, between the starting points of consecutive lines in the Y-component destination image.
- **pMag**: Pointer to the magnitude of the gradient destination image ROI (in polar gradient form).
- **magStep**: Distance, in bytes, between the starting points of consecutive lines in the magnitude of the gradient destination image.
- **pAngle**: Pointer to the destination image ROI for the angle (in polar gradient form).
- **angleStep**: Distance, in bytes, between the starting points of consecutive lines in the angle destination image.
- **dstRoiSize**: Size of the source and destination image ROI in pixels.
- **maskSize**: Predefined mask of IppiMaskSize type.
- **normType**: Normalization mode of IppNormType type.
- **borderType**: Type of border. Possible values are:
  - ippBorderConst: Values of all border pixels are set to constant.
  - ippBorderRepl: Border is replicated from the edge pixels.
  - ippBorderInMem: Border is obtained from the source image pixels in memory.
  - ippBorderMirror: Border pixels are mirrored from the source image boundary pixels.
- **borderValue**: Constant value to assign to pixels in the constant border (not applicable for other border types).
- **pBuffer**: Pointer to the work buffer.

**Description**

This function operates with ROI (see Regions of Interest in Intel IPP).

Before using this function, compute the size of the work buffer using the ippiGradientVectorGetBufferSize function.
NOTE
Any of the $pGx$, $pGy$, $pMag$, and $pAngle$ output parameters can be NULL. This means that the parameter(s) is not requested.

This function operates on "gray" single-channel (C1 flavors) and color (C3 flavors) images.

**Single-channel image (C1) input:**

If input is a single-channel image, the `ippiGradientVectorPrewitt` function computes the gradient vector at each pixel of the source image ROI and stores the result either in Cartesian ($pGx$ and $pGy$) and/or polar ($pMag$ and $pAngle$) form, or any combination of these possible outputs.

Cartesian projections $G_x$ and $G_y$ are stored in the $pGx$ and $pGy$ buffers, respectively. The formulas below describe the algorithm for the 3x3 Prewitt operator:

$$
G_x = \begin{bmatrix}
-1 & 0 & 1 \\
-1 & 0 & 1 \\
-1 & 0 & 1 \\
\end{bmatrix} \ast A \quad \text{and} \quad G_y = \begin{bmatrix}
1 & 1 & 1 \\
0 & 0 & 0 \\
-1 & -1 & -1 \\
\end{bmatrix} \ast A
$$

where
- $A$ is the source image
- $\ast$ means two-dimensional convolution
- $G_x$ and $G_y$ are X and Y components of the gradient

The magnitude of the gradient is computed according to the `normType` value by the following formulas:

**L1 normalization:**

$$
G = ||G_x|| + ||G_y||
$$

**L2 normalization:**

$$
G = \sqrt{G_x^2 + G_y^2}
$$

The value of angle between $G_x$ and $G_y$ is computed by the formula:

$$
\text{angle} = \arctan \frac{G_y}{G_x}
$$

**Color image (C3) input:**

If input is a color image, the `ippiGradientVectorPrewitt` function computes the spatial image derivatives $G_x$ and $G_y$ for each channel of the image using the specified differential operator. For each pixel $(x, y)$ this function chooses the derivatives for which $L2(G_x, G_y)$ is the maximal value and stores them in the $pGx$ and $pGy$ output arrays. In other words, for each pixel of a color image the function returns the derivatives composing the largest gradient across all channels.

The examples of using this function are similar to the examples provided with the `GradientVectorSobel` function description.
Return Values

ippStsNoErr  
Indicates no error.

ippStsNullPtrErr  
Indicates an error when pSrc or pBuffer is NULL.

ippStsSizeErr  
Indicates an error when one of the fields of dstRoiSize has a zero or negative value.

ippStsMaskSizeErr  
Indicates an error when maskSize has an illegal value.

ippStsStepErr  
Indicates an error when:
• srcStep, gxStep, gyStep, magStep, or angleStep has a zero or negative value
• srcStep, gxStep, gyStep, magStep, or angleStep is not a multiple of the image data size (4 for floating-point images or 2 for short integer images)

ippStsBadArgErr  
Indicates an error when normType has an illegal value.

ippStsBorderErr  
Indicates an error when borderType has an incorrect value.

See Also

Structures and Enumerators

Regions of Interest in Intel IPP

GradientVectorGetBufferSize  
Computes the size of the work buffer for the ippiGradientVector{Sobel|Scharr|Prewitt} functions.

GradientVectorScharr  
Computes gradient vectors of an image using the Scharr operator.

Syntax

IppStatus ippiGradientVectorScharr_<mod> (const Ipp<srcDatatype>** pSrc, int srcStep, Ipp<dstDatatype>* pGx, int gxStep, Ipp<dstDatatype>* pGy, int gyStep, Ipp<dstDatatype>* pMag, int magStep, Ipp32f* pAngle, int angleStep, IppiSize dstRoiSize, IppiMaskSize maskSize, IppNormType normType, IppiBorderType borderType, Ipp<srcDatatype> borderValue, Ipp8u* pBuffer);

Supported values for mod:

8u16s_C1R 16s32f_C1R 16u32f_C1R 32f_C1R

IppStatus ippiGradientVectorScharr_<mod> (const Ipp<srcDatatype>** pSrc, int srcStep, Ipp<dstDatatype>* pGx, int gxStep, Ipp<dstDatatype>* pGy, int gyStep, Ipp<dstDatatype>* pMag, int magStep, Ipp32f* pAngle, int angleStep, IppiSize dstRoiSize, IppiMaskSize maskSize, IppNormType normType, IppiBorderType borderType, const Ipp<srcDatatype> borderValue[3], Ipp8u* pBuffer);

Supported values for mod:

8u16s_C3C1R 16s32f_C3C1R 16u32f_C3C1R 32f_C3C1R

Include Files

ippi.h
Domain Dependencies

Headers: ippcore.h, ippvm.h, ipps.h
Libraries: ippcore.lib, ippvm.lib, ipps.lib

Parameters

- **pSrc**: Pointer to the source image ROI.
- **srcStep**: Distance, in bytes, between the starting points of consecutive lines in the source image.
- **pGx**: Pointer to the destination image ROI for the X component of the gradient vector (in Cartesian form).
- **gxStep**: Distance, in bytes, between the starting points of consecutive lines in the X-component destination image.
- **pGy**: Pointer to the destination image ROI for the Y component of the gradient vector (in Cartesian form).
- **gyStep**: Distance, in bytes, between the starting points of consecutive lines in the Y-component destination image.
- **pMag**: Pointer to the magnitude of the gradient destination image ROI (in polar gradient form).
- **magStep**: Distance, in bytes, between the starting points of consecutive lines in the magnitude of the gradient destination image.
- **pAngle**: Pointer to the destination image ROI for the angle (in polar gradient form).
- **angleStep**: Distance, in bytes, between the starting points of consecutive lines in the angle destination image.
- **dstRoiSize**: Size of the source and destination image ROI in pixels.
- **maskSize**: Predefined mask of IppiMaskSize type.
- **normType**: Normalization mode of IppNormType type.
- **borderType**: Type of border. Possible values are:
  - ippBorderConst: Values of all border pixels are set to constant.
  - ippBorderRepl: Border is replicated from the edge pixels.
  - ippBorderInMem: Border is obtained from the source image pixels in memory.
  - ippBorderMirror: Border pixels are mirrored from the source image boundary pixels.
- **borderValue**: Constant value to assign to pixels in the constant border (not applicable for other border types).
- **pBuffer**: Pointer to the work buffer.

Description

This function operates with ROI (see Regions of Interest in Intel IPP).
Before using this function, compute the size of the work buffer using the `ippiGradientVectorGetBufferSize` function.

---

**NOTE**

Any of the `pGx`, `pGy`, `pMag`, and `pAngle` output parameters can be `NULL`. This means that the parameter(s) is not requested.

---

This function operates on "gray" single-channel (C1 flavors) and color (C3 flavors) images.

**Single-channel image (C1) input :**

If input is a single-channel image, the `ippiGradientVectorScharr` function computes the gradient vector at each pixel of the source image ROI and stores the result either in Cartesian (`pGx` and `pGy`) and/or polar (`pMag` and `pAngle`) form, or any combination of these possible outputs.

Cartesian projections $G_x$ and $G_y$ are stored in the `pGx` and `pGy` buffers, respectively. The formulas below describe the algorithm for the 3x3 Scharr operator:

$$
G_x = \begin{bmatrix} 3 & 0 & -3 \\ 10 & 0 & -10 \\ 3 & 0 & -3 \end{bmatrix} * A \\
G_y = \begin{bmatrix} 3 & 10 & 3 \\ 0 & 0 & 0 \\ -3 & -10 & -3 \end{bmatrix} * A
$$

where

- $A$ is the source image
- $*$ means two-dimensional convolution
- $G_x$ and $G_y$ are X and Y components of the gradient

The magnitude of the gradient is computed according to the `normType` value by the following formulas:

**L1 normalization:**

$$ G = |G_x| + |G_y| $$

**L2 normalization:**

$$ G = \sqrt{G_x^2 + G_y^2} $$

The value of angle between $G_x$ and $G_y$ is computed by the formula:

$$ angle = \arctan \frac{G_y}{G_x} $$

**Color image (C3) input :**
If input is a color image, the ippiGradientVectorScharr function computes the spatial image derivatives \( G_x \) and \( G_y \) for each channel of the image using the specified differential operator. For each pixel \((x, y)\) this function chooses the derivatives for which \( L2(G_x, G_y) \) is the maximal value and stores them in the \( pGx \) and \( pGy \) output arrays. In other words, for each pixel of a color image the function returns the derivatives composing the largest gradient across all channels.

The examples of using this function are similar to the examples provided with the GradientVectorSobel function description.

**Return Values**

- **ippStsNoErr**: Indicates no error.
- **ippStsNullPtrErr**: Indicates an error when \( pSrc \) or \( pBuffer \) is NULL.
- **ippStsSizeErr**: Indicates an error when one of the fields of \( dstRoiSize \) has a zero or negative value.
- **ippStsMaskSizeErr**: Indicates an error when \( maskSize \) has an illegal value.
- **ippStsStepErr**: Indicates an error when:
  - \( srcStep, gxStep, gyStep, magStep, \) or \( angleStep \) has a zero or negative value
  - \( srcStep, gxStep, gyStep, magStep, \) or \( angleStep \) is not a multiple of the image data size (4 for floating-point images or 2 for short integer images)
- **ippStsBadArgErr**: Indicates an error when \( normType \) has an illegal value.
- **ippStsBorderErr**: Indicates an error when \( borderType \) has an incorrect value.

**See Also**

- **Structures and Enumerators**
- **Regions of Interest in Intel IPP**
- **GradientVectorGetBufferSize** Computes the size of the work buffer for the ippiGradientVector{Sobel|Scharr|Prewitt} functions.

**GradientVectorSobel**

*Computes gradient vectors of an image using the Sobel operator.*

**Syntax**

```c
IppStatus ippiGradientVectorSobel_<mod> (const Ipp<srcDatatype>* pSrc, int srcStep, Ipp<dstDatatype>* pGx, int gxStep, Ipp<dstDatatype>* pGy, int gyStep, Ipp<dstDatatype>* pMag, int magStep, Ipp32f* pAngle, int angleStep, IppiSize dstRoiSize, IppiMaskSize maskSize, IppNormType normType, IppiBorderType borderType, Ipp<srcDatatype> borderValue, Ipp8u* pBuffer);
```

**Supported values for** \( \text{mod} \):

- 8u16s_C1R
- 16s32f_C1R
- 16u32f_C1R
- 32f_C1R
IppStatus ippiGradientVectorSobel_<mod> (const Ipp<srcDatatype>* pSrc, int srcStep, Ipp<dstDatatype>* pGx, int gxStep, Ipp<dstDatatype>* pGy, int gyStep, Ipp<dstDatatype>* pMag, int magStep, Ipp32f* pAngle, int angleStep, IppiSize dstRoiSize, IppiMaskSize maskSize, IppNormType normType, IppiBorderType borderType, const Ipp<srcDatatype> borderValue[3], Ipp8u* pBuffer);

Supported values for mod:

8u16s_C3C1R 16s32f_C3C1R 16u32f_C3C1R 32f_C3C1R

Include Files

ippi.h

Domain Dependencies

Headers: ippcore.h, ippvm.h, ipps.h
Libraries: ippcore.lib, ippvm.lib, ipps.lib

Parameters

pSrc

Pointer to the source image ROI.

srcStep

Distance, in bytes, between the starting points of consecutive lines in the source image.

pGx

Pointer to the destination image ROI for the X component of the gradient vector (in Cartesian form).

gxStep

Distance, in bytes, between the starting points of consecutive lines in the X-component destination image.

pGy

Pointer to the destination image ROI for the Y component of the gradient vector (in Cartesian form).

gyStep

Distance, in bytes, between the starting points of consecutive lines in the Y-component destination image.

pMag

Pointer to the magnitude of the gradient destination image ROI (in polar gradient form).

magStep

Distance, in bytes, between the starting points of consecutive lines in the magnitude of the gradient destination image.

pAngle

Pointer to the destination image ROI for the angle (in polar gradient form).

angleStep

Distance, in bytes, between the starting points of consecutive lines in the angle destination image.

dstRoiSize

Size of the source and destination image ROI in pixels.

maskSize

Predefined mask of IppiMaskSize type.

normType

Normalization mode of IppNormType type.

borderType

Type of border. Possible values are:

ippBorderConst

Values of all border pixels are set to constant.

ippBorderRepl

Border is replicated from the edge pixels.
ippBorderInMem

Border is obtained from the source image pixels in memory.

ippBorderMirror

Border pixels are mirrored from the source image boundary pixels.

borderValue

Constant value to assign to pixels in the constant border (not applicable for other border types).

pBuffer

Pointer to the work buffer.

Description

This function operates with ROI (see Regions of Interest in Intel IPP).

Before using this function, compute the size of the work buffer using the GradientVectorGetBufferSize function.

NOTE

Any of the pGx, pGy, pMag, and pAngle output parameters can be NULL. This means that the parameter(s) is not requested.

This function operates on "gray" single-channel (C1 flavors) and color (C3 flavors) images.

Single-channel image (C1) input:

If input is a single-channel image, the ippiGradientVectorSobel function computes the gradient vector at each pixel of the source image ROI and stores the result either in Cartesian (pGx and pGy) and/or polar (pMag and pAngle) form, or any combination of these possible outputs.

Cartesian projections \( G_x \) and \( G_y \) are stored in the pGx and pGy buffers, respectively. The formulas below describe the algorithm for the 3x3 Sobel operator:

\[
G_x = \begin{bmatrix}
-1 & 0 & 1 \\
-2 & 0 & 2 \\
-1 & 0 & 1 \\
\end{bmatrix} * A \\
G_y = \begin{bmatrix}
1 & 2 & 1 \\
0 & 0 & 0 \\
-1 & -2 & -1 \\
\end{bmatrix} * A
\]

for the 5x5 Sobel operator:

\[
G_x = \begin{bmatrix}
-1 & -2 & 0 & 2 & 1 \\
-4 & -8 & 0 & 8 & 4 \\
-6 & -12 & 0 & 12 & 6 \\
-4 & -8 & 0 & 8 & 4 \\
-1 & -2 & 0 & 2 & 1 \\
\end{bmatrix} * A \\
G_y = \begin{bmatrix}
1 & 4 & 6 & 4 & 1 \\
0 & 8 & 12 & 8 & 2 \\
2 & 0 & 0 & 0 & 0 \\
-2 & -8 & -12 & -8 & -2 \\
-1 & -4 & -6 & -4 & -1 \\
\end{bmatrix} * A
\]

where
is the source image
* means two-dimensional convolution
$G_x$ and $G_y$ are X and Y components of the gradient

The magnitude of the gradient is computed according to the normType value by the following formulas:

L1 normalization:

$$G = |G_x| + |G_y|$$

L2 normalization:

$$G = \sqrt{G_x^2 + G_y^2}$$

The value of angle between $G_x$ and $G_y$ is computed by the formula:

$$angle = \arctan \frac{G_y}{G_x}$$

**Color image (C3) input:**

If input is a color image, the `ippiGradientVectorSobel` function computes the spatial image derivatives $G_x$ and $G_y$ for each channel of the image using the specified differential operator. For each pixel $(x, y)$ this function chooses the derivatives for which $L2(G_x, G_y)$ is the maximal value and stores them in the $pGx$ and $pGy$ output arrays. In other words, for each pixel of a color image the function returns the derivatives composing the largest gradient across all channels.

**Return Values**

- `ippStsNoErr`: Indicates no error.
- `ippStsNullPtrErr`: Indicates an error when `pSrc` or `pBuffer` is NULL.
- `ippStsSizeErr`: Indicates an error when one of the fields of `dstRoiSize` has a zero or negative value.
- `ippStsMaskSizeErr`: Indicates an error when `maskSize` has an illegal value.
- `ippStsStepErr`: Indicates an error when:
  - `srcStep`, `gxStep`, `gyStep`, `magStep`, or `angleStep` has a zero or negative value
  - `srcStep`, `gxStep`, `gyStep`, `magStep`, or `angleStep` is not a multiple of the image data size (4 for floating-point images or 2 for short integer images)
- `ippStsBadArgErr`: Indicates an error when `normType` has an illegal value.
- `ippStsBorderErr`: Indicates an error when `borderType` has an incorrect value.

**Example**

To better understand usage of the `ippiGradientVectorSobel` function, refer to the following examples in the examples archive available for download from https://software.intel.com/en-us/ipp-manual-examples:
GradientVectorSobel1.c
GradientVectorSobel2.c

See Also
Structures and Enumerators
Regions of Interest in Intel IPP
GradientVectorGetBufferSize Computes the size of the work buffer for the
ippiGradientVector{Sobel|Scharr|Prewitt} functions.

Flood Fill Functions

This section describes functions performing flood filling of connected areas. Flood filling means that a group of connected pixels with close values is filled with, or is set to, a certain value. The flood filling process starts with a specified point ("seed") and continues until it reaches the image ROI boundary or cannot find any new pixels to fill due to a large difference in pixel values. For every pixel filled, the functions analyze neighbor pixels:
- 4 neighbors (except diagonal neighbors); this kind of connectivity is called 4-connectivity and the corresponding function name includes 4Con, or
- 8 neighbors (diagonal neighbors included); this kind of connectivity is called 8-connectivity and the corresponding function name includes 8Con.

**Pixels Connectivity Patterns**

![Diagram showing pixels connectivity patterns]

These functions can be used for:
- segmenting a grayscale image into a set of uni-color areas,
- marking each connected component with individual color for bi-level images.

**FloodFillGetSize**

*Calculates size of temporary buffer for flood filling operation.*

**Syntax**

```c
IppStatus ippiFloodFillGetSize(IppiSize roiSize, int* pBufSize);
```

**Include Files**

ippcv.h

**Domain Dependencies**

Headers: ippcore.h, ippvm.h, ipps.h,ippi.h
Libraries: ippcore.lib, ippvm.lib, ipps.lib,ippi.lib
Parameters

roiSize
Size of the source image ROI in pixels.

pBufSize
Pointer to the variable that returns the size of the temporary buffer.

Description

This function calculates the size of the temporary buffer to be used by the function ippiFloodFill.

Return Values

ippStsNoErr
Indicates no error. Any other value indicates an error or a warning.

ippStsNullPtrErr
Indicates an error condition if pBufSize pointer is NULL.

ippStsSizeErr
Indicates an error condition if roiSize has a field with zero or negative value.

FloodFillGetSize_Grad

Calculates size of temporary buffer for the gradient flood filling.

Syntax

IppStatus ippiFloodFillGetSize_Grad(IppiSize roiSize, int* pBufSize);

Include Files

ippcv.h

Domain Dependencies

Headers: ippcore.h, ippvm.h, ipps.h,ippi.h
Libraries: ippcore.lib, ippvm.lib, ipps.lib,ippi.lib

Parameters

roiSize
Size of the source image ROI in pixels.

pBufSize
Pointer to the variable that returns the size of the temporary buffer.

Description

This function calculates the size of the temporary buffer to be used by the function ippiFloodFill_Grad.

Return Values

ippStsNoErr
Indicates no error. Any other value indicates an error or a warning.

ippStsNullPtrErr
Indicates an error condition if pBufSize pointer is NULL.

ippStsSizeErr
Indicates an error condition if roiSize has a field with zero or negative value.
FloodFill
Performs flood filling of connected area.

Syntax

Case 1: Operations on one-channel data

IppStatus ippiFloodFill_4Con_<mod>(Ipp<datatype>* pImage, int imageStep, IppiSize roiSize, IppiPoint seed, Ipp<datatype> newVal, IppiConnectedComp* pRegion, Ipp8u* pBuffer);

IppStatus ippiFloodFill_8Con_<mod>(Ipp<DataType>* pImage, int imageStep, IppiSize roiSize, IppiPoint seed, Ipp<datatype> newVal, IppiConnectedComp* pRegion, Ipp8u* pBuffer);

Supported values for mod:

8u_C1IR   16u_C1IR   32s_C1IR   32f_C1IR

Case 2: Operations on three-channel data

IppStatus ippiFloodFill_4Con_<mod>(Ipp<datatype>* pImage, int imageStep, IppiSize roiSize, IppiPoint seed, Ipp<datatype>* pNewVal, IppiConnectedComp* pRegion, Ipp8u* pBuffer);

IppStatus ippiFloodFill_8Con_<mod>(Ipp<DataType>* pImage, int imageStep, IppiSize roiSize, IppiPoint seed, Ipp<datatype>* pNewVal, IppiConnectedComp* pRegion, Ipp8u* pBuffer);

Supported values for mod:

8u_C3IR   16u_C3IR   32f_C3IR

Include Files
ippcv.h

Domain Dependencies
Headers: ippcore.h, ippvm.h, ipps.h, ippi.h
Libraries: ippcore.lib, ippvm.lib, ipps.lib, ippi.lib

Parameters

pImage       Pointer to the ROI in the source and destination image (for the in-place operation).
imageStep    Distance in bytes between starts of consecutive lines in the image buffer.
roiSize      Size of the image ROI in pixels.
seed         Initial point.
newVal       Value to fill with for one-channel data.
pNewVal      Pointer to the vector containing values to fill with for three-channel data.
**pRegion**
Pointer to the connected components structure that stores information about the refilled area.

**pBuffer**
Pointer to the temporary buffer.

**Description**
This function operates with ROI (see Regions of Interest in Intel IPP).
This function performs flood filling (see Flood Filling Functions) of the group of connected pixels whose pixel values are equal to the value in the seed point. Values of these pixel is set to the newVal value.

The function requires a temporary buffer whose size should be computed with the function ippiFloodFillGetSize beforehand.

The functions with the "_4con" suffixes check 4-connected neighborhood of each pixel, that is, side neighbors. The functions with the "_8con" suffixes check 8-connected neighborhood of each pixel, that is, side and corner neighbors. See Figure Pixels Connectivity Patterns”.

**Return Values**
- **ippStsNoErr**
  Indicates no error. Any other value indicates an error or a warning.
- **ippStsNullPtrErr**
  Indicates an error condition if one of the specified pointers is NULL.
- **ippStsSizeErr**
  Indicates an error condition if roiSize has a field with zero or negative value.
- **ippStsStepErr**
  Indicates an error condition if imageStep is less than pRoiSize.width*<pixelSize>.
- **ippStsNotEvenStepErr**
  Indicates an error condition if steps for floating-point images are not divisible by 4, or steps for 16-bit integer images are not divisible by 2.
- **ippStsOutOfRangeErr**
  Indicates an error condition if the seed point is out of ROI.

**Example**
To better understand usage of the ippiFloodFill function, refer to the following example in the examples archive available for download from https://software.intel.com/en-us/ipp-manual-examples:

FloodFill.c

**FloodFill_Grad**
Performs gradient flood filling of connected area on an image.

**Syntax**

**Case 1: Operations on one-channel data**

IppStatus ippiFloodFill_Grad4Con_<mod>(Ipp<DataType>* pImage, int imageStep, IppiSize roiSize, IppiPoint seed, Ipp<datatype> newVal, Ipp<datatype> minDelta, Ipp<datatype> maxDelta, IppiConnectedComp* pRegion, Ipp8u* pBuffer);
IppStatus ippiFloodFill_Grad8Con_<mod>(Ipp<DataType>* pImage, int imageStep, IppiSize roiSize, IppiPoint seed, Ipp<datatype> newVal, Ipp<datatype> minDelta, Ipp<datatype> maxDelta, IppiConnectedComp* pRegion, Ipp8u* pBuffer);

Supported values for mod:

8u_C1IR   16u_C1IR   32f_C1IR

Case 2: Operations on three-channel data
IppStatus ippiFloodFill_Grad4Con_<mod>(Ipp<DataType>* pImage, int imageStep, IppiSize roiSize, IppiPoint seed, Ipp<datatype>* pNewVal, Ipp<datatype>* pMinDelta, Ipp<datatype>* pMaxDelta, IppiConnectedComp* pRegion, Ipp8u* pBuffer);

IppStatus ippiFloodFill_Grad8Con_<mod>(Ipp<DataType>* pImage, int imageStep, IppiSize roiSize, IppiPoint seed, Ipp<datatype>* pNewVal, Ipp<datatype>* pMinDelta, Ipp<datatype>* pMaxDelta, IppiConnectedComp* pRegion, Ipp8u* pBuffer);

Supported values for mod:

8u_C3IR   16u_C3IR   32f_C3IR

Include Files

ippcv.h

Domain Dependencies

Headers: ippcore.h, ippvm.h, ipps.h,ippi.h
Libraries: ippcore.lib, ippvm.lib, ipps.lib,ippi.lib

Parameters

pImage        Pointer to the ROI in the source and destination image (in-place operation).
imageStep     Distance in bytes between starts of consecutive lines in the image buffer.
roiSize       Size of the image ROI in pixels.
seed          Initial point.
minDelta      Minimum difference between neighbor pixels for one-channel data.
maxDelta      Maximum difference between neighbor pixels for one-channel data.
newVal        Value to fill with for one-channel data.
pMinDelta     Pointer to the minimum differences between neighbor pixels for three-channel images.
pMaxDelta     Pointer to the maximum differences between neighbor pixels for three-channel images.
pNewVal       Pointer to the vector containing values to fill with for three-channel data.
pRegion       Pointer to the connected components structure that stores information about the refilled area.
pBuffer       Pointer to the temporary buffer.
Description
This function operates with ROI (see Regions of Interest in Intel IPP).

This function performs flood filling (see Flood Filling Functions) of the group of connected pixels in the seed pixel neighborhoods whose pixel values \( v \) satisfy the following conditions:

\[ v_0 - d_{lw} \leq v \leq v_0 + d_{up}, \]

where \( v_0 \) is the value of at least one of the current pixel neighbors, which already belongs to the refilled area, and \( d_{lw}, d_{up} \) are \( \text{minDelta} \), \( \text{maxDelta} \), respectively. Values of these pixels are set to the \( \text{newVal} \) value.

The function requires a temporary buffer whose size should be computed with the function \text{ippiFloodFillGetSize_Grad} beforehand.

The functions with the "\_4con" suffixes check 4-connected neighborhood of each pixel, i.e., side neighbors. The functions with the "\_8con" suffixes check 8-connected neighborhood of each pixel, i.e., side and corner neighbors. See Figure Pixels Connectivity Patterns”.

Return Values

- \text{ippStsNoErr} Indicates no error. Any other value indicates an error or a warning.
- \text{ippStsNullPtrErr} Indicates an error condition if one of the specified pointers is NULL.
- \text{ippStsSizeErr} Indicates an error condition if \text{roiSize} has a field with zero or negative value.
- \text{ippStsStepErr} Indicates an error condition if \text{imageStep} is less than \text{pRoiSize.width} \times \text{<pixelSize>}
- \text{ippStsNotEvenStepErr} Indicates an error condition if steps for floating-point images are not divisible by 4, or steps for 16-bit integer images are not divisible by 2.
- \text{ippStsOutOfRangeErr} Indicates an error condition if the \text{seed} point is out of ROI.

Example
To better understand usage of the \text{ippiFloodFill_Grad} function, refer to the following example in the examples archive available for download from https://software.intel.com/en-us/ipp-manual-examples:

\text{FloodFill_Grad.c}

FloodFill_Range
Performs flood filling of pixels with values in the specified range in the connected area on an image.

Syntax
Case 1: Operations on one-channel data

\begin{verbatim}
IppStatus ippiFloodFill_Range4Con_<mod>(Ipp<DataType>* pImage, int imageStep, IppiSize roiSize, IppiPoint seed, Ipp<datatype> newVal, Ipp<datatype> minDelta, Ipp<datatype> maxDelta, IppConnectedComp* pRegion, Ipp8u* pBuffer);
\end{verbatim}
IppStatusippiFloodFill_Range8Con_<mod>(Ipp<DataType>* pImage, int imageStep, IppiSize roiSize, IppiPoint seed, Ipp<datatype> newVal, Ipp<datatype> minDelta, Ipp<datatype> maxDelta, IppiConnectedComp* pRegion, Ipp8u* pBuffer);

Supported values for mod:

8u_C1IR      16u_C1IR      32f_C1IR

Case 2: Operations on three-channel data
IppStatusippiFloodFill_Range4Con_<mod>(Ipp<DataType>* pImage, int imageStep, IppiSize roiSize, IppiPoint seed, Ipp<datatype>* pNewVal, Ipp<datatype>* pMinDelta, Ipp<datatype>* pMaxDelta, IppiConnectedComp* pRegion, Ipp8u* pBuffer);
IppStatusippiFloodFill_Range8Con_<mod>(Ipp<DataType>* pImage, int imageStep, IppiSize roiSize, IppiPoint seed, Ipp<datatype>* pNewVal, Ipp<datatype>* pMinDelta, Ipp<datatype>* pMaxDelta, IppiConnectedComp* pRegion, Ipp8u* pBuffer);

Supported values for mod:

8u_C3IR      16u_C3IR      32f_C3IR

Include Files
ippcv.h

Domain Dependencies
Headers: ippcore.h, ippvm.h, ipps.h,ippi.h
Libraries: ippcore.lib, ippvm.lib, ipps.lib,ippi.lib

Parameters
pImage
Pointer to the ROI in the source and destination image (in-place operation).
imageStep
Distance in bytes between starts of consecutive lines in the image buffer.
roiSize
Size of the image ROI in pixels.
seed
Initial point.
minDelta
Minimum difference between neighbor pixels for one-channel data.
maxDelta
Maximum difference between neighbor pixels for one-channel data.
newVal
Value to fill with for one-channel data.
pMinDelta
Pointer to the minimum differences between neighbor pixels for three-channel images.
pMaxDelta
Pointer to the maximum differences between neighbor pixels for three-channel images.
pNewVal
Pointer to the vector containing values to fill with for three-channel data.
pRegion
Pointer to the connected components structure that stores information about the refilled area.
pBuffer
Pointer to the temporary buffer.
Description
This function operates with ROI (see Regions of Interest in Intel IPP).

This function performs flood filling (see Flood Filling Functions) of the group of connected pixels in the seed pixel neighborhoods whose pixel values v satisfy the following conditions:

\[ v_0 - d_{lw} \leq v \leq v_0 + d_{up}, \]

where \( v_0 \) is the pixel value of the seed point, and \( d_{lw}, d_{up} \) are minDelta, maxDelta, respectively. Values of these pixels are set to the newVal value.

The function requires a temporary buffer whose size should be computed with the function ippiFloodFillGetSize_Grad beforehand.

The functions with the "_4con" suffixes check 4-connected neighborhood of each pixel, i.e., side neighbors. The functions with the "_8con" suffixes check 8-connected neighborhood of each pixel, i.e., side and corner neighbors. See Figure Pixels Connectivity Patterns

Return Values
ippStsNoErr Indicates no error. Any other value indicates an error or a warning.
ippStsNullPtrErr Indicates an error condition if one of the specified pointers is NULL.
ippStsSizeErr Indicates an error condition if roiSize has a field with zero or negative value.
ippStsStepErr Indicates an error condition if imageSize is less than pRoiSize.width*<pixelSize>.
ippStsNotEvenStepErr Indicates an error condition if steps for floating-point images are not divisible by 4, or steps for 16-bit integer images are not divisible by 2.
ippStsOutOfRangeErr Indicates an error condition if the seed point is out of ROI.

Motion Analysis and Object Tracking

FGMMGetBufferSize
Computation of the state structure for the Gaussian mixture model foreground/background subtraction.

Syntax
IppStatus ippiFGMMGetBufferSize_8u_C3R(IppiSize roi, int maxNGauss, int* pSpecSize);

Include Files
ippcv.h

Domain Dependencies
Headers: ippcore.h, ippvm.h, ipps.h,ippi.h
Libraries: ippcore.lib, ippvm.lib, ipps.lib,ippi.lib
Parameters

roi
Size of the source image ROI, in pixels.

maxNGauss
Maximal size of the Gaussian mixture components.

pSpecSize
Pointer to the size of the IppFGMMSpec_8u_C3R structure.

Description
This function operates with ROI.
This function computes the size of the IppFGMMSpec_8u_C3R structure for the FGMMForeground and FGMMBackground functions.

Return Values

ippStsNoErr
Indicates no error.

ippStsNullPtrErr
Indicates an error when pSpecSize is NULL.

ippStsSizeErr
Indicates an error when roi is less than, or equal to zero.

ippStsBadArgErr
Indicates an error when maxNumGauss is less than, or equal to zero.

See Also
Regions of Interest in Intel IPP
FGMMForeground Performs the Gaussian mixture model foreground subtraction.
FGMMBackground Returns the updated background image.

FGMInit
Initializes the state structure for the Gaussian mixture model foreground/background subtraction.

Syntax
IppStatus ippiFGMInit_8u_C3R(IppiSize roi, int maxNGauss, IppFGMModel* pModel, IppFGMMState_8u_C3R* pState);

Include Files
ippcv.h

Domain Dependencies
Headers: ippcore.h, ippvm.h, ipps.h,ippi.h
Libraries: ippcore.lib, ippvm.lib, ipps.lib,ippi.lib

Parameters

roi
Size of the source image ROI, in pixels.

maxNGauss
Maximal size of the Gaussian mixture components.

pModel
Pointer to the IppFGMModel structure containing parameters for the model. If pModel is NULL, the default parameters are applied.

pState
Pointer to the IppFGMMState_8u_C3R state structure.
Description
This function operates with ROI.

This function initializes the IppFGMMState_8u_C3R state structure for the FGMMForeground and FGMMBackground functions.

Before using this function, you need to compute the size of the state structure using the FGMMGetBufferSize function.

Return Values
- **ippStsNoErr**: Indicates no error.
- **ippStsNullPtrErr**: Indicates an error when pModel or pState is NULL.
- **ippStsSizeErr**: Indicates an error when roi is less than, or equal to zero.
- **ippStsBadArgErr**: Indicates an error when maxNGauss is less than, or equal to zero.

See Also
- **Regions of Interest in Intel IPP**
- **FGMMForeground**: Performs the Gaussian mixture model foreground subtraction.
- **FGMMBackground**: Returns the updated background image.
- **FGMMGetBufferSize**: Computes the size of the state structure for the Gaussian mixture model foreground/background subtraction.

**FGMMForeground**
Performs the Gaussian mixture model foreground subtraction.

Syntax

```c
IppStatus ippiFGMMForeground_8u_C3R(const Ipp8u* pSrc, int srcStep, Ipp8u* pDst, int dstStep, IppiSize roi, IppFGMMState_8u_C3R* pState, IppFGMModel* pModel, double learning_rate);
```

Include Files
```
ippcv.h
```

Domain Dependencies
- **Headers**: ippcore.h, ippvm.h, ipps.h, ippi.h
- **Libraries**: ippcore.lib, ippvm.lib, ipps.lib, ippi.lib

Parameters
- **pSrc**: Pointer to the source image.
- **srcStep**: Distance, in bytes, between the starting points of consecutive lines in the source image.
- **pDst**: Pointer to the one-channel Ipp8u mask of foreground.
- **dstStep**: Distance, in bytes, between the starting points of consecutive lines in the destination image.
- **roi**: Size of the source image ROI, in pixels.
**learning_rate**

Speed of algorithm learning.

**pState**

Pointer to the IppFGMMState_8u_C3R state structure.

**pModel**

Pointer to the IppFGMModel structure containing parameters for the model. If **pModel** is NULL, the parameters are the same as in a previous call.

## Description

This function operates with ROI.

This function implements the Gaussian mixture model foreground subtraction described in [ZIVKOVIC04]. The foreground mask is stored in **pDst**.

Before using this function, you need to compute the size of the IppFGMMState_8u_C3R state structure and initialize the structure using the FGMMGetBufferSize and FGMMInit functions, respectively.

For an example on how to use this function, refer to the example provided with the FGMMBackground function description.

### Return Values

- **ippStsNoErr**: Indicates no error.
- **ippStsNullPtrErr**: Indicates an error when **pSrc**, **pDst**, **pModel**, or **pState** is NULL.
- **ippStsSizeErr**: Indicates an error when **roi** is less than, or equal to zero.
- **ippStsStepErr**: Indicates an error when **srcStep** or **dstStep** is less than, or equal to zero.

## See Also

- Regions of Interest in Intel IPP
- FGMMGetBufferSize Computes the size of the state structure for the Gaussian mixture model foreground/background subtraction.
- FGMMInit Initializes the state structure for the Gaussian mixture model foreground/background subtraction.
- FGMMBackground Returns the updated background image.

### FGMMBackground

**Returns the updated background image.**

## Syntax

```c
IppStatus ippiFGMMBackground_8u_C3R(Ipp8u* pDst, int dstStep, IppiSize roi,
IppFGMMState_8u_C3R* pState);
```

### Include Files

ippcv.h

### Domain Dependencies

**Headers**: ippcore.h, ippvm.h, ipps.h, ippi.h

**Libraries**: ippcore.lib, ippvm.lib, ipps.lib, ippi.lib
Parameters

- **pDst**: Pointer to the three-channel background image.
- **dstStep**: Distance, in bytes, between the starting points of consecutive lines in the destination image.
- **roi**: Size of the source image ROI, in pixels.
- **pState**: Pointer to the IppFGMState_8u_C3R state structure.

Description

This function implements the Gaussian mixture model background subtraction described in [ZIVKOVIC04]. The function returns the three-channel Ipp8u background image.

Before using this function, you need to compute the size of the IppFGMState_8u_C3R state structure and initialize the structure using the FGMMGetBufferSize and FGMMInit functions, respectively.

Return Values

- **ippStsNoErr**: Indicates no error.
- **ippStsNullPtrErr**: Indicates an error when *pDst* or *pState* is NULL.
- **ippStsSizeErr**: Indicates an error when *roi* is less than, or equal to zero.
- **ippStsStepErr**: Indicates an error when *dstStep* is less than, or equal to zero.

Example

To better understand usage of the ippiFGMMBackground function, refer to the following example in the examples archive available for download from https://software.intel.com/en-us/ipp-manual-examples:

FGMMBackground.c

See Also

- Regions of Interest in Intel IPP
- FGMMGetBufferSize Computes the size of the state structure for the Gaussian mixture model foreground/background subtraction.
- FGMMInit Initializes the state structure for the Gaussian mixture model foreground/background subtraction.

Motion Template Functions

This section describes a motion templates function. This function generates motion templates images to rapidly determine where, how, and in which direction the motion occurred. The algorithms are based on [Davis97], and [Davis99]. The function operates on images that are the output of frame or background differencing, or other image segmentation operations. Thus, the input and output image types are all grayscale, that is, one color channel. The pixel types can be 8u, 8s, or 32f.

Motion Representation

Figure Motion Image History” (a) shows capturing a foreground silhouette of the moving object or person. As the person or object moves, copying the most recent foreground silhouette as the highest values in the motion history image creates a “layered history” of the resulting motion. Typically, this “highest value” is just a floating-point timestamp of time since the code has been running in milliseconds. Figure Motion Image
History” (b) shows the result that may be called the **Motion History Image (MHI)**. The MHI in Figure 6a represents how the motion took place. A pixel level or a time delta threshold, as appropriate, is set such that pixel values in the MHI that fall below that threshold are set to zero.

**Motion Image History**

![Image showing motion history](image)

The most recent motion has the highest value, earlier motions have decreasing values subject to a threshold below which the value is set to zero.

**Updating MHI Images**

Generally, floating point images are used because system time differences, that is, time elapsing since the application was launched, are read in milliseconds to be further converted into a floating point number which is the value of the most recent silhouette. Then follows writing this current silhouette over the past silhouettes with subsequent thresholding away pixels that are too old to create the MHI.

**UpdateMotionHistory**

*Updates motion history image using motion silhouette at given timestamp.*

**Syntax**

```c
IppStatus ippiUpdateMotionHistory_<mod>(const Ipp<srcDatatype>* pSilhouette, int silhStep, Ipp32f* pMhi, int mhiStep, IppSize roiSize, Ipp32f timeStamp, Ipp32f mhiDuration);
```

Supported values for `mod`:

- `8u32f_C1IR`
- `16u32f_C1IR`
- `32f_C1IR`

**Include Files**

`ippcv.h`
Domain Dependencies

Headers:  ippcore.h, ippvm.h, ipps.h,ippi.h
Libraries:  ippcore.lib, ippvm.lib, ipps.lib,ippi.lib

Parameters

pSilhouette  Pointer to the silhouette image ROI that has non-zero values for those pixels where the motion occurs.
roiSize  Size of the image ROI in pixels.
silhStep  Distance in bytes between starts of consecutive lines in the silhouette image.
pMhi  Pointer to the motion history image which is both an input and output parameter.
mhiStep  Distance in bytes between starts of consecutive lines in the motion history image.
timeStamp  Timestamp in milliseconds.
mhiDuration  Threshold for MHI pixels. MHI motions older than this threshold are deleted.

Description

This function operates with ROI (see Regions of Interest in Intel IPP).
This function updates the motion history image. It sets MHI pixels to the current timestamp value, if their values are non-zero.
The function deletes MHI pixels, if their values are less than the mhiDuration timestamp, that is, the pixels are "old."

Return Values

ippStsNoErr  Indicates no error. Any other value indicates an error or a warning.
ippStsNullPtrErr  Indicates an error condition if one of the specified pointers is NULL.
ippStsSizeErr  Indicates an error condition if roiSize has a field with zero or negative value.
ippStsStepErr  Indicates an error condition if mhiStep or silhStep is less than roiSize.width * <pixelSize>.
ippStsNotEvenStepErr  Indicates an error condition if the step value is not divisible by 2 for 16u images, and by 4 for 32f images.
ippStsOutOfRangeErr  Indicates an error when mhiDuration is negative.

Optical Flow

This section describes the functions that calculate the optical flow using the pyramidal Lucas-Kanade algorithm [Bou99].
The optical flow between two images is generally defined as an apparent motion of image brightness. Let \( I(x, y, t) \) be the image brightness that changes in time to provide an image sequence.

Optical flow coordinates
\[ u = \frac{\partial x}{\partial t}, \quad v = \frac{\partial y}{\partial t} \]

can be found from so called *optical flow constraint equation*:

\[ \frac{\partial I}{\partial t} = \frac{\partial I}{\partial x} u + \frac{\partial I}{\partial y} v \]

The Lucas-Kanade algorithm assumes that the group of adjacent pixels has the same velocity and founds the approximate solution of the above equation using the least square method.

**OpticalFlowPyrLKGetSize**

*Computes the size of the pyramidal optical flow state structure.*

**Syntax**

```c
IppStatus ippiOpticalFlowPyrLKGetSize(int winSize, IppiSize roiSize, IppDataType dataType, IppHintAlgorithm hint, int* pStateSize);
```

**Include Files**

ippcv.h

**Domain Dependencies**

- **Headers**: ippcore.h, ippvm.h, ipps.h,ippi.h
- **Libraries**: ippcore.lib, ippvm.lib, ipps.lib,ippi.lib

**Parameters**

- `winSize`: Size of the search window of each pyramid level.
- `roiSize`: Maximal size of the source image (zero level of the pyramid) ROI, in pixels.
- `dataType`: Data type of the image. Possible values: ipp8u, ipp16u, or ipp32f.
- `hint`: Option to select the algorithmic implementation of the transform function.
- `pStateSize`: Pointer to the size of the state structure.

**Description**

This function operates with ROI (see **Regions of Interest in Intel IPP**).

This function computes the size of the `pState` structure that is required to calculate the optical flow between two images in the centered window of size `winSize*winSize` using the pyramidal Lucas-Kanade [Bou99] algorithm. The `hint` argument specifies the computation algorithm. The `pState` structure is used by the `ippiOpticalFlowPyrLK` function and can be applied to process images with size not greater than `roiSize`.

**Return Values**

- `ippStsNoErr`: Indicates no error. Any other value indicates an error or a warning.
- `ippStsNullPtrErr`: Indicates an error when `pStateSize` is NULL.
- `ippStsDataTypeErr`: Indicates an error when `dataType` has an illegal value.
Indicates an error when \( \text{roiSize} \) has a field with a zero or negative value; or when \( \text{winSize} \) is less than, or equal to zero.

See Also
Regions of Interest in Intel IPP
OpticalFlowPyrLK Calculates optical flow for the set of feature points using the pyramidal Lucas-Kanade algorithm.
OpticalFlowPyrLKInit Initializes the state structure for optical flow calculation.

**OpticalFlowPyrLKInit**
Initializes the state structure for optical flow calculation.

**Syntax**

\[
\text{IppStatus } \text{ippiOpticalFlowPyrLKInit}_\langle \text{mod} \rangle (\text{IppiOptFlowPyrLK}_\langle \text{mod} \rangle** \text{ppState}, \text{IppiSize} \text{roiSize}, \text{int} \text{winSize}, \text{IppHintAlgorithm} \text{hint}, \text{Ipp8u* } \text{pStateBuf});
\]

Supported values for \( \text{mod} \):

- 8u_C1R
- 16u_C1R
- 32f_C1R

**Include Files**
ippcv.h

**Domain Dependencies**
Headers: ippcore.h, ippvm.h, ipps.h,ippi.h
Libraries: ippcore.lib, ippvm.lib, ipps.lib,ippi.lib

**Parameters**

- **ppState**
  Double pointer to the optical flow state structure to be initialized.
- **roiSize**
  Maximal size of the source image (zero level of the pyramid) ROI, in pixels.
- **winSize**
  Size of the search window of each pyramid level.
- **hint**
  Option to select the algorithmic implementation of the transform function.
- **pStateBuf**
  Pointer to the work buffer for the state structure.

**Description**
This function operates with ROI (see Regions of Interest in Intel IPP).

This function initializes the \( \text{ppState} \) structure that is required to calculate the optical flow between two images in the centered window of size \( \text{winSize} \times \text{winSize} \) using the pyramidal Lucas-Kanade [Bou99] algorithm. The \( \text{hint} \) argument specifies the computation algorithm. The \( \text{ppState} \) structure is used by the \text{ippiOpticalFlowPyrLK} function and can be applied to process images with size not greater than \( \text{roiSize} \).

**Return Values**

- **ippStsNoErr**
  Indicates no error. Any other value indicates an error or a warning.
Indicates an error when ppState is NULL.

Indicates an error when roiSize has a field with a zero or negative value; or when winSize is less than, or equal to zero.

See Also

Regions of Interest in Intel IPP
OpticalFlowPyrLK Calculates optical flow for the set of feature points using the pyramidal Lucas-Kanade algorithm.
OpticalFlowPyrLKGetSize Computes the size of the pyramidal optical flow state structure.

OpticalFlowPyrLK
Calculates optical flow for the set of feature points using the pyramidal Lucas-Kanade algorithm.

Syntax

IppStatus ippiOpticalFlowPyrLK_<mod>(IppiPyramid* pPyr1, IppiPyramid* pPyr2, const IppiPoint_32f* pPrev, IppiPoint_32f* pNext, Ipp8s* pStatus, Ipp32f* pError, int numFeat, int winSize, int maxLev, int maxIter, Ipp32f threshold, IppiOptFlowPyrLK_<mod>* pState);

Supported values for mod:

8u_C1R  16u_C1R  32f_C1R

Include Files

ippcv.h

Domain Dependencies

Headers: ippcore.h, ippvm.h, ipps.h,ippi.h
Libraries: ippcore.lib, ippvm.lib, ipps.lib,ippi.lib

Parameters

pPyr1 Pointer to the ROI in the first image pyramid structure.
pPyr2 Pointer to the ROI in the second image pyramid structure.
pPrev Pointer to the array of initial coordinates of the feature points.
pNext Pointer to the array of new coordinates of feature point; as input it contains hints for new coordinates.
pStatus Pointer to the array of result indicators.
pError Pointer to the array of differences between neighborhoods of old and new point positions.
numFeat Number of feature points.
winSize Size of the search window of each pyramid level.
maxLev Pyramid level to start the operation.
maxIter Maximum number of algorithm iterations for each pyramid level.
threshold

Threshold value.

pState

Pointer to the pyramidal optical flow structure.

Description

This function operates with ROI (see Regions of Interest in Intel IPP).

This function implements the iterative version of the Lucas-Kanade algorithms [Bou99]. It computes with sub-pixel accuracy new coordinates of the numFeat feature points of two images at time \( t \) and \( t+dt \). Their initial coordinates are places in the pPrev array. Computed values of new coordinates of the feature points are stored in the array pNext, that initially contains estimations of them (or hints), for example, based on the flow values for the previous image in sequence. If there are not such hints, the pNext array contains the same initial coordinates as the pPrev array.

pStatus and pError are arrays of size numFeat with the respective data type.

The images are presented by the pyramid structures pPyr1 and pPyr2 respectively (see description of the PyramidGetSize andippiPyramidInit functions for more details). These structures should be initialized by calling the function PyramidGetSize and ippiPyramidInit functions beforehand. The function uses the pyramidal optical flow structure pState that also should be previously initialized using OpticalFlowPyrLKGetSize and OpticalFlowPyrLKInit.

The function starts operation on the highest pyramid level (smallest image) that is specified by the maxLev parameter in the centered search window which size winSize could not exceed the corresponding value winSize that is specified in OpticalFlowPyrLKGetSize and OpticalFlowPyrLKInit. The operation for \( i \)-th feature point on the given pyramid level finishes if:

- New position of the point is found:

\[
\sqrt{dx^2 + dy^2} < \text{threshold},
\]

- Specified number of iteration maxIter is performed
- Intersection between the pyramid layer and the search window became empty

In first two cases for non-zero levels the new position coordinates are scaled to the next pyramid level and the operation continues on the next level. For zero level or for third case the operation stops, the number of the corresponding level is written to the pStatus[i] element, the new coordinates are scaled to zero level and are written to pNext[i]. The square root of the average squared difference between neighborhoods of old and new positions is written to pError[i].

Return Values

- ippStsNoErr Indicates no error. Any other value indicates an error or a warning.
- ippStsNullPtrErr Indicates an error if one of the specified pointer is NULL.
- ippStsSizeErr Indicates an error condition if numFeat or winSize has zero or negative value.
- ippStsBadArgErr Indicates an error condition if maxLev or threshold has negative value, or maxIter has zero or negative value.

Example

To better understand usage of the ippiOpticalFlowPyrLK function, refer to the following example in the examples archive available for download from https://software.intel.com/en-us/ipp-manual-examples:

OpticalFlowPyrLK.c
Universal Pyramids

The functions described in this section operate with universal image pyramids. These pyramids use separable symmetric kernel (not only Gaussian type) and downsampling/upsampling with arbitrary factor (not only 2). The next pyramid layer can be built for an image of an arbitrary size. These pyramids are used in some computer vision algorithms, for example, in optical flow calculations.

NOTE
All universal pyramid functions use the mirrored border.

Example shows how to build pyramids and calculate the optical flow for two images.

PyramidGetSize

Computes the size of the pyramid structure and the size of the temporary buffer for the ippiPyramidInit function.

Syntax

IppStatus ippiPyramidGetSize(int* pPyrSize, int* pBufSize, int level, IppiSize roiSize, Ipp32f rate);

Include Files

ippcv.h

Domain Dependencies

Headers: ippcore.h, ippvm.h, ipps.h,ippi.h
Libraries: ippcore.lib, ippvm.lib, ipps.lib,ippi.lib

Parameters

pPyrSize  Pointer to the size of the pyramid structure.
pBufSize  Pointer to the size of the external work buffer for pyramid processing.
level     Maximum value for the pyramid level.
roiSize   Size of zero level image ROI, in pixels.
rate      Ratio between neighbouring levels (1 < rate ≤ 10).

Description

This function computes the size of the pyramid structure and the size of the temporary buffer, in bytes, for the ippiPyramidInit function. For an example on how to use this function, refer to the example provided with the ippiPyramidLayerDown function description.

Return Values

ippStsNoErr Indicates no error.
ippStsNullPtrErr Indicates an error when any of the specified pointers is NULL.
Indicates an error when roiSize has a field with a zero or negative value.
Indicates an error when level is equal to or less than 0, or when rate is out of the range.

See Also
PyramidInit Initializes the pyramid structure and calculates the ROI size for pyramid layers.
PyramidLayerDown Creates a lower pyramid layer.

PyramidInit
Initializes the pyramid structure and calculates the ROI size for pyramid layers.

Syntax
IppStatus ippiPyramidInit(IppiPyramid** pPyr, int level, IppiSize roiSize, Ipp32f rate, Ipp8u* pPyrBuffer, Ipp8u* pBuffer);

Include Files
ippcv.h

Domain Dependencies
Headers: ippcore.h, ippvm.h, ipps.h,ippi.h
Libraries: ippcore.lib, ippvm.lib, ipps.lib,ippi.lib

Parameters
pPyr Pointer to the pointer to the pyramid structure.
level Maximum value for the pyramid level.
roiSize Size of zero level image ROI, in pixels.
rate Ratio between neighbouring levels (1 < rate≤ 10).
pPyrBuffer Pointer to the buffer for the pyramid structure initialization.
pBuffer Pointer to the work buffer.

Description
This function initializes the structure for pyramid with (level+1) levels. This structure is used by the ippiOpticalFlowPyrLK function for optical flow calculation. The IppiPyramid structure contains the following fields:

pImage Pointer to the array of (level+1) layer images.
pStep Pointer to the array of (level+1) image row step values.
pRoi Pointer to the array of (level+1) layer image ROIs.
pRate Pointer to the array of (level+1) ratios of i-th levels to the zero level (rate^{-i}).
pState Pointer to the structure to compute the next pyramid layer.
level Number of levels in the pyramid.
The ippiPyramidInit function fills the pRoi and pRate arrays and the level field. The value of the level field is equal to the minimum of the input value of the level parameter and the maximum possible number of layers of the pyramid with given rate and zero level size.

You need to specify other fields. To initialize the pyramid layer structure pState, use the ippiPyramidLayerDownInit or ippiPyramidLayerUpInit functions. To obtain the pyramid layer images, you can use the ippiPyramidLayerDown and ippiPyramidLayerUp functions.

For an example on how to use this function, refer to the example provided with the ippiPyramidLayerDown function description.

**Return Values**

- **ippStsNoErr** Indicates no error.
- **ippStsNullPtrErr** Indicates an error when at least one of the pointers is NULL.
- **ippStsSizeErr** Indicates an error when roiSize has a field with a zero or negative value.
- **ippStsBadArgErr** Indicates an error when level is equal to or less than 0, or when rate is out of the range.

**See Also**

- **OpticalFlowPyrLK** Calculates optical flow for the set of feature points using the pyramidal Lucas-Kanade algorithm.
- **PyramidLayerDownInit** Initializes the structure for creating a lower pyramid layer.
- **PyramidLayerUpInit** Initializes the structure for creating an upper pyramid layer.
- **PyramidLayerDown** Creates a lower pyramid layer.
- **PyramidLayerUp** Creates an upper pyramid layer.

**GetPyramidDownROI**

*Computes the size of the lower pyramid layer.*

**Syntax**

```c
IppStatus ippiGetPyramidDownROI(IppiSize srcRoiSize, IppiSize* pDstRoiSize, Ipp32f rate);
```

**Include Files**

ippcv.h

**Domain Dependencies**

Headers: ippcore.h, ippvm.h, ipps.h,ippi.h  
Libraries: ippcore.lib, ippvm.lib, ipps.lib,ippi.lib

**Parameters**

- **srcRoiSize** Size of the source pyramid layer ROI in pixels.
- **pDstRoiSize** Pointer to the size of the destination (lower) pyramid layer ROI in pixels.
- **rate** Ratio between source and destination layers (1 < rate ≤ 10).

**Description**
This function operates with ROI (see Regions of Interest in Intel IPP).

This function computes the size of the lower pyramid layer \(p_{DstRoiSize}\) for a source layer of a given size \(srcRoiSize\) and specified size ratio \(rate\) between them in accordance with the following formulas:

\[
\begin{align*}
p_{DstRoiSize}.width &= \text{max}(1, \text{min}\left(\frac{srcRoiSize.width}{rate}, srcRoiSize.width-1\right)) \\
p_{DstRoiSize}.height &= \text{max}(1, \text{min}\left(\frac{srcRoiSize.height}{rate}, srcRoiSize.height-1\right))
\end{align*}
\]

**NOTE**
Since for the non-integer \(rate\) results depend on the computational precision, it is strongly recommended to use this function in computations.

**Return Values**
- ippStsNoErr: Indicates no error.
- ippStsNullPtrErr: Indicates an error if \(p_{DstRoiSize}\) pointer is NULL.
- ippStsSizeErr: Indicates an error condition if \(srcRoiSize\) has a field with zero or negative value.
- ippStsBadArgErr: Indicates an error condition if \(rate\) is out of the range.

**GetPyramidUpROI**

*Computes the size of the upper pyramid layer.*

**Syntax**
```
IppStatus ippiGetPyramidUpROI(IppiSize srcRoiSize, IppiSize* pDstRoiSizeMin, IppiSize* pDstRoiSizeMax, Ipp32f rate);
```

**Include Files**
ippcv.h

**Domain Dependencies**
- Headers: ippcore.h, ippvm.h, ipps.h, ipp.i
- Libraries: ippcore.lib, ippvm.lib, ipps.lib,ippi.lib

**Parameters**
- \(srcRoiSize\): Size of the source pyramid layer ROI in pixels.
- \(p_{DstRoiSizeMin}\): Pointer to the minimal size of the destination (upper) pyramid layer ROI in pixels.
- \(p_{DstRoiSizeMax}\): Pointer to the maximal size of the destination (upper) pyramid layer ROI in pixels.
- \(rate\): Ratio between source and destination layers (1 < \(rate\) ≤ 10).

**Description**
This function operates with ROI (see Regions of Interest in Intel IPP).
This function computes possible sizes of the upper pyramid layer \( pDstRoiSizeMin \) and \( pDstRoiSizeMax \) for a source layer of a given size \( srcRoiSize \) and specified size ratio \( rate \) between them in accordance with the following formulas:

**maximum size** \( pDstRoiSizeMax \):

\[
\begin{align*}
pDstRoiMax.width &= \max(srcRoiSize.width + 1, \lfloor srcRoiSize.width \cdot rate \rfloor) \\
pDstRoiMax.height &= \max(srcRoiSize.height + 1, \lfloor srcRoiSize.height \cdot rate \rfloor)
\end{align*}
\]

**minimum size** \( pDstRoiSizeMin \):

if the width and height of the source layer ROI is greater than 1,

\[
\begin{align*}
pDstRoiMin.width &= \max(srcRoiSize.width + 1, \lfloor (srcRoiSize.width - 1) \cdot rate \rfloor) \\
pDstRoiMin.height &= \max(srcRoiSize.height + 1, \lfloor (srcRoiSize.height - 1) \cdot rate \rfloor)
\end{align*}
\]

if the width and height of the source layer ROI is equal to 1,

\[
\begin{align*}
pDstRoiMin.width &= 1 \\
pDstRoiMin.height &= 1
\end{align*}
\]

**NOTE**
Since for the non-integer \( rate \) results depend on the computational precision, it is strongly recommended to use this function in computations.

**Return Values**

- ippStsNoErr: Indicates no error.
- ippStsNullPtrErr: Indicates an error if one of the specified pointers is NULL.
- ippStsSizeErr: Indicates an error condition if \( srcRoiSize \) has a field with zero or negative value.
- ippStsBadArgErr: Indicates an error condition if \( rate \) is out of the range.

**PyramidLayerDownGetSize**

*Computes the size of the structure for creating a lower pyramid layer and the size of the temporary buffer.*

**Syntax**

```c
IppStatusippiPyramidLayerDownGetSize_8u_C1R(IppSize srcRoi, Ipp32f rate, int kerSize, int* pStateSize, int* pBufSize);
IppStatusippiPyramidLayerDownGetSize_8u_C3R(IppSize srcRoi, Ipp32f rate, int kerSize, int* pStateSize, int* pBufSize);
IppStatusippiPyramidLayerDownGetSize_16u_C1R(IppSize srcRoi, Ipp32f rate, int kerSize, int* pStateSize, int* pBufSize);
IppStatusippiPyramidLayerDownGetSize_16u_C3R(IppSize srcRoi, Ipp32f rate, int kerSize, int* pStateSize, int* pBufSize);
IppStatusippiPyramidLayerDownGetSize_32f_C1R(IppSize srcRoi, Ipp32f rate, int kerSize, int* pStateSize, int* pBufSize);
IppStatusippiPyramidLayerDownGetSize_32f_C3R(IppSize srcRoi, Ipp32f rate, int kerSize, int* pStateSize, int* pBufSize);
```
Include Files
ippcv.h

Domain Dependencies
Headers: ippcore.h, ippvm.h, ipps.h,ippi.h
Libraries: ippcore.lib, ippvm.lib, ipps.lib,ippi.lib

Parameters

tsrcRoi
Size of the source image ROI.

trate
Ratio between neighbouring levels (1 < rate ≤ 10).

tkerSize
Size of the kernel.
	pStateSize
Pointer to the size of the pyramid layer state structure.
	pBufSize
Pointer to the size of the external work buffer.

Description
This function operates with ROI (see Regions of Interest in Intel IPP).
This function computes the size of the *pState* structure to build a lower pyramid layer and the size of the temporary buffer, in bytes. This structure is used by the *ippiPyramidLayerDown* function and can be applied to process images with size not greater than *srcRoi*. For an example on how to use this function, refer to the example provided with the *ippiPyramidLayerDown* function description.

Return Values

- ippStsNoErr: Indicates no error.
- ippStsNullPtrErr: Indicates an error when at least one of the pointers is NULL.
- ippStsSizeErr: Indicates an error when the width or height of images is less than, or equal to zero.
- ippStsBadArgErr: Indicates an error when *kerSize* is even, or equal to or less than 0; or when *rate* is out of the range.

See Also
Regions of Interest in Intel IPP
PyramidLayerDown Creates a lower pyramid layer.

PyramidLayerDownInit
Initializes the structure for creating a lower pyramid layer.

Syntax

Case 1: Operating on integer data

```
IppStatus ippiPyramidLayerDownInit_8u_C1R(IppiPyramidDownState_8u_C1R** ppState,
IppiSize srcRoi, Ipp32f rate, Ipp16s* pKernel, int kerSize, int mode, Ipp8u* pStateBuf,
Ipp8u* pBuffer);
IppStatus ippiPyramidLayerDownInit_8u_C3R(IppiPyramidDownState_8u_C3R** ppState,
IppiSize srcRoi, Ipp32f rate, Ipp16s* pKernel, int kerSize, int mode, Ipp8u* pStateBuf,
Ipp8u* pBuffer);
```
IppStatus ippiPyramidLayerDownInit_16u_C1R(IppiPyramidDownState_16u_C1R** ppState,
IppSize srcRoi, Ipp32f rate, Ipp16s* pKernel, int kerSize, int mode, Ipp8u* pStateBuf,
Ipp8u* pBuffer);

IppStatus ippiPyramidLayerDownInit_16u_C3R(IppiPyramidDownState_16u_C3R** ppState,
IppSize srcRoi, Ipp32f rate, Ipp16s* pKernel, int kerSize, int mode, Ipp8u* pStateBuf,
Ipp8u* pBuffer);

Case 2: Operating on floating point data

IppStatus ippiPyramidLayerDownInit_32f_C1R(IppiPyramidDownState_32f_C1R** ppState,
IppSize srcRoi, Ipp32f rate, Ipp32f* pKernel, int kerSize, int mode, Ipp8u* pStateBuf,
Ipp8u* pBuffer);

IppStatus ippiPyramidLayerDownInit_32f_C3R(IppiPyramidDownState_32f_C3R** ppState,
IppSize srcRoi, Ipp32f rate, Ipp32f* pKernel, int kerSize, int mode, Ipp8u* pStateBuf,
Ipp8u* pBuffer);

Include Files
ippcv.h

Domain Dependencies
Headers: ippcore.h, ippvm.h, ipps.h, ippi.h
Libraries: ippcore.lib, ippvm.lib, ipps.lib, ippi.lib

Parameters

ppState
Pointer to the pointer to the initialized pyramid layer state structure.

srcRoi
Size of the source image ROI.

rate
Ratio between neighbouring levels (1 < rate ≤ 10).

pKernel
Separable symmetric kernel of odd length.

kerSize
Size of the kernel.

mode
Interpolation method, possible value is:
IPPI_INTER_LINEAR  Bilinear interpolation.

pStateBuf
Pointer to the buffer to initialize the pyramid layer state structure.

pBuffer
Pointer to the external buffer.

Description
This function operates with ROI (see Regions of Interest in Intel IPP).

This function initializes the pState structure to build a lower pyramid layer. This structure is used by the
ippiPyramidLayerDown function and can be applied to process images with size not greater than dstRoi.

The specified kernel pKernel should be symmetric. If it is not symmetric, the function builds the symmetric
kernel using the first part of the specified kernel and returns a warning. The symmetric separable kernel can
be not Gaussian. If the sum of kernel elements is not equal to zero, the kernel is normalized.

For integer rates, the function performs downsampling by discarding rows and columns that are not multiples
of the rate value. For non-integer rates, the function uses bilinear interpolation (see Linear Interpolation for
more information).
For an example on how to use this function, refer to the example provided with the \texttt{ippiPyramidLayerDown} function description.

\section*{Return Values}

- \texttt{ippStsNoErr} Indicates no error.
- \texttt{ippStsNullPtrErr} Indicates an error when at least one of the pointers is \texttt{NULL}.
- \texttt{ippStsSizeErr} Indicates an error when the width or height of images is less than, or equal to zero.
- \texttt{ippStsBadArgErr} Indicates an error when \texttt{kerSize} is even, or equal to or less than 0; or when \texttt{rate} is out of the range.

\section*{See Also}

- \texttt{Regions of Interest in Intel IPP}
- \texttt{PyramidLayerDown} Creates a lower pyramid layer.
- \texttt{Linear Interpolation}

\textbf{PyramidLayerDown}  
\textit{Creates a lower pyramid layer.}

\section*{Syntax}

\begin{verbatim}
IppStatus ippiPyramidLayerDown_<mod>(const Ipp<datatype>* pSrc, int srcStep, IppiSize srcRoiSize, Ipp<datatype>* pDst, int dstStep, IppiSize dstRoiSize, IppiPyramidDownState_<mod>* pState);
\end{verbatim}

Supported values for \texttt{mod}:

- 8u\_C1R  
- 16u\_C1R  
- 32f\_C1R  
- 8u\_C3R  
- 16u\_C3R  
- 32f\_C3R

\section*{Include Files}

- ippcv.h

\section*{Domain Dependencies}

- Headers: ippcore.h, ippvm.h, ipps.h,ippi.h
- Libraries: ippcore.lib, ippvm.lib, ipps.lib,ippi.lib

\section*{Parameters}

- \texttt{pSrc}  
  Pointer to the source image ROI.
- \texttt{srcStep}  
  Distance, in bytes, between the starting points of consecutive lines in the source image.
- \texttt{srcRoiSize}  
  Size of the source image ROI, in pixels.
- \texttt{pDst}  
  Pointer to the destination image ROI.
- \texttt{dstStep}  
  Distance, in bytes, between the starting points of consecutive lines in the destination image.
- \texttt{dstRoiSize}  
  Size of the destination image ROI, in pixels.
**pState**

Pointer to the pyramid layer structure.

**Description**

This function operates with ROI (see Regions of Interest in Intel IPP).

This function creates a lower pyramid layer \( p_{Dst} \) from the source image \( p_{Src} \). The function applies the convolution kernel to the source image using the mirror border and then performs downsampling. Before calling `ippiPyramidLayerDown`, compute the size of the `pState` structure and work buffer using the `PyramidLayerDownGetSize` function and initialize the structure using the `PyramidLayerDownInit` function. The function can process images with \( srcRoiSize \) not greater than the \( srcRoi \) parameter specified in the `PyramidLayerDownInit` function.

**NOTE**

This function uses the mirrored border.

**Return Values**

- `ippStsNoErr` Indicates no error.
- `ippStsNullPtrErr` Indicates an error if one of the specified pointers is NULL.
- `ippStsSizeErr` Indicates an error condition if \( srcRoiSize \) or \( dstRoiSize \) has a field with zero or negative value.
- `ippStsStepErr` Indicates an error condition if \( srcStep \) is less than \( srcRoiSize.width * <\text{pixelSize}> \), or \( dstStep \) is less than \( dstRoiSize.width * <\text{pixelSize}> \).
- `ippStsNotEvenStepErr` Indicates an error condition if one of the step values is not divisible by 4 for floating-point images, or by 2 for short-integer images.
- `ippStsBadArgErr` Indicates an error condition if \( pState->rate \) has wrong value.

**Example**

To better understand usage of the `ippiPyramidLayerDown` function, refer to the following example in the examples archive available for download from https://software.intel.com/en-us/ipp-manual-examples:

PyramidLayerDown.c

**See Also**

Regions of Interest in Intel IPP

`PyramidLayerDownGetSize` Computes the size of the structure for creating a lower pyramid layer and the size of the temporary buffer.

`PyramidLayerDownInit` Initializes the structure for creating a lower pyramid layer.

**PyramidLayerUpGetSize**

*Computes the size of the structure for creating an upper pyramid layer and the size of the temporary buffer.*

**Syntax**

```c
IppStatus ippiPyramidLayerUpGetSize_8u_C1R(IppiSize dstRoi, Ipp32f rate, int kerSize, int* pStateSize);
```
IppStatus ippiPyramidLayerUpGetSize_8u_C3R(IppiSize dstRoi, Ipp32f rate, int kerSize, int* pStateSize);
IppStatus ippiPyramidLayerUpGetSize_16u_C1R(IppiSize dstRoi, Ipp32f rate, int kerSize, int* pStateSize);
IppStatus ippiPyramidLayerUpGetSize_16u_C3R(IppiSize dstRoi, Ipp32f rate, int kerSize, int* pStateSize);
IppStatus ippiPyramidLayerUpGetSize_32f_C1R(IppiSize dstRoi, Ipp32f rate, int kerSize, int* pStateSize);
IppStatus ippiPyramidLayerUpGetSize_32f_C3R(IppiSize dstRoi, Ipp32f rate, int kerSize, int* pStateSize);

Include Files
ippcv.h

Domain Dependencies
Headers: ippcore.h, ippvm.h, ipps.h, ippi.h
Libraries: ippcore.lib, ippvm.lib, ipps.lib, ippi.lib

Parameters

dstRoi
Size of the destination image ROI.

rate
Ratio between neighbouring levels (1 < rate ≤ 10).

kerSize
Size of the kernel.

pStateSize
Pointer to the size of the pyramid layer state structure.

Description
This function operates with ROI (see Regions of Interest in Intel IPP).
This function computes the size of the pState structure to build an upper pyramid layer and the size of the temporary buffer, in bytes. This structure is used by the ippiPyramidLayerUp function and can be applied to process images with size not greater than dstRoi. For an example on how to use this function, refer to the example provided with the ippiPyramidLayerDown function description.

Return Values

ippStsNoErr
Indicates no error.

ippStsNullPtrErr
Indicates an error when at least one of the pointers is NULL.

ippStsSizeErr
Indicates an error when the width or height of images is less than, or equal to zero.

ippStsBadArgErr
Indicates an error when kerSize is even, or equal to or less than 0; or when rate is out of the range.

See Also
Regions of Interest in Intel IPP
PyramidLayerUp Creates an upper pyramid layer.
PyramidLayerDown Creates a lower pyramid layer.
PyramidLayerUpInit
Initializes the structure for creating an upper pyramid layer.

Syntax

Case 1: Operating on integer data

IppStatus ippiPyramidLayerUpInit_8u_C1R(IppiPyramidUpState_8u_C1R** ppState, IppiSize dstRoi, Ipp32f rate, Ipp16s* pKernel, int kerSize, int mode, Ipp8u* pStateBuf);
IppStatus ippiPyramidLayerUpInit_8u_C3R(IppiPyramidUpState_8u_C3R** ppState, IppiSize dstRoi, Ipp32f rate, Ipp16s* pKernel, int kerSize, int mode, Ipp8u* pStateBuf);
IppStatus ippiPyramidLayerUpInit_16u_C1R(IppiPyramidUpState_16u_C1R** ppState, IppiSize dstRoi, Ipp32f rate, Ipp16s* pKernel, int kerSize, int mode, Ipp8u* pStateBuf);
IppStatus ippiPyramidLayerUpInit_16u_C3R(IppiPyramidUpState_16u_C3R** ppState, IppiSize dstRoi, Ipp32f rate, Ipp16s* pKernel, int kerSize, int mode, Ipp8u* pStateBuf);

Case 2: Operating on floating point data

IppStatus ippiPyramidLayerUpInit_32f_C1R(IppiPyramidUpState_32f_C1R** ppState, IppiSize dstRoi, Ipp32f rate, Ipp32f* pKernel, int kerSize, int mode, Ipp8u* pStateBuf, Ipp8u* pBuffer);
IppStatus ippiPyramidLayerUpInit_32f_C3R(IppiPyramidUpState_32f_C3R** ppState, IppiSize dstRoi, Ipp32f rate, Ipp32f* pKernel, int kerSize, int mode, Ipp8u* pStateBuf, Ipp8u* pBuffer);

Include Files
ippcv.h

Domain Dependencies

Headers: ippcore.h, ippvm.h, ipps.h,ippi.h
Libraries: ippcore.lib, ippvm.lib, ipps.lib,ippi.lib

Parameters

ppState
Pointer to the pointer to the initialized pyramid state structure.
dstRoi
Size of the destination image ROI.
rate
Ratio between neighbouring levels (1 < rate ≤ 10).
pKernel
Separable symmetric kernel of odd length.
kerSize
Size of the kernel.
mode
Interpolation method, possible value is:

| IPPI_INTER_LINEAR | Bilinear interpolation. |

pStateBuf
Pointer to the buffer to initialize the pyramid layer state structure.

Description

This function operates with ROI (see Regions of Interest in Intel IPP).
This function allocates memory and initializes the pState structure to build an upper pyramid layer. This structure is used by the ippiPyramidLayerUp function and can be applied to process images with size not greater than dstRoi.

The specified kernel pKernel should be symmetric. If it is not symmetric, the function builds the symmetric kernel using the first part of the specified kernel and returns a warning. The symmetric separable kernel can be not Gaussian. If the sum of kernel elements is not equal to zero, the kernel is normalized.

For integer rates, the function performs upsampling by inserting zero rows and columns that are not multiples of the rate value. For non-integer rates, the function uses bilinear interpolation (see Linear Interpolation for more information) to calculate kernel coefficients for pixels with non-integer coordinates.

For an example on how to use this function, refer to the example provided with the ippiPyramidLayerDown function description.

**Return Values**

- **ippStsNoErr**: Indicates no error.
- **ippStsNullPtrErr**: Indicates an error when at least one of the pointers is NULL.
- **ippStsSizeErr**: Indicates an error when the width or height of images is less than, or equal to zero.
- **ippStsBadArgErr**: Indicates an error when kerSize is even, or equal to or less than 0; or when rate is out of the range.

**See Also**

Regions of Interest in Intel IPP

PyramidLayerUp Creates an upper pyramid layer.

PyramidLayerDown Creates a lower pyramid layer.

Linear Interpolation

**PyramidLayerUp**

*Creates an upper pyramid layer.*

**Syntax**

```c
IppStatus ippiPyramidLayerUp_<mod>(const Ipp<datatype>* pSrc, int srcStep, IppiSize srcRoiSize, Ipp<datatype>* pDst, int dstStep, IppiSize dstRoiSize, IppiPyramidUpState_<mod>* pState);
```

Supported values for mod:

- 8u_C1R
- 16u_C1R
- 32f_C1R
- 8u_C3R
- 16u_C3R
- 32f_C3R

**Include Files**

ippcv.h

**Domain Dependencies**

Headers: ippcore.h, ippvm.h, ipps.h,ippi.h

Libraries: ippcore.lib, ippvm.lib, ipps.lib,ippi.lib
Parameters

- **pSrc**: Pointer to the source image ROI.
- **srcStep**: Distance, in bytes, between the starting points of consecutive lines in the source image.
- **srcRoiSize**: Size of source image ROI, in pixels.
- **pDst**: Pointer to the destination image ROI.
- **dstStep**: Distance, in bytes, between the starting points of consecutive lines in the destination image.
- **dstRoiSize**: Size of destination image ROI, in pixels.
- **pState**: The pointer to the pyramid layer structure.

Description

This function operates with ROI (see *Regions of Interest in Intel IPP*).

This function creates an upper pyramid layer `pDst` from the source image `pSrc`. The function performs upsampling of the source image and then applies the convolution kernel using the mirror border. Before calling the `ippiPyramidLayerUp` function, compute the size of the pyramid layer structure `pState` using the `PyramidLayerUpGetSize` function and initialize the structure using the `PyramidLayerUpInit` function. The function can process images with `srcRoiSize` not greater than the `roiSize` parameter specified in the `PyramidLayerUpInit` function.

**NOTE**

This function uses the mirrored border.

Return Values

- **ippStsNoErr**: Indicates no error.
- **ippStsNullPtrErr**: Indicates an error if one of the specified pointers is NULL.
- **ippStsSizeErr**: Indicates an error condition if `srcRoiSize` or `dstRoiSize` has a field with zero or negative value.
- **ippStsStepErr**: Indicates an error condition if `srcStep` is less than `srcRoiSize.width * <pixelSize>`, or `dstStep` is less than `dstRoiSize.width * <pixelSize>`.
- **ippStsNotEvenStepErr**: Indicates an error condition if one of the step values is not divisible by 4 for floating-point images, or by 2 for short-integer images.
- **ippStsBadArgErr**: Indicates an error condition if `pState->rate` has wrong value.

See Also

- *Regions of Interest in Intel IPP*
- `PyramidLayerDownGetSize` Computes the size of the structure for creating a lower pyramid layer and the size of the temporary buffer.
- `PyramidLayerDownInit` Initializes the structure for creating a lower pyramid layer.
Example of Using General Pyramid Functions

Refer to the following example to understand how different general pyramids functions can be used to create the Gaussian and Laplacian pyramids:

Pyramid.c

Image Inpainting

The functions described in this section allows to restore the unknown image portions. They could be used to repair damaged parts of images and to remove some objects from images. Fast direct methods of inpainting that allow for run-time correcting of video frames are supported.

InpaintGetSize

*Computes the size of the state structure and work buffer for image inpainting.*

**Syntax**

```
IppStatusippiInpaintGetSize(constIpp8upMask,intmaskStep,IppiSizeroiSize,Ipp32fradius,IppiInpaintFlagflags,intchannels,int*pStateSize,int*pBufSize);
```

**Include Files**

ippcv.h

**Domain Dependencies**

Headers: ippcore.h, ippvm.h, ipps.h,ippi.h

Libraries: ippcore.lib, ippvm.lib, ipps.lib,ippi.lib

**Parameters**

- **pMask**: Pointer to the mask image ROI.
- **maskStep**: Distance, in bytes, between the starting points of consecutive lines in the mask image.
- **roiSize**: Size of the image ROI, in pixels.
- **radius**: Radius of the neighborhood used for inpainting.
- **flags**: Specifies algorithm for image inpainting. Possible values:
  - IPP_INPAINT_TELEA: Telea algorithm
  - IPP_INPAINT_NS: Navier-Stokes equation
- **channels**: Number of channels in the image.
- **pStateSize**: Pointer to the size of the state structure.
- **pBufSize**: Pointer to the size of the external work buffer.

**Description**

This function operates with ROI (see Regions of Interest in Intel IPP).
This function computes the size of the state structure for direct methods of image inpainting and the size of the external work buffer. Call this function before usingippiInpaintInit. For an example on how to use this function, refer to the example provided with theippiInpaint function description.

**Return Values**

<table>
<thead>
<tr>
<th>Return Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ippStsNoErr</td>
<td>Indicates no error. Any other value indicates an error or a warning.</td>
</tr>
<tr>
<td>ippStsNullPtrErr</td>
<td>Indicates an error when one of the specified pointers is NULL.</td>
</tr>
<tr>
<td>ippStsSizeErr</td>
<td>Indicates an error when width or height of the image is less than, or equal to zero.</td>
</tr>
<tr>
<td>ippStsStepErr</td>
<td>Indicates an error when the step in the mask image is too small.</td>
</tr>
<tr>
<td>ippStsBadArgErr</td>
<td>Indicates an error when radius is less than 1, or flags has an illegal value.</td>
</tr>
<tr>
<td>ippStsNumChannelsErr</td>
<td>Indicates an error when the specified number of image channels is invalid or not supported.</td>
</tr>
</tbody>
</table>

**See Also**

**Regions of Interest in Intel IPP**

InpaintMODIFIED API. Restores unknown image pixels.

InpaintInit Initializes the state structure for image inpainting.

**InpaintInit**

*Initializes the state structure for image inpainting.*

**Syntax**

IppStatusippiInpaintInit_8u_C1R(IppiInpaintState_8u_C1R** ppState, const Ipp32f* pDist, int distStep, const Ipp8u* pMask, int maskStep, IppiSize roiSize, Ipp32f radius, IppiInpaintFlag flags, Ipp8u* pStateBuf, Ipp8u* pBuf);

IppStatusippiInpaintInit_8u_C3R(IppiInpaintState_8u_C3R** ppState, const Ipp32f* pDist, int distStep, const Ipp8u* pMask, int maskStep, IppiSize roiSize, Ipp32f radius, IppiInpaintFlag flags, Ipp8u* pStateBuf, Ipp8u* pBuf);

**Include Files**

ippcv.h

**Domain Dependencies**

**Headers:** ippcore.h, ippvm.h, ipps.h, ippi.h

**Libraries:** ippcore.lib, ippvm.lib, ipps.lib, ippi.lib

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ppState</td>
<td>Double pointer to the state structure for image inpainting.</td>
</tr>
<tr>
<td>pDist</td>
<td>Pointer to the ROI of the image of distances.</td>
</tr>
<tr>
<td>distStep</td>
<td>Distance, in bytes, between the starting points of consecutive lines in the image of distances.</td>
</tr>
<tr>
<td>pMask</td>
<td>Pointer to the mask image ROI.</td>
</tr>
</tbody>
</table>
maskStep
Distance, in bytes, between the starting points of consecutive lines in the mask image.

roiSize
Size of the image ROI, in pixels.

radius
Radius of the neighborhood used for inpainting (dist ≤ radius pixels are processed).

flags
Specifies algorithm for image inpainting. Possible values:
- IPP_INPAINT_TELEA: Telea algorithm
- IPP_INPAINT_NS: Navier-Stokes equation

pStateBuf
Pointer to the buffer for the state structure initialization.

pBuf
Pointer to the external work buffer.

Description
This function operates with ROI (see Regions of Interest in Intel IPP).
This function initializes the ppState structure for direct methods of image inpainting. This structure is used
by theippiInpaint function and can be applied to process images of the same size roiSize.

Zero pixels of the pMask image correspond to the known image pixels, non-zero pixels - to the unknown
image pixels that should be restored. The distance image pDist specifies the order of pixel inpainting. Values
of unknown pixels are restored in ascending order depending on their distances. The radius parameter
specifies the radius of the circular neighborhood that affects the restoration of the central pixel. The flag parameter specifies the method of direct inpainting. Two methods are supported: Telea algorithm [Telea04]
and Navier-Stokes equation [Bert01].

For an example on how to use this function, refer to the example provided with theippiInpaint function
description.

NOTE The image ROI must not exceed the maximum width and height of roiSize specified in the
initialization function.

Return Values
- ippStsNoErr: Indicates no error. Any other value indicates an error or a warning.
- ippStsNullPtrErr: Indicates an error when one of the specified pointers is NULL.
- ippStsSizeErr: Indicates an error when width or height of the image is less than, or equal to zero.
- ippStsStepErr: Indicates an error when the step of the mask or distance image ROI is too small.
- ippStsNotEvenStepErr: Indicates an error when the step value is not divisible by the pDist element.
- ippStsBadArgErr: Indicates an error when radius is less than 1, or flags has an illegal value.

See Also
- Regions of Interest in Intel IPP
- InpaintMODIFIED API. Restores unknown image pixels.
- InpaintGetSize Computes the size of the state structure and work buffer for image inpainting.
Inpaint
MODIFIED API. Restores unknown image pixels.

Syntax
IppStatus ippiInpaint_8u_C1R(const Ipp8u* pSrc, int srcStep, Ipp8u* pDst, int dstStep, IppiSize roiSize, IppiInpaintState_8u_C1R* pState, Ipp8u* pBuffer);
IppStatus ippiInpaint_8u_C3R(const Ipp8u* pSrc, int srcStep, Ipp8u* pDst, int dstStep, IppiSize roiSize, IppiInpaintState_8u_C1R* pState, Ipp8u* pBuffer);

Include Files
ippcv.h

Domain Dependencies
Headers: ippcore.h, ippvm.h, ipps.h,ippi.h
Libraries: ippcore.lib, ippvm.lib, ipps.lib,ippi.lib

Parameters
pSrc Pointer to the source image ROI.
srcStep Distance in bytes between starts of consecutive lines in the source image.
pDst Pointer to the destination image ROI.
dstStep Distance in bytes between starts of consecutive lines in the destination image.
roiSize Size of the image ROI in pixels.
pState The pointer to the inpainting structure.
pBuffer Pointer to the work buffer.

Description

Important The API of this function has been modified in Intel IPP 9.0 release.

Before using this function, compute the size of the state structure and work buffer using InpaintGetSize and initialize the structure using InpaintInit.

This function operates with ROI (see Regions of Interest in Intel IPP).

This function reconstructs damaged part of the image, or removes a selected object (see Figure "Image Inpainting"). The image part to restore is defined by the mask that is created when the inpainting structure pState is initialized by the InpaintInit function. Different distant transforms can be used, but the Fast Marching method (ippiFastMarching) provides the best results. The order of pixel restoration is defined by the distance through the initialization the inpainting structure pState by the InpaintInit function. Pixels are restored in according to the growing of their distance value. When a pixel is inpainted, it is treated as the known one.

Two algorithms of direct inpainting are supported (controlled by the parameter flags of the InpaintInit function):
- image restoration of the unknown pixel by the weighted sum of approximations by known pixels in the neighborhood \((flags = IPP_INPAINT_TELEA)\) [Telea04],
- image restoration based on the Navier-Stokes equations \((flags = IPP_INPAINT_NS)\) [Bert01].

The inpainting structure \(pState\) can be used to perform restoration of several different images of the same size \(roiSize\).

### Image Inpainting

![Image Inpainting](image)

**Initial image**  
**restored image**

### Return Values

- **ippStsNoErr**: Indicates no error. Any other value indicates an error or a warning.
- **ippStsNullPtrErr**: Indicates an error when one of the specified pointers is `NULL`.
- **ippStsSizeErr**: Indicates an error when \(roiSize\) has a field with zero or negative value, or if differs from the corresponding parameter that is specified when the inpainting structure is initialized by `InpaintInit`.
- **ippStsStepErr**: Indicates an error condition if \(srcStep\) or \(dstStep\) is less than \(roiSize.width \times pixelSize\).

### Example

To better understand usage of the `ippiInpaint` function, refer to the following example in the examples archive available for download from [https://software.intel.com/en-us/ipp-manual-examples](https://software.intel.com/en-us/ipp-manual-examples):

`Inpaint.c`

### See Also

- **InpaintGetSize** Computes the size of the state structure and work buffer for image inpainting.
- **InpaintInit** Initializes the state structure for image inpainting.

### Image Segmentation

This section describes the functions that perform image segmentation using different techniques. These functions allow to extract parts of the image that can be associated with objects of the real world. Watershed and gradient segmentation are region-based methods to split image into the distinctive areas.

Background/foreground segmentation allows for distinguishing between moving objects and stable areas of the background.
**LabelMarkersGetBufferSize**

*Computes the size of the working buffer for the marker labeling.*

**Syntax**

IppStatus ippiLabelMarkersGetBufferSize_8u_C1R(IppiSize roiSize, int* pBufSize);
IppStatus ippiLabelMarkersGetBufferSize_8u32s_C1R(IppiSize roiSize, int* pBufSize);
IppStatus ippiLabelMarkersGetBufferSize_16u_C1R(IppiSize roiSize, int* pBufSize);

**Include Files**

ippcv.h

**Domain Dependencies**

**Headers:** ippcore.h, ippvm.h, ipps.h,ippi.h
**Libraries:** ippcore.lib, ippvm.lib, ipps.lib,ippi.lib

**Parameters**

- **roiSize**
  Size of the source image ROI in pixels.
- **pBufSize**
  Pointer to the computed size of the working buffer.

**Description**

This function operates with ROI (see Regions of Interest in Intel IPP).

This function computes the size of the working buffer required for the ippiLabelMarkers function. The buffer with the length pBufSize[0] can be used to segment images with width and/or height that is equal to or less than the corresponding field of the parameter roiSize.

**Return Values**

- **ippStsNoErr**
  Indicates no error. Any other value indicates an error or a warning.
- **ippStsNullPtrErr**
  Indicates an error condition if the pointer pBufSize is NULL.
- **ippStsSizeErr**
  Indicates an error condition if roiSize has a field with zero or negative value.

**LabelMarkers**

*Labels markers in image with different values.*

**Syntax**

IppStatus ippiLabelMarkers_8u_C1IR(Ipp8u* pMarker, int markerStep, IppiSize roiSize, int minLabel, int maxLabel, IppiNorm norm, int* pNumber, Ipp8u* pBuffer);
IppStatus ippiLabelMarkers_8u32s_C1R(Ipp8u* pSrcMarker, int srcMarkerStep, Ipp32s* pDstMarker, int dstMarkerStep, IppiSize roiSize, int minLabel, int maxLabel, IppiNorm norm, int* pNumber, Ipp8u* pBuffer);
IppStatus ippiLabelMarkers_16u_C1R(Ipp16u* pMarker, int markerStep, IppiSize roiSize, int minLabel, int maxLabel, IppiNorm norm, int* pNumber, Ipp8u* pBuffer);
Include Files
ippcv.h

Domain Dependencies
Headers: ippcore.h, ippvm.h, ipps.h,ippi.h
Libraries: ippcore.lib, ippvm.lib, ipps.lib,ippi.lib

Parameters
pMarker
Pointer to the source and destination image ROI (for in-place flavors).

markerStep
Distance in bytes between starts of consecutive lines in the source and destination image.

pSrcMarker
Pointer to the source image ROI (for not-in-place flavors).

srcMarkerStep
Distance, in bytes, between the starting points of consecutive lines in the source image (for not-in-place flavors).

pDstMarker
Pointer to the source and destination image ROI (for not-in-place flavors).

dstMarkerStep
Distance, in bytes, between the starting points of consecutive lines in the destination image (for not-in-place flavors).

minLabel
Minimal value of the marker label (0 < minLabel ≤ maxLabel).

maxLabel
Maximal value of the marker label (minLabel ≤ maxLabel < 255 for 8-bit markers, and minLabel ≤ maxLabel < 65535 for 16-bit markers, and minLabel ≤ maxLabel < (2^{31}-1) for 32-bit markers).

roiSize
Size of the source and destination image ROI in pixels.

norm
Specifies type of the norm to form the mask for marker propagation:

ippiNormInf     Infinity norm (8-connectivity);
ippiNormL1      L1 norm (4-connectivity).

pNumber
Pointer to the number of markers.

pBuffer
Pointer to the working buffer.

Description
This function operates with ROI (see Regions of Interest in Intel IPP).

This function labels markers in the destination image with different integer values. Each connected set of non-zero image pixels is treated as the separate marker. 4- or 8-connectivity can be used depending on the norm type. All pixels belonging to the same marker are set to the same value from the interval [minLabel, maxLabel]. Two markers can be labeled with the same value if the number of connected components exceeds minLabel-maxLabel+1. The image with labeled markers can be used as the seed image for segmentation by functions ippiSegmentWatershed or ippiSegmentGradient functions.

The function requires the working buffer pBuffer whose size should be computed by the function ippiLabelMarkersGetBufferSize beforehand.
## Return Values

<table>
<thead>
<tr>
<th>Error Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ippStsNoErr</td>
<td>Indicates no error. Any other value indicates an error or a warning.</td>
</tr>
<tr>
<td>ippStsNullPtrErr</td>
<td>Indicates an error condition if one of the specified pointers is NULL.</td>
</tr>
<tr>
<td>ippStsSizeErr</td>
<td>Indicates an error condition if <code>roiSize</code> has a field with zero or negative value.</td>
</tr>
<tr>
<td>ippStsStepErr</td>
<td>Indicates an error condition if <code>markerStep</code> is less than <code>roiSize.width * &lt;pixelSize&gt;</code>.</td>
</tr>
<tr>
<td>ippStsNotEvenStepErr</td>
<td>Indicates an error condition if <code>markerStep</code>, <code>srcMarkerStep</code>, or <code>dstMarkerStep</code> is not divisible by respective <code>&lt;pixelSize&gt;</code>.</td>
</tr>
<tr>
<td>ippStsBadArgErr</td>
<td>Indicates an error condition if one of the <code>minLabel</code>, <code>maxLabel</code>, and <code>norm</code> has an illegal value.</td>
</tr>
</tbody>
</table>

## Example

To better understand usage of the `ippiLabelMarkers` function, refer to the following example in the examples archive available for download from https://software.intel.com/en-us/ipp-manual-examples:

LabelMarkers.c

---

## MarkSpecklesGetBufferSize

**Computes the size of the external work buffer for speckle marking.**

### Syntax

```c
IppStatus ippiMarkSpecklesGetBufferSize(IppiSize roiSize, IppDataType dataType, int numChannels, int* pBufferSize);
```

### Include Files

ippcv.h

### Domain Dependencies

**Headers:** ippcore.h, ippvm.h, ipps.h,ippi.h  
**Libraries:** ippcore.lib, ippvm.lib, ipps.lib,ippi.lib

### Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>roiSize</code></td>
<td>Size of the source and destination image ROI in pixels.</td>
</tr>
<tr>
<td><code>dataType</code></td>
<td>Data type of the source and destination image.</td>
</tr>
<tr>
<td><code>numChannels</code></td>
<td>Number of channels in the image. Possible value is 1.</td>
</tr>
<tr>
<td><code>pBufferSize</code></td>
<td>Pointer to the computed size (in bytes) of the external work buffer.</td>
</tr>
</tbody>
</table>

### Description

This function computes the size of the external work buffer for the `MarkSpeckles` function.

For an example on how to use this function, refer to the example provided with the `MarkSpeckles` function description.
Return Values

- ippStsNoErr: Indicates no error. Any other value indicates an error.
- ippStsNullPtrErr: Indicates an error when pBufferSize is NULL.
- ippStsSizeErr: Indicates an error when roiSize has a field with a zero or negative value.
- ippStsDataTypeErr: Indicates an error when dataType has an illegal value.
- ippStsNumChannelErr: Indicates an error when numChannels has an illegal value.

See Also
MarkSpeckles Marks small noise blobs (speckles) in an image.

MarkSpeckles
Marks small noise blobs (speckles) in an image.

Syntax
IppStatus ippiMarkSpeckles_<mod>(Ipp<datatype>* pSrcDst, int srcDstStep, IppSize roiSize, Ipp<datatype> speckleVal, int maxSpeckleSize, Ipp<datatype> maxPixDiff, IppiNorm norm, Ipp8u* pBuffer);

Supported values for mod:
- 8u_C1IR
- 16u_C1IR
- 16s_C1IR
- 32f_C1IR

Include Files
ippcv.h

Domain Dependencies
Headers: ippcore.h, ippvm.h, ipps.h, ippi.h
Libraries: ippcore.lib, ippvm.lib, ipps.lib, ippi.lib

Parameters

- pSrcDst: Pointer to the source and destination image.
- srcDstStep: Distance, in bytes, between the starting points of consecutive lines in the source and destination image.
- roiSize: Size of the source and destination image ROI in pixels.
- speckleVal: Value to set to the speckles.
- maxSpeckleSize: Maximum size of the image component to consider it as a speckle.
- maxPixDiff: Maximum difference between neighboring disparity pixels to put them into the same component.
- norm: Type of the norm to form the mask for marker propagation. Possible value is:
  - ippiNormL1: L1 norm (4-connectivity)
Pointer to the work buffer.

**Description**

This function operates with ROI (see Regions of Interest in Intel IPP).

This function marks small noise blobs (speckles) in the source image.

The `pSrcDst` parameter points to the processed source and destination image ROI.

The function finds small connected components and set them to the `speckleVal` value. This function marks only components with size that is less than, or equal to `maxSpeckleSize`. Pixels of the image belong to the same connected component if the difference between adjacent pixels (considering 4-connected adjacency) is less than, or equal to the `maxSpeckleSize` value.

**NOTE**

This release does not support 8-connectivity.

The function does not process the pixels of the image that already have the `speckleVal` value.

Before using the `ippiMarkSpeckles` function, compute the size of the external buffer using the `MarkSpecklesGetBufferSize` function.

**Return Values**

- `ippStsNoErr` Indicates no error.
- `ippStsNullPtrErr` Indicates an error when `pSrcDst` or `pBufferSize` is NULL.
- `ippStsSizeErr` Indicates an error when `roiSize` has a field with a zero or negative value.
- `ippStsNotEvenStepErr` Indicates an error when one of the step values is not divisible by 4 for floating-point images, or by 2 for short-integer images.
- `ippStsNormErr` Indicates an error when `norm` has an incorrect or not supported value.

**Example**

To better understand usage of the `ippiMarkSpeckles` function, refer to the following example in the examples archive available for download from https://software.intel.com/en-us/ipp-manual-examples: `MarkSpeckles.c`

**See Also**

- Regions of Interest in Intel IPP
- User-defined Border Types
- `FilterGaussianGetBufferSize` Computes the size of the Gaussian specification structure and the size of the external work buffer for Gaussian filtering with user-defined borders.
- `FilterGaussianInit` Initializes the Gaussian context structure.

**SegmentWatershedGetBufferSize**

*Computes the size of the working buffer for the watershed segmentation.*
Syntax

IppStatus ippiSegmentWatershedGetSize_8u_C1R(IppSize roiSize, int* pBufSize);
IppStatus ippiSegmentWatershedGetSize_8u16u_C1R(IppSize roiSize, int* pBufSize);
IppStatus ippiSegmentWatershedGetSize_32f16u_C1R(IppSize roiSize, int* pBufSize);

Include Files

ippcv.h

Domain Dependencies

Headers: ippcore.h, ippvm.h, ipps.h, ippi.h
Libraries: ippcore.lib, ippvm.lib, ipps.lib, ippi.lib

Parameters

roiSize  Size of the source image ROI in pixels.

pBufSize  Pointer to the computed size of the working buffer.

Description

This function operates with ROI (see Regions of Interest in Intel IPP).

This function computes the size of the working buffer required for the ippiSegmentWatershed function. The buffer with the length pBufSize[0] can be used to segment images with width and/or height that is equal to or less than the corresponding field of the parameter roiSize.

Return Values

ippStsNoErr  Indicates no error. Any other value indicates an error or a warning.

ippStsNullPtrErr  Indicates an error condition if the pointer pBufSize is NULL.

ippStsSizeErr  Indicates an error condition if roiSize has a field with zero or negative value.

SegmentWatershed

Performs marker-controlled watershed segmentation of an image.

Syntax

IppStatus ippiSegmentWatershed_8u_C1IR(const Ipp8u* pSrc, int srcStep, Ipp8u* pMarker, int markerStep, IppSize roiSize, IppiNorm norm, int flag, Ipp8u* pBuffer);
IppStatus ippiSegmentWatershed_8u16u_C1IR(const Ipp8u* pSrc, int srcStep, Ipp16u* pMarker, int markerStep, IppSize roiSize, IppiNorm norm, int flag, Ipp8u* pBuffer);
IppStatus ippiSegmentWatershed_32f16u_C1IR(const Ipp32f* pSrc, int srcStep, Ipp16u* pMarker, int markerStep, IppSize roiSize, IppiNorm norm, int flag, Ipp8u* pBuffer);

Include Files

ippcv.h
Domain Dependencies
Headers: ippcore.h, ippvm.h, ipps.h,ippi.h
Libraries: ippcore.lib, ippvm.lib, ipps.lib,ippi.lib

Parameters

pSrc  
Pointer to the source image ROI.

srcStep  
Distance, in bytes, between the starting points of consecutive lines in the source image.

pMarker  
Pointer to the ROI of the source and destination image of markers.

markerStep  
Distance, in bytes, between the starting points of consecutive lines in the image with markers.

roiSize  
Size of the source and destination image ROI, in pixels.

Specifies the type of the norm to form the mask for marker propagation:

ippiNormInf  
Infinity norm (8-connectivity, 3x3 rectangular mask)

ippiNormL1  
L1 norm (4-connectivity, 3x3 cross mask)

ippiNormL2  
Approximation of L2 norm (8-connectivity, 3x3 mask) [Bor86]

ippiNormFM  
Fast marching distance [Telea04]

flag  
Specifies the algorithm of segmentation. The value is a logical sum of the following mandatory values:

IPP_SEGMENT_QUEUE  
Priority queue is used to define the order of pixel processing.

IPP_SEGMENT_DISTANCE  
Distance transform algorithm is used for segmentation.

and the following optional values:

IPP_SEGMENT_BORDER_4  
Pixels of the 4-connectivity border between image segments are marked with the IPP_MAX_8U (255) value.

IPP_SEGMENT_BORDER_8  
Pixels of the 8-connectivity border between image segments are marked with the IPP_MAX_8U (255) value.

 pBuffer  
Pointer to the working buffer.

Description
This function operates with ROI (see Regions of Interest in Intel IPP).

This function performs marker-controlled watershed segmentation of the source image. Non-zero pixels of the pMarker image belong to water source markers. Marker values propagate through the whole image according to the watershed algorithm. Image segments are formed by groups of connected pMarker pixels with the same value. The parameter norm controls marker propagation connectivity. Watershed segmentation is preferable for images with local minimums, for example, gradient images. Image markers generally correspond to these local minimums and can be created, for example, manually or using morphological reconstruction.
The parameter *flag* specifies how watershed segmentation is performed. This parameter is a logical sum of two values among the following supported values:

- **Mandatory values specifying the algorithm of segmentation:**
  
  ```
  IPP_SEGMENT_QUEUE
  Classic watershed segmentation scheme with the priority queue
  [Vincent91]
  
  IPP_SEGMENT_DISTANCE
  Watershed segmentation by calculating the topographic distance for
  each pixel [Lotufo00], [Meyer94]
  ```

- **Optional additional values of the *flag*:** IPP_SEGMENT_BORDER_4 and IPP_SEGMENT_BORDER_8 specify the border of the segments. All pixels adjacent to the differently marked pixels are considered as border pixels, and their values are set to IPP_MAX_8U (255) for 8-bit markers, or IPP_MAX_16U (65535) for 16-bit markers. In this case, the value IPP_MAX_8U (IPP_MAX_16U) should not be used to mark segments. If these optional values are not specified, segments are formed without borders.

The function requires the working buffer *pBuffer*, which size should be computed by the function ippiSegmentWatershedGetBufferSize beforehand.

**Figure "Watershed Segmentation with Different Norms"** shows the plateau filling through the watershed segmentation with different values of the *norm* parameters. Initial image (a) has the labeled with markers upper and lower fourths with low pixel value, the central plateau between them, the ridge between the plateau and the lower forth with one pixel hole in the center of it. The following pictures are segmentation results: b) - for L1 norm (block distance), c) - Linf norm (chessboard distance), d) - approximate L2 (Euclidian) norm [Bor86], e) Fast Marching distance.

**Watershed Segmentation with Different Norms**

![Watershed Segmentation with Different Norms](image)
Return Values

ippStsNoErr Indicates no error. Any other value indicates an error or a warning.
ippStsNullPtrErr Indicates an error condition if one of the specified pointers is NULL.
ippStsSizeErr Indicates an error condition if roiSize has a field with zero or negative value.
ippStsStepErr Indicates an error condition if one of the srcStep or markerStep is less than roiSize.width * < pixelSize>.
ippNotEvenStsStepErr Indicates an error condition if one of the srcStep or markerStep for 16-bit integer images is not divisible by 2.
ippStsBadArgErr Indicates an error condition if norm has an illegal value.

SegmentGradientGetBufferSize
Computes the size of the working buffer for the gradient segmentation.

Syntax
IppStatus ippiSegmentGradientGetBufferSize_8u_C1R(IppiSize roiSize, int* pBufferSize);
IppStatus ippiSegmentGradientGetBufferSize_8u_C3R(IppiSize roiSize, int* pBufSize);

Include Files
ippcv.h

Domain Dependencies
Headers: ippcore.h, ippvm.h, ipps.h,ippi.h
Libraries: ippcore.lib, ippvm.lib, ipps.lib,ippi.lib

Parameters
roiSize Size of the source image ROI in pixels.
pBufSize, pBufferSize Pointer to the computed size of the working buffer.

Description
This function operates with ROI (see Regions of Interest in Intel IPP).
This function computes the size of the working buffer required for the ippiSegmentGradient function. The buffer with the length pBufSize[0] can be used to segment images with width and/or height that is equal to or less than the corresponding field of the parameter roiSize.

Return Values

ippStsNoErr Indicates no error. Any other value indicates an error or a warning.
ippStsNullPtrErr Indicates an error condition if the pointer pBufSize is NULL.
ippStsSizeErr Indicates an error condition if roiSize has a field with zero or negative value.
**SegmentGradient**

Performs image segmentation by region growing to the least gradient direction.

**Syntax**

```c
IppStatus ippiSegmentGradient_8u_C1IR(const Ipp8u* pSrc, int srcStep, Ipp8u* pMarker,
int markerStep, IppSize roiSize, IppiNorm norm, int flags, Ipp8u* pBuffer);

IppStatus ippiSegmentGradient_8u_C3IR(const Ipp8u* pSrc, int srcStep, Ipp8u* pMarker,
int markerStep, IppSize roiSize, IppiNorm norm, int flags, Ipp8u* pBuffer);
```

**Include Files**

ippcv.h

**Domain Dependencies**

Headers: ippcore.h, ippvm.h, ipps.h,ippi.h

Libraries: ippcore.lib, ippvm.lib, ipps.lib,ippi.lib

**Parameters**

- **pSrc**: Pointer to the source image ROI.
- **srcStep**: Distance in bytes between starts of consecutive lines in the source image.
- **pMarker**: Pointer to the ROI of the source and destination image of markers.
- **markerStep**: Distance in bytes between starts of consecutive lines in the image of markers.
- **roiSize**: Size of the source and destination image ROI in pixels.
- **norm**: Specifies type of the norm to form the mask for marker propagation:
  - ippiNormInf: Infinity norm (8-connectivity, 3x3 rectangular mask);
  - ippiNormL1: L1 norm (4-connectivity, 3x3 cross mask);
- **flags**: optional flag:
  - IPP_SEGMENT_BORDER_4: pixels of the 4-connectivity border between image segments are marked with value (IPP_MAX_8U)-1 (254).
  - IPP_SEGMENT_BORDER_8: pixels of the 8-connectivity border between image segments are marked with value (IPP_MAX_8U)-1 (254).
- **pBuffer**: Pointer to the working buffer.

**Description**

This function operates with ROI (see Regions of Interest in Intel IPP).

This function performs image segmentation by region growing with markers. Non-zero pixels of pMarker image belong to initial image regions. Marker values propagate through the whole image in the direction of the least value of the absolute value of the image gradient. For 3-channel image the gradient is calculated as
the maximum of channel gradients. Image segments are formed by groups of connected \( p\text{Marker} \) pixels with the same value. The parameter \( \text{norm} \) controls marker propagation connectivity. Gradient segmentation is generally done for an image without explicit calculation of the image gradient. [Meyer92]

If \( \text{IPP\_SEGMENT\_BORDER} \) flag is defined, then the pixels adjacent to differently marked pixels are assumed to be border pixels and are set to a special value (254). This value must not be used to mark segments in this case.

Another special value (255) is used inside the function and can not be used to mark segment in any case.

The function requires the working buffer \( \text{pBuffer} \) whose size should be computed by the function \( \text{ippiSegmentGradientGetBufferSize} \) beforehand.

**Return Values**

- **ippStsNoErr**
  Indicates no error. Any other value indicates an error or a warning.

- **ippStsNullPtrErr**
  Indicates an error condition if one of the specified pointers is NULL.

- **ippStsSizeErr**
  Indicates an error condition if \( \text{roiSize} \) has a field with zero or negative value.

- **ippStsStepErr**
  Indicates an error condition if one of the \( \text{srcStep} \) or \( \text{markerStep} \) is less than \( \text{roiSize.width} < \text{pixelSize} \).

- **ippStsBadArgErr**
  Indicates an error condition if \( \text{norm} \) has an illegal value.

**BoundSegments**

Marks pixels belonging to segment boundaries.

**Syntax**

```c
IppStatus ippiBoundSegments_8u_C1IR(Ipp8u* pMarker, int markerStep, IppiSize roiSize, Ipp8u val, IppiNorm norm);
IppStatus ippiBoundSegments_16u_C1IR(Ipp16u* pMarker, int markerStep, IppiSize roiSize, Ipp16u val, IppiNorm norm);
```

**Include Files**

ippcv.h

**Domain Dependencies**

**Headers:** ippcore.h, ippvm.h, ipps.h, ippi.h

**Libraries:** ippcore.lib, ippvm.lib, ipps.lib, ippi.lib

**Parameters**

- **pMarker**
  Pointer to the ROI of the source and destination image of markers.

- **markerStep**
  Distance in bytes between starts of consecutive lines in the image of markers.

- **roiSize**
  Size of the source and destination image ROI in pixels.

- **val**
  Value of the boundary pixel.

- **norm**
  Specifies type of the norm gor pixel neighborhood:
ippiNormInf
ippiNormL1

Description
This function operates with ROI (see Regions of Interest in Intel IPP).
This function detects segment boundaries in the pMarker image and sets border pixels to the value val. A segment is the set of connected pixels of the pMarker image with the same value not equal to val. After boundaries are marked, the pMarker image does not contain any pair of adjacent in norm pixels with the same value that not equal to val.

Return Values

<table>
<thead>
<tr>
<th>IPP Status Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ippStsNoErr</td>
<td>Indicates no error. Any other value indicates an error or a warning.</td>
</tr>
<tr>
<td>ippStsNullPtrErr</td>
<td>Indicates an error condition if one of the specified pointers is NULL.</td>
</tr>
<tr>
<td>ippStsSizeErr</td>
<td>Indicates an error condition if roiSize has a field with zero or negative value.</td>
</tr>
<tr>
<td>ippStsStepErr</td>
<td>Indicates an error condition if markerStep is less than roiSize.width * pixelSize.</td>
</tr>
<tr>
<td>ippNotEvenStsStepErr</td>
<td>Indicates an error condition if markerStep for 16-bit integer images is not divisible by 2.</td>
</tr>
<tr>
<td>ippStsBadArgErr</td>
<td>Indicates an error condition if norm has an illegal value.</td>
</tr>
</tbody>
</table>

Pattern Recognition
Object Detection Using Haar-like Features

The object detector described in [ Viola01] and [ Lein02] is based on Haar classifiers. Each classifier uses $k$ rectangular areas (Haar features) to make decision if the region of the image looks like the predefined image or not. Figure “Types of Haar Features” shows different types of Haar features.

Types of Haar Features

In the Intel IPP Haar features are represented using IppRect structure. Figure “Representing Haar Features” shows how it can be done for common and tilted features.

Representing Haar Features

When the classifier $K_t$ is applied to the pixel $(i, j)$ of the image $A$, it yields the value $val_1(t)$ if

$$
\sum_{i=1}^{k} \left( \sum_{u=i+R_1y}^{u+R_1height-1} \sum_{v=j+R_1width}^{v+R_1width-1} A_{uv} \right) < \text{norm}(i, j) \cdot \text{threshold}(t)
$$

and $val_2(t)$ otherwise.

Here $w_l$ is a feature weight, $\text{norm}(i, j)$ is the norm factor (generally the standard deviation on the rectangle containing all features), $\text{threshold}(t)$, $val_1(t)$ and $val_2(t)$ are parameters of the classifier. For fast computation the integral representation of an image is used. Haar classifiers are organized in sequences.
called stages (classification stages). The stage value is the sum of its classifier values. During feature detecting stages are consequently applied to the region of the image until the stage value becomes less than the threshold value or all stages are passed.

**HaarClassifierGetSize**

*Computes the size of the structure for standard Haar classifiers.*

**Syntax**

```c
IppStatus ippiHaarClassifierGetSize(IppDataType dataType, IppiSize roiSize, const int* pNum, int length, int* pSize);
```

**Include Files**

ippcv.h

**Domain Dependencies**

*Headers:* ippcore.h, ippvm.h, ipps.h,ippi.h

*Libraries:* ippcore.lib, ippvm.lib, ipps.lib,ippi.lib

**Parameters**

- `dataType`  
  Data type of the source image. Possible values: ipp32f, ipp32s.
- `roiSize`  
  Maximal size of the source image ROI, in pixels.
- `pNum`  
  Pointer to the array of Haar classifier lengths.
- `length`  
  Number of classifiers in the stage.
- `pSize`  
  Pointer to the size of Haar classifier structure.

**Description**

This function operates with ROI (see Regions of Interest in Intel IPP).

This function computes the size of the `pState` structure that is required to calculate the sequence of Haar classifiers - classification stage. The \( i \)-th classifier in the stage has \( pNum[i] \) rectangular features.

The length of the `pThreshold`, `pVal1`, and `pVal2` vectors used in the `ippiHaarClassifierInit` function is equal to `length`.

The length of the `pFeature` and `pWeight` vectors is equal to:

\[
\sum_{i=0}^{length-1} pNum[i]
\]

**Return Values**

- `ippStsNoErr`  
  Indicates no error. Any other value indicates an error or a warning.
Indicates an error when any of the specified pointers is NULL.

Indicates an error when:
- \( \text{length} \) or one of the \( pNum[i] \) values is less than, or equal to zero
- \( \text{roiSize} \) has a field with a zero or negative value

Indicates an error when one of the features is defined incorrectly.

Indicates an error when \( \text{dataType} \) has an illegal value.

**See Also**

Regions of Interest in Intel IPP

**HaarClassifierInit** Initializes the structure for standard Haar classifiers.

**ApplyHaarClassifier** Applies a Haar classifier to an image.

**HaarClassifierInit**

*Initializes the structure for standard Haar classifiers.*

**Syntax**

```c
IppStatus ippiHaarClassifierInit_32f(IppiHaarClassifier_32f** ppState, const IppiRect* pFeature, const Ipp32f* pWeight, const Ipp32f* pThreshold, const Ipp32f* pVal1, const Ipp32f* pVal2, const int* pNum, int length);  
IppStatus ippiHaarClassifierInit_32s(IppiHaarClassifier_32s** ppState, const IppiRect* pFeature, const Ipp32s* pWeight, const Ipp32s* pThreshold, const Ipp32s* pVal1, const Ipp32s* pVal2, const int* pNum, int length);
```

**Include Files**

ippcv.h

**Domain Dependencies**

Headers: ippcore.h, ippvm.h, ipps.h,ippi.h  
Libraries: ippcore.lib, ippvm.lib, ipps.lib,ippi.lib

**Parameters**

- **ppState**
  - Double pointer to the Haar classifier structure.
- **pFeature**
  - Pointer to the array of features.
- **pWeight**
  - Pointer to the array of feature weights.
- **pThreshold**
  - Pointer to the array of classifier threshold values.
- **pVal1, pVal2**
  - Pointers to the arrays of classifier result values.
- **pNum**
  - Pointer to the array of Haar classifier lengths.
- **length**
  - Number of classifiers in the stage.

**Description**

This function initializes the state structure that is required to calculate the sequence of Haar classifiers - classification stage. The \( i \)-th classifier in the stage has \( pNum[i] \) rectangular features. Each feature is defined by a certain rectangle with horizontal and vertical sides. The length of the \( pThreshold, pVal1, \) and \( pVal2 \) vectors is equal to \( \text{length} \). The length of \( pFeature \) and \( pWeight \) is equal to:
Result of applying classifiers to the image is computed using the formula in "Object Detection Using Haar-like Features".

All features of the classifier initialized by the `ippiHaarClassifierInit` function have vertical and horizontal sides (left part of Figure "Representing Haar Features"). Some of these features then can be tilted using the `ippiTiltHaarFeatures` function.

**Return Values**

- `ippStsNoErr` Indicates no error. Any other value indicates an error or a warning.
- `ippStsNullPtrErr` Indicates an error when any of the specified pointers is `NULL`.
- `ippStsSizeErr` Indicates an error when `length` or one of the `pNum[i]` values is less than, or equal to zero.
- `ippStsBadArgErr` Indicates an error when one of the features is defined incorrectly.

**See Also**

- `HaarClassifierGetSize` Computes the size of the structure for standard Haar classifiers.
- `ApplyHaarClassifier` Applies a Haar classifier to an image.
- `Object Detection Using Haar-like Features`
- `TiltHaarFeatures` Modifies a Haar classifier by tilting specified features.

**GetHaarClassifierSize**

 Returns the size of the Haar classifier.

**Syntax**

```c
IppStatus ippiGetHaarClassifierSize_32f(IppiHaarClassifier_32f* pState, IppiSize* pSize);
IppStatus ippiGetHaarClassifierSize_32s(IppiHaarClassifier_32s* pState, IppiSize* pSize);
```

**Include Files**

`ippcv.h`

**Domain Dependencies**

- **Headers**: `ippcore.h`, `ippvm.h`, `ipps.h`, `ippi.h`
- **Libraries**: `ippcore.lib`, `ippvm.lib`, `ipps.lib`, `ippi.lib`

**Parameters**

- `pState` Pointer to the Haar classifier structure.
**pSize**

Pointer to the size of Haar classifier structure.

**Description**

This function computes the minimum size of the window containing all features of the Haar classifier described by the `pState`.

**Return Values**

- `ippStsNoErr` Indicates no error.
- `ippStsNullPtrErr` Indicates an error condition if the `pState` pointer is `NULL`.

**TiltedHaarClassifierInit**

*Initializes the structure for tilted Haar classifiers.*

**Syntax**

```c
IppStatusippiTiltedHaarClassifierInit_32f(IppiHaarClassifier_32f* pState, const IppiRect* pFeature, const Ipp32f* pWeight, const Ipp32f* pThreshold, const Ipp32f* pVal1, const Ipp32f* pVal2, const int* pNum, int length);
IppStatusippiTiltedHaarClassifierInit_32s(IppiHaarClassifier_32s* pState, const IppiRect* pFeature, const Ipp32s* pWeight, const Ipp32s* pThreshold, const Ipp32s* pVal1, const Ipp32s* pVal2, const int* pNum, int length);
```

**Include Files**

`ippcv.h`

**Domain Dependencies**

*Headers:* `ippcore.h`, `ippvm.h`, `ipps.h`, `ippi.h`

*Libraries:* `ippcore.lib`, `ippvm.lib`, `ipps.lib`, `ippi.lib`

**Parameters**

- `pState` Double pointer to the Haar classifier structure.
- `pFeature` Pointer to the array of features.
- `pWeight` Pointer to the array of feature weights.
- `pThreshold` Pointer to the array of classifier threshold values.
- `pVal1, pVal2` Pointers to the arrays of classifier result values.
- `pNum` Pointer to the array of Haar classifier lengths.
- `length` Number of classifiers in the stage.

**Description**

This function initializes the state structure that is required to calculate the sequence of Haar classifiers - classification stage. The `i`-th classifier in the stage has `pNum[i]` rectangular features. Each feature is defined by a certain rectangle with sides tilted by 45 degrees. You should specify the points with minimum and maximum row numbers. The length of the `pFeature, pFeature, pWeight, pVal1, and pVal2` vectors is equal to:
Result of applying classifiers to the image is computed using the formula in "Object Detection Using Haar-like Features".

All features of the classifier initialized by the ippiTiltedHaarClassifierInit function have tilted sides (right part of Figure "Representing Haar Features").

**Return Values**

<table>
<thead>
<tr>
<th>Status Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ippStsNoErr</td>
<td>Indicates no error. Any other value indicates an error or a warning.</td>
</tr>
<tr>
<td>ippStsNullPtrErr</td>
<td>Indicates an error when any of the specified pointers is NULL.</td>
</tr>
<tr>
<td>ippStsSizeErr</td>
<td>Indicates an error when:</td>
</tr>
<tr>
<td></td>
<td>• length or one of the pNum[i] values is less than, or equal to zero</td>
</tr>
<tr>
<td></td>
<td>• Sum of all elements of pNum is not equal to length</td>
</tr>
<tr>
<td>ippStsBadArgErr</td>
<td>Indicates an error when one of the features is defined incorrectly.</td>
</tr>
</tbody>
</table>

**See Also**

*ApplyHaarClassifier* Applies a Haar classifier to an image.

*Object Detection Using Haar-like Features*

*TiltHaarFeatures* Modifies a Haar classifier by tilting specified features.

**TiltHaarFeatures**

*Modifies a Haar classifier by tilting specified features.*

**Syntax**

```c
IppStatus ippiTiltHaarFeatures_32f(const Ipp8u* pMask, int flag, IppiHaarClassifier_32f* pState);
IppStatus ippiTiltHaarFeatures_32s(const Ipp8u* pMask, int flag, IppiHaarClassifier_32s* pState);
```

**Include Files**

ippcv.h

**Domain Dependencies**

*Headers:* ippcore.h, ippvm.h, ipps.h, ippi.h

*Libraries:* ippcore.lib, ippvm.lib, ipps.lib, ippi.lib

**Parameters**

*pMask* Pointer to the mask vector.
Flag to choose the direction of feature tilting.

Pointer to the Haar classifier structure.

Description
This function tilts specified features of the Haar classifier. Before using this function, compute the size of the Haar classifier state structure using `HaarClassifierGetSize` and initialize the structure using `TiltedHaarClassifierInit`. Non-zero elements of previously prepared vector `pMask` indicates the features that are tilted. The `flag` parameter specifies how the features are tilted:

- if `flag` is equal to 0, the feature is tilted around the left top corner clockwise
- if `flag` is equal to 1, the feature is tilted around the bottom left corner counter-clockwise

This mixed classifier containing both common and tilted features can be used by the function `ippiApplyMixedHaarClassifier`.

Return Values
- `ippStsNoErr` Indicates no error.
- `ippStsNullPtrErr` Indicates an error when one of the specified pointers is NULL.
- `ippStsBadArgErr` Indicates an error when the classifier is tilted already.

See Also
- `HaarClassifierGetSize` Computes the size of the structure for standard Haar classifiers.
- `TiltedHaarClassifierInit` Initializes the structure for tilted Haar classifiers.

ApplyHaarClassifier
Applies a Haar classifier to an image.

Syntax

```c
IppStatus ippiApplyHaarClassifier_32f_C1R(const Ipp32f* pSrc, int srcStep, const Ipp32f* pNorm, int normStep, Ipp8u* pMask, int maskStep, IppiSize roiSize, int* pPositive, Ipp32f threshold, IppiHaarClassifier_32f* pState);

IppStatus ippiApplyHaarClassifier_32s32f_C1R(const Ipp32s* pSrc, int srcStep, const Ipp32f* pNorm, int normStep, Ipp8u* pMask, int maskStep, IppiSize roiSize, int* pPositive, Ipp32f threshold, IppiHaarClassifier_32f* pState);

IppStatus ippiApplyHaarClassifier_32s_C1RSfs(const Ipp32s* pSrc, int srcStep, const Ipp32s* pNorm, int normStep, Ipp8u* pMask, int maskStep, IppiSize roiSize, int* pPositive, Ipp32s threshold, IppiHaarClassifier_32s* pState, int scaleFactor);
```

Include Files

`ippcv.h`

Domain Dependencies

Headers: `ippcore.h`, `ippvm.h`, `ipps.h`, `ippi.h`

Libraries: `ippcore.lib`, `ippvm.lib`, `ipps.lib`, `ippi.lib`

Parameters

- `pSrc` Pointer to the ROI in the source image of integrals.
**Description**

This function operates with ROI (see Regions of Interest in Intel IPP).

This function applies the Haar classifier to pixels of the source image ROI \( pSrc \). The source image should be in the integral representation, it can be obtained by calling one of the integral functions beforehand. The sum of pixels on feature rectangles is computed as:

\[
\sum_{i=1}^{k} (pSrc[i+y_1,j+x_1] - pSrc[i+y_2,j+x_1]) - pSrc[i+y_2,j+x_1] + pSrc[i+y_2,j+x_1]) \cdot w_i
\]

Here \((y_1, x_1)\) and \((y_2, x_2)\) are coordinates of top left and right bottom pixels of \( l \)-th rectangle of the feature, and \( w_i \) is the feature weight. For \( i = 0. \text{roiSize.height} - 1, j = 0. \text{roiSize.width} - 1 \) all pixels referred in the above formula should be allocated in memory.

The input value of \( pPositive[0] \) is used as a hint to choose the calculation algorithm. If it is greater than or equal to \( \text{roiSize.width} \cdot \text{roiSize.height} \), the value of the classifier is calculated in accordance with the above formula for all pixels of the input image. Otherwise the value of the classifier is calculated for all non-zero pixels of \( pMask \) image. If the sum is less than \( \text{threshold} \) than the negative decision is made and the value of the corresponding pixel of the \( pMask \) image is set to zero. The number of positive decisions is assigned to the \( pPositive[0] \).

Before using this function, you need to compute the size of the state structure using HaarClassifierGetSize and initialize the structure using HaarClassifierInit or TiltedHaarClassifierInit.

**Return Values**

- **ippStsNoErr**: Indicates no error.
- **ippStsNullPtrErr**: Indicates an error when one of the specified pointers is NULL.
- **ippStsSizeErr**: Indicates an error when \( \text{roiSize} \) has a field with a zero or negative value.
- **ippStsStepErr**: Indicates an error when one of the image step values is less than \( \text{roiSize.width} \cdot \text{pixelSize} \).
- **ippStsNorEvenStepErr**: Indicates an error when one of the image step values is not divisible by 4 for 32-bit images.
See Also
HaarClassifierGetSize Computes the size of the structure for standard Haar classifiers.
HaarClassifierInit Initializes the structure for standard Haar classifiers.
TiltedHaarClassifierInit Initializes the structure for tilted Haar classifiers.

ApplyMixedHaarClassifier
Applies a mixed Haar classifier to an image.

Syntax
IppStatus ippiApplyMixedHaarClassifier_32f_C1R(const Ipp32f* pSrc, int srcStep, const Ipp32f* pTilt, int tiltStep, const Ipp32f* pNorm, int normStep, Ipp8u* pMask, int maskStep, IppiSize roiSize, int* pPositive, Ipp32f threshold, IppiHaarClassifier_32f* pState);
IppStatus ippiApplyMixedHaarClassifier_32s32f_C1R(const Ipp32s* pSrc, int srcStep, const Ipp32s* pTilt, int tiltStep, const Ipp32f* pNorm, int normStep, Ipp8u* pMask, int maskStep, IppiSize roiSize, int* pPositive, Ipp32f threshold, IppiHaarClassifier_32f* pState);
IppStatus ippiApplyMixedHaarClassifier_32s_C1RSfs(const Ipp32s* pSrc, int srcStep, const Ipp32s* pTilt, int tiltStep, const Ipp32s* pNorm, int normStep, Ipp8u* pMask, int maskStep, IppiSize roiSize, int* pPositive, Ipp32s threshold, IppiHaarClassifier_32s* pState, int scaleFactor);

Include Files
ippcv.h

Domain Dependencies
Headers: ippcore.h, ippvm.h, ipps.h,ippi.h
Libraries: ippcore.lib, ippvm.lib, ipps.lib,ippi.lib

Parameters
pSrc Pointer to the ROI in the source image of integrals.
srcStep Distance, in bytes, between the starting points of consecutive lines in the source image of integrals.
pTilt Pointer to the ROI in the source image of tilted integrals.
tiltStep Distance, in bytes, between the starting points of consecutive lines in the source image of tilted integrals.
pNorm Pointer to the ROI in the source image of norm factors.
normStep Distance, in bytes, between the starting points of consecutive lines in the image of the norm factors.
pMask Pointer to the source and destination image of classification decisions.
maskStep Distance, in bytes, between the starting points of consecutive lines in the image of classification decisions.
pPositive Pointer to the number of positive decisions.
roiSize Size of the source and destination images ROI in pixels.
threshold Stage threshold value.
Description

This function operates with ROI (see Regions of Interest in Intel IPP).

This function applies the mixed Haar classifier \( pState \) to the ROI of the source images \( pSrc \) and \( pTilt \). The mixed Haar classifier is a classifier initialized by \texttt{HaarClassifierInit} and then modified by the \texttt{TiltHaarFeatures} function. The source images must be in the integral representation, they can be obtained by calling one of the integral functions beforehand. Common features are applied to the \( pSrc \) image, and tilted features are applied to the \( pTilt \) image. The sum of pixels on feature rectangles is computed as:

\[
\sum_{i=1}^{k} (pSrc[i+y_1,f+x_1] - pSrc[i+y_1,f+x_1] - pSrc[i+y_1,f+x_1])w_i
\]

Here \((y_1, x_1)\) and \((Y_1, X_1)\) are coordinates of top left and right bottom pixels of \( l \)-th rectangle of the feature, and \( w_i \) is the feature weight. For \( i = 0. \) \textit{roiSize.height - 1}, \( j = 0. \) \textit{roiSize.width - 1} all pixels referred in the above formula should be allocated in memory.

The input value of \( pPositive[0] \) is used as a hint to choose the calculation algorithm. If it is greater than or equal to \( \text{roiSize.width*roiSize.height} \) the value of the classifier is calculated in accordance with the above formula for all pixels of the input image. Otherwise the value of the classifier is calculated for all non-zero pixels of \( pMask \) image. If the sum is less than \textit{threshold} than the negative decision is made and the value of the corresponding pixel of the \( pMask \) image is set to zero. The number of positive decisions is assigned to the \( pPositive[0] \).

Return Values

- \texttt{ippStsNoErr} Indicates no error.
- \texttt{ippStsNullPtrErr} Indicates an error when one of the specified pointers is \texttt{NULL}.
- \texttt{ippStsSizeErr} Indicates an error when \textit{roiSize} has a field with a zero or negative value.
- \texttt{ippStsStepErr} Indicates an error when one of the image step values is less than \textit{roiSize.width*<pixelSize>}.
- \texttt{ippStsNorEvenStepErr} Indicates an error when one of the image step values is not divisible by 4 for 32-bit images.

See Also

- Regions of Interest in Intel IPP
- \texttt{HaarClassifierInit} Initializes the structure for standard Haar classifiers.
- \texttt{TiltHaarFeatures} Modifies a Haar classifier by tilting specified features.

Local Binary Pattern (LBP) Operator

The local binary pattern (LBP) operator transforms an image into an array, or to an image with integer labels. Integer labels describe small-scale view of the image. For grayscale images, these labels represent a texture descriptor of the image. Integer labels statistics are used for image analysis. Changes of the monotonic gray level do not affect the LBP operator.
Intel® IPP functions described in this section use LBP operators with mask size 3x3 and 5x5. The LBPImageMode functions support four modes of LBP calculation set by the mode parameter. The LBPImage functions compute LBP similar to the LBPImageMode functions with the mode value equal to 1. The LBP operator with 3x3 mask uses neighborhood consisting of eight pixels, as shown in the figures below.

**mode=0:**

<table>
<thead>
<tr>
<th></th>
<th>1</th>
<th>8</th>
<th>7</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>A</td>
<td>6</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>4</td>
<td>5</td>
<td></td>
</tr>
</tbody>
</table>

**Anchor Point**

**mode=1:**

<table>
<thead>
<tr>
<th></th>
<th>1</th>
<th>2</th>
<th>3</th>
</tr>
</thead>
<tbody>
<tr>
<td>8</td>
<td>A</td>
<td>4</td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>6</td>
<td>5</td>
<td></td>
</tr>
</tbody>
</table>

**mode=2:**

<table>
<thead>
<tr>
<th>8</th>
<th>7</th>
<th>6</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>A</td>
<td>5</td>
</tr>
<tr>
<td>2</td>
<td>3</td>
<td>4</td>
</tr>
</tbody>
</table>

**mode=3:**

<table>
<thead>
<tr>
<th>2</th>
<th>3</th>
<th>4</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>A</td>
<td>5</td>
</tr>
<tr>
<td>8</td>
<td>7</td>
<td>6</td>
</tr>
</tbody>
</table>

The LBP operator with 5x5 mask uses neighborhood consisting of 16 pixels, as shown in the figures below.

**mode=0:**
### mode = 1:

<table>
<thead>
<tr>
<th></th>
<th>0</th>
<th>15</th>
<th>14</th>
<th>13</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>16</td>
<td>0</td>
<td>12</td>
<td>11</td>
</tr>
<tr>
<td>1</td>
<td>16</td>
<td>0</td>
<td>0</td>
<td>10</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>0</td>
<td>A</td>
<td>0</td>
<td>10</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>4</td>
<td>0</td>
<td>8</td>
<td>9</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>5</td>
<td>6</td>
<td>7</td>
<td>0</td>
<td></td>
</tr>
</tbody>
</table>

### mode = 2:

<table>
<thead>
<tr>
<th></th>
<th>0</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>2</td>
<td>0</td>
<td>6</td>
<td>7</td>
</tr>
<tr>
<td>1</td>
<td>2</td>
<td>0</td>
<td>6</td>
<td>7</td>
<td></td>
</tr>
<tr>
<td>16</td>
<td>0</td>
<td>A</td>
<td>0</td>
<td>8</td>
<td></td>
</tr>
<tr>
<td>15</td>
<td>14</td>
<td>0</td>
<td>10</td>
<td>9</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>13</td>
<td>12</td>
<td>11</td>
<td>0</td>
<td></td>
</tr>
</tbody>
</table>

### mode = 3:

<table>
<thead>
<tr>
<th></th>
<th>0</th>
<th>14</th>
<th>13</th>
<th>12</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>16</td>
<td>15</td>
<td>0</td>
<td>11</td>
<td>10</td>
</tr>
<tr>
<td>16</td>
<td>15</td>
<td>0</td>
<td>11</td>
<td>10</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>A</td>
<td>0</td>
<td>9</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>3</td>
<td>0</td>
<td>7</td>
<td>8</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>4</td>
<td>5</td>
<td>6</td>
<td>0</td>
<td></td>
</tr>
</tbody>
</table>
In the above figures:
- Numbers indicate the position of the corresponding bit in a resulting label
- The A letter indicates the anchor point position.

The LBP operator does the following when processing an image:
- Compares each pixel neighboring to the anchor with the anchor pixel in accordance with the neighboring pixel order. If the neighboring pixel value is more than, or equal to the anchor point value, the result is 1. If the neighboring pixel value is less than the anchor point value, the result is 0.
- Puts the result of comparison to the corresponding bit of the resulting label, as shown in the figure below.

**LBPI mageMode**
*Calculates LBP of the image according to the specified mode.*

**Syntax**

```
IppStatus ippiLBPImageMode3x3_<mod>(const Ipp<srcDatatype>* pSrc, int srcStep,
Ipp<dstDatatype>* pDst, int dstStep, IppiSize dstRoiSize, int mode, IppiBorderType
borderType, const Ipp<srcDatatype>* borderValue);
```

Supported values for `mod`:

- 8u_C1R
- 32f8u_C1R

```
IppStatus ippiLBPImageMode5x5_<mod>(const Ipp<srcDatatype>* pSrc, int srcStep,
Ipp<dstDatatype>* pDst, int dstStep, IppiSize dstRoiSize, int mode, IppiBorderType
borderType, const Ipp<srcDatatype>* borderValue);
```
Include Files
ippi.h

Domain Dependencies
Headers: ippcore.h, ippvm.h, ipps.h
Libraries: ippcore.lib, ippvm.lib, ipps.lib

Parameters

- **pSrc**: Pointer to the source image ROI.
- **srcStep**: Distance, in bytes, between the starting points of consecutive lines in the source image.
- **pDst**: Pointer to the destination image ROI.
- **dstStep**: Distance, in bytes, between the starting points of consecutive lines in the destination image.
- **dstRoiSize**: Size of the destination ROI, in pixels.
- **mode**: Mode for LBP calculation. Supported values are 0, 1, 2, 3.
- **borderType**: Type of border. Possible values are:
  - ippBorderRepl: Border is replicated from the edge pixels.
  - ippBorderInMem: Border is obtained from the source image pixels in memory.
  - Mixed borders are also supported. They can be obtained by the bitwise operation OR between ippBorderRepl and ippBorderInMemTop, ippBorderInMemBottom, ippBorderInMemLeft, ippBorderInMemRight.
- **borderValue**: Constant value to assign to pixels of the constant border. This parameter is applicable only to the ippBorderConst border type.

Description
These functions operate with ROI (see Regions of Interest in Intel IPP).

The ippILBPImageMode3x3 and ippILBPImageMode5x5 functions calculate LBP of the pSrc image ROI according to the mode value. The result is stored in the pDst destination image.

Return Values

- ippStsNoErr: Indicates no error.
- ippStsNullPtrErr: Indicates an error when one of the specified pointers is NULL.
- ippStsSizeErr: Indicates an error if dstRoiSize has a field with a zero or negative value.
- ippStsBadArgErr: Indicates an error when border has an illegal value.

Example
To better understand usage of the ippILBPImageMode function, refer to the following example in the examples archive available for download from https://software.intel.com/en-us/ipp-manual-examples:
LBPI mageMode.c

**See Also**
Regions of Interest in Intel IPP

**LBPI mageHorizCorr**
*Calculates a correlation between two LBPs.*

**Syntax**

```c
IppStatus ippiLBPI mageHorizCorr_<mod>(const Ipp<datatype>* pSrc1, int src1Step, const Ipp<datatype>* pSrc2, int src2Step, Ipp<datatype>* pDst, int dstStep, IppiSize dstRoiSize, int horShift, IppiBorderType borderType, const Ipp<datatype>* borderValue);
```

Supported values for `mod`:

- 8u_C1R
- 16u_C1R

**Include Files**
ippi.h

**Domain Dependencies**
Headers: ippcore.h, ippvm.h, ipps.h
Libraries: ippcore.lib, ippvm.lib, ipps.lib

**Parameters**

- `pSrc1`, `pSrc2`: Pointers to the source image ROI.
- `src1Step`, `src2Step`: Distance, in bytes, between the starting points of consecutive lines in the source image.
- `pDst`: Pointer to the destination image ROI.
- `dstStep`: Distance, in bytes, between the starting points of consecutive lines in the destination image.
- `dstRoiSize`: Size of the destination ROI in pixels.
- `horShift`: Horizontal shift of the `pSrc2` image.
- `borderType`: Type of border. Possible values are:
  - ippBorderRepl: Border is replicated from the edge pixels.
  - ippBorderInMem: Border is obtained from the source image pixels in memory.
  - Mixed borders are also supported. They can be obtained by the bitwise operation OR between ippBorderRepl and ippBorderInMemTop, ippBorderInMemBottom, ippBorderInMemLeft, ippBorderInMemRight.
- `borderValue`: Constant value to assign to pixels of the constant border. This parameter is applicable only to the ippBorderConst border type.

**Description**
This function operates with ROI.
This function calculates the difference between two LBP images. The result is stored in the `pDst` destination image.

**Return Values**

- **ippStsNoErr**: Indicates no error.
- **ippStsNullPtrErr**: Indicates an error when one of the specified pointers is NULL.
- **ippStsSizeErr**: Indicates an error if `dstRoiSize` has a field with zero or negative value.
- **ippStsBadArgErr**: Indicates an error when `border` has an illegal value.

**Example**

To better understand usage of the `ippiLBPImageHorizCorr` function, refer to the following example in the examples archive available for download from https://software.intel.com/en-us/ipp-manual-examples:

`LBPImageHorizCorr.c`

**See Also**

- Regions of Interest in Intel IPP
- Borders in Neighborhood Operations
- User-defined Border Types
- Camera Calibration and 3D Reconstruction

**Camera Calibration and 3D Reconstruction**

**Correction of Camera Lens Distortion**

Digital camera usually introduces significant distortion caused by the camera and lens. These distortions cause errors in any analysis of the image. The functions described in this section correct these distortion using intrinsic camera parameters and distortion coefficients. These intrinsic camera parameters are focal lengths $f_x$, $f_y$, and principal point coordinates $c_x$, $c_y$. The distortion is characterized by two coefficients of radial distortions $k_1$, $k_2$ and two coefficients of tangential distortions $p_1$, $p_2$.

The undistorted coordinates $x_u$ and $y_u$ of point with coordinates $(x_d, y_d)$ are computed in accordance with the following formulas:

$$
x_u = x_d \cdot \left(1 + k_1 r^2 + k_2 r^4 \right) + 2 p_1 x_d y_d + p_2 \cdot \left(r^2 + 2x^2_d\right)
$$

$$
y_u = y_d \cdot \left(1 + k_1 r^2 + k_2 r^4 \right) + 2 p_2 x_d y_d + p_1 \cdot \left(r^2 + 2y^2_d\right)
$$

Here $r^2 = x_d^2 + y_d^2$, $x_d = (j-cx)/f_x$, $y_d = (i-cy)/f_y$; $i$ and $j$ are row and columns numbers of the pixel. The pixel value is computed using bilinear interpolation of four nearest pixel of the source image. If undistorted coordinates are outside the image, then the destination pixel is not changed.

**UndistortGetSize**

*Computes the size of the external buffer.*

**Syntax**

```c
IppStatus ippiUndistortGetSize(IppSize roiSize, int* pBufferSize);
```
Include Files
ippcv.h

Domain Dependencies
Headers: ippcore.h, ippvm.h, ipps.h, ippi.h
Libraries: ippcore.lib, ippvm.lib, ipps.lib, ippi.lib

Parameters

roiSize
Size of source and destination images ROI in pixels.
pBufferSize
Pointer to the computed value of the buffer size.

Description
This function computes the size of the temporary external buffer that is used by the functions
ippiUndistortRadial. The buffer of the computed size can be used to process smaller images as well.

Return Values

ippStsNoErr
Indicates no error.
ippStsNullPtrErr
Indicates an error if pBufferSize is NULL.
ippStsSizeErr
Indicates an error condition if roiSize has a field with zero or negative value.

UndistortRadial
Corrects radial distortions of the single image.

Syntax
IppStatus ippiUndistortRadial_<mod>(const Ipp<datatype>* pSrc, int srcStep, Ipp<datatype>* pDst, int dstStep, IppSize roiSize, Ipp32f fx, Ipp32f fy, Ipp32f cx, Ipp32f cy, Ipp32f k1, Ipp32f k2, Ipp8u* pBuffer);

Supported values for mod:

8u_C1R  16u_C1R  32f_C1R
8u_C3R  16u_C3R  32f_C3R

Include Files
ippcv.h

Domain Dependencies
Headers: ippcore.h, ippvm.h, ipps.h, ippi.h
Libraries: ippcore.lib, ippvm.lib, ipps.lib, ippi.lib

Parameters

pSrc
Pointer to the ROI in the source distorted image.
srcStep
Distance in bytes between starts of consecutive lines in the source image.
pDst
Pointer to the ROI in the destination corrected image.

dstStep
Distance in bytes between starts of consecutive lines in the
destination image.

roiSize
Size of source and destination images ROI in pixels.

fx
Focal lengths along the x axis.

fy
Focal lengths along the y axis.

cx
x-coordinate of the principal point.

cy
y-coordinate of the principal point.

k1
First coefficient of radial distortion.

k2
Second coefficient of radial distortion.

pBuffer
Pointer to the external buffer.

Description
This function operates with ROI (see Regions of Interest in Intel IPP).
This function corrects radial distortions of the single source image pSrc and stores corrected image in the
pDst. Correction is performed accounting camera parameters fx, fy, cx, cy and radial distortion parameters
k1, k2. The function can also pass the pointer to the external buffer pBuffer whose size should be computed
previously using the function ippiUndistortGetSize. If a null pointer is passed, slower computations
without an external buffer will be performed.

Return Values
ippStsNoErr Indicates no error.
ippStsNullPtrErr Indicates an error if pSrc or pDst is NULL.
ippStsSizeErr Indicates an error condition if roiSize has a field with zero or
negative value.
ippStsStepErr Indicates an error condition if srcStep is less than
roiSize.width * <pixelSize>, or dstStep is less than
roiSize.width * <pixelSize>.
ippStsNotEvenStepErr Indicates an error condition if one of the step values is not
divisible by 4 for floating-point images, or by 2 for short-
integer images.
ippStsBadArgErr Indicates an error if fx or fy is equal to 0.

CreateMapCameraUndistort
Creates look-up tables of coordinates of corrected
image.

Syntax
IppStatus ippiCreateMapCameraUndistort_32f_C1R(Ipp32f* pxMap, int xStep, Ipp32f* pyMap,
int yStep, IppiSize roiSize, Ipp32f fx, Ipp32f fy, Ipp32f cx, Ipp32f cy, Ipp32f k1,
Ipp32f k2, Ipp32f p1, Ipp32f p2, Ipp8u* pBuffer);

Include Files
ippcv.h
Domain Dependencies

Headers: ippcore.h, ippvm.h, ipps.h,ippi.h
Libraries: ippcore.lib, ippvm.lib, ipps.lib,ippi.lib

Parameters

pxMap Pointer to the destination x coordinate look-up buffer.
xStep Distance in bytes between starts of consecutive lines in the pxMap image.
pyMap Pointer to the destination y coordinate look-up buffer.
yStep Distance in bytes between starts of consecutive lines in the pyMap image.
roiSize Size of source and destination images ROI in pixels.
fx Focal lengths along the x axis.
fy Focal lengths along the y axis.
cx x-coordinate of the principal point.
cy y-coordinate of the principal point.
k1 First coefficient of radial distortion.
k2 Second coefficient of radial distortion.
p1 First coefficient of tangential distortion.
p2 Second coefficient of tangential distortion.
pBuffer Pointer to the external buffer.

Description

This function operates with ROI (see Regions of Interest in Intel IPP ).

This function creates the look-up tables of x- and y-coordinates pxMap and pyMap respectively. These coordinates are computed in accordance with camera parameters fx, fy, cx, cy, and distortion parameters k1, k2, p1, p2. The created tables can be used by the Intel IPP functionippiRemap to remap the distorted source image and get the corrected image.

To accelerate the computations the function can pass the pointer to the external buffer pBuffer whose size should be computed previously using the functionippiUndistortGetSize. If a null pointer is passed, slower computations without an external buffer will be performed.

Return Values

ippStsNoErr Indicates no error.
ippStsNullPtrErr Indicates an error if pxMap or pyMap is NULL.
ippStsSizeErr Indicates an error condition if roiSize has a field with zero or negative value.
ippStsStepErr Indicates an error condition if: xStep is less than roiSize.width * <pixelSize>, or yStep is less than roiSize.width * <pixelSize>.
ippStsNotEvenStepErr Indicates an error condition if one of the step values is not divisible by 4 for floating-point images.
ippStsBadArgErr Indicates an error when $fx$ or $fy$ is equal to 0.

Example
To better understand usage of the ippCreateMapCameraUndistort function, refer to the following example in the examples archive available for download from https://software.intel.com/en-us/ipp-manual-examples:

CreateMapCameraUndistort.c
This section describes the Intel® Integrated Performance Primitives (Intel® IPP) functions that perform 3D data transforms - resizing, affine transform, and remapping, as well as functions for 3D data linear filtering.

## CopyConstBorder

Copies pixel values between two 3D images and adds border pixels with a constant value.

### Syntax

```c
```

```c
```

```c
IppStatus ipprCopyConstBorder_16s_C1V_L(const Ipp16s* pSrc, IppSizeL srcPlaneStep, IppSizeL srcStep, IpprVolumeL srcRoiVolume, Ipp16s* pDst, IppSizeL dstPlaneStep, IppSizeL dstStep, IpprVolumeL dstRoiVolume, IppSizeL topBorderHeight, IppSizeL leftBorderWidth, IppSizeL forwardBorderDepth, const Ipp16s* value);
```

```c
```

```c
IppStatus ipprCopyConstBorder_64f_C1V_L(const Ipp64f* pSrc, IppSizeL srcPlaneStep, IppSizeL srcStep, IpprVolumeL srcRoiVolume, Ipp64f* pDst, IppSizeL dstPlaneStep, IppSizeL dstStep, IpprVolumeL dstRoiVolume, IppSizeL topBorderHeight, IppSizeL leftBorderWidth, IppSizeL forwardBorderDepth, const Ipp64f* value);
```

### Include Files

ipp_i.h

### Domain Dependencies

**Headers:** ippcore.h, ippvm.h, ipps.h

**Libraries:** ippcore.lib, ippvm.lib, ipps.lib

### Parameters

- **pSrc**
  Array of pointers to the planes in the source volume.

- **srcStep**
  Distance, in bytes, between the starting points of consecutive lines in each plane of the source volume.

- **srcPlaneStep**
  Distance, in bytes, between the starting points of consecutive images in every plane of the source volume.
**srcRoiVolume**

Volume of the source ROI in pixels.

**pDst**

Array of pointers to the planes in the destination volume.

**dstStep**

Distance, in bytes, between the starting points of consecutive lines in each plane of the destination volume.

**dstPlaneStep**

Distance, in bytes, between the starting points of consecutive images in every plane of the source volume.

**dstRoiVolume**

Volume of the destination ROI in pixels.

**topBorderHeight**

Height of the top border in pixels.

**leftBorderWidth**

Width of the left border in pixels.

**forwardBorderDepth**

Depth of the forward border in pixels.

**value**

Constant value to assign to the border pixels.

**Description**

This function operates with VOI. This function copies the source image `pSrc` with the volume `srcRoiVolume` to the destination image `pDst` with the volume `dstRoiVolume` and creates a border outside the copied area. The function sets pixel values of the border to the specified constant value that is passed by the `value` argument.

The image below shows the mapping of the parameters `topBorderHeight`, `leftBorderWidth`, and `forwardBorderDepth` onto the dimensions of the three-dimensional space.

**Return Values**

- **ippStsNullPtrErr**
  
  Indicates an error condition if `pSrc`, `pDst` or `value` pointer is NULL.

- **ippStsStepErr**
  
  Indicates an error condition if `srcPlaneStep`, `srcStep`, `dstPlaneStep` or `dstStep` has a field with negative value.

- **ippStsSizeErr**
  
  Indicates an error condition if `leftBorderWidth`, `topBorderHeight` or `forwardBorderDepth` has a field with negative value.
See Also
CopyReplicateBorder Copies pixel values between two 3D images and adds replicated border pixels.

Structures and Enumerators for Platform-Aware Functions

CopyReplicateBorder

Copies pixel values between two 3D images and adds replicated border pixels.

Syntax

```c
```

```c
```

```c
```

```c
```

```c
IppStatus ipprCopyReplicateBorder_64f_C1V_L(const Ipp64f* pSrc, IppSizeL srcPlaneStep, IppSizeL srcStep, IpprVolumeL SrcRoiVolume, Ipp64f* pDst, IppSizeL dstPlaneStep, IppSizeL dstStep, IpprVolumeL DstRoiVolume, IppSizeL topBorderHeight, IppSizeL leftBorderWidth, IppSizeL forwardBorderDepth);
```

Include Files
ippi_l.h

Domain Dependencies

Headers: ippcore.h, ippvm.h, ipps.h
Libraries: ippcore.lib, ippvm.lib, ipps.lib

Parameters

- **pSrc**
  - Array of pointers to the planes in the source volume.
- **srcStep**
  - Distance, in bytes, between the starting points of consecutive lines in each plane of the source volume.
- **srcPlaneStep**
  - Distance, in bytes, between the starting points of consecutive images in every plane of the source volume.
- **srcRoiVolume**
  - Volume of the source ROI in pixels.
- **pDst**
  - Array of pointers to the planes in the destination volume.
**dstStep**
Distance, in bytes, between the starting points of consecutive lines in each plane of the destination volume.

**dstPlaneStep**
Distance, in bytes, between the starting points of consecutive images in every plane of the source volume.

**dstRoiVolume**
Volume of the destination ROI in pixels.

**topBorderHeight**
Height of the top border in pixels.

**leftBorderWidth**
Width of the left border in pixels.

**forwardBorderDepth**
Depth of the forward border in pixels.

**Description**
This function operates with VOI. This function copies the source image `pSrc` with the volume `srcRoiVolume` to the destination image `pDst` with the volume `dstRoiVolume`. The function fills pixels ('border') outside the copied area in the destination image with the values of the source image pixels.

The image below shows the mapping of the parameters `topBorderHeight`, `leftBorderWidth`, and `forwardBorderWidth` onto the dimensions of the three-dimensional space.

**Return Values**

- **ippStsNullPtrErr**
  Indicates an error condition if `pSrc` or `pDst` pointer is NULL.

- **ippStsStepErr**
  Indicates an error condition if `srcPlaneStep` value is less than `srcStep` value or if `dstPlaneStep` value is less than `dstStep` value.

- **ippStsSizeErr**
  Indicates an error condition if `leftBorderWidth`, `topBorderHeight` or `forwardBorderDepth` has a field with negative value.

**See Also**

- **CopyConstBorder** Copies pixel values between two 3D images and adds border pixels with a constant value.

**Structures and Enumerators for Platform-Aware Functions**
Filter

Filters a volume using a general cuboidal kernel.

Syntax

IppStatus ipprFilter_16s_C1PV(const Ipp16s* const pSrc[], int srcStep, const Ipp16s* pDst[], int dstStep, IpprVolume dstVolume, const Ipp32s* pKernel, IpprVolume kernelVolume, IpprPoint anchor, int divisor, Ipp8u* pBuffer);

IppStatus ipprFilter_16s_C1V(const Ipp16s* pSrc, int srcStep, int srcPlaneStep, Ipp16s* pDst, int dstStep, int dstPlaneStep, IpprVolume dstVolume, const Ipp32s* pKernel, IpprVolume kernelVolume, IpprPoint anchor, int divisor, Ipp8u* pBuffer);

Include Files

ippi.h

Domain Dependencies

Headers: ippcore.h, ippvm.h, ipps.h
Libraries: ippcore.lib, ippvm.lib, ipps.lib

Parameters

pSrc

Array of pointers to the planes in the source volume.

srcStep

Distance, in bytes, between the starting points of consecutive lines in each plane of the source volume.

call srcPlaneStep

Distance, in bytes, between the starting points of consecutive lines in every plane of the source volume (for the 16s_C1V flavor).

pDst

Array of pointers to the planes in the destination volume.

dstStep

Distance, in bytes, between the starting points of consecutive lines in each plane of the destination volume.

call dstPlaneStep

Distance, in bytes, between the starting points of consecutive lines in every plane of the destination volume (for the 16s_C1V flavor).

dstVolume

Size of the processed volume.

pKernel

Pointers to the kernel values.

kernelVolume

Size of the kernel volume.

anchor

Anchor 3d-cell specifying the cuboidal kernel alignment with respect to the position of the input voxel.

divisor

The integer value by which the computed result is divided.

pBuffer

Pointer to the external buffer.

Description

This function operates with VOI. This function uses the general cuboidal kernel of size kernelVolume to filter a volume VOI. This function sums the products between the kernel coefficients pKernel and voxel values taken over the source voxel neighborhood defined by kernelVolume and anchor. The anchor 3d-cell is specified by its coordinates anchor.x, anchor.y and anchor.z in the coordinate system associated with the volume.
right bottom back corner of the kernel. Note the kernel coefficients are used in inverse order. The sum is written to the destination voxel. To ensure valid operation when volume boundary voxels are processed, the application must correctly define additional border voxels.

**Return Values**

- **ippStsNoErr**: Indicates no error. Any other value indicates an error.
- **ippStsNullPtrErr**: Indicates an error condition if `pSrc`, `pDst`, `pKernel` or `pBuffer` pointer is `NULL`.
- **ippStsSizeErr**: Indicates an error condition if `dstVolume` or `kernelVolume` has a field with zero or negative value.
- **ippStsDivisorErr**: Indicates an error condition if the divisor value is zero.

**FilterGetBufSize**

*Calculates the size of the working buffer.*

**Syntax**

```c
IppStatus ipprFilterGetBufSize(IpprVolume dstVolume, IpprVolume kernelVolume, int nChannel, int* pSize);
```

**Include Files**

`ippi.h`

**Domain Dependencies**

*Headers: ippcore.h, ippvm.h, ipps.h*

*Libraries: ippcore.lib, ippvm.lib, ipps.lib*

**Parameters**

- `dstVolume`: Size of the processed volume.
- `kernelVolume`: Size of the kernel volume.
- `nChannel`: Number of channels or planes, possible value is one.
- `pSize`: Pointer to the size of the external buffer.

**Description**

This function operates with VOI. This function computes the size of the working buffer `pSize` that is required for the function `ipprFilter`.

**Return Values**

- **ippStsNoErr**: Indicates no error. Any other value indicates an error.
- **ippStsNullPtrErr**: Indicates an error condition if `pSize` pointer is `NULL`.
- **ippStsNumChannelErr**: Indicates an error condition if `nChannel` has an illegal value.
- **ippStsSizeErr**: Indicates an error condition if `dstVolume` or `kernelVolume` has a field with zero or negative value.
FilterBorder

Filters a 3D image using a rectangular filter.

Syntax

IppStatus ipprFilterBorder_8u_C1V(const Ipp8u* pSrc, int srcPlaneStep, int srcStep, Ipp8u* pDst, int dstPlaneStep, int dstStep, IpprVolume dstRoiVolume, IpprBorderType borderType, const Ipp8u borderValue[1], const IppFilterBorderSpec* pSpec, Ipp8u* pBuffer);

IppStatus ipprFilterBorder_16s_C1V(const Ipp16s* pSrc, int srcPlaneStep, int srcStep, Ipp16s* pDst, int dstPlaneStep, int dstStep, IpprVolume dstRoiVolume, IpprBorderType borderType, const Ipp16s borderValue[1], const IppFilterBorderSpec* pSpec, Ipp8u* pBuffer);

IppStatus ipprFilterBorder_16u_C1V(const Ipp16u* pSrc, int srcPlaneStep, int srcStep, Ipp16u* pDst, int dstPlaneStep, int dstStep, IpprVolume dstRoiVolume, IpprBorderType borderType, const Ipp16u borderValue[1], const IppFilterBorderSpec* pSpec, Ipp8u* pBuffer);

IppStatus ipprFilterBorder_32f_C1V(const Ipp32f* pSrc, int srcPlaneStep, int srcStep, Ipp32f* pDst, int dstPlaneStep, int dstStep, IpprVolume dstRoiVolume, IpprBorderType borderType, const Ipp32f borderValue[1], const IppFilterBorderSpec* pSpec, Ipp8u* pBuffer);

IppStatus ipprFilterBorder_64f_C1V(const Ipp64f* pSrc, int srcPlaneStep, int srcStep, Ipp64f* pDst, int dstPlaneStep, int dstStep, IpprVolume dstRoiVolume, IpprBorderType borderType, const Ipp64f borderValue[1], const IppFilterBorderSpec* pSpec, Ipp8u* pBuffer);

Platform-aware functions

IppStatus ipprFilterBorder_8u_C1V_L(const Ipp8u* pSrc, IppSizeL srcPlaneStep, IppSizeL srcStep, Ipp8u* pDst, IppSizeL dstPlaneStep, IppSizeL dstStep, IpprVolumeL dstRoiVolume, IpprBorderType borderType, const Ipp8u borderValue[1], const IppFilterBorderSpec* pSpec, Ipp8u* pBuffer);

IppStatus ipprFilterBorder_16s_C1V_L(const Ipp16s* pSrc, IppSizeL srcPlaneStep, IppSizeL srcStep, Ipp16s* pDst, IppSizeL dstPlaneStep, IppSizeL dstStep, IpprVolumeL dstRoiVolume, IpprBorderType borderType, const Ipp16s borderValue[1], const IppFilterBorderSpec* pSpec, Ipp8u* pBuffer);

IppStatus ipprFilterBorder_16u_C1V_L(const Ipp16u* pSrc, IppSizeL srcPlaneStep, IppSizeL srcStep, Ipp16u* pDst, IppSizeL dstPlaneStep, IppSizeL dstStep, IpprVolumeL dstRoiVolume, IpprBorderType borderType, const Ipp16u borderValue[1], const IppFilterBorderSpec* pSpec, Ipp8u* pBuffer);

IppStatus ipprFilterBorder_32f_C1V_L(const Ipp32f* pSrc, IppSizeL srcPlaneStep, IppSizeL srcStep, Ipp32f* pDst, IppSizeL dstPlaneStep, IppSizeL dstStep, IpprVolumeL dstRoiVolume, IpprBorderType borderType, const Ipp32f borderValue[1], const IppFilterBorderSpec* pSpec, Ipp8u* pBuffer);

IppStatus ipprFilterBorder_64f_C1V_L(const Ipp64f* pSrc, IppSizeL srcPlaneStep, IppSizeL srcStep, Ipp64f* pDst, IppSizeL dstPlaneStep, IppSizeL dstStep, IpprVolumeL dstRoiVolume, IpprBorderType borderType, const Ipp64f borderValue[1], const IppFilterBorderSpec* pSpec, Ipp8u* pBuffer);
Threading Layer (TL) functions based on the Platform Aware API

IppStatus ipprFilterBorder_8u_C1V_LT(const Ipp8u* pSrc, IppSizeL srcPlaneStep, IppSizeL srcStep, Ipp8u* pDst, IppSizeL dstPlaneStep, IppSizeL dstStep, IpprVolumeL dstRoiVolume, IpprBorderType borderType, const Ipp8u borderValue[1], const IpprFilterBorderSpec_LT* pSpec, Ipp8u* pBuffer);

IppStatus ipprFilterBorder_16s_C1V_LT(const Ipp16s* pSrc, IppSizeL srcPlaneStep, IppSizeL srcStep, Ipp16s* pDst, IppSizeL dstPlaneStep, IppSizeL dstStep, IpprVolumeL dstRoiVolume, IpprBorderType borderType, const Ipp16s borderValue[1], const IpprFilterBorderSpec_LT* pSpec, Ipp8u* pBuffer);

IppStatus ipprFilterBorder_16u_C1V_LT(const Ipp16u* pSrc, IppSizeL srcPlaneStep, IppSizeL srcStep, Ipp16u* pDst, IppSizeL dstPlaneStep, IppSizeL dstStep, IpprVolumeL dstRoiVolume, IpprBorderType borderType, const Ipp16u borderValue[1], const IpprFilterBorderSpec_LT* pSpec, Ipp8u* pBuffer);

IppStatus ipprFilterBorder_32f_C1V_LT(const Ipp32f* pSrc, IppSizeL srcPlaneStep, IppSizeL srcStep, Ipp32f* pDst, IppSizeL dstPlaneStep, IppSizeL dstStep, IpprVolumeL dstRoiVolume, IpprBorderType borderType, const Ipp32f borderValue[1], const IpprFilterBorderSpec_LT* pSpec, Ipp8u* pBuffer);

IppStatus ipprFilterBorder_64f_C1V_LT(const Ipp64f* pSrc, IppSizeL srcPlaneStep, IppSizeL srcStep, Ipp64f* pDst, IppSizeL dstPlaneStep, IppSizeL dstStep, IpprVolumeL dstRoiVolume, IpprBorderType borderType, const Ipp64f borderValue[1], const IpprFilterBorderSpec_LT* pSpec, Ipp8u* pBuffer);

Threading Layer (TL) functions based on the Classic API

IppStatus ipprFilterBorder_8u_C1V_T(const Ipp8u* pSrc, int srcPlaneStep, int srcStep, Ipp8u* pDst, int dstPlaneStep, int dstStep, IpprVolume dstRoiVolume, IpprBorderType borderType, const Ipp8u borderValue[1], const IpprFilterBorderSpec_T* pSpec, Ipp8u* pBuffer);

IppStatus ipprFilterBorder_16s_C1V_T(const Ipp16s* pSrc, int srcPlaneStep, int srcStep, Ipp16s* pDst, int dstPlaneStep, int dstStep, IpprVolume dstRoiVolume, IpprBorderType borderType, const Ipp16s borderValue[1], const IpprFilterBorderSpec_T* pSpec, Ipp8u* pBuffer);

IppStatus ipprFilterBorder_16u_C1V_T(const Ipp16u* pSrc, int srcPlaneStep, int srcStep, Ipp16u* pDst, int dstPlaneStep, int dstStep, IpprVolume dstRoiVolume, IpprBorderType borderType, const Ipp16u borderValue[1], const IpprFilterBorderSpec_T* pSpec, Ipp8u* pBuffer);

IppStatus ipprFilterBorder_32f_C1V_T(const Ipp32f* pSrc, int srcPlaneStep, int srcStep, Ipp32f* pDst, int dstPlaneStep, int dstStep, IpprVolume dstRoiVolume, IpprBorderType borderType, const Ipp32f borderValue[1], const IpprFilterBorderSpec_T* pSpec, Ipp8u* pBuffer);

IppStatus ipprFilterBorder_64f_C1V_T(const Ipp64f* pSrc, int srcPlaneStep, int srcStep, Ipp64f* pDst, int dstPlaneStep, int dstStep, IpprVolume dstRoiVolume, IpprBorderType borderType, const Ipp64f borderValue[1], const IpprFilterBorderSpec_T* pSpec, Ipp8u* pBuffer);

Include Files

ippi.h
ippi_l.h
ippi_tl.h
Domain Dependencies

Headers: ippcore.h, ippvm.h, ipps.h
Libraries: ippcore.lib, ippvm.lib, ipps.lib

Parameters

pSrc
srcStep
srcPlaneStep
pDst
dstStep
dstPlaneStep
dstRoiVolume
borderType
borderValue
pSpec
pBuffer

Array of pointers to the planes in the source volume.
Distance, in bytes, between the starting points of consecutive lines in each plane of the source volume.
Distance, in bytes, between the starting points of consecutive images in every plane of the source volume.
Array of pointers to the planes in the destination volume.
Distance, in bytes, between the starting points of consecutive lines in each plane of the destination volume.
Distance, in bytes, between the starting points of consecutive images in every plane of the source volume.
Volume of the destination ROI in pixels.
Type of the border. Possible values are:

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ipprBorderInMem</td>
<td>Border is obtained from the source image pixels in memory.</td>
</tr>
<tr>
<td>ipprBorderRepl</td>
<td>Border is replicated from the edge pixels.</td>
</tr>
<tr>
<td>ipprBorderConst</td>
<td>Border is replicated from the edge pixels.</td>
</tr>
</tbody>
</table>

Constant value to assign to pixels of the constant border.
Pointer to the filter specification structure.
Pointer to the work buffer for filtering operations.

Description

Before using this function, you need to initialize the filter specification structure for 3D image processing using the ipprFilterBorderInit function.

This function operates with VOI. This function performs linear filtering on a source image with the volume. Type of the image border is defined by the value of the border parameter.

Return Values

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ippStsNoErr</td>
<td>Indicates no error condition. Any other value indicates an error condition.</td>
</tr>
<tr>
<td>ippStsStepErr</td>
<td>Indicates an error condition if srcPlaneStep, srcStep, dstPlaneStep, or dstStep has a field with negative value.</td>
</tr>
<tr>
<td>ippStsNullPtrErr</td>
<td>Indicates an error condition if the pSrc, pDst, pSpec, or pBuffer pointer is NULL.</td>
</tr>
<tr>
<td>ippStsSizeErr</td>
<td>Indicates an error condition if dstRoiVolume has a field with zero or negative value.</td>
</tr>
</tbody>
</table>
Example
To better understand usage of this function, refer to the following example in the examples archive available for download from https://software.intel.com/en-us/ipp-manual-examples:
FilterBorder3d.c

See Also
FilterBorderInit Initializes the filter specification structure for 3D image processing.
FilterBorderGetSize Computes the size of the filter specification structure and the size of the work buffer for 3D image processing.

Structures and Enumerators
Structures and Enumerators for Platform-Aware Functions

FilterBorderInit

Initializes the filter specification structure for 3D image processing.

Syntax
IppStatus ipprFilterBorderInit_16s(const Ipp16s* pKernel, IpprVolume kernelVolume, int divisor, IppDataType dataType, int numChannels, IpprFilterBorderSpec* pSpec);
IppStatus ipprFilterBorderInit_32f(const Ipp32f* pKernel, IpprVolume kernelVolume, IppDataType dataType, int numChannels, IpprFilterBorderSpec* pSpec);
IppStatus ipprFilterBorderInit_64f(const Ipp64f* pKernel, IpprVolume kernelVolume, IppDataType dataType, int numChannels, IpprFilterBorderSpec* pSpec);

Platform-aware functions
IppStatus ipprFilterBorderInit_16s_L(const Ipp16s* pKernel, IpprVolumeL kernelVolume, int divisor, IppDataType dataType, int numChannels, IpprFilterBorderSpec* pSpec);
IppStatus ipprFilterBorderInit_32f_L(const Ipp32f* pKernel, IpprVolumeL kernelVolume, IppDataType dataType, int numChannels, IpprFilterBorderSpec* pSpec);
IppStatus ipprFilterBorderInit_64f_L(const Ipp64f* pKernel, IpprVolumeL kernelVolume, IppDataType dataType, int numChannels, IpprFilterBorderSpec* pSpec);

Threading Layer (TL) functions based on the Platform Aware API
IppStatus ipprFilterBorderInit_16s_LT(const Ipp16s* pKernel, IpprVolumeL kernelVolume, int divisor, IppDataType dataType, int numChannels, IpprFilterBorderSpec_LT* pSpec);
IppStatus ipprFilterBorderInit_32f_LT(const Ipp32f* pKernel, IpprVolumeL kernelVolume, IppDataType dataType, int numChannels, IpprFilterBorderSpec_LT* pSpec);
IppStatus ipprFilterBorderInit_64f_LT(const Ipp64f* pKernel, IpprVolumeL kernelVolume, IppDataType dataType, int numChannels, IpprFilterBorderSpec_LT* pSpec);

Threading Layer (TL) functions based on the Classic API
IppStatus ipprFilterBorderInit_16s_T(const Ipp16s* pKernel, IpprVolume kernelVolume, int divisor, IppDataType dataType, int numChannels, IpprFilterBorderSpec_T* pSpec);
IppStatus ipprFilterBorderInit_32f_T(const Ipp32f* pKernel, IpprVolume kernelVolume, IppDataType dataType, int numChannels, IpprFilterBorderSpec_T* pSpec);
IppStatus ipprFilterBorderInit_64f_T(const Ipp64f* pKernel, IpprVolume kernelVolume, IppDataType dataType, int numChannels, IpprFilterBorderSpec_T* pSpec);
Include Files
ippi.h
ippi_l.h
ippi_t1.h

Domain Dependencies
Headers: ippcore.h, ippvm.h, ipps.h
Libraries: ippcore.lib, ippvm.lib, ipps.lib

Parameters
kernelVolume  Size of the kernel volume.
pKernel  Pointers to the kernel values.
dataType  Data type of the source image. Possible values are ipp8u, ipp16u, ipp16s, ipp32f, and ipp64f.
divisor  The integer value by which the computed result is divided.
numChannels  Number of channels in the image. Possible value is 1.
pSpec  Pointer to the filter specification structure.

Description
This function operates with VOI. This function initializes the filter specification structure pSpec. Before using this function, you need to compute the size of the specification structure for 3D image processing using the ipprFilterBorderGetSize function.

Return Values
ippStsNoErr  Indicates no error condition. Any other value indicates an error condition.
ippStsNullPtrErr  Indicates an error condition if the pSpec or pKernel pointer is NULL.
ippStsSizeErr  Indicates an error condition if kernelVolume has a field with zero or negative value.
ippStsChannelErr  Indicates an error condition if numChannels has an illegal value.
ippStsDataTypeErr  Indicates an error condition if dataType has an illegal value.
ippStsDivisorErr  Indicates an error condition if the divisor value is zero.

See Also
FilterBorder Filters a 3D image using a rectangular filter.
FilterBorderGetSize Computes the size of the filter specification structure and the size of the work buffer for 3D image processing.
Structures and Enumerators for Platform-Aware Functions
FilterBorderGetSize

Computes the size of the filter specification structure and the size of the work buffer for 3D image processing.

Syntax

IppStatus ipprFilterBorderGetSize(IpprVolume kernelVolume, IpprVolume dstRoiVolume, IppDataType dataType, IppDataType kernelType, int numChannels, int* pSpecSize, int* pBufferSize);

Platform-aware function

IppStatus ipprFilterBorderGetSize_L(IpprVolumeL kernelVolume, IpprVolumeL dstRoiVolume, IppDataType dataType, IppDataType kernelType, int numChannels, IppSizeL* pSpecSize, IppSizeL* pBufferSize);

Threading Layer (TL) function based on the Platform Aware API

IppStatus ipprFilterBorderGetSize_LT(IpprVolumeL kernelVolume, IpprVolumeL dstRoiVolume, IppDataType dataType, IppDataType kernelType, int numChannels, IppSizeL* pSpecSize, IppSizeL* pBufferSize);

Threading Layer (TL) function based on the Classic API

IppStatus ipprFilterBorderGetSize_T(IpprVolume kernelVolume, IpprVolume dstRoiVolume, IppDataType dataType, IppDataType kernelType, int numChannels, int* pSpecSize, int* pBufferSize);

Include Files

ippi.h
ippi_l.h
ippi_tl.h

Domain Dependencies

Headers: ippcore.h, ippvm.h, ipps.h
Libraries: ippcore.lib, ippvm.lib, ipps.lib

Parameters

kernelVolume  Size of the kernel volume.
dstRoiVolume  Maximal size of the destination image ROI (in pixels).
dataType  Data type of the source image. Possible values are ipp8u, ipp16u, ipp16s, ipp32f, and ipp64f.
kernelType  Data type of the filter kernel. Possible values are ipp16s, ipp32f, and ipp64f.
umChannels  Number of channels in the image. Possible value is 1.
pSpecSize  Pointer to the size of the filter specification structure.
pBufferSize  Pointer to the size of the work buffer required for filtering.
Description
This function operates with VOI. This function computes the size of the filter specification structure \( p\text{Spec} \) and the size of the buffer required for 3D image filtering operations. Call this function before using the `ippiFilterBorderInit` function.

Return Values
- `ippStsNoErr` Indicates no error condition. Any other value indicates an error condition.
- `ippStsNullPtrErr` Indicates an error condition if the `pSpecSize` or `pBufferSize` pointer is `NULL`.
- `ippStsSizeErr` Indicates an error condition if `dstRoiVolume` or `kernelVolume` has a field with zero or negative value.
- `ippStsChannelErr` Indicates an error condition if `numChannels` has an illegal value.
- `ippStsDataTypeErr` Indicates an error condition if the combination of `kernelType` and `dataType` has an illegal value.

See Also
- `FilterBorder` Filters a 3D image using a rectangular filter.
- `FilterBorderInit` Initializes the filter specification structure for 3D image processing.
- Structures and Enumerators for Platform-Aware Functions

FilterMedian

*Filters a 3D image using a median filter.*

Syntax

```c
IppStatus ipprFilterMedian_8u_C1V(const Ipp8u* pSrc, int srcPlaneStep, int srcStep, Ipp8u* pDst, int dstPlaneStep, int dstStep, IpprVolume dstRoiVolume, IpprBorderType borderType, const Ipp8u* pBorderValue, const IpprFilterMedianSpec* pSpec, Ipp8u* pBuffer);

IppStatus ipprFilterMedian_16u_C1V(const Ipp16u* pSrc, int srcPlaneStep, int srcStep, Ipp16u* pDst, int dstPlaneStep, int dstStep, IpprVolume dstRoiVolume, IpprBorderType borderType, const Ipp16u* pBorderValue, const IpprFilterMedianSpec* pSpec, Ipp8u* pBuffer);

IppStatus ipprFilterMedian_16s_C1V(const Ipp16s* pSrc, int srcPlaneStep, int srcStep, Ipp16s* pDst, int dstPlaneStep, int dstStep, IpprVolume dstRoiVolume, IpprBorderType borderType, const Ipp16s* pBorderValue, const IpprFilterMedianSpec* pSpec, Ipp8u* pBuffer);

IppStatus ipprFilterMedian_32f_C1V(const Ipp32f* pSrc, int srcPlaneStep, int srcStep, Ipp32f* pDst, int dstPlaneStep, int dstStep, IpprVolume dstRoiVolume, IpprBorderType borderType, const Ipp32f* pBorderValue, const IpprFilterMedianSpec* pSpec, Ipp8u* pBuffer);

IppStatus ipprFilterMedian_64f_C1V(const Ipp64f* pSrc, int srcPlaneStep, int srcStep, Ipp64f* pDst, int dstPlaneStep, int dstStep, IpprVolume dstRoiVolume, IpprBorderType borderType, const Ipp64f* pBorderValue, const IpprFilterMedianSpec* pSpec, Ipp8u* pBuffer);
```


Threading Layer (TL) functions

IppStatus ipprFilterMedian_8u_C1V_T(const Ipp8u* pSrc, int srcPlaneStep, int srcStep, Ipp8u* pDst, int dstPlaneStep, int dstStep, IpprVolume dstRoiVolume, IpprBorderType borderType, const Ipp8u* pBorderValue, const IpprFilterMedianSpec_T* pSpec, Ipp8u* pBuffer);

IppStatus ipprFilterMedian_16u_C1V_T(const Ipp16u* pSrc, int srcPlaneStep, int srcStep, Ipp16u* pDst, int dstPlaneStep, int dstStep, IpprVolume dstRoiVolume, IpprBorderType borderType, const Ipp16u* pBorderValue, const IpprFilterMedianSpec_T* pSpec, Ipp8u* pBuffer);

IppStatus ipprFilterMedian_16s_C1V_T(const Ipp16s* pSrc, int srcPlaneStep, int srcStep, Ipp16s* pDst, int dstPlaneStep, int dstStep, IpprVolume dstRoiVolume, IpprBorderType borderType, const Ipp16s* pBorderValue, const IpprFilterMedianSpec_T* pSpec, Ipp8u* pBuffer);

IppStatus ipprFilterMedian_32f_C1V_T(const Ipp32f* pSrc, int srcPlaneStep, int srcStep, Ipp32f* pDst, int dstPlaneStep, int dstStep, IpprVolume dstRoiVolume, IpprBorderType borderType, const Ipp32f* pBorderValue, const IpprFilterMedianSpec_T* pSpec, Ipp8u* pBuffer);

IppStatus ipprFilterMedian_64f_C1V_T(const Ipp64f* pSrc, int srcPlaneStep, int srcStep, Ipp64f* pDst, int dstPlaneStep, int dstStep, IpprVolume dstRoiVolume, IpprBorderType borderType, const Ipp64f* pBorderValue, const IpprFilterMedianSpec_T* pSpec, Ipp8u* pBuffer);

Include Files

ippi.h
ippi_t1l.h

Domain Dependencies

Headers: ippcore.h, ippvm.h, ipps.h
Libraries: ippcore.lib, ippvm.lib, ipps.lib

Parameters

pSrc
Array of pointers to the planes in the source volume.

srcStep
Distance, in bytes, between the starting points of consecutive lines in each plane of the source volume.

csrcPlaneStep
Distance, in bytes, between the starting points of consecutive images in each plane of the source volume.

pDst
Array of pointers to the planes in the destination volume.

dstStep
Distance, in bytes, between the starting points of consecutive lines in each plane of the destination volume.

dstPlaneStep
Distance, in bytes, between the starting points of consecutive images in each plane of the destination volume.

dstRoiVolume
Volume of the destination ROI in pixels.

borderType
Type of the border. Possible values are:
ipprBorderInMem  Border is obtained from the source image pixels in memory.

ipprBorderRepl  Border is replicated from the edge pixels.

ipprBorderConst  Border is replicated from the edge pixels.

pBorderValue  Pointer to the constant value to assign to pixels of the constant border.

pSpec  Pointer to the filter specification structure.

pBuffer  Pointer to the work buffer for filtering operations.

**Description**

Before using this function, you need to initialize the filter specification structure for 3D image processing using the `ipprFilterMedianInit` function.

This function operates with VOI. This function performs median filtering on a source image with the volume. Type of the image border is defined by the value of the border parameter.

**Return Values**

- **ippStsNoErr**  Indicates no error condition. Any other value indicates an error condition.
- **ippStsStepErr**  Indicates an error condition if `srcPlaneStep`, `srcStep`, `dstPlaneStep`, or `dstStep` has a field with negative value.
- **ippStsNullPtrErr**  Indicates an error condition if the `pSrc`, `pDst`, `pSpec`, `pBorderValue`, or `pBuffer` pointer is NULL.
- **ippStsSizeErr**  Indicates an error condition if `dstRoiVolume` has a field with zero or negative value.

**See Also**

- **FilterMedianInit**  Initializes the filter specification structure for 3D image processing with a median filter.
- **FilterMedianGetSize**  Computes the size of the filter specification structure and the size of the work buffer for 3D image processing with a median filter.

**FilterMedianInit**

*Initializes the filter specification structure for 3D image processing with a median filter.*

**Syntax**

```c
IppStatus ipprFilterMedianInit(IpprVolume maskVolume, IppDataType dataType, int numChannels, IpprFilterMedianSpec* pSpec);
```

**Threading Layer (TL) function**

```c
IppStatus ipprFilterMedianInit_T(IpprVolume maskVolume, IppDataType dataType, int numChannels, IpprFilterMedianSpec_T* pSpec);
```

**Include Files**

- ippi.h
- ippi_tl.h
Domain Dependencies
Headers: ippcore.h, ippvm.h, ipps.h
Libraries: ippcore.lib, ippvm.lib, ipps.lib

Parameters

maskVolume  
Size of the mask volume.

dataType  
Data type of the source image. Possible values are ipp8u, ipp16u, ipp16s, ipp32f, and ipp64f.

numChannels  
Number of channels in the image. Possible value is 1.

pSpec  
Pointer to the filter specification structure.

Description
This function operates with VOI. This function initializes the filter specification structure pSpec for 3D image processing with a median filter. Before using this function, you need to compute the size of the corresponding specification structure using the ipprFilterMedianGetSize function.

Return Values

ippStsNoErr  
Indicates no error condition. Any other value indicates an error condition.

ippStsNullPtrErr  
Indicates an error condition if the pSpec pointer is NULL.

ippStsSizeErr  
Indicates an error condition if maskVolume has a field with zero or negative value.

ippStsChannelErr  
Indicates an error condition if numChannels has an illegal value.

ippStsDataTypeErr  
Indicates an error condition if dataType has an illegal value.

See Also
FilterMedianGetSize Computes the size of the filter specification structure and the size of the work buffer for 3D image processing with a median filter.
FilterMedian Filters a 3D image using a median filter.

FilterMedianGetSize
Computes the size of the filter specification structure and the size of the work buffer for 3D image processing with a median filter.

Syntax

IppStatus ipprFilterMedianGetSize(IpprVolume maskVolume, IpprVolume dstRoiVolume, IppDataType dataType, int numChannels, int* pSpecSize, int* pBufferSize);

Threading Layer (TL) function

IppStatus ipprFilterMedianGetSize_T(IpprVolume maskVolume, IpprVolume dstRoiVolume, IppDataType dataType, int numChannels, int* pSpecSize, int* pBufferSize);

Include Files
ippi.h
ippi_tl.h
Domain Dependencies

Headers: ippcore.h, ippvm.h, ipps.h
Libraries: ippcore.lib, ippvm.lib, ipps.lib

Parameters

maskVolume
Size of the mask volume.

dstRoiVolume
Maximal size of the destination image ROI (in pixels).

dataType
Data type of the source image. Possible values are ipp8u, ipp16u, ipp16s, ipp32f, and ipp64f.

numChannels
Number of channels in the image. Possible value is 1.

pSpecSize
Pointer to the size of the filter specification structure.

pBufferSize
Pointer to the size of the work buffer required for filtering.

Description

This function operates with VOI. This function computes the size of the filter specification structure pSpec
and the size of the buffer required for 3D image filtering operations with a median filter. Call this function
before using the ipprFilterMedianInit function.

Return Values

ippStsNoErr
Indicates no error condition. Any other value indicates an error condition.

ippStsNullPtrErr
Indicates an error condition if the pSpecSize or pBufferSize pointer is NULL.

ippStsSizeErr
Indicates an error condition if dstRoiVolume or maskVolume has a field with zero or negative value.

ippStsChannelErr
Indicates an error condition if numChannels has an illegal value.

ippStsDataTypeErr
Indicates an error condition if dataType has an illegal value.

See Also

FilterMedianInit Initializes the filter specification structure for 3D image processing with a median filter.

FilterMedian Filters a 3D image using a median filter.

ResizeGetBufSize

Calculates the size of the external work buffer for the function ipprResize.

Syntax

IppStatus ipprResizeGetBufSize(IpprCuboid srcVoi, IpprCuboid dstVoi, int nChannel, int interpolation, int* pSize);

Include Files

ippi.h
Domain Dependencies

Headers: ippcore.h, ippvm.h, ipps.h
Libraries: ippcore.lib, ippvm.lib, ipps.lib

Parameters

srcVoi Volume of interest in the source volume.
dstVoi Volume of interest in the destination volume.
nChannel Number of channels, possible value: 1.
interpolation Type of interpolation, the following values are possible:

- IPPI_INTER_NN - nearest neighbor interpolation,
- IPPI_INTER_LINEAR - trilinear interpolation,
- IPPI_INTER_CUBIC - tricubic interpolation,
- IPPI_INTER_CUBIC2P_BSPLINE - B-spline,
- IPPI_INTER_CUBIC2P_CATMULLROM - Catmull-Rom spline,
- IPPI_INTER_CUBIC2P_B05C03 - special two-parameters filter (1/2, 3/10).
pSize Pointer to the size of the external buffer.

Description

This function calculates the size of the external buffer required for the ipprResize function.

Return Values

- ippStsNoErr Indicates no error. Any other value indicates an error.
- ippStsNullPtrErr Indicates an error when the pSize pointer is NULL.
- ippStsSizeErr Indicates an error when width, or height, or depth of the srcVoi or dstVoi is less than, or equal to 0.
- ippStsNumChannelErr Indicates an error when nChannel is not equal to 1.
- ippStsInterpolationErr Indicates an error when interpolation has an illegal value.

See Also

Resize Resizes the source volume.

GetResizeCuboid

*Computes coordinates of the destination cuboid.*

Syntax

IppStatus ipprGetResizeCuboid(IpprCuboid srcVoi, IpprCuboid* pDstCuboid, double xFactor, double yFactor, double zFactor, double xShift, double yShift, double zShift, int interpolation);

Include Files

ippi.h
Domain Dependencies

Headers: ippcore.h, ippvm.h, ipps.h
Libraries: ippcore.lib, ippvm.lib, ipps.lib

Parameters

srcVoi
Volume of interest of the source volume.

pDstCuboid
Pointer to the destination cuboid.

x-, y-, zFactor
Factors by which the x, y, z dimensions of the source VOI are changed.

x-, y-, zShift
Shift values in the x, y, and z directions respectively.

interpolation
Type of interpolation, the following values are possible:
   IPPI_INTER_NN - nearest neighbor interpolation,
   IPPI_INTER_LINEAR - trilinear interpolation,
   IPPI_INTER_CUBIC - tricubic interpolation,
   IPPI_INTER_CUBIC2P_BSPLINE - B-spline,
   IPPI_INTER_CUBIC2P_CATMULLROM - Catmull-Rom spline,
   IPPI_INTER_CUBIC2P_B05C03 - special two-parameters filter (1/2, 3/10).

Description

This function operates with volume of interest (VOI).
This function computes the coordinates of the resultant cuboid which is obtained if the source volume srcVoi is resized with the specified parameters. The resize operation is not performed.

Return Values

ippStsNoErr
Indicates no error. Any other value indicates an error or a warning.

ippStsNullPtrErr
Indicates an error when pDstCuboid is NULL.

ippStsSizeErr
Indicates an error when width, or height, or depth of the source and destination volumes is less than, or equal to 0.

ippStsResizeFactorErr
Indicates an error when one of the xFactor, yFactor, zFactor values is less than, or equal to 0.

ippStsInterpolationErr
Indicates an error when interpolation has an illegal value.

Resize

Resizes the source volume.

Syntax

IppStatus ipprResize_8u_C1V(const Ipp8u* pSrc, IppVolume srcVolume, int srcStep, int srcPlaneStep, IpprCuboid srcVoi, Ipp8u* pDst, int dstStep, int dstPlaneStep, IpprCuboid dstVoi, double xFactor, double yFactor, double zFactor, double xShift, double yShift, double zShift, int interpolation, Ipp8u* pBuffer);
IppStatus ipprResize_16u_C1V(const Ipp16u* pSrc, IpprVolume srcVolume, int srcStep, int srcPlaneStep, IpprCuboid srcVoi, Ipp16u* pDst, int dstStep, int dstPlaneStep, IpprCuboid dstVoi, double xFactor, double yFactor, double zFactor, double xShift, double yShift, double zShift, int interpolation, Ipp8u* pBuffer);

IppStatus ipprResize_8u_C1PV(const Ipp8u* const pSrc[], IpprVolume srcVolume, int srcStep, IpprCuboid srcVoi, Ipp8u* const pDst[], int dstStep, IpprCuboid dstVoi, double xFactor, double yFactor, double zFactor, double xShift, double yShift, double zShift, int interpolation, Ipp8u* pBuffer);

IppStatus ipprResize_16u_C1PV(const Ipp16u* const pSrc[], IpprVolume srcVolume, int srcStep, IpprCuboid srcVoi, Ipp16u* const pDst[], int dstStep, IpprCuboid dstVoi, double xFactor, double yFactor, double zFactor, double xShift, double yShift, double zShift, int interpolation, Ipp8u* pBuffer);

IppStatus ipprResize_32f_C1PV(const Ipp32f* const pSrc[], IpprVolume srcVolume, int srcStep, IpprCuboid srcVoi, Ipp32f* const pDst[], int dstStep, IpprCuboid dstVoi, double xFactor, double yFactor, double zFactor, double xShift, double yShift, double zShift, int interpolation, Ipp8u* pBuffer);

IppStatus ipprResize_32f_C1V(const Ipp32f* pSrc, IpprVolume srcVolume, int srcStep, int srcPlaneStep, IpprCuboid srcVoi, Ipp32f* pDst, int dstStep, int dstPlaneStep, IpprCuboid dstVoi, double xFactor, double yFactor, double zFactor, double xShift, double yShift, double zShift, int interpolation, Ipp8u* pBuffer);

Include Files
ippi.h

Domain Dependencies

Headers: ippcore.h, ippvm.h, ipp.h
Libraries: ippcore.lib, ippvm.lib, ipp.lib

Parameters

pSrc  
Pointer to the source volume origin. An array of pointers to the source planes for non-contiguous volume.

srcVolume  
Size of the source volume.

srcStep  
Distance, in bytes, between the starting points of consecutive lines in each plane of the source volume.

srcPlaneStep  
Distance, in bytes, between the planes of the source contiguous volume.

srcVoi  
Volume of interest of the source volume.

pDst  
Pointer to the destination volume origin. An array of pointers to the destination planes for non-contiguous volume.

dstStep  
Distance, in bytes, between the starting points of consecutive lines in each plane of the destination volume.

dstPlaneStep  
Distance, in bytes, between the planes of the destination contiguous volume.

dstVoi  
Volume of interest of the destination volume.

x-, y-, zFactor  
Factors by which the x, y, z dimensions of the source VOI are changed.
Shift values in the $x$, $y$, and $z$ directions respectively.

Type of interpolation, the following values are possible:

- IPPI_INTER_NN: nearest neighbor interpolation,
- IPPI_INTER_LINEAR: trilinear interpolation,
- IPPI_INTER_CUBIC: tricubic interpolation,
- IPPI_INTER_CUBIC2P_BSPLINE: B-spline,
- IPPI_INTER_CUBIC2P_CATMULLROM: Catmull-Rom spline,
- IPPI_INTER_CUBIC2P_B05C03: special two-parameters filter (1/2, 3/10).

Pointer to the external buffer.

**Description**

This function operates with volume of interest (VOI).

This function resizes the source volume $srcVoi$ by $xFactor$ in the $x$ direction, $yFactor$ in the $y$ direction and $zFactor$ in the $z$ direction. The volume size can be reduced or increased in each direction, depending on the values of $xFactor$, $yFactor$, $zFactor$. If the value of the certain factor is greater than 1, the volume size is increased, and if it is less than 1, the volume size is reduced in the corresponding direction. The result is resampled using the interpolation method specified by the `interpolation` parameter, and written to the destination volume $VOI$.

Coordinates $x'$, $y'$, and $z'$ in the resized volume are obtained from the equations:

\[
\begin{align*}
    x' &= xFactor \times x + xShift \\
    y' &= yFactor \times x + yShift \\
    z' &= zFactor \times x + zShift
\end{align*}
\]

where $x$, $y$, and $z$ denote the coordinates of the element in the source volume. The right coordinate system (RCS) is used here.

Before using this function, compute the size of the external buffer $pBuffer$ using `ipprResizeGetBufSize`.

**Return Values**

- ippStsNoErr: Indicates no error. Any other value indicates an error or a warning.
- ippStsNullPtrErr: Indicates an error when one of the specified pointers is NULL.
- ippStsSizeErr: Indicates an error when width, or height, or depth of the source and destination volumes is less than or equal to 0.
- ippStsResizeFactorErr: Indicates an error when one of the $xFactor$, $yFactor$, $zFactor$ pointers is less than, or equal to 0.
- ippStsInterpolationErr: Indicates an error when `interpolation` has an illegal value.
- ippStsWrongIntersectVOI: Indicates a warning when $srcVoi$ has no intersection with the source volume, operation is not performed.

**See Also**

- **ResizeGetBufSize**: Calculates the size of the external work buffer for the function `ipprResize`.
Remap

Performs the look-up coordinate mapping of the elements of the source volume.

Syntax

Case 1: Operation on non-contiguous volume data

IppStatus ipprRemap_8u_C1PV(const Ipp8u* pSrc[], IprrVolume srcVolume, int srcStep, IpprCuboid srcVoi, const Ipp32f* pxMap[], const Ipp32f* pyMap[], const Ipp32f* pzMap[], int mapStep, Ipp8u* pDst[], int dstStep, IprrVolume dstVolume, int interpolation);

IppStatus ipprRemap_16u_C1PV(const Ipp16u* pSrc[], IprrVolume srcVolume, int srcStep, IpprCuboid srcVoi, const Ipp32f* pxMap[], const Ipp32f* pyMap[], const Ipp32f* pzMap[], int mapStep, Ipp16u* pDst[], int dstStep, IprrVolume dstVolume, int interpolation);

IppStatus ipprRemap_32f_C1PV(const Ipp32f* pSrc[], IprrVolume srcVolume, int srcStep, IpprCuboid srcVoi, const Ipp32f* pxMap[], const Ipp32f* pyMap[], const Ipp32f* pzMap[], int mapStep, Ipp32f* pDst[], int dstStep, IprrVolume dstVolume, int interpolation);

Case 2: Operation on contiguous volume data

IppStatus ipprRemap_8u_C1V(const Ipp8u* pSrc, IprrVolume srcVolume, int srcStep, int srcPlaneStep, IpprCuboid srcVoi, const Ipp32f* pxMap, const Ipp32f* pyMap, const Ipp32f* pzMap, int mapStep, int mapPlaneStep, Ipp8u* pDst, int dstStep, int dstPlaneStep, IprrVolume dstVolume, int interpolation);

IppStatus ipprRemap_16u_C1V(const Ipp16u* pSrc, IprrVolume srcVolume, int srcStep, int srcPlaneStep, IpprCuboid srcVoi, const Ipp32f* pxMap, const Ipp32f* pyMap, const Ipp32f* pzMap, int mapStep, int mapPlaneStep, Ipp16u* pDst, int dstStep, int dstPlaneStep, IprrVolume dstVolume, int interpolation);

IppStatus ipprRemap_32f_C1V(const Ipp32f* pSrc, IprrVolume srcVolume, int srcStep, int srcPlaneStep, IpprCuboid srcVoi, const Ipp32f* pxMap, const Ipp32f* pyMap, const Ipp32f* pzMap, int mapStep, int mapPlaneStep, Ipp32f* pDst, int dstStep, int dstPlaneStep, IprrVolume dstVolume, int interpolation);

Include Files

ippi.h

Domain Dependencies

Headers: ippcore.h, ippvm.h, ipps.h
Libraries: ippcore.lib, ippvm.lib, ipps.lib

Parameters

pSrc

Array of pointers to the planes in the source volume.

srcVolume

Size of the source volume.

srcStep

Distance, in bytes, between the starting points of consecutive lines in every plane of the source volume.

srcPlaneStep

Distance, in bytes, between the starting points of consecutive lines in every plane of the source volume (for 8u_C1V, 16u_C1V, and 32f_C1V flavors).

srcVoi

Region of interest in the source volume.
Arrays of pointers to the starts of the 2D buffers, containing tables of the x-, y- and z-coordinates. If the referenced coordinates correspond to a voxel outside of the source VOI, no mapping of the source pixel is done.

Step, in bytes, through the buffers containing tables of the x-, y- and z-coordinates.

Distance, in bytes, between the starting points of consecutive lines in every plane of the buffers containing tables (for 8u_C1V, 16u_C1V, and 32f_C1V flavors).

Array of the pointers to the planes in the destination volume.

Distance, in bytes, between the starting points of consecutive lines in every plane of the destination volume.

Distance, in bytes, between the starting points of consecutive lines in every plane of the destination volume (for 8u_C1V, 16u_C1V, and 32f_C1V flavors).

Size of the destination volume.

The type of interpolation, the following values are possible:

- IPPI_INTER_NN - nearest neighbor interpolation,
- IPPI_INTER_LINEAR - trilinear interpolation,
- IPPI_INTER_CUBIC - tricubic interpolation,
- IPPI_INTER_CUBIC2P_BSPLINE - B-spline,
- IPPI_INTER_CUBIC2P_CATMULLROM - Catmull-Rom spline,
- IPPI_INTER_CUBIC2P_B05C03 - special two-parameters filter (1/2, 3/10).

**Description**

This function operates with volume of interest (VOI).

This function transforms the source volume by remapping its voxels. Voxel remapping is performed using pxMap, pyMap and pzMap buffers to look-up the coordinates of the source volume voxel that is written to the target destination volume voxel. The application has to supply these look-up tables.

The remapping of the source voxels to the destination voxels is made according to the following formulas:

\[
dst\_voxel[i, j, k] = src\_voxel[ pxMap[i, j, k], pyMap[i, j, k], pzMap[i, j, k] ]
\]

where \( i, j, k \) are the x-, y- and z-coordinates of the target destination volume voxel \( dst\_voxel \);

\( pxMap[i, j, k] \) contains the x-coordinates of the source volume voxels \( src\_voxel \) that are written to \( dst\_voxel \).

\( pyMap[i, j, k] \) contains the y-coordinates of the source volume voxels \( src\_voxel \) that are written to \( dst\_voxel \).

\( pzMap[i, j, k] \) contains the z-coordinates of the source volume voxels \( src\_voxel \) that are written to \( dst\_voxel \).

**Return Values**

- ippStsNoErr: Indicates no error. Any other value indicates an error or a warning.

- ippStsNullPtrErr: Indicates an error when one of the specified pointers is NULL.
WarpAffineGetBufSize

Calculates the size of the external buffer for the affine transform.

Syntax

IppStatus ipprWarpAffineGetBufSize(IpprVolume srcVolume, IpprCuboid srcVoi, IpprCuboid dstVoi, const double coeffs[3][4], int nChannel, int interpolation, int* pSize);

Include Files

ippi.h

Domain Dependencies

Headers: ippcore.h, ippvm.h, ipps.h
Libraries: ippcore.lib, ippvm.lib, ipps.lib

Parameters

srcVolume  Size of the source volume.
srcVoi  Region of interest of the source volume.
dstVoi  Region of interest of the destination volume.
coeffs  Affine transform matrix.
nChannel  Number of channel or planes, possible value is 1.
interpolation  Type of interpolation, the following values are possible:

- IPPI_INTER_NN - nearest neighbor interpolation,
- IPPI_INTER_LINEAR - trilinear interpolation,
- IPPI_INTER_CUBIC - tricubic interpolation,
- IPPI_INTER_CUBIC2P_BSPLINE - B-spline,
- IPPI_INTER_CUBIC2P_CATMULLROM - Catmull-Rom spline,
- IPPI_INTER_CUBIC2P_B05C03 - special two-parameters filter (1/2, 3/10).

pSize  Pointer to the size of the external buffer.

Description

This function calculates the size (in bytes) of the external buffer that is required for the ipprWarpAffine function. (In some cases the function returns zero size of the buffer).
Return Values

ippStsNoErr
Indicates no error. Any other value indicates an error or a warning.

ippStsNullPtrErr
Indicates an error when pSize or coeffs is NULL.

ippStsSizeErr
Indicates an error if width, or height, or depth of the srcVoi or dstVoi is less than, or equal to zero.

ippStsNumChannelErr
Indicates an error when nChannel has an illegal value.

ippStsInterpolationErr
Indicates an error when interpolation has an illegal value.

See Also
WarpAffine Performs the general affine transform of the source volume.

WarpAffine
Performs the general affine transform of the source volume.

Syntax
IppStatus ipprWarpAffine_8u_C1PV(const Ipp8u* const pSrc[], IpprVolume srcVolume, int srcStep, IpprCuboid srcVoi, Ipp8u* const pDst[], int dstStep, IpprCuboid dstVoi, const double coeffs[3][4], int interpolation, Ipp8u* pBuffer);
IppStatus ipprWarpAffine_16u_C1PV(const Ipp16u* const pSrc[], IpprVolume srcVolume, int srcStep, IpprCuboid srcVoi, Ipp16u* const pDst[], int dstStep, IpprCuboid dstVoi, const double coeffs[3][4], int interpolation, Ipp8u* pBuffer);
IppStatus ipprWarpAffine_32f_C1PV(const Ipp32f* const pSrc[], IpprVolume srcVolume, int srcStep, IpprCuboid srcVoi, Ipp32f* const pDst[], int dstStep, IpprCuboid dstVoi, const double coeffs[3][4], int interpolation, Ipp8u* pBuffer);
IppStatus ipprWarpAffine_8u_C1V(const Ipp8u* pSrc, IpprVolume srcVolume, int srcStep, int srcPlaneStep, IpprCuboid srcVoi, Ipp8u* pDst, int dstStep, int dstPlaneStep, IpprCuboid dstVoi, const double coeffs[3][4], int interpolation, Ipp8u* pBuffer);
IppStatus ipprWarpAffine_16u_C1V(const Ipp16u* pSrc, IpprVolume srcVolume, int srcStep, int srcPlaneStep, IpprCuboid srcVoi, Ipp16u* pDst, int dstStep, int dstPlaneStep, IpprCuboid dstVoi, const double coeffs[3][4], int interpolation, Ipp8u* pBuffer);
IppStatus ipprWarpAffine_32f_C1V(const Ipp32f* pSrc, IpprVolume srcVolume, int srcStep, int srcPlaneStep, IpprCuboid srcVoi, Ipp32f* pDst, int dstStep, int dstPlaneStep, IpprCuboid dstVoi, const double coeffs[3][4], int interpolation, Ipp8u* pBuffer);

Include Files
ippi.h

Domain Dependencies
Headers: ippcore.h, ippvm.h, ipps.h
Libraries: ippcore.lib, ippvm.lib, ipps.lib
Parameters

- **pSrc**: Array of pointers to the planes in the source volume.
- **srcVolume**: Size, in pixels, of the source volume.
- **srcStep**: Distance, in bytes, between the starting points of consecutive lines in each plane of the source volume.
- **srcPlaneStep**: Distance, in bytes, between the starting points of consecutive lines in every plane of the source volume (for `8u_C1V`, `16u_C1V`, and `32f_C1V` flavors).
- **dstStep**: Distance, in bytes, between the starting points of consecutive lines in each plane of the destination volume.
- **srcVoi**: Volume of interest of the source volume.
- **pDst**: Array of pointers to the planes in the destination volume.
- **dstVoi**: Volume of interest of the destination volume.
- **dstStep**: Distance, in bytes, between the starting points of consecutive lines in every plane of the destination volume.
- **dstPlaneStep**: Distance, in bytes, between the starting points of consecutive lines in every plane of the destination volume (for `8u_C1V`, `16u_C1V`, and `32f_C1V` flavors).
- **coeffs**: Coefficients of the affine transform.
- **interpolation**: Type of interpolation, the following values are possible:
  - `IPPI_INTER_NN` - nearest neighbor interpolation,
  - `IPPI_INTER_LINEAR` - trilinear interpolation,
  - `IPPI_INTER_CUBIC` - tricubic interpolation,
  - `IPPI_INTER_CUBIC2P_BSPLINE` - B-spline,
  - `IPPI_INTER_CUBIC2P_CATMULLROM` - Catmull-Rom spline,
  - `IPPI_INTER_CUBIC2P_B05C03` - special two-parameters filter (1/2, 3/10).
- **pBuffer**: Pointer to the external buffer.

Description

This function operates with volume of interest (VOI).

This affine warp function transforms the coordinates \((x, y, z)\) of the source volume voxels according to the following formulas:

- \(x' = c_{00}x + c_{01}y + c_{02}z + c_{03}\)
- \(y' = c_{10}x + c_{11}y + c_{12}z + c_{13}\)
- \(z' = c_{20}x + c_{21}y + c_{22}z + c_{23}\)

where \(x', y', z'\) denote the voxel coordinates in the transformed volume, and \(c_{ij}\) are the affine transform coefficients stored in the array **coeffs**.

Before calling this function, compute the size of the external buffer **pBuffer** using the `ipprWarpAffineGetBufSize` function.
Return Values

- ippStsNoErr: Indicates no error. Any other value indicates an error or a warning.
- ippStsNullPtrErr: Indicates an error when one of the specified pointers is NULL.
- ippStsSizeErr: Indicates an error when width, or height, or depth of the source and destination volumes is less than, or equal to zero.
- ippStsCoeffErr: Indicates an error when determinant of the transform matrix $c_{ij}$ is equal to zero.
- ippStsInterpolationErr: Indicates an error when interpolation has an illegal value.
- ippStsWrongIntersectVOI: Indicates a warning when srcVoi has no intersection with the source volume, operation is not performed.

See Also
WarpAffineGetBufSize Calculates the size of the external buffer for the affine transform.
Handling of Special Cases

Some mathematical functions implemented in Intel IPP are not defined for all possible argument values. This appendix describes how the corresponding Intel IPP image processing functions handle situations when their input arguments fall outside the range of function definition or may lead to ambiguously determined output results.

Table A-1 below summarizes these special cases for different functions and lists result values together with status codes returned by the functions. The status codes ending with \textit{Err} (except for the \textit{ippStsNoErr} status) indicate an error. When an error occurs, the function execution is interrupted. All other status codes indicate that the input argument is outside the range, but the function execution is continued with the corresponding result value.

**Special Cases for Intel IPP Image Processing Functions**

<table>
<thead>
<tr>
<th>Function Base Name</th>
<th>Data Type</th>
<th>Case Description</th>
<th>Result Value</th>
<th>Status Code</th>
</tr>
</thead>
<tbody>
<tr>
<td>\textit{ippiSqrt}</td>
<td>16s</td>
<td>Sqrt (x &lt;0)</td>
<td>0</td>
<td>\textit{ippStsSqrtNegArg}</td>
</tr>
<tr>
<td></td>
<td>32f</td>
<td>Sqrt (x &lt;0)</td>
<td>\textit{NAN_32F}</td>
<td>\textit{ippStsSqrtNegArg}</td>
</tr>
<tr>
<td>\textit{ippiDiv}</td>
<td>8u</td>
<td>Div (0/0)</td>
<td>0</td>
<td>\textit{ippStsDivByZero}</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Div (x/0)</td>
<td>\textit{IPP_MAX_8U}</td>
<td>\textit{ippStsDivByZero}</td>
</tr>
<tr>
<td></td>
<td>16s</td>
<td>Div (0/0)</td>
<td>0</td>
<td>\textit{ippStsDivByZero}</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Div (x/0), x&gt;0</td>
<td>\textit{IPP_MAX_16S}</td>
<td>\textit{ippStsDivByZero}</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Div (x/0), x&lt;0</td>
<td>\textit{IPP_MIN_16S}</td>
<td>\textit{ippStsDivByZero}</td>
</tr>
<tr>
<td></td>
<td>32f</td>
<td>Div (0/0)</td>
<td>\textit{NAN_32F}</td>
<td>\textit{ippStsDivByZero}</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Div (x/0), x&gt;0</td>
<td>\textit{INF_32F}</td>
<td>\textit{ippStsDivByZero}</td>
</tr>
<tr>
<td></td>
<td>16sc</td>
<td>Div (0/0)</td>
<td>0</td>
<td>\textit{ippStsDivByZero}</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Div (x/0), x&gt;0</td>
<td>\textit{IPP_MAX_16S}</td>
<td>\textit{ippStsDivByZero}</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Div (x/0), x&lt;0</td>
<td>\textit{IPP_MIN_16S}</td>
<td>\textit{ippStsDivByZero}</td>
</tr>
<tr>
<td></td>
<td>32sc</td>
<td>Div (0/0)</td>
<td>0</td>
<td>\textit{ippStsDivByZero}</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Div (x/0), x&gt;0</td>
<td>\textit{IPP_MAX_32S}</td>
<td>\textit{ippStsDivByZero}</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Div (x/0), x&lt;0</td>
<td>\textit{IPP_MIN_32S}</td>
<td>\textit{ippStsDivByZero}</td>
</tr>
<tr>
<td></td>
<td>32fc</td>
<td>Div (0/0)</td>
<td>\textit{NAN_32F}</td>
<td>\textit{ippStsDivByZero}</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Div (x/0), x&gt;0</td>
<td>\textit{INF_32F}</td>
<td>\textit{ippStsDivByZero}</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Div (x/0), x&lt;0</td>
<td>\textit{INF_NEG_32F}</td>
<td></td>
</tr>
<tr>
<td>\textit{ippiDivC}</td>
<td>all</td>
<td>Div(x/const), const=0</td>
<td>-</td>
<td>\textit{ippStsDivByZeroErr}</td>
</tr>
<tr>
<td>Function Base Name</td>
<td>Data Type</td>
<td>Case Description</td>
<td>Result Value</td>
<td>Status Code</td>
</tr>
<tr>
<td>--------------------</td>
<td>-----------</td>
<td>------------------</td>
<td>--------------</td>
<td>-------------</td>
</tr>
<tr>
<td>ippiLn</td>
<td>8u</td>
<td>Ln (0)</td>
<td>0</td>
<td>ippStsLnZeroArg</td>
</tr>
<tr>
<td></td>
<td>16s</td>
<td>Ln (0)</td>
<td>IPP_MIN_16S</td>
<td>ippStsLnZeroArg</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Ln (x&lt;0)</td>
<td>IPP_MIN_16S</td>
<td>ippStsLnNegArg</td>
</tr>
<tr>
<td></td>
<td>32f</td>
<td>Ln (x&lt;0)</td>
<td>NAN_32F</td>
<td>ippStsLnNegArg</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Ln(x&lt;IPP_MINABS_32F)</td>
<td>INF_NEG_32F</td>
<td>ippStsLnZeroArg</td>
</tr>
</tbody>
</table>

| ippiExp            | 8u        | overflow         |IPP_MAX_8U    | ippStsNoErr  |
|                    | 16s       | overflow         |IPP_MAX_16S   | ippStsNoErr  |
|                    | 32f       | overflow         |INF_32F       | ippStsNoErr  |

Here $x$ denotes an input value. For the definition of the constants used, see Image Data Types and Ranges in chapter 2.

Note that flavors of the same math function operating on different data types may produce different results for the equal argument values. However, for a given function and a fixed data type, handling of special cases is the same for all function flavors that have different descriptors in their names. For example, logarithm function ippiLn operating on 16s data treats zero argument values in the same way for all its flavors ippiLn_16s_C1RSfs, ippiLn_16s_C3RSfs, ippiLn_16s_C1IRSfs, and ippiLn_16s_C3IRSfs.
Interpolation in Image Geometric Transform Functions

This appendix describes the interpolation algorithms used in the geometric transformation functions of Intel IPP. For more information about each of the geometric transform functions, see Chapter 12.

Overview of Interpolation Modes

In geometric transformations, the grid of input image pixels is not necessarily mapped onto the grid of pixels in the output image. Therefore, to compute the pixel intensities in the output image, the geometric transform functions need to interpolate the intensity values of several input pixels that are mapped to a certain neighborhood of the output pixel.

Geometric transformations can use various interpolation algorithms. The library supports the following interpolation modes depending on how the type of interpolation algorithm is specified:

- **Type 1:** Application code specifies the interpolation mode by passing the \textit{interpolation} parameter of \texttt{int} type to a processing function. The following interpolation modes are supported:
  - Nearest neighbor interpolation (\texttt{interpolation = IPP\_INTER\_NN})
  - Linear interpolation (\texttt{interpolation = IPP\_INTER\_LINEAR})
  - Cubic interpolation (\texttt{interpolation = IPP\_INTER\_CUBIC})
  - Interpolation with Lanczos window function (\texttt{interpolation = IPP\_INTER\_LANCZOS})
  - Interpolation with two-parameter cubic filters with the fixed coefficients (\texttt{interpolation} can be set to the following:
    - \texttt{IPPI\_INTER\_CUBIC2P\_BSPLINE (B=1; C=0)}
    - \texttt{IPPI\_INTER\_CUBIC2P\_CATHULLROM (B=0; C=0.5)}
    - \texttt{IPPI\_INTER\_CUBIC2P\_ B05C03 (B=0.5; C=0.3)}

- **Type 2:** Interpolation mode is specified explicitly in a function name suffix:
  - Nearest neighbor interpolation (pass \texttt{interpolation = ippNearest} to \texttt{GetSize} functions, use the functions with the \texttt{Nearest} suffix, for example, \texttt{ResizeNearestInit} or \texttt{ResizeNearest})
  - Linear interpolation (pass \texttt{interpolation = ippLinear} to \texttt{GetSize} functions, use the functions with the \texttt{Linear} suffix, for example, \texttt{ResizeLinearInit} or \texttt{ResizeLinear})
  - Interpolation with two-parameter cubic filters (pass \texttt{interpolation = ippCubic} to \texttt{GetSize} functions, use the functions with the \texttt{Cubic} suffix, for example, \texttt{ResizeCubicInit} or \texttt{ResizeCubic})
  - Supersampling (pass \texttt{interpolation = ippSuper} to \texttt{GetSize} functions, use the functions with the \texttt{Super} suffix, for example, \texttt{ResizeSuperInit} or \texttt{ResizeSuper})
  - Interpolation with Lanczos window function (pass \texttt{interpolation = ippLanczos} to \texttt{GetSize} functions, use the functions with the \texttt{Lanczos} suffix, for example, \texttt{ResizeLanczosInit} or \texttt{ResizeLanczos})

For certain functions of type 1, the specified interpolation modes can be combined with additional \textit{smoothing of edges} to which the borders of the original image are transformed. To use this option, for the \texttt{interpolation} parameter, specify the edge smoothing flag and the desired interpolation mode using the bitwise OR operation. For example, in order to rotate an image with cubic interpolation and smooth the rotated image edges, pass the following value to \texttt{ippiRotate()}:

\begin{verbatim}
interpolation = IPPI\_INTER\_CUBIC | IPPI\_SMOOTH\_EDGE.
\end{verbatim}

To enable edge smoothing for functions of type 2, pass the special flag to the \texttt{Init} function (if it exists), for example, you can pass this flag to \texttt{Init} functions for \texttt{WarpAffine} and \texttt{WarpPerspective} function groups.

Interpolation with edge smoothing option can be used only in those geometric transform functions where this option is explicitly listed in the parameters definition section.
Table B-1 lists the supported interpolation modes for the main geometric transform functions that use interpolation.

### Interpolation Modes Supported by Image Geometric Transform Functions

<table>
<thead>
<tr>
<th>Function Base Name</th>
<th>NN</th>
<th>Lin</th>
<th>Cub2P</th>
<th>Cub</th>
<th>CR</th>
<th>La2</th>
<th>La3</th>
<th>SS</th>
<th>AA</th>
<th>ES</th>
</tr>
</thead>
<tbody>
<tr>
<td>ResizeCubic</td>
<td></td>
<td>x</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ResizeLanczos</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ResizeLinear</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ResizeNearest</td>
<td>x</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ResizeSuper</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>WarpAffineCubic</td>
<td>x</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>WarpAffineLinear</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>WarpAffineNearest</td>
<td>x</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>WarpPerspectiveCubic</td>
<td>x</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>WarpPerspectiveLinear</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>WarpPerspectiveNearest</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Remap</td>
<td>x</td>
<td>x</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>WarpBilinear</td>
<td>x</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>WarpBilinearBack</td>
<td>x</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>WarpBilinearQuad</td>
<td>x</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

*) The function `ippiResizeCubic` supports the interpolation with two-parameter cubic filters, where parameters B and C can be specified explicitly.

Here **NN** - nearest neighbor interpolation, **Lin** - linear interpolation, **Cub2P** - interpolation with two-parameter cubic filters, **Cub** - cubic interpolation, **CR** - Catmull-Rom spline, **La2, La3** - interpolation with the Lanczos window, **SS** - super sampling interpolation, **AA** - antialiasing, **ES** - edge smoothing.

The sections that follow provide more details on each interpolation mode.

### Mathematical Notation

In this appendix the following notation is used:

- \((x_D, y_D)\) \hspace{1cm} pixel coordinates in the destination image (integer values);
- \((x_S, y_S)\) \hspace{1cm} the computed coordinates of a point in the source image that is mapped exactly to \((x_D, y_D)\);
- \(S(x, y)\) \hspace{1cm} pixel value (intensity) in the source image;
- \(D(x, y)\) \hspace{1cm} pixel value (intensity) in the destination image.

### Nearest Neighbor Interpolation

This is the fastest and least accurate interpolation mode. The pixel value in the destination image is set to the value of the source image pixel closest to the point \((x_S, y_S)\):

\[ D(x_D, y_D) = S(\text{round}(x_S), \text{round}(y_S)) \]

To use the nearest neighbor interpolation, set the interpolation parameter to **IPPI_INTER_NN** or use the functions with the **Nearest** suffix (pass **interpolation=ippNearest** to GetSize functions).
Linear Interpolation

This is the fastest and least accurate interpolation mode. The pixel value in the destination image is set to the value of the source image pixel closest to the point

\((x_S, y_S) : D(x_D, y_D) = S(\text{round}(x_S), \text{round}(y_S))\).

The linear interpolation is slower but more accurate than the nearest neighbor interpolation. On the other hand, it is faster but less accurate than cubic interpolation. The linear interpolation algorithm uses source image intensities at the four pixels \((x_{S0}, y_{S0}), (x_{S1}, y_{S0}), (x_{S0}, y_{S1}), (x_{S1}, y_{S1})\) that are closest to \((x_S, y_S)\) in the source image:

\[ x_{S0} = \text{int}(x_S), \quad x_{S1} = x_{S0} + 1, \quad y_{S0} = \text{int}(y_S), \quad y_{S1} = y_{S0} + 1. \]

First, the intensity values are interpolated along the x-axis to produce two intermediate results \(I_0\) and \(I_1\) (see Figure B-1):

\[ I_0 = S(x_S, y_{S0}) = S(x_{S0}, y_{S0})* (x_{S1} - x_S) + S(x_{S1}, y_{S0})* (x_S - x_{S0}) \]
\[ I_1 = S(x_S, y_{S1}) = S(x_{S0}, y_{S1})* (x_{S1} - x_S) + S(x_{S1}, y_{S1})* (x_S - x_{S0}) \]

Then, the sought-for intensity \(D(x_D, y_D)\) is computed by interpolating the intermediate values \(I_0\) and \(I_1\) along the y-axis:

\[ D(x_D, y_D) = I_0* (y_{S1} - y_S) + I_1* (y_S - y_{S0}). \]

To use the linear interpolation, set the interpolation parameter to IPPI_INTER_LINEAR or use the functions with the Linear suffix (pass interpolation=ippLinear to GetSize functions). For images with 8-bit unsigned color channels, the ippiWarpAffine and ippiResizeLinear functions compute the coordinates \((x_S, y_S)\) with the accuracy \(2^{-16} = 1/65536\). For images with 16-bit unsigned color channels, these functions compute the coordinates with floating-point precision.

Cubic Interpolation

The cubic interpolation algorithm (see Figure B-2) uses source image intensities at sixteen pixels in the neighborhood of the point \((x_S, y_S)\) in the source image:

\[ x_{S0} = \text{int}(x_S) - 1; \quad x_{S1} = x_{S0} + 1; \quad x_{S2} = x_{S0} + 2; \quad x_{S3} = x_{S0} + 3; \]
\[ y_{S0} = \text{int}(y_S) - 1; \ y_{S1} = y_{S0} + 1; \ y_{S2} = y_{S0} + 2; \ y_{S3} = y_{S0} + 3. \]

First, for each \( y_{Sk} \) the algorithm determines four cubic polynomials \( F_0(x), F_1(x), F_2(x), \) and \( F_3(x) \):
\[
F_k(x) = a_kx^3 + b_kx^2 + c_kx + d_k, \quad 0 \leq k \leq 3
\]
such that
\[
F_k(x_{S0}) = S(x_{S0}, y_{Sk}); \ F_k(x_{S1}) = S(x_{S1}, y_{Sk}),
\]
\[
F_k(x_{S2}) = S(x_{S2}, y_{Sk}); \ F_k(x_{S3}) = S(x_{S3}, y_{Sk}).
\]
In Figure B-2, these polynomials are shown by solid curves.

Then, the algorithm determines a cubic polynomial \( F_y(y) \) such that
\[
F_y(y_{S0}) = F_0(x_{S0}), \ F_y(y_{S1}) = F_1(x_{S0}), \ F_y(y_{S2}) = F_2(x_{S0}), \ F_y(y_{S3}) = F_3(x_{S0}).
\]
The polynomial \( F_y(y) \) is represented by the dashed curve in Figure B-2.

Finally, the sought intensity \( D(x_D, y_D) \) is set to the value \( F_y(y_S) \).

To use the cubic interpolation, set the interpolation parameter to IPPI_INTER_CUBIC.

### Cubic Interpolation

![Cubic Interpolation](image)

### Super Sampling

If the destination image is much smaller than the source image, the above interpolation algorithms may skip some pixels in the source image (that is, these algorithms not necessarily use all source pixels when computing intensity of the destination pixels).

The super-sampling algorithm is as follows:

1. Divide the source image rectangular ROI (or the whole image, if there is no ROI) into equal rectangles, each rectangle corresponding to some pixel in the destination image. Note that each source pixel is represented by a 1x1 square.

2. Compute a weighted sum of source pixel values for all pixels that are in the rectangle or have a non-zero intersection with the rectangle. If a source pixel is fully contained in the rectangle, the value of that pixel is taken with weight 1. If the rectangle and the square of the source pixel have an intersection of area \( a < 1 \), that pixel's value is taken with weight \( a \).

   Figure B-3 shows the corresponding weight value for each source pixel intersecting with the rectangle.

3. To compute the pixel value in the destination image, divide this weighted sum by the ratio of the source and destination rectangle areas \( S_{Src}/S_{Dst} = 1/xFactor*yFactor \).

   Here \( xFactor \), and \( yFactor \) are the parameters of the functions that specify the factors by which the \( x \) and \( y \) dimensions of the source image ROI are changed.

Note that supersampling interpolation can be used only for \( xFactor < 1 \), and \( yFactor < 1 \).
To use the supersampling interpolation, use the functions with the `Super` suffix (pass `interpolation=ippSuper` to `GetSize` functions).

**Supersampling Weights**

![](image)

**Lanczos Interpolation**

This method is based on the 2-lobed or 3-lobed Lanczos window function as the interpolation function.

**Interpolation with the 2-lobed Lanczos Window Function**

The interpolation algorithm uses source image intensities at 16 pixels in the neighborhood of the point \((x_S, y_S)\) in the source image:

\[
x_{S0} = \text{int}(x_S) - 1; \quad x_{S1} = x_{S0} + 1; \quad x_{S2} = x_{S0} + 2; \quad x_{S3} = x_{S0} + 3;
\]

\[
y_{S0} = \text{int}(y_S) - 1; \quad y_{S1} = y_{S0} + 1; \quad y_{S2} = y_{S0} + 2; \quad y_{S3} = y_{S0} + 3;
\]

First, the intensity values are interpolated along the \(x\)-axis to produce four intermediate results \(I_0, I_1, I_2, I_3\):

\[
I_k = \sum_{i=0}^{3} a_i \cdot g(x_{S_i}, y_{S_k}), \quad 0 \leq k \leq 3
\]

Then the intensity \(D( x_D, y_D)\) is computed by interpolating the intermediate values \(I_k\) along the \(y\)-axis:

\[
D(x_D, y_D) = \sum_{k=0}^{3} b_k \cdot I_k
\]

Here \(a_i\) and \(b_k\) are the coefficients defined as

\[
a_i = L(x_{S_i} - x_S), \quad b_k = L(y_{S_k} - y_S),
\]

where \(L(x)\) is the Lanczos windowed sinc function:

\[
L(x) = \sin{x} / \pi x \cdot \text{Lanczos}_2(x) = \begin{cases} \sin(\pi x) / \pi x \cdot \sin((\pi x) / 2), & 0 \leq |x| < 2 \\ 0, & 2 \leq |x| \end{cases}
\]

To use this interpolation, use the `ippiResizeLanczos` function.
Interpolation with the 3-lobed Lanczos Window Function

The interpolation algorithm uses source image intensities at 36 pixels in the neighborhood of the point \((x_S, y_S)\) in the source image:

\[x_{S0} = \text{int}(x_S) - 2;\]
\[x_{S1} = x_{S0} + 1;\]
\[x_{S2} = x_{S0} + 2;\]
\[x_{S3} = x_{S0} + 3;\]
\[x_{S4} = x_{S0} + 4;\]
\[x_{S5} = x_{S0} + 5;\]
\[y_{S0} = \text{int}(y_S) - 2;\]
\[y_{S1} = y_{S0} + 1;\]
\[y_{S2} = y_{S0} + 2;\]
\[y_{S3} = y_{S0} + 3;\]
\[y_{S4} = y_{S0} + 4;\]
\[y_{S5} = y_{S0} + 5;\]

First, the intensity values are interpolated along the \(x\)-axis to produce six intermediate results \(I_0, I_1, ..., I_5\):

\[I_k = \sum_{i=0}^{5} a_i \cdot s(x_{Si}, y_{Si}), 0 \leq k \leq 5\]

Then the intensity \(D(x_D, y_D)\) is computed by interpolating the intermediate values \(I_k\) along the \(y\)-axis:

\[D(x_D, y_D) = \sum_{k=0}^{5} b_k \cdot I_k\]

Here \(a_i\) and \(b_k\) are the coefficients defined as

\[a_i = L(x_S - x_{Si}),\]
\[b_k = L(y_S - y_{Si}),\]

where \(L(x)\) is the Lanczos windowed sinc function:

\[L(x) = \sin c(x) \cdot \text{Lanczos3}(x) = \begin{cases} \frac{\sin \left( \pi x \right)}{\pi x} \cdot \frac{\sin \left( \frac{\pi x}{3} \right)}{\frac{\pi x}{3}}, & 0 \leq |x| < 3 \\ 0, & 3 \leq |x| \end{cases}\]

To use this interpolation, set the \textit{interpolation} parameter to \texttt{IPPI_INTER_LANCZOS}, or use the functions with the \texttt{Lanczos} suffix (pass \texttt{interpolation=ippLanczos} to \texttt{GetSize} functions).

Interpolation with Two-Parameter Cubic Filters

The two-parameter family of cubic filters have kernels of the form:

\[k(x) = \begin{cases} \frac{1}{6} \left[ (12 - 9B - 6C)|x|^3 + (-18 + 12B + 6C)|x|^2 + (6 - 2B) \right], & |x| < 1 \\ \frac{1}{6} \left[ (-B - 6C)|x|^3 + (6B + 30C)|x|^2 + (-12B - 48C)|x| + (8B + 24C) \right], & 1 \leq |x| < 2 \\ 0, & \text{otherwise} \end{cases}\]

where \(B\) and \(C\) are two parameters; their variations give different approximation.

These interpolation methods additionally filter the output to improve quality of an image.

To get more information about how \(B\) and \(C\) values affect the result, refer to [Mitchell88].

To use the interpolation with two-parameter cubic filters, use the functions with the \texttt{Cubic} suffix (pass \texttt{interpolation=ippCubic} to \texttt{GetSize} functions).
Appendix C: Removed Functions for Image and Video Processing

This appendix contains tables that list the functions removed from Intel IPP 9.0. If an application created with the previous versions calls a function listed here, then the source code must be modified. The tables specify the corresponding Intel IPP 9.0 functions or workaround to replace the removed functions:

- ippcc.h
- ippcv.h
-ippi.h
-ippj.h - the whole domain is removed
-ippvc.h - the whole domain is removed

NOTE
To get information on possible alternatives to the removed functions that do not have substitution or workaround in Intel IPP 2019, refer to https://software.intel.com/en-us/articles/the-alternatives-for-intel-ipp-legacy-domains-and-functions or file a support request at Online Service Center.

### ippcc.h:

<table>
<thead>
<tr>
<th>Removed from 9.0</th>
<th>Substitution or Workaround</th>
</tr>
</thead>
<tbody>
<tr>
<td>ippiBGR555ToYCbCr411_16u8u_C3P3R</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiBGR555ToYCbCr420_16u8u_C3P3R</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiBGR555ToYCbCr422_16u8u_C3C2R</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiBGR555ToYCbCr422_16u8u_C3P3R</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiBGR555ToYCrCb420_16u8u_C3P3R</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiBGR555ToYUV420_16u8u_C3P3R</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiBGR565ToBGR_16u8u_C3R</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiBGR565ToYCbCr411_16u8u_C3P3R</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiBGR565ToYCbCr420_16u8u_C3P3R</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiBGR565ToYCbCr422_16u8u_C3C2R</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiBGR565ToYCbCr422_16u8u_C3P3R</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiBGR565ToYCrCb420_16u8u_C3P3R</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiBGR565ToYUV420_16u8u_C3P3R</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiBGRToBGR565_8u16u_C3R</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiColorTwist32f_8s_AC4IR</td>
<td>8s is not supported anymore, use another data type</td>
</tr>
<tr>
<td>ippiColorTwist32f_8s_AC4R</td>
<td>8s is not supported anymore, use another data type</td>
</tr>
<tr>
<td>ippiColorTwist32f_8s_C3IR</td>
<td>8s is not supported anymore, use another data type</td>
</tr>
</tbody>
</table>
### Removed from 9.0

<table>
<thead>
<tr>
<th>Function</th>
<th>Substitution or Workaround</th>
</tr>
</thead>
<tbody>
<tr>
<td>ippiColorTwist32f_8s_C3R</td>
<td>8s is not supported anymore, use another data type</td>
</tr>
<tr>
<td>ippiColorTwist32f_8s_IP3R</td>
<td>8s is not supported anymore, use another data type</td>
</tr>
<tr>
<td>ippiColorTwist32f_8s_P3R</td>
<td>8s is not supported anymore, use another data type</td>
</tr>
<tr>
<td>ippiRGB565ToRGB_16u8u_C3R</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiRGB565ToYUV420_16u8u_C3P3R</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiRGB565ToYUV422_16u8u_C3P3R</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiRGBToRGB565_8u16u_C3R</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiYCbCr411ToBGR555_8u16u_P3C3R</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiYCbCr411ToBGR565_8u16u_P3C3R</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiYCbCr420ToBGR444Dither_8u16u_P3C3R</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiYCbCr420ToBGR444_8u16u_P3C3R</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiYCbCr420ToBGR555Dither_8u16u_P3C3R</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiYCbCr420ToBGR555_8u16u_P3C3R</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiYCbCr420ToBGR565Dither_8u16u_P3C3R</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiYCbCr420ToBGR565_8u16u_P3C3R</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiYCbCr420ToBGR565Dither_8u16u_P3C3R</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiYCbCr420ToBGR565_8u16u_P3C3R</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiYCbCr422ToBGR444Dither_8u16u_P3C3R</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiYCbCr422ToBGR444_8u16u_P3C3R</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiYCbCr422ToBGR555Dither_8u16u_P3C3R</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiYCbCr422ToBGR555_8u16u_P3C3R</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiYCbCr422ToBGR565Dither_8u16u_P3C3R</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiYCbCr422ToBGR565_8u16u_P3C3R</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiYCbCr422ToBGR565Dither_8u16u_P3C3R</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiYCbCr422ToBGR565_8u16u_P3C3R</td>
<td>N/A</td>
</tr>
<tr>
<td>Removed from 9.0</td>
<td>Substitution or Workaround</td>
</tr>
<tr>
<td>---------------------------------------------------------------------------------</td>
<td>-----------------------------</td>
</tr>
<tr>
<td>ippiYCbCr422ToBGR565_8u16u_C2C3R</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiYCbCr422ToBGR565_8u16u_P3C3R</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiYCbCr422ToRGB444Dither_8u16u_C2C3R</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiYCbCr422ToRGB444Dither_8u16u_P3C3R</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiYCbCr422ToRGB444_8u16u_C2C3R</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiYCbCr422ToRGB444_8u16u_P3C3R</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiYCbCr422ToRGB555Dither_8u16u_C2C3R</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiYCbCr422ToRGB555Dither_8u16u_P3C3R</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiYCbCr422ToRGB555_8u16u_C2C3R</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiYCbCr422ToRGB555_8u16u_P3C3R</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiYCbCr422ToRGB565Dither_8u16u_C2C3R</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiYCbCr422ToRGB565Dither_8u16u_P3C3R</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiYCbCr422ToRGB565_8u16u_C2C3R</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiYCbCr422ToRGB565_8u16u_P3C3R</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiYCbCrToBGR444Dither_8u16u_C3R</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiYCbCrToBGR444Dither_8u16u_P3C3R</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiYCbCrToBGR444_8u16u_C3R</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiYCbCrToBGR444_8u16u_P3C3R</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiYCbCrToBGR555Dither_8u16u_C3R</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiYCbCrToBGR555Dither_8u16u_P3C3R</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiYCbCrToBGR555_8u16u_C3R</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiYCbCrToBGR555_8u16u_P3C3R</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiYCbCrToBGR565Dither_8u16u_C3R</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiYCbCrToBGR565Dither_8u16u_P3C3R</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiYCbCrToBGR565_8u16u_C3R</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiYCbCrToBGR565_8u16u_P3C3R</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiYCbCrToRGB444Dither_8u16u_C3R</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiYCbCrToRGB444Dither_8u16u_P3C3R</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiYCbCrToRGB444_8u16u_C3R</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiYCbCrToRGB444_8u16u_P3C3R</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiYCbCrToRGB555Dither_8u16u_C3R</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiYCbCrToRGB555Dither_8u16u_P3C3R</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiYCbCrToRGB555_8u16u_C3R</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiYCbCrToRGB555_8u16u_P3C3R</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiYCbCrToRGB565Dither_8u16u_C3R</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiYCbCrToRGB565Dither_8u16u_P3C3R</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiYCbCrToRGB565_8u16u_C3R</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiYCbCrToRGB565_8u16u_P3C3R</td>
<td>N/A</td>
</tr>
</tbody>
</table>
## Appendix C: Removed Functions for Image and Video Processing

**Removed from 9.0**

<table>
<thead>
<tr>
<th>Function</th>
<th>Substitution or Workaround</th>
</tr>
</thead>
<tbody>
<tr>
<td>ippiYCbCrToRGB555Dither_8u16u_P3C3R</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiYCbCrToRGB555_8u16u_C3R</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiYCbCrToRGB555_8u16u_P3C3R</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiYCbCrToRGB565Dither_8u16u_C3R</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiYCbCrToRGB565Dither_8u16u_P3C3R</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiYCbCrToRGB565_8u16u_C3R</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiYCbCrToRGB565_8u16u_P3C3R</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiYUV420ToBGR444Dither_8u16u_P3C3R</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiYUV420ToBGR444_8u16u_P3C3R</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiYUV420ToBGR555Dither_8u16u_P3C3R</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiYUV420ToBGR555_8u16u_P3C3R</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiYUV420ToBGR565Dither_8u16u_P3C3R</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiYUV420ToBGR565_8u16u_P3C3R</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiYUV420ToRGB444Dither_8u16u_P3C3R</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiYUV420ToRGB444_8u16u_P3C3R</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiYUV420ToRGB555Dither_8u16u_P3C3R</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiYUV420ToRGB555_8u16u_P3C3R</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiYUV420ToRGB565Dither_8u16u_P3C3R</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiYUV420ToRGB565_8u16u_P3C3R</td>
<td>N/A</td>
</tr>
</tbody>
</table>

### ippcv.h:

**Removed from 9.0**

<table>
<thead>
<tr>
<th>Function</th>
<th>Substitution or Workaround</th>
</tr>
</thead>
<tbody>
<tr>
<td>ippiAddProduct_8s32f_C1IMR</td>
<td>8s is not supported anymore, use another data type</td>
</tr>
<tr>
<td>ippiAddProduct_8s32f_C1IR</td>
<td>8s is not supported anymore, use another data type</td>
</tr>
<tr>
<td>ippiAddSquare_8s32f_C1IMR</td>
<td>8s is not supported anymore, use another data type</td>
</tr>
<tr>
<td>ippiAddSquare_8s32f_C1IR</td>
<td>8s is not supported anymore, use another data type</td>
</tr>
<tr>
<td>ippiAddWeighted_8s32f_C1IMR</td>
<td>8s is not supported anymore, use another data type</td>
</tr>
<tr>
<td>ippiAddWeighted_8s32f_C1IR</td>
<td>8s is not supported anymore, use another data type</td>
</tr>
<tr>
<td>ippiAdd_8s32f_C1IMR</td>
<td>8s is not supported anymore, use another data type</td>
</tr>
<tr>
<td>ippiAdd_8s32f_C1IR</td>
<td>8s is not supported anymore, use another data type</td>
</tr>
<tr>
<td>ippiDilateBorderReplicate_32f_C1R</td>
<td>ippiDilateBorder_32f_C1R</td>
</tr>
<tr>
<td>Removed from 9.0</td>
<td>Substitution or Workaround</td>
</tr>
<tr>
<td>-------------------------------------------------------</td>
<td>-----------------------------------------------------------------------</td>
</tr>
<tr>
<td>ippiDilateBorderReplicate_32f_C3R</td>
<td>ippiDilateBorder_32f_C3R</td>
</tr>
<tr>
<td>ippiDilateBorderReplicate_32f_C4R</td>
<td>ippiDilateBorder_32f_C4R</td>
</tr>
<tr>
<td>ippiDilateBorderReplicate_8u_C1R</td>
<td>ippiDilateBorder_8u_C1R</td>
</tr>
<tr>
<td>ippiDilateBorderReplicate_8u_C3R</td>
<td>ippiDilateBorder_8u_C3R</td>
</tr>
<tr>
<td>ippiDilateBorderReplicate_8u_C4R</td>
<td>ippiDilateBorder_8u_C4R</td>
</tr>
<tr>
<td>ippiErodeBorderReplicate_32f_C1R</td>
<td>ippiErodeBorder_32f_C1R</td>
</tr>
<tr>
<td>ippiErodeBorderReplicate_32f_C3R</td>
<td>ippiErodeBorder_32f_C3R</td>
</tr>
<tr>
<td>ippiErodeBorderReplicate_32f_C4R</td>
<td>ippiErodeBorder_32f_C4R</td>
</tr>
<tr>
<td>ippiErodeBorderReplicate_8u_C1R</td>
<td>ippiErodeBorder_8u_C1R</td>
</tr>
<tr>
<td>ippiErodeBorderReplicate_8u_C3R</td>
<td>ippiErodeBorder_8u_C3R</td>
</tr>
<tr>
<td>ippiErodeBorderReplicate_8u_C4R</td>
<td>ippiErodeBorder_8u_C4R</td>
</tr>
<tr>
<td>ippiFilterGaussBorder_32f_C1R</td>
<td>ippiFilterGaussianBorder_32f_C1R</td>
</tr>
<tr>
<td>ippiFilterGaussBorder_32f_C3R</td>
<td>ippiFilterGaussianBorder_32f_C3R</td>
</tr>
<tr>
<td>ippiFilterLaplacianBorder_8u8s_C1R</td>
<td>8s data type is not supported anymore - use another data type</td>
</tr>
<tr>
<td>ippiFilterLaplacianBorder_8u8s_C3R</td>
<td>8s data type is not supported anymore - use another data type</td>
</tr>
<tr>
<td>ippiFilterMaxBorderReplicate_32f_C1R</td>
<td>ippiFilterMaxBorder_32f_C1R</td>
</tr>
<tr>
<td>ippiFilterMaxBorderReplicate_32f_C3R</td>
<td>ippiFilterMaxBorder_32f_C3R</td>
</tr>
<tr>
<td>ippiFilterMaxBorderReplicate_32f_C4R</td>
<td>ippiFilterMaxBorder_32f_C4R</td>
</tr>
<tr>
<td>ippiFilterMaxBorderReplicate_8u_C1R</td>
<td>ippiFilterMaxBorder_8u_C1R</td>
</tr>
<tr>
<td>ippiFilterMaxBorderReplicate_8u_C3R</td>
<td>ippiFilterMaxBorder_8u_C3R</td>
</tr>
<tr>
<td>ippiFilterMaxBorderReplicate_8u_C4R</td>
<td>ippiFilterMaxBorder_8u_C4R</td>
</tr>
<tr>
<td>ippiFilterMaxGetBufferSize_32f_C1R</td>
<td>ippiFilterMaxBorderGetBufferSize with dataType=32f and numChannels=1</td>
</tr>
<tr>
<td>ippiFilterMaxGetBufferSize_32f_C3R</td>
<td>ippiFilterMaxBorderGetBufferSize with dataType=32f and numChannels=3</td>
</tr>
<tr>
<td>ippiFilterMaxGetBufferSize_32f_C4R</td>
<td>ippiFilterMaxBorderGetBufferSize with dataType=32f and numChannels=4</td>
</tr>
<tr>
<td>ippiFilterMaxGetBufferSize_8u_C1R</td>
<td>ippiFilterMaxBorderGetBufferSize with dataType=8u and numChannels=1</td>
</tr>
<tr>
<td>ippiFilterMaxGetBufferSize_8u_C3R</td>
<td>ippiFilterMaxBorderGetBufferSize with dataType=8u and numChannels=3</td>
</tr>
</tbody>
</table>
### Removed from 9.0

<table>
<thead>
<tr>
<th>Function</th>
<th>Substitution or Workaround</th>
</tr>
</thead>
<tbody>
<tr>
<td>ippiFilterMaxGetSize_8u_C4R</td>
<td>ippiFilterMaxBorderGetSize with dataType=8u and numChannels=4</td>
</tr>
<tr>
<td>ippiFilterMinBorderReplicate_32f_C1R</td>
<td>ippiFilterMinBorder_32f_C1R</td>
</tr>
<tr>
<td>ippiFilterMinBorderReplicate_32f_C3R</td>
<td>ippiFilterMinBorder_32f_C3R</td>
</tr>
<tr>
<td>ippiFilterMinBorderReplicate_32f_C4R</td>
<td>ippiFilterMinBorder_32f_C4R</td>
</tr>
<tr>
<td>ippiFilterMinBorderReplicate_8u_C1R</td>
<td>ippiFilterMinBorder_8u_C1R</td>
</tr>
<tr>
<td>ippiFilterMinBorderReplicate_8u_C3R</td>
<td>ippiFilterMinBorder_8u_C3R</td>
</tr>
<tr>
<td>ippiFilterMinBorderReplicate_8u_C4R</td>
<td>ippiFilterMinBorder_8u_C4R</td>
</tr>
<tr>
<td>ippiFilterMinGetSize_32f_C1R</td>
<td>ippiFilterMinBorderGetSize with dataType=32f and numChannels=1</td>
</tr>
<tr>
<td>ippiFilterMinGetSize_32f_C3R</td>
<td>ippiFilterMinBorderGetSize with dataType=32f and numChannels=3</td>
</tr>
<tr>
<td>ippiFilterMinGetSize_32f_C4R</td>
<td>ippiFilterMinBorderGetSize with dataType=32f and numChannels=4</td>
</tr>
<tr>
<td>ippiFilterMinGetSize_8u_C1R</td>
<td>ippiFilterMinBorderGetSize with dataType=8u and numChannels=1</td>
</tr>
<tr>
<td>ippiFilterMinGetSize_8u_C3R</td>
<td>ippiFilterMinBorderGetSize with dataType=8u and numChannels=3</td>
</tr>
<tr>
<td>ippiFilterMinGetSize_8u_C4R</td>
<td>ippiFilterMinBorderGetSize with dataType=8u and numChannels=4</td>
</tr>
<tr>
<td>ippiFilterScharrHorizBorder_32f_C1R</td>
<td>ippiFilterScharrHorizMaskBorder_32f_C1R from the ippIP domain</td>
</tr>
<tr>
<td>ippiFilterScharrHorizBorder_8u16s_C1R</td>
<td>ippiFilterScharrHorizMaskBorderGetSize from the ippIP domain</td>
</tr>
<tr>
<td>ippiFilterScharrHorizBorder_8u8s_C1R</td>
<td>8s data type is not supported anymore - use another data type</td>
</tr>
<tr>
<td>ippiFilterScharrHorizGetSize_32f_C1R</td>
<td>ippiFilterScharrHorizMaskBorderGetSize from the ippIP domain</td>
</tr>
<tr>
<td>ippiFilterScharrHorizGetSize_8u16s_C1R</td>
<td>ippiFilterScharrHorizMaskBorderGetSize from the ippIP domain</td>
</tr>
<tr>
<td>ippiFilterScharrHorizGetSize_8u8s_C1R</td>
<td>8s data type is not supported anymore - use another data type</td>
</tr>
<tr>
<td>ippiFilterScharrVertBorder_32f_C1R</td>
<td>ippiFilterScharrVertMaskBorder_32f_C1R from the ippIP domain</td>
</tr>
<tr>
<td>ippiFilterScharrVertBorder_8u16s_C1R</td>
<td>ippiFilterScharrVertMaskBorderGetSize from the ippIP domain</td>
</tr>
<tr>
<td>Removed from 9.0</td>
<td>Substitution or Workaround</td>
</tr>
<tr>
<td>------------------------------------------------------</td>
<td>----------------------------------------------------------------</td>
</tr>
<tr>
<td>ippiFilterScharrVertBorder_8u8s_C1R</td>
<td>8s data type is not supported anymore - use another data type</td>
</tr>
<tr>
<td>ippiFilterScharrVertGetBufferSize_32f_C1R</td>
<td>ippiFilterScharrVertMaskBorderGetBufferSize from the ippIP domain</td>
</tr>
<tr>
<td>ippiFilterScharrVertGetBufferSize_8u16s_C1R</td>
<td>ippiFilterScharrVertMaskBorderGetBufferSize from the ippIP domain</td>
</tr>
<tr>
<td>ippiFilterScharrVertGetBufferSize_8u8s_C1R</td>
<td>8s data type is not supported anymore - use another data type</td>
</tr>
<tr>
<td>ippiFilterSobelCrossBorder_8u8s_C1R</td>
<td>8s data type is not supported anymore - use another data type</td>
</tr>
<tr>
<td>ippiFilterSobelCrossGetBufferSize_8u8s_C1R</td>
<td>8s data type is not supported anymore - use another data type</td>
</tr>
<tr>
<td>ippiFilterSobelHorizBorder_8u8s_C1R</td>
<td>8s data type is not supported anymore - use another data type</td>
</tr>
<tr>
<td>ippiFilterSobelHorizGetBufferSize_8u8s_C1R</td>
<td>8s data type is not supported anymore - use another data type</td>
</tr>
<tr>
<td>ippiFilterSobelHorizSecondBorder_32f_C1R</td>
<td>ippiFilterSobelHorizSecondBorder_32f_C1R from the ippIP domain</td>
</tr>
<tr>
<td>ippiFilterSobelHorizSecondBorder_8u16s_C1R</td>
<td>ippiFilterSobelHorizSecondBorderGetBufferSize from the ippIP domain</td>
</tr>
<tr>
<td>ippiFilterSobelHorizSecondBorder_8u8s_C1R</td>
<td>8s data type is not supported anymore - use another data type</td>
</tr>
<tr>
<td>ippiFilterSobelHorizSecondGetBufferSize_32f_C1R</td>
<td>ippiFilterSobelHorizSecondBorderGetBufferSize from the ippIP domain</td>
</tr>
<tr>
<td>ippiFilterSobelHorizSecondGetBufferSize_8u16s_C1R</td>
<td>ippiFilterSobelHorizSecondBorderGetBufferSize from the ippIP domain</td>
</tr>
<tr>
<td>ippiFilterSobelHorizSecondGetBufferSize_8u8s_C1R</td>
<td>8s data type is not supported anymore - use another data type</td>
</tr>
<tr>
<td>ippiFilterSobelNegVertBorder_32f_C1R</td>
<td>ippiFilterSobelNegVertBorder_32f_C1R from the ippIP domain</td>
</tr>
<tr>
<td>ippiFilterSobelNegVertBorder_8u16s_C1R</td>
<td>ippiFilterSobelNegVertBorderGetBufferSize from the ippIP domain</td>
</tr>
<tr>
<td>ippiFilterSobelNegVertBorder_8u8s_C1R</td>
<td>8s data type is not supported anymore - use another data type</td>
</tr>
<tr>
<td>ippiFilterSobelNegGetBufferSize_32f_C1R</td>
<td>ippiFilterSobelNegVertBorderGetBufferSize from the ippIP domain</td>
</tr>
<tr>
<td>ippiFilterSobelNegGetBufferSize_8u16s_C1R</td>
<td>ippiFilterSobelNegVertBorderGetBufferSize from the ippIP domain</td>
</tr>
</tbody>
</table>
### Removed from 9.0

<table>
<thead>
<tr>
<th>Function</th>
<th>Substitution or Workaround</th>
</tr>
</thead>
<tbody>
<tr>
<td>ippiFilterSobelNegVertGetBufferSize_8u8s_C1R</td>
<td>8s data type is not supported anymore - use another data type</td>
</tr>
<tr>
<td>ippiFilterSobelVertBorder_8u8s_C1R</td>
<td>8s data type is not supported anymore - use another data type</td>
</tr>
<tr>
<td>ippiFilterSobelVertGetBufferSize_8u8s_C1R</td>
<td>8s data type is not supported anymore - use another data type</td>
</tr>
<tr>
<td>ippiFilterSobelVertSecondBorder_32f_C1R</td>
<td>ippiFilterSobelVertSecondBorder_32f_C1R from the ippIP domain</td>
</tr>
<tr>
<td>ippiFilterSobelVertSecondBorder_8u16s_C1R</td>
<td>ippiFilterSobelVertSecondBorderGetBufferSize from the ippIP domain</td>
</tr>
<tr>
<td>ippiFilterSobelVertSecondBorder_8u8s_C1R</td>
<td>8s data type is not supported anymore - use another data type</td>
</tr>
<tr>
<td>ippiFilterSobelVertSecondGetBufferSize_32f_C1R</td>
<td>ippiFilterSobelVertSecondBorderGetBufferSize from the ippIP domain</td>
</tr>
<tr>
<td>ippiFilterSobelVertSecondGetBufferSize_8u16s_C1R</td>
<td>ippiFilterSobelVertSecondBorderGetBufferSize from the ippIP domain</td>
</tr>
<tr>
<td>ippiFilterSobelVertSecondGetBufferSize_8u8s_C1R</td>
<td>8s data type is not supported anymore - use another data type</td>
</tr>
<tr>
<td>ippiForegroundGaussianFree_8u_C1R</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiForegroundGaussianFree_8u_C3R</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiForegroundGaussianInitAlloc_8u_C1R</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiForegroundGaussianInitAlloc_8u_C3R</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiForegroundGaussian_8u_C1R</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiForegroundGaussian_8u_C3R</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiForegroundHistogramFree_8u_C1R</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiForegroundHistogramFree_8u_C3R</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiForegroundHistogramInitAlloc_8u_C1R</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiForegroundHistogramInitAlloc_8u_C3R</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiForegroundHistogramUpdate_8u_C1R</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiForegroundHistogramUpdate_8u_C3R</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiForegroundHistogram_8u_C1R</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiForegroundHistogram_8u_C3R</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiHaarClassifierFree_32f</td>
<td>ippFree</td>
</tr>
<tr>
<td>ippiHaarClassifierFree_32s</td>
<td>ippFree</td>
</tr>
<tr>
<td>Removed from 9.0</td>
<td>Substitution or Workaround</td>
</tr>
<tr>
<td>--------------------------------------</td>
<td>-----------------------------------------------------------------</td>
</tr>
<tr>
<td>ippiHaarClassifierInitAlloc_32f</td>
<td>ippiHaarClassifierGetSize+ippMalloc</td>
</tr>
<tr>
<td></td>
<td>+ippiHaarClassifierInit_32f</td>
</tr>
<tr>
<td>ippiHaarClassifierInitAlloc_32s</td>
<td>ippiHaarClassifierGetSize+ippMalloc</td>
</tr>
<tr>
<td></td>
<td>+ippiHaarClassifierInit_32s</td>
</tr>
<tr>
<td>ippiInpaintFree_8u_C1R</td>
<td>ippiFree</td>
</tr>
<tr>
<td>ippiInpaintFree_8u_C3R</td>
<td>ippiFree</td>
</tr>
<tr>
<td>ippiInpaintInitAlloc_8u_C1R</td>
<td>ippiInpaintGetSize+ippMalloc</td>
</tr>
<tr>
<td></td>
<td>+ippiInpaintInit_8u_C1R</td>
</tr>
<tr>
<td>ippiInpaintInitAlloc_8u_C3R</td>
<td>ippiInpaintGetSize+ippMalloc</td>
</tr>
<tr>
<td></td>
<td>+ippiInpaintInit_8u_C3R</td>
</tr>
<tr>
<td>ippiMean_8s_C1MR</td>
<td>8s data type is not supported anymore - use another data type</td>
</tr>
<tr>
<td>ippiMean_8s_C3CMR</td>
<td>8s data type is not supported anymore - use another data type</td>
</tr>
<tr>
<td>ippiMean_StdDev_8s_C1MR</td>
<td>8s data type is not supported anymore - use another data type</td>
</tr>
<tr>
<td>ippiMean_StdDev_8s_C1R</td>
<td>8s data type is not supported anymore - use another data type</td>
</tr>
<tr>
<td>ippiMean_StdDev_8s_C3CMR</td>
<td>8s data type is not supported anymore - use another data type</td>
</tr>
<tr>
<td>ippiMean_StdDev_8s_C3CR</td>
<td>8s data type is not supported anymore - use another data type</td>
</tr>
<tr>
<td>ippiMinMaxIndx_8s_C1MR</td>
<td>8s data type is not supported anymore - use another data type</td>
</tr>
<tr>
<td>ippiMinMaxIndx_8s_C1R</td>
<td>8s data type is not supported anymore - use another data type</td>
</tr>
<tr>
<td>ippiMinMaxIndx_8s_C3CMR</td>
<td>8s data type is not supported anymore - use another data type</td>
</tr>
<tr>
<td>ippiMinMaxIndx_8s_C3CR</td>
<td>8s data type is not supported anymore - use another data type</td>
</tr>
<tr>
<td>ippiMorphAdvFree</td>
<td>ippFree</td>
</tr>
<tr>
<td>ippiMorphAdvInitAlloc_32f_C1R</td>
<td>ippiMorphAdvGetSize_32f_C1R+ippMalloc</td>
</tr>
<tr>
<td></td>
<td>+ippiMorphAdvInit_32f_C1R</td>
</tr>
<tr>
<td>ippiMorphAdvInitAlloc_32f_C3R</td>
<td>ippiMorphAdvGetSize_32f_C3R+ippMalloc</td>
</tr>
<tr>
<td></td>
<td>+ippiMorphAdvInit_32f_C3R</td>
</tr>
<tr>
<td>ippiMorphAdvInitAlloc_32f_C4R</td>
<td>ippiMorphAdvGetSize_32f_C4R+ippMalloc</td>
</tr>
<tr>
<td></td>
<td>+ippiMorphAdvInit_32f_C4R</td>
</tr>
<tr>
<td>Removed from 9.0</td>
<td>Substitution or Workaround</td>
</tr>
<tr>
<td>-------------------------------------------------------</td>
<td>-----------------------------------------------------------------</td>
</tr>
<tr>
<td>ippiMorphAdvInitAlloc_8u_C1R</td>
<td>ippiMorphAdvGetSize_8u_C1R+ippMalloc+ippiMorphAdvInit_8u_C1R</td>
</tr>
<tr>
<td>ippiMorphAdvInitAlloc_8u_C3R</td>
<td>ippiMorphAdvGetSize_8u_C3R+ippMalloc+ippiMorphAdvInit_8u_C3R</td>
</tr>
<tr>
<td>ippiMorphAdvInitAlloc_8u_C4R</td>
<td>ippiMorphAdvGetSize_8u_C4R+ippMalloc+ippiMorphAdvInit_8u_C4R</td>
</tr>
<tr>
<td>ippiMorphGrayFree_32f_C1R</td>
<td>ippFree</td>
</tr>
<tr>
<td>ippiMorphGrayFree_8u_C1R</td>
<td>ippFree</td>
</tr>
<tr>
<td>ippiMorphGrayInitAlloc_32f_C1R</td>
<td>ippiMorphGrayGetSize_32f_C1R+ippMalloc+ippiMorphGrayInit_32f_C1R</td>
</tr>
<tr>
<td>ippiMorphGrayInitAlloc_8u_C1R</td>
<td>ippiMorphGrayGetSize_8u_C1R+ippMalloc+ippiMorphGrayInit_8u_C1R</td>
</tr>
<tr>
<td>ippiMorphInitAlloc_32f_C1R</td>
<td>Use newippiMorphInitAlloc_32f_C1R</td>
</tr>
<tr>
<td>ippiMorphInitAlloc_32f_C3R</td>
<td>Use newippiMorphInitAlloc_32f_C3R</td>
</tr>
<tr>
<td>ippiMorphInitAlloc_32f_C4R</td>
<td>Use newippiMorphInitAlloc_32f_C4R</td>
</tr>
<tr>
<td>ippiMorphGrayGetBufferSize_16u_C1</td>
<td>ippFree</td>
</tr>
<tr>
<td>ippiMorphGrayGetBufferSize_32f_C1</td>
<td>ippFree</td>
</tr>
<tr>
<td>ippiMorphGrayGetBufferSize_64f_C1</td>
<td>ippFree</td>
</tr>
<tr>
<td>ippiMorphGrayGetBufferSize_8u_C1</td>
<td>ippFree</td>
</tr>
<tr>
<td>ippiMorphologyGetBufferSize_16u_C1</td>
<td>Use newippiMorphologyGetBufferSize_16u_C1</td>
</tr>
<tr>
<td>ippiMorphologyGetBufferSize_32f_C1</td>
<td>Use newippiMorphologyGetBufferSize_32f_C1</td>
</tr>
<tr>
<td>ippiMorphologyGetBufferSize_64f_C1</td>
<td>Use newippiMorphologyGetBufferSize_64f_C1</td>
</tr>
<tr>
<td>ippiMorphologyGetBufferSize_8u_C1</td>
<td>Use newippiMorphologyGetBufferSize_8u_C1</td>
</tr>
<tr>
<td>ippiMorphologyGetBufferSize_32f_C3R</td>
<td>ippMorphologyBorderGetBufferSize_32f_C3R</td>
</tr>
<tr>
<td>ippiMorphologyGetBufferSize_32f_C4R</td>
<td>ippMorphologyBorderGetBufferSize_32f_C4R</td>
</tr>
<tr>
<td>ippiMorphologyGetBufferSize_8u_C3R</td>
<td>ippMorphologyBorderGetBufferSize_8u_C3R</td>
</tr>
<tr>
<td>ippiMorphologyGetBufferSize_8u_C4R</td>
<td>ippMorphologyBorderGetBufferSize_8u_C4R</td>
</tr>
<tr>
<td>ippiMorphologyInitAlloc_32f_C1R</td>
<td>ippMorphologyBorderInit_32f_C1R</td>
</tr>
<tr>
<td>ippiMorphologyInitAlloc_32f_C3R</td>
<td>ippMorphologyBorderInit_32f_C3R</td>
</tr>
<tr>
<td>ippiMorphologyInitAlloc_32f_C4R</td>
<td>ippMorphologyBorderInit_32f_C4R</td>
</tr>
<tr>
<td>ippiMorphologyInitAlloc_8u_C1R</td>
<td>ippMorphologyBorderInit_8u_C1R</td>
</tr>
<tr>
<td>ippiMorphologyInitAlloc_8u_C3R</td>
<td>ippMorphologyBorderInit_8u_C3R</td>
</tr>
<tr>
<td>ippiMorphologyInitAlloc_8u_C4R</td>
<td>ippMorphologyBorderInit_8u_C4R</td>
</tr>
<tr>
<td>Removed from 9.0</td>
<td>Substitution or Workaround</td>
</tr>
<tr>
<td>------------------</td>
<td>---------------------------</td>
</tr>
<tr>
<td>ippiMorphologyInitAlloc_8u_C3R</td>
<td>ippiMorphologyBorderGetSize_8u_C3R +ippiMalloc_8u +ippiMorphologyBorderInit_8u_C3R</td>
</tr>
<tr>
<td>ippiMorphologyInitAlloc_8u_C4R</td>
<td>ippiMorphologyBorderGetSize_8u_C4R +ippiMalloc_8u +ippiMorphologyBorderInit_8u_C4R</td>
</tr>
<tr>
<td>ippiMorphologyInit_32f_C1R</td>
<td>ippiMorphologyBorderInit_32f_C1R</td>
</tr>
<tr>
<td>ippiMorphologyInit_32f_C3R</td>
<td>ippiMorphologyBorderInit_32f_C3R</td>
</tr>
<tr>
<td>ippiMorphologyInit_32f_C4R</td>
<td>ippiMorphologyBorderInit_32f_C4R</td>
</tr>
<tr>
<td>ippiMorphologyInit_8u_C1R</td>
<td>ippiMorphologyBorderInit_8u_C1R</td>
</tr>
<tr>
<td>ippiMorphologyInit_8u_C3R</td>
<td>ippiMorphologyBorderInit_8u_C3R</td>
</tr>
<tr>
<td>ippiMorphologyInit_8u_C4R</td>
<td>ippiMorphologyBorderInit_8u_C4R</td>
</tr>
<tr>
<td>ippiNormDiff_Inf_8s_C1MR</td>
<td>8s data type is not supported anymore - use another data type</td>
</tr>
<tr>
<td>ippiNormDiff_Inf_8s_C3CMR</td>
<td>8s data type is not supported anymore - use another data type</td>
</tr>
<tr>
<td>ippiNormDiff_L1_8s_C1MR</td>
<td>8s data type is not supported anymore - use another data type</td>
</tr>
<tr>
<td>ippiNormDiff_L1_8s_C3CMR</td>
<td>8s data type is not supported anymore - use another data type</td>
</tr>
<tr>
<td>ippiNormDiff_L2_8s_C1MR</td>
<td>8s data type is not supported anymore - use another data type</td>
</tr>
<tr>
<td>ippiNormDiff_L2_8s_C3CMR</td>
<td>8s data type is not supported anymore - use another data type</td>
</tr>
<tr>
<td>ippiNormRel_Inf_8s_C1MR</td>
<td>8s data type is not supported anymore - use another data type</td>
</tr>
<tr>
<td>ippiNormRel_Inf_8s_C3CMR</td>
<td>8s data type is not supported anymore - use another data type</td>
</tr>
<tr>
<td>ippiNormRel_L1_8s_C1MR</td>
<td>8s data type is not supported anymore - use another data type</td>
</tr>
<tr>
<td>ippiNormRel_L1_8s_C3CMR</td>
<td>8s data type is not supported anymore - use another data type</td>
</tr>
<tr>
<td>ippiNormRel_L2_8s_C1MR</td>
<td>8s data type is not supported anymore - use another data type</td>
</tr>
<tr>
<td>ippiNormRel_L2_8s_C3CMR</td>
<td>8s data type is not supported anymore - use another data type</td>
</tr>
<tr>
<td>Removed from 9.0</td>
<td>Substitution or Workaround</td>
</tr>
<tr>
<td>---------------------------------------------</td>
<td>----------------------------------------------------</td>
</tr>
<tr>
<td>ippiNorm_Inf_8s_C1MR</td>
<td>8s data type is not supported anymore - use another data type</td>
</tr>
<tr>
<td>ippiNorm_Inf_8s_C3CMR</td>
<td>8s data type is not supported anymore - use another data type</td>
</tr>
<tr>
<td>ippiNorm_L1_8s_C1MR</td>
<td>8s data type is not supported anymore - use another data type</td>
</tr>
<tr>
<td>ippiNorm_L1_8s_C3CMR</td>
<td>8s data type is not supported anymore - use another data type</td>
</tr>
<tr>
<td>ippiNorm_L2_8s_C1MR</td>
<td>8s data type is not supported anymore - use another data type</td>
</tr>
<tr>
<td>ippiNorm_L2_8s_C3CMR</td>
<td>8s data type is not supported anymore - use another data type</td>
</tr>
<tr>
<td>ippiOpticalFlowPyrLKFree_16u_C1R</td>
<td>ippFree</td>
</tr>
<tr>
<td>ippiOpticalFlowPyrLKFree_32f_C1R</td>
<td>ippFree</td>
</tr>
<tr>
<td>ippiOpticalFlowPyrLKFree_8u_C1R</td>
<td>ippFree</td>
</tr>
<tr>
<td>ippiOpticalFlowPyrLKInitAlloc_16u_C1R</td>
<td>ippiOpticalFlowPyrLKGetSize (with 16u dataType)+ippMalloc+ippiOpticalFlowPyrLKInit_16u_C1R</td>
</tr>
<tr>
<td>ippiOpticalFlowPyrLKInitAlloc_32f_C1R</td>
<td>ippiOpticalFlowPyrLKGetSize (with 32f dataType)+ippMalloc+ippiOpticalFlowPyrLKInit_32f_C1R</td>
</tr>
<tr>
<td>ippiOpticalFlowPyrLKInitAlloc_8u_C1R</td>
<td>ippiOpticalFlowPyrLKGetSize (with 8u dataType)+ippMalloc+ippiOpticalFlowPyrLKInit_8u_C1R</td>
</tr>
<tr>
<td>ippiPyrDownGetBufSize_Gauss5x5</td>
<td>ippiPyramidLayerDownGetSize_8u_C1R</td>
</tr>
<tr>
<td>ippiPyrDown_Gauss5x5_32f_C1R</td>
<td>ippiPyramidLayerDown_32f_C1R</td>
</tr>
<tr>
<td>ippiPyrDown_Gauss5x5_32f_C3R</td>
<td>ippiPyramidLayerDown_32f_C3R</td>
</tr>
<tr>
<td>ippiPyrDown_Gauss5x5_8s_C1R</td>
<td>8s data type is not supported anymore - use another data type</td>
</tr>
<tr>
<td>ippiPyrDown_Gauss5x5_8s_C3R</td>
<td>8s data type is not supported anymore - use another data type</td>
</tr>
<tr>
<td>ippiPyrDown_Gauss5x5_8u_C1R</td>
<td>ippiPyramidLayerDown_8u_C1R</td>
</tr>
<tr>
<td>ippiPyrDown_Gauss5x5_8u_C3R</td>
<td>ippiPyramidLayerDown_8u_C3R</td>
</tr>
<tr>
<td>ippiPyrUpGetSize_Gauss5x5</td>
<td>ippiPyramidLayerUpGetSize_8u_C1R</td>
</tr>
<tr>
<td>ippiPyrUp_Gauss5x5_32f_C1R</td>
<td>ippiPyramidLayerUp_32f_C1R</td>
</tr>
<tr>
<td>ippiPyrUp_Gauss5x5_32f_C3R</td>
<td>ippiPyramidLayerUp_32f_C3R</td>
</tr>
<tr>
<td>Removed from 9.0</td>
<td>Substitution or Workaround</td>
</tr>
<tr>
<td>----------------------------------------------</td>
<td>---------------------------------------------</td>
</tr>
<tr>
<td>ippiPyrUp_Gauss5x5_8s_C1R</td>
<td>8s data type is not supported anymore - use another data type</td>
</tr>
<tr>
<td>ippiPyrUp_Gauss5x5_8s_C3R</td>
<td>8s data type is not supported anymore - use another data type</td>
</tr>
<tr>
<td>ippiPyrUp_Gauss5x5_8u_C1R</td>
<td>ippiPyramidLayerUp_8u_C1R</td>
</tr>
<tr>
<td>ippiPyrUp_Gauss5x5_8u_C3R</td>
<td>ippiPyramidLayerUp_8u_C3R</td>
</tr>
<tr>
<td>ippiPyramidFree</td>
<td>ippFree</td>
</tr>
<tr>
<td>ippiPyramidInitAlloc</td>
<td>ippiPyramidGetSize+ippiPyramidInit</td>
</tr>
<tr>
<td>ippiPyramidLayerDownFree_16u_C1R</td>
<td>ippFree</td>
</tr>
<tr>
<td>ippiPyramidLayerDownFree_16u_C3R</td>
<td>ippFree</td>
</tr>
<tr>
<td>ippiPyramidLayerDownFree_32f_C1R</td>
<td>ippFree</td>
</tr>
<tr>
<td>ippiPyramidLayerDownFree_32f_C3R</td>
<td>ippFree</td>
</tr>
<tr>
<td>ippiPyramidLayerDownFree_8u_C1R</td>
<td>ippFree</td>
</tr>
<tr>
<td>ippiPyramidLayerDownFree_8u_C3R</td>
<td>ippFree</td>
</tr>
<tr>
<td>ippiPyramidLayerDownInitAlloc_16u_C1R</td>
<td>ippiPyramidLayerDownGetSize_16u_C1R +ippMalloc +ippiPyramidLayerDownInit_16u_C1R</td>
</tr>
<tr>
<td>ippiPyramidLayerDownInitAlloc_16u_C3R</td>
<td>ippiPyramidLayerDownGetSize_16u_C3R +ippMalloc +ippiPyramidLayerDownInit_16u_C3R</td>
</tr>
<tr>
<td>ippiPyramidLayerDownInitAlloc_32f_C1R</td>
<td>ippiPyramidLayerDownGetSize_32f_C1R +ippMalloc +ippiPyramidLayerDownInit_32f_C1R</td>
</tr>
<tr>
<td>ippiPyramidLayerDownInitAlloc_32f_C3R</td>
<td>ippiPyramidLayerDownGetSize_32f_C3R +ippMalloc +ippiPyramidLayerDownInit_32f_C3R</td>
</tr>
<tr>
<td>ippiPyramidLayerDownInitAlloc_8u_C1R</td>
<td>ippiPyramidLayerDownGetSize_8u_C1R +ippMalloc +ippiPyramidLayerDownInit_8u_C1R</td>
</tr>
<tr>
<td>ippiPyramidLayerDownInitAlloc_8u_C3R</td>
<td>ippiPyramidLayerDownGetSize_8u_C3R +ippMalloc +ippiPyramidLayerDownInit_8u_C3R</td>
</tr>
<tr>
<td>ippiPyramidLayerUpFree_16u_C1R</td>
<td>ippFree</td>
</tr>
<tr>
<td>ippiPyramidLayerUpFree_16u_C3R</td>
<td>ippFree</td>
</tr>
<tr>
<td>ippiPyramidLayerUpFree_32f_C1R</td>
<td>ippFree</td>
</tr>
</tbody>
</table>
### Appendix C: Removed Functions for Image and Video Processing

<table>
<thead>
<tr>
<th>Removed from 9.0</th>
<th>Substitution or Workaround</th>
</tr>
</thead>
<tbody>
<tr>
<td>ippiPyramidLayerUpFree_32f_C3R</td>
<td>ippFree</td>
</tr>
<tr>
<td>ippiPyramidLayerUpFree_8u_C1R</td>
<td>ippFree</td>
</tr>
<tr>
<td>ippiPyramidLayerUpFree_8u_C3R</td>
<td>ippFree</td>
</tr>
<tr>
<td>ippiPyramidLayerUpInitAlloc_16u_C1R</td>
<td>ippiPyramidLayerUpGetSize_16u_C1R + ippMalloc + ippiPyramidLayerUpInit_16u_C1R</td>
</tr>
<tr>
<td>ippiPyramidLayerUpInitAlloc_16u_C3R</td>
<td>ippiPyramidLayerUpGetSize_16u_C3R + ippMalloc + ippiPyramidLayerUpInit_16u_C3R</td>
</tr>
<tr>
<td>ippiPyramidLayerUpInitAlloc_32f_C1R</td>
<td>ippiPyramidLayerUpGetSize_32f_C1R + ippMalloc + ippiPyramidLayerUpInit_32u_C1R</td>
</tr>
<tr>
<td>ippiPyramidLayerUpInitAlloc_32f_C3R</td>
<td>ippiPyramidLayerUpGetSize_32f_C3R + ippMalloc + ippiPyramidLayerUpInit_32f_C3R</td>
</tr>
<tr>
<td>ippiPyramidLayerUpInitAlloc_8u_C1R</td>
<td>ippiPyramidLayerUpGetSize_8u_C1R + ippMalloc + ippiPyramidLayerUpInit_8u_C1R</td>
</tr>
<tr>
<td>ippiPyramidLayerUpInitAlloc_8u_C3R</td>
<td>ippiPyramidLayerUpGetSize_8u_C3R + ippMalloc + ippiPyramidLayerUpInit_8u_C3R</td>
</tr>
<tr>
<td>ippiSRHNCalcResidual_PSF2x2_16u32f_C1R</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiSRHNCalcResidual_PSF2x2_8u32f_C1R</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiSRHNCalcResidual_PSF3x3_16u32f_C1R</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiSRHNCalcResidual_PSF3x3_8u32f_C1R</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiSRHNFree_PSF2x2</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiSRHNFree_PSF3x3</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiSRHInitAlloc_PSF2x2</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiSRHInitAlloc_PSF3x3</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiSRHUpdateGradient_PSF2x2_32f_C1R</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiSRHUpdateGradient_PSF3x3_32f_C1R</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiTiltedHaarClassifierInitAlloc_32f</td>
<td>ippiHaarClassifierGetSize + ippMalloc + ippiTiltedHaarClassifierInit_32f</td>
</tr>
<tr>
<td>ippiTiltedHaarClassifierInitAlloc_32s</td>
<td>ippiHaarClassifierGetSize + ippMalloc + ippiTiltedHaarClassifierInit_32s</td>
</tr>
<tr>
<td>ippibFastArctan_32f</td>
<td>ippsAtan_32f_A11</td>
</tr>
</tbody>
</table>
**ippi.h:**

<table>
<thead>
<tr>
<th>Removed from 9.0</th>
<th>Substitution or Workaround</th>
</tr>
</thead>
<tbody>
<tr>
<td>ippiAddC_16sc_AC4IRSfs</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiAddC_16sc_AC4RSfs</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiAddC_16sc_C1IRSfs</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiAddC_16sc_C1RSfs</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiAddC_16sc_C3IRSfs</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiAddC_16sc_C3RSfs</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiAddC_32fc_AC4IR</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiAddC_32fc_AC4R</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiAddC_32fc_C1IR</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiAddC_32fc_C1R</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiAddC_32fc_C3IR</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiAddC_32fc_C3R</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiAddC_32sc_AC4IRSfs</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiAddC_32sc_AC4RSfs</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiAddC_32sc_C1IRSfs</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiAddC_32sc_C1RSfs</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiAddC_32sc_C3IRSfs</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiAddC_32sc_C3RSfs</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiAddRandGauss_Direct_16s_AC4IR</td>
<td>ippiAddRandGauss_16s_AC4IR</td>
</tr>
<tr>
<td>ippiAddRandGauss_Direct_16s_C1IR</td>
<td>ippiAddRandGauss_16s_C1IR</td>
</tr>
<tr>
<td>ippiAddRandGauss_Direct_16s_C3IR</td>
<td>ippiAddRandGauss_16s_C3IR</td>
</tr>
<tr>
<td>ippiAddRandGauss_Direct_16s_C4IR</td>
<td>ippiAddRandGauss_16s_C4IR</td>
</tr>
<tr>
<td>ippiAddRandGauss_Direct_16u_AC4IR</td>
<td>ippiAddRandGauss_16u_AC4IR</td>
</tr>
<tr>
<td>ippiAddRandGauss_Direct_16u_C1IR</td>
<td>ippiAddRandGauss_16u_C1IR</td>
</tr>
<tr>
<td>ippiAddRandGauss_Direct_16u_C3IR</td>
<td>ippiAddRandGauss_16u_C3IR</td>
</tr>
<tr>
<td>ippiAddRandGauss_Direct_16u_C4IR</td>
<td>ippiAddRandGauss_16u_C4IR</td>
</tr>
<tr>
<td>ippiAddRandGauss_Direct_32f_AC4IR</td>
<td>ippiAddRandGauss_32f_AC4IR</td>
</tr>
<tr>
<td>ippiAddRandGauss_Direct_32f_C1IR</td>
<td>ippiAddRandGauss_32f_C1IR</td>
</tr>
<tr>
<td>ippiAddRandGauss_Direct_32f_C3IR</td>
<td>ippiAddRandGauss_32f_C3IR</td>
</tr>
<tr>
<td>ippiAddRandGauss_Direct_32f_C4IR</td>
<td>ippiAddRandGauss_32f_C4IR</td>
</tr>
</tbody>
</table>
## Removed from 9.0

<table>
<thead>
<tr>
<th>Removed from 9.0</th>
<th>Substitution or Workaround</th>
</tr>
</thead>
<tbody>
<tr>
<td>ippiAddRandGauss_Direct_8u_AC4IR</td>
<td>ippiAddRandGauss_8u_AC4IR</td>
</tr>
<tr>
<td>ippiAddRandGauss_Direct_8u_C1IR</td>
<td>ippiAddRandGauss_8u_C1IR</td>
</tr>
<tr>
<td>ippiAddRandGauss_Direct_8u_C3IR</td>
<td>ippiAddRandGauss_8u_C3IR</td>
</tr>
<tr>
<td>ippiAddRandGauss_Direct_8u_C4IR</td>
<td>ippiAddRandGauss_8u_C4IR</td>
</tr>
<tr>
<td>ippiAddRandUniform_Direct_16s_AC4IR</td>
<td>ippiAddRandUniform_16s_AC4IR</td>
</tr>
<tr>
<td>ippiAddRandUniform_Direct_16s_C1IR</td>
<td>ippiAddRandUniform_16s_C1IR</td>
</tr>
<tr>
<td>ippiAddRandUniform_Direct_16s_C3IR</td>
<td>ippiAddRandUniform_16s_C3IR</td>
</tr>
<tr>
<td>ippiAddRandUniform_Direct_16s_C4IR</td>
<td>ippiAddRandUniform_16s_C4IR</td>
</tr>
<tr>
<td>ippiAddRandUniform_Direct_16u_AC4IR</td>
<td>ippiAddRandUniform_16u_AC4IR</td>
</tr>
<tr>
<td>ippiAddRandUniform_Direct_16u_C1IR</td>
<td>ippiAddRandUniform_16u_C1IR</td>
</tr>
<tr>
<td>ippiAddRandUniform_Direct_16u_C3IR</td>
<td>ippiAddRandUniform_16u_C3IR</td>
</tr>
<tr>
<td>ippiAddRandUniform_Direct_16u_C4IR</td>
<td>ippiAddRandUniform_16u_C4IR</td>
</tr>
<tr>
<td>ippiAddRandUniform_Direct_32f_AC4IR</td>
<td>ippiAddRandUniform_32f_AC4IR</td>
</tr>
<tr>
<td>ippiAddRandUniform_Direct_32f_C1IR</td>
<td>ippiAddRandUniform_32f_C1IR</td>
</tr>
<tr>
<td>ippiAddRandUniform_Direct_32f_C3IR</td>
<td>ippiAddRandUniform_32f_C3IR</td>
</tr>
<tr>
<td>ippiAddRandUniform_Direct_32f_C4IR</td>
<td>ippiAddRandUniform_32f_C4IR</td>
</tr>
<tr>
<td>ippiAddRandUniform_Direct_8u_AC4IR</td>
<td>ippiAddRandUniform_8u_AC4IR</td>
</tr>
<tr>
<td>ippiAddRandUniform_Direct_8u_C1IR</td>
<td>ippiAddRandUniform_8u_C1IR</td>
</tr>
<tr>
<td>ippiAddRandUniform_Direct_8u_C3IR</td>
<td>ippiAddRandUniform_8u_C3IR</td>
</tr>
<tr>
<td>ippiAddRandUniform_Direct_8u_C4IR</td>
<td>ippiAddRandUniform_8u_C4IR</td>
</tr>
<tr>
<td>ippiAddRotateShift</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiAdd_16sc_AC4IRsfs</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiAdd_16sc_AC4RSfs</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiAdd_16sc_C1IRSfs</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiAdd_16sc_C1RSfs</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiAdd_16sc_C3IRSfs</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiAdd_16sc_C3RSfs</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiAdd_32fc_AC4IR</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiAdd_32fc_AC4R</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiAdd_32fc_C1IR</td>
<td>N/A</td>
</tr>
<tr>
<td>Removed from 9.0</td>
<td>Substitution or Workaround</td>
</tr>
<tr>
<td>----------------------------------</td>
<td>------------------------------------------------</td>
</tr>
<tr>
<td>ippiAdd_32fc_C1R</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiAdd_32fc_C3IR</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiAdd_32fc_C3R</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiAdd_32sc_AC4IRSfs</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiAdd_32sc_AC4RSfs</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiAdd_32sc_C1IRSfs</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiAdd_32sc_C1RSfs</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiAdd_32sc_C3IRSfs</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiAdd_32sc_C3RSfs</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiAlphaCompC_8s_C1R</td>
<td>ippiConvert_8s16s+ippiAlphaCompC_16s</td>
</tr>
<tr>
<td>ippiAlphaComp_8s_AC1R</td>
<td>ippiConvert_8s16s+ippiAlphaCompC_16s</td>
</tr>
<tr>
<td>ippiAlphaComp_8s_AC4R</td>
<td>ippiConvert_8s16s+ippiAlphaCompC_16s</td>
</tr>
<tr>
<td>ippiComplement_32s_C1IR</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiConvFull_16s_AC4R</td>
<td>ippiConv_16s_C4R (with algType=ippiROIFull)</td>
</tr>
<tr>
<td>ippiConvFull_16s_C1R</td>
<td>ippiConv_16s_C1R (with algType=ippiROIFull)</td>
</tr>
<tr>
<td>ippiConvFull_16s_C3R</td>
<td>ippiConv_16s_C3R (with algType=ippiROIFull)</td>
</tr>
<tr>
<td>ippiConvFull_32f_AC4R</td>
<td>ippiConv_32f_C4R (with algType=ippiROIFull)</td>
</tr>
<tr>
<td>ippiConvFull_32f_C1R</td>
<td>ippiConv_32f_C1R (with algType=ippiROIFull)</td>
</tr>
<tr>
<td>ippiConvFull_32f_C3R</td>
<td>ippiConv_32f_C3R (with algType=ippiROIFull)</td>
</tr>
<tr>
<td>ippiConvFull_8u_AC4R</td>
<td>ippiConv_8u_C4R (with algType=ippiROIFull)</td>
</tr>
<tr>
<td>ippiConvFull_8u_C1R</td>
<td>ippiConv_8u_C1R (with algType=ippiROIFull)</td>
</tr>
<tr>
<td>ippiConvFull_8u_C3R</td>
<td>ippiConv_8u_C3R (with algType=ippiROIFull)</td>
</tr>
<tr>
<td>ippiConvValid_16s_AC4R</td>
<td>ippiConv_16s_C4R (with algType=ippiROIValid)</td>
</tr>
<tr>
<td>ippiConvValid_16s_C1R</td>
<td>ippiConv_16s_C1R (with algType=ippiROIValid)</td>
</tr>
<tr>
<td>ippiConvValid_16s_C3R</td>
<td>ippiConv_16s_C3R (with algType=ippiROIValid)</td>
</tr>
<tr>
<td>ippiConvValid_32f_AC4R</td>
<td>ippiConv_32f_C4R (with algType=ippiROIValid)</td>
</tr>
<tr>
<td>ippiConvValid_32f_C1R</td>
<td>ippiConv_32f_C1R (with algType=ippiROIValid)</td>
</tr>
<tr>
<td>ippiConvValid_32f_C3R</td>
<td>ippiConv_32f_C3R (with algType=ippiROIValid)</td>
</tr>
<tr>
<td>ippiConvValid_8u_AC4R</td>
<td>ippiConv_8u_C4R (with algType=ippiROIValid)</td>
</tr>
<tr>
<td>ippiConvValid_8u_C1R</td>
<td>ippiConv_8u_C1R (with algType=ippiROIValid)</td>
</tr>
<tr>
<td>Removed from 9.0</td>
<td>Substitution or Workaround</td>
</tr>
<tr>
<td>----------------------------------------</td>
<td>-------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>ippiConvValid_8u_C3R</td>
<td>ippiConv8u_C3R (with algType=ippiROIValid)</td>
</tr>
<tr>
<td>ippiConvert_1u8u_C1R</td>
<td>ippiBinToGray_1u8u_C1R</td>
</tr>
<tr>
<td>ippiCplxExtendToPack_16sc16s_C1R</td>
<td>ippiConvert_16s32f</td>
</tr>
<tr>
<td></td>
<td>+ippiCplxExtendToPack_32f</td>
</tr>
<tr>
<td>ippiCplxExtendToPack_16sc16s_C3R</td>
<td>ippiConvert_16s32f</td>
</tr>
<tr>
<td></td>
<td>+ippiCplxExtendToPack_32f</td>
</tr>
<tr>
<td>ippiCplxExtendToPack_32sc32s_C1R</td>
<td>ippiConvert_32s32f</td>
</tr>
<tr>
<td></td>
<td>+ippiCplxExtendToPack_32f</td>
</tr>
<tr>
<td>ippiCplxExtendToPack_32sc32s_C3R</td>
<td>ippiConvert_32s32f</td>
</tr>
<tr>
<td></td>
<td>+ippiCplxExtendToPack_32f</td>
</tr>
<tr>
<td>ippiCrossCorrFull_NormLevel_16u32f_AC4R</td>
<td>ippiCopy_16u_C4C1R</td>
</tr>
<tr>
<td></td>
<td>+ippiCrossCorrNorm_16u32f_C1R</td>
</tr>
<tr>
<td></td>
<td>+algType=ippiROIFull</td>
</tr>
<tr>
<td>ippiCrossCorrFull_NormLevel_16u32f_C1R</td>
<td>ippiCrossCorrNorm_16u32f_C1R</td>
</tr>
<tr>
<td></td>
<td>+algType=ippiROIFull</td>
</tr>
<tr>
<td>ippiCrossCorrFull_NormLevel_16u32f_C3R</td>
<td>ippiCopy_16u_C3C1R</td>
</tr>
<tr>
<td></td>
<td>+ippiCrossCorrNorm_16u32f_C1R</td>
</tr>
<tr>
<td></td>
<td>+algType=ippiROIFull</td>
</tr>
<tr>
<td>ippiCrossCorrFull_NormLevel_16u32f_C4R</td>
<td>ippiCopy_16u_C4C1R</td>
</tr>
<tr>
<td></td>
<td>+ippiCrossCorrNorm_16u32f_C1R</td>
</tr>
<tr>
<td></td>
<td>+algType=ippiROIFull</td>
</tr>
<tr>
<td>ippiCrossCorrFull_NormLevel_32f_AC4R</td>
<td>ippiCopy_32f_C4C1R</td>
</tr>
<tr>
<td></td>
<td>+ippiCrossCorrNorm_32f_C1R</td>
</tr>
<tr>
<td></td>
<td>+algType=ippiROIFull</td>
</tr>
<tr>
<td>ippiCrossCorrFull_NormLevel_32f_C1R</td>
<td>ippiCrossCorrNorm_32f_C1R</td>
</tr>
<tr>
<td></td>
<td>+algType=ippiROIFull</td>
</tr>
<tr>
<td>ippiCrossCorrFull_NormLevel_32f_C3R</td>
<td>ippiCopy_32f_C3C1R</td>
</tr>
<tr>
<td></td>
<td>+ippiCrossCorrNorm_32f_C1R</td>
</tr>
<tr>
<td></td>
<td>+algType=ippiROIFull</td>
</tr>
<tr>
<td>ippiCrossCorrFull_NormLevel_32f_C4R</td>
<td>ippiCopy_32f_C4C1R</td>
</tr>
<tr>
<td></td>
<td>+ippiCrossCorrNorm_32f_C1R</td>
</tr>
<tr>
<td></td>
<td>+algType=ippiROIFull</td>
</tr>
<tr>
<td>ippiCrossCorrFull_NormLevel_64f_C1R</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiCrossCorrFull_NormLevel_8s32f_AC4R</td>
<td>ippiConvert_8s32f+ippiCopy_32f_C4C1R</td>
</tr>
<tr>
<td></td>
<td>+ippiCrossCorrNorm_32f_C1R</td>
</tr>
<tr>
<td></td>
<td>+algType=ippiROIFull</td>
</tr>
<tr>
<td>Removed from 9.0</td>
<td>Substitution or Workaround</td>
</tr>
<tr>
<td>-------------------------------------------------------</td>
<td>-------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>ippiCrossCorrFull_NormLevel_8s32f_C1R</td>
<td>ippiConvert_8s32f +ippiCrossCorrNorm_32f_C1R +algType=ippiROIFull</td>
</tr>
<tr>
<td>ippiCrossCorrFull_NormLevel_8s32f_C3R</td>
<td>ippiConvert_8s32f +ippiCopy_32f_C3C1R +ippiCrossCorrNorm_32f_C1R +algType=ippiROIFull</td>
</tr>
<tr>
<td>ippiCrossCorrFull_NormLevel_8s32f_C4R</td>
<td>ippiConvert_8s32f +ippiCopy_32f_C4C1R +ippiCrossCorrNorm_32f_C1R +algType=ippiROIFull</td>
</tr>
<tr>
<td>ippiCrossCorrFull_NormLevel_8u32f_C1R</td>
<td>ippiCrossCorrNorm_8u32f_C1R +algType=ippiROIFull</td>
</tr>
<tr>
<td>ippiCrossCorrFull_NormLevel_8u32f_C3R</td>
<td>ippiCopy_8u_C3C1R +ippiCrossCorrNorm_8u32f_C1R +algType=ippiROIFull</td>
</tr>
<tr>
<td>ippiCrossCorrFull_NormLevel_8u32f_C4R</td>
<td>ippiCopy_8u_C4C1R +ippiCrossCorrNorm_8u32f_C1R +algType=ippiROIFull</td>
</tr>
<tr>
<td>ippiCrossCorrFull_NormLevel_8u_AC4RSfs</td>
<td>ippiCopy_8u_C4C1R +ippiCrossCorrNorm_8u_C1RSfs +algType=ippiROIFull</td>
</tr>
<tr>
<td>ippiCrossCorrFull_NormLevel_8u_C1RSfs</td>
<td>ippiCrossCorrNorm_8u_C1RSfs +algType=ippiROIFull</td>
</tr>
<tr>
<td>ippiCrossCorrFull_NormLevel_8u_C3RSfs</td>
<td>ippiCopy_8u_C3C1R +ippiCrossCorrNorm_8u_C1RSfs +algType=ippiROIFull</td>
</tr>
<tr>
<td>ippiCrossCorrFull_NormLevel_8u_C4RSfs</td>
<td>ippiCopy_8u_C4C1R +ippiCrossCorrNorm_8u_C1RSfs +algType=ippiROIFull</td>
</tr>
<tr>
<td>ippiCrossCorrFull_Norm_16u32f_AC4R</td>
<td>ippiCopy_16u_C4C1R +ippiCrossCorrNorm_16u32f_C1R +algType=ippiROIFull</td>
</tr>
<tr>
<td>ippiCrossCorrFull_Norm_16u32f_C1R</td>
<td>ippiCrossCorrNorm_16u32f_C1R +algType=ippiROIFull</td>
</tr>
<tr>
<td>ippiCrossCorrFull_Norm_16u32f_C3R</td>
<td>ippiCrossCorrNorm_16u32f_C1R +algType=ippiROIFull</td>
</tr>
<tr>
<td>Removed from 9.0</td>
<td>Substitution or Workaround</td>
</tr>
<tr>
<td>-------------------------------------------------------</td>
<td>-----------------------------------------------------------------</td>
</tr>
<tr>
<td>ippiCrossCorrFull_Norm_16u32f_C4R</td>
<td>ippiCopy_16u_C4C1R</td>
</tr>
<tr>
<td></td>
<td>+ippiCrossCorrNorm_16u32f_C1R</td>
</tr>
<tr>
<td></td>
<td>+algType=ippiROIFull</td>
</tr>
<tr>
<td>ippiCrossCorrFull_Norm_32f_AC4R</td>
<td>ippiCopy_32f_C4C1R</td>
</tr>
<tr>
<td></td>
<td>+ippiCrossCorrNorm_32f_C1R</td>
</tr>
<tr>
<td></td>
<td>+algType=ippiROIFull</td>
</tr>
<tr>
<td>ippiCrossCorrFull_Norm_32f_C1R</td>
<td>ippiCrossCorrNorm_32f_C1R</td>
</tr>
<tr>
<td></td>
<td>+algType=ippiROIFull</td>
</tr>
<tr>
<td>ippiCrossCorrFull_Norm_32f_C3R</td>
<td>ippiCopy_32f_C3C1R</td>
</tr>
<tr>
<td></td>
<td>+ippiCrossCorrNorm_32f_C1R</td>
</tr>
<tr>
<td></td>
<td>+algType=ippiROIFull</td>
</tr>
<tr>
<td>ippiCrossCorrFull_Norm_32f_C4R</td>
<td>ippiCopy_32f_C4C1R</td>
</tr>
<tr>
<td></td>
<td>+ippiCrossCorrNorm_32f_C1R</td>
</tr>
<tr>
<td></td>
<td>+algType=ippiROIFull</td>
</tr>
<tr>
<td>ippiCrossCorrFull_Norm_8s32f_AC4R</td>
<td>ippiConvert_8s32f+ippiCopy_32f_C4C1R</td>
</tr>
<tr>
<td></td>
<td>+ippiCrossCorrNorm_32f_C1R</td>
</tr>
<tr>
<td></td>
<td>+algType=ippiROIFull</td>
</tr>
<tr>
<td>ippiCrossCorrFull_Norm_8s32f_C1R</td>
<td>ippiConvert_8s32f</td>
</tr>
<tr>
<td></td>
<td>+ippiCrossCorrNorm_32f_C1R</td>
</tr>
<tr>
<td></td>
<td>+algType=ippiROIFull</td>
</tr>
<tr>
<td>ippiCrossCorrFull_Norm_8s32f_C3R</td>
<td>ippiConvert_8s32f+ippiCopy_32f_C3C1R</td>
</tr>
<tr>
<td></td>
<td>+ippiCrossCorrNorm_32f_C1R</td>
</tr>
<tr>
<td></td>
<td>+algType=ippiROIFull</td>
</tr>
<tr>
<td>ippiCrossCorrFull_Norm_8s32f_C4R</td>
<td>ippiConvert_8s32f+ippiCopy_32f_C4C1R</td>
</tr>
<tr>
<td></td>
<td>+ippiCrossCorrNorm_32f_C1R</td>
</tr>
<tr>
<td></td>
<td>+algType=ippiROIFull</td>
</tr>
<tr>
<td>ippiCrossCorrFull_Norm_8u32f_AC4R</td>
<td>ippiCopy_8u_C4C1R</td>
</tr>
<tr>
<td></td>
<td>+ippiCrossCorrNorm_8u32f_C1R</td>
</tr>
<tr>
<td></td>
<td>+algType=ippiROIFull</td>
</tr>
<tr>
<td>ippiCrossCorrFull_Norm_8u32f_C1R</td>
<td>ippiCrossCorrNorm_8u32f_C1R</td>
</tr>
<tr>
<td></td>
<td>+algType=ippiROIFull</td>
</tr>
<tr>
<td>ippiCrossCorrFull_Norm_8u32f_C3R</td>
<td>ippiCopy_8u_C3C1R</td>
</tr>
<tr>
<td></td>
<td>+ippiCrossCorrNorm_8u32f_C1R</td>
</tr>
<tr>
<td></td>
<td>+algType=ippiROIFull</td>
</tr>
<tr>
<td>ippiCrossCorrFull_Norm_8u32f_C4R</td>
<td>ippiCopy_8u_C4C1R</td>
</tr>
<tr>
<td></td>
<td>+ippiCrossCorrNorm_8u32f_C1R</td>
</tr>
<tr>
<td></td>
<td>+algType=ippiROIFull</td>
</tr>
<tr>
<td>ippiCrossCorrFull_Norm_8u_AC4RSfs</td>
<td>ippiCopy_8u_C4C1R</td>
</tr>
<tr>
<td></td>
<td>+ippiCrossCorrNorm_8u_C1RSfs</td>
</tr>
<tr>
<td></td>
<td>+algType=ippiROIFull</td>
</tr>
<tr>
<td>Removed from 9.0</td>
<td>Substitution or Workaround</td>
</tr>
<tr>
<td>------------------------------------------------------</td>
<td>----------------------------------------------------------------</td>
</tr>
<tr>
<td>ippicrosscorrfull_norm_8u_c1rsfs</td>
<td>ippicrosscorr_norm_8u_c1rsfs +algType=ippiroifull</td>
</tr>
<tr>
<td>ippicrosscorrfull_norm_8u_c3rsfs</td>
<td>ippicopy_8u_c3c1r +ippicrosscorrnorm_8u_c1rsfs +algType=ippiroifull</td>
</tr>
<tr>
<td>ippicrosscorrfull_norm_8u_c4rsfs</td>
<td>ippicopy_8u_c4c1r +ippicrosscorrnorm_8u_c1rsfs +algType=ippiroifull</td>
</tr>
<tr>
<td>ippicrosscorrsame_normlevel_16u32f_ac4r</td>
<td>ippicopy_16u_c4c1r +ippicrosscorrnorm_16u32f_c1r +algType=ippiroisame</td>
</tr>
<tr>
<td>ippicrosscorrsame_normlevel_16u32f_c1r</td>
<td>ippicrosscorrnorm_16u32f_c1r +algType=ippiroisame</td>
</tr>
<tr>
<td>ippicrosscorrsame_normlevel_16u32f_c3r</td>
<td>ippicopy_16u_c3c1r +ippicrosscorrnorm_16u32f_c1r +algType=ippiroisame</td>
</tr>
<tr>
<td>ippicrosscorrsame_normlevel_16u32f_c4r</td>
<td>ippicopy_16u_c4c1r +ippicrosscorrnorm_16u32f_c1r +algType=ippiroisame</td>
</tr>
<tr>
<td>ippicrosscorrsame_normlevel_32f_ac4r</td>
<td>ippicopy_32f_c4c1r +ippicrosscorrnorm_32f_c1r +algType=ippiroisame</td>
</tr>
<tr>
<td>ippicrosscorrsame_normlevel_32f_c1r</td>
<td>ippicrosscorrnorm_32f_c1r +algType=ippiroisame</td>
</tr>
<tr>
<td>ippicrosscorrsame_normlevel_32f_c3r</td>
<td>ippicopy_32f_c3c1r +ippicrosscorrnorm_32f_c1r +algType=ippiroisame</td>
</tr>
<tr>
<td>ippicrosscorrsame_normlevel_32f_c4r</td>
<td>ippicopy_32f_c4c1r +ippicrosscorrnorm_32f_c1r +algType=ippiroisame</td>
</tr>
<tr>
<td>ippicrosscorrsame_normlevel_8s32f_ac4r</td>
<td>ippiconvert_8s32f+ippicopy_32f_c4c1r +ippicrosscorrnorm_32f_c1r +algType=ippiroisame</td>
</tr>
<tr>
<td>ippicrosscorrsame_normlevel_8s32f_c1r</td>
<td>ippiconvert_8s32f +ippicrosscorrnorm_32f_c1r +algType=ippiroisame</td>
</tr>
<tr>
<td>ippicrosscorrsame_normlevel_8s32f_c3r</td>
<td>ippiconvert_8s32f+ippicopy_32f_c3c1r +ippicrosscorrnorm_32f_c1r +algType=ippiroisame</td>
</tr>
<tr>
<td>Removed from 9.0</td>
<td>Substitution or Workaround</td>
</tr>
<tr>
<td>------------------</td>
<td>-----------------------------</td>
</tr>
<tr>
<td>ippiCrossCorrSame_NormLevel_8s32f_C4R</td>
<td>ippiConvert_8s32f+ippiCopy_32f_C4C1R +ippiCrossCorrNorm_32f_C1R +algType=ippiROISameippiNormCoefficient</td>
</tr>
<tr>
<td>ippiCrossCorrSame_NormLevel_8u32f_AC4R</td>
<td>ippiCopy_8u_C4C1R +ippiCrossCorrNorm_8u32f_C1R +algType=ippiROISameippiNormCoefficient</td>
</tr>
<tr>
<td>ippiCrossCorrSame_NormLevel_8u32f_C1R</td>
<td>ippiCrossCorrNorm_8u32f_C1R +algType=ippiROISameippiNormCoefficient</td>
</tr>
<tr>
<td>ippiCrossCorrSame_NormLevel_8u32f_C3R</td>
<td>ippiCopy_8u_C3C1R +ippiCrossCorrNorm_8u32f_C1R +algType=ippiROISameippiNormCoefficient</td>
</tr>
<tr>
<td>ippiCrossCorrSame_NormLevel_8u32f_C4R</td>
<td>ippiCopy_8u_C4C1R +ippiCrossCorrNorm_8u32f_C1R +algType=ippiROISameippiNormCoefficient</td>
</tr>
<tr>
<td>ippiCrossCorrSame_NormLevel_8u_AC4RSfs</td>
<td>ippiCopy_8u_C4C1R +ippiCrossCorrNorm_8u_C1RSfs +algType=ippiROISameippiNormCoefficient</td>
</tr>
<tr>
<td>ippiCrossCorrSame_NormLevel_8u_C1RSfs</td>
<td>ippiCrossCorrNorm_8u_C1RSfs +algType=ippiROISameippiNormCoefficient</td>
</tr>
<tr>
<td>ippiCrossCorrSame_NormLevel_8u_C3RSfs</td>
<td>ippiCopy_8u_C3C1R +ippiCrossCorrNorm_8u_C1RSfs +algType=ippiROISameippiNormCoefficient</td>
</tr>
<tr>
<td>ippiCrossCorrSame_NormLevel_8u_C4RSfs</td>
<td>ippiCopy_8u_C4C1R +ippiCrossCorrNorm_8u_C1RSfs +algType=ippiROISameippiNormCoefficient</td>
</tr>
<tr>
<td>ippiCrossCorrSame_Norm_16u32f_AC4R</td>
<td>ippiCopy_16u_C4C1R +ippiCrossCorrNorm_16u32f_C1R +algType=ippiROISameippiNorm</td>
</tr>
<tr>
<td>ippiCrossCorrSame_Norm_16u32f_C1R</td>
<td>ippiCrossCorrNorm_16u32f_C1R +algType=ippiROISameippiNorm</td>
</tr>
<tr>
<td>ippiCrossCorrSame_Norm_16u32f_C3R</td>
<td>ippiCopy_16u_C3C1R +ippiCrossCorrNorm_16u32f_C1R +algType=ippiROISameippiNorm</td>
</tr>
<tr>
<td>ippiCrossCorrSame_Norm_16u32f_C4R</td>
<td>ippiCopy_16u_C4C1R +ippiCrossCorrNorm_16u32f_C1R +algType=ippiROISameippiNorm</td>
</tr>
<tr>
<td>ippiCrossCorrSame_Norm_32f_AC4R</td>
<td>ippiCopy_32f_C4C1R +ippiCrossCorrNorm_32f_C1R +algType=ippiROISameippiNorm</td>
</tr>
<tr>
<td>Removed from 9.0</td>
<td>Substitution or Workaround</td>
</tr>
<tr>
<td>-----------------</td>
<td>-----------------------------</td>
</tr>
</tbody>
</table>
| ippiCrossCorrSame_Norm_32f_C1R | ippiCrossCorrNorm_32f_C1R  
+algType=ippiROISameippiNorm |
| ippiCrossCorrSame_Norm_32f_C3R | ippiCopy_32f_C3C1R  
+ippiCrossCorrNorm_32f_C1R  
+algType=ippiROISameippiNorm |
| ippiCrossCorrSame_Norm_32f_C4R | ippiCopy_32f_C4C1R  
+ippiCrossCorrNorm_32f_C1R  
+algType=ippiROISameippiNorm |
| ippiCrossCorrSame_Norm_8s32f_AC4R | ippiConvert_8s32fippiCopy_32f_C4C1R  
+ippiCrossCorrNorm_32f_C1R  
+algType=ippiROISameippiNorm |
| ippiCrossCorrSame_Norm_8s32f_C1R | ippiConvert_8s32fippiCrossCorrNorm_32f_C1R  
+algType=ippiROISameippiNorm |
| ippiCrossCorrSame_Norm_8s32f_C3R | ippiConvert_8s32fippiCopy_32f_C3C1R  
+ippiCrossCorrNorm_32f_C1R  
+algType=ippiROISameippiNorm |
| ippiCrossCorrSame_Norm_8s32f_C4R | ippiConvert_8s32fippiCopy_32f_C4C1R  
+ippiCrossCorrNorm_32f_C1R  
+algType=ippiROISameippiNorm |
| ippiCrossCorrSame_Norm_8u32f_AC4R | ippiCopy_8u_C4C1R  
+ippiCrossCorrNorm_8u32f_C1R  
+algType=ippiROISameippiNorm |
| ippiCrossCorrSame_Norm_8u32f_C1R | ippiCrossCorrNorm_8u32f_C1R  
+algType=ippiROISameippiNorm |
| ippiCrossCorrSame_Norm_8u32f_C3R | ippiCopy_8u_C3C1R  
+ippiCrossCorrNorm_8u32f_C1R  
+algType=ippiROISameippiNorm |
| ippiCrossCorrSame_Norm_8u32f_C4R | ippiCopy_8u_C4C1R  
+ippiCrossCorrNorm_8u32f_C1R  
+algType=ippiROISameippiNorm |
| ippiCrossCorrSame_Norm_8u_AC4RSfs | ippiCopy_8u_C4C1R  
+ippiCrossCorrNorm_8u_C1RSfs  
+algType=ippiROISameippiNorm |
| ippiCrossCorrSame_Norm_8u_C1RSfs | ippiCrossCorrNorm_8u_C1RSfs  
+algType=ippiROISameippiNorm |
| ippiCrossCorrSame_Norm_8u_C3RSfs | ippiCopy_8u_C3C1R  
+ippiCrossCorrNorm_8u_C1RSfs  
+algType=ippiROISameippiNorm |
### Appendix C: Removed Functions for Image and Video Processing

<table>
<thead>
<tr>
<th>Removed from 9.0</th>
<th>Substitution or Workaround</th>
</tr>
</thead>
<tbody>
<tr>
<td>ippiCrossCorrSame_Norm_8u_C4RSfs</td>
<td>ippiCopy_8u_C4C1R</td>
</tr>
<tr>
<td></td>
<td>+ippiCrossCorrNorm_8u_C1RSfs</td>
</tr>
<tr>
<td></td>
<td>+algType=ippiROISameippiNorm</td>
</tr>
<tr>
<td>ippiCrossCorrValid_16u32f_C1R</td>
<td>ippiCrossCorrNorm_16u32f_C1R</td>
</tr>
<tr>
<td></td>
<td>+algType=ippiROIValidippiNormNone</td>
</tr>
<tr>
<td>ippiCrossCorrValid_32f_C1R</td>
<td>ippiCrossCorrNorm_32f_C1R</td>
</tr>
<tr>
<td></td>
<td>+algType=ippiROIValidippiNormNone</td>
</tr>
<tr>
<td>ippiCrossCorrValid_8s32f_C1R</td>
<td>ippiConvert_8s32f</td>
</tr>
<tr>
<td></td>
<td>+ippiCrossCorrNorm_32f_C1R</td>
</tr>
<tr>
<td></td>
<td>+algType=ippiROIValidippiNormNone</td>
</tr>
<tr>
<td>ippiCrossCorrValid_8u32f_C1R</td>
<td>ippiCrossCorrNorm_8u32f_C1R</td>
</tr>
<tr>
<td></td>
<td>+algType=ippiROIValidippiNormNone</td>
</tr>
<tr>
<td>ippiCrossCorrValid_NormLevel_16u32f_AC4R</td>
<td>ippiCopy_16u_C4C1R</td>
</tr>
<tr>
<td></td>
<td>+ippiCrossCorrNorm_16u32f_C1R</td>
</tr>
<tr>
<td></td>
<td>+algType=ippiROIValidippiNormCoefficient</td>
</tr>
<tr>
<td>ippiCrossCorrValid_NormLevel_16u32f_C1R</td>
<td>ippiCrossCorrNorm_16u32f_C1R</td>
</tr>
<tr>
<td></td>
<td>+algType=ippiROIValidippiNormCoefficient</td>
</tr>
<tr>
<td>ippiCrossCorrValid_NormLevel_16u32f_C3R</td>
<td>ippiCopy_16u_C3C1R</td>
</tr>
<tr>
<td></td>
<td>+ippiCrossCorrNorm_16u32f_C1R</td>
</tr>
<tr>
<td></td>
<td>+algType=ippiROIValidippiNormCoefficient</td>
</tr>
<tr>
<td>ippiCrossCorrValid_NormLevel_16u32f_C4R</td>
<td>ippiCopy_16u_C4C1R</td>
</tr>
<tr>
<td></td>
<td>+ippiCrossCorrNorm_16u32f_C1R</td>
</tr>
<tr>
<td></td>
<td>+algType=ippiROIValidippiNormCoefficient</td>
</tr>
<tr>
<td>ippiCrossCorrValid_NormLevel_32f_AC4R</td>
<td>ippiCopy_32f_C4C1R</td>
</tr>
<tr>
<td></td>
<td>+ippiCrossCorrNorm_32f_C1R</td>
</tr>
<tr>
<td></td>
<td>+algType=ippiROIValidippiNormCoefficient</td>
</tr>
<tr>
<td>ippiCrossCorrValid_NormLevel_32f_C1R</td>
<td>ippiCrossCorrNorm_32f_C1R</td>
</tr>
<tr>
<td></td>
<td>+algType=ippiROIValidippiNormCoefficient</td>
</tr>
<tr>
<td>ippiCrossCorrValid_NormLevel_32f_C3R</td>
<td>ippiCopy_32f_C3C1R</td>
</tr>
<tr>
<td></td>
<td>+ippiCrossCorrNorm_32f_C1R</td>
</tr>
<tr>
<td></td>
<td>+algType=ippiROIValidippiNormCoefficient</td>
</tr>
<tr>
<td>ippiCrossCorrValid_NormLevel_32f_C4R</td>
<td>ippiCopy_32f_C4C1R</td>
</tr>
<tr>
<td></td>
<td>+ippiCrossCorrNorm_32f_C1R</td>
</tr>
<tr>
<td></td>
<td>+algType=ippiROIValidippiNormCoefficient</td>
</tr>
<tr>
<td>ippiCrossCorrValid_NormLevel_64f_C1R</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiCrossCorrValid_NormLevel_8s32f_AC4R</td>
<td>ippiConvert_8s32f+ippiCopy_32f_C4C1R</td>
</tr>
<tr>
<td></td>
<td>+ippiCrossCorrNorm_32f_C1R</td>
</tr>
<tr>
<td></td>
<td>+algType=ippiROIValidippiNormCoefficient</td>
</tr>
<tr>
<td>Removed from 9.0</td>
<td>Substitution or Workaround</td>
</tr>
<tr>
<td>------------------------------------------------------</td>
<td>-------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>ippiCrossCorrValid_NormLevel_8s32f_C1R</td>
<td>ippiConvert_8s32f +ippiCrossCorrNorm_32f_C1R +algType=ippiROIValid</td>
</tr>
<tr>
<td>ippiCrossCorrValid_NormLevel_8s32f_C3R</td>
<td>ippiConvert_8s32f +ippiCopy_32f_C3C1R +ippiCrossCorrNorm_32f_C1R +algType=ippiROIValid</td>
</tr>
<tr>
<td>ippiCrossCorrValid_NormLevel_8s32f_C4R</td>
<td>ippiConvert_8s32f +ippiCopy_32f_C4C1R +ippiCrossCorrNorm_32f_C1R +algType=ippiROIValid</td>
</tr>
<tr>
<td>ippiCrossCorrValid_NormLevel_8u32f_AC4R</td>
<td>ippiCopy_8u_C4C1R +ippiCrossCorrNorm_8u32f_C1R +algType=ippiROIValid</td>
</tr>
<tr>
<td>ippiCrossCorrValid_NormLevel_8u32f_C1R</td>
<td>+ippiCrossCorrNorm_8u32f_C1R +algType=ippiROIValid</td>
</tr>
<tr>
<td>ippiCrossCorrValid_NormLevel_8u32f_C3R</td>
<td>+ippiCopy_8u_C3C1R +ippiCrossCorrNorm_8u32f_C1R +algType=ippiROIValid</td>
</tr>
<tr>
<td>ippiCrossCorrValid_NormLevel_8u32f_C4R</td>
<td>+ippiCopy_8u_C4C1R +ippiCrossCorrNorm_8u32f_C1R +algType=ippiROIValid</td>
</tr>
<tr>
<td>ippiCrossCorrValid_NormLevel_8u_AC4RSfs</td>
<td>+ippiCrossCorrNorm_8u_C1RSfs +algType=ippiROIValid</td>
</tr>
<tr>
<td>ippiCrossCorrValid_NormLevel_8u_C1RSfs</td>
<td>+ippiCrossCorrNorm_8u_C1RSfs +algType=ippiROIValid</td>
</tr>
<tr>
<td>ippiCrossCorrValid_NormLevel_8u_C3RSfs</td>
<td>+ippiCopy_8u_C3C1R +ippiCrossCorrNorm_8u_C1RSfs +algType=ippiROIValid</td>
</tr>
<tr>
<td>ippiCrossCorrValid_NormLevel_8u_C4RSfs</td>
<td>+ippiCopy_8u_C4C1R +ippiCrossCorrNorm_8u_C1RSfs +algType=ippiROIValid</td>
</tr>
<tr>
<td>ippiCrossCorrValid_Norm_16u32f_AC4R</td>
<td>+ippiCopy_16u_C4C1R +ippiCrossCorrNorm_16u32f_C1R +algType=ippiROIValid</td>
</tr>
<tr>
<td>ippiCrossCorrValid_Norm_16u32f_C1R</td>
<td>+ippiCrossCorrNorm_16u32f_C1R +algType=ippiROIValid</td>
</tr>
<tr>
<td>ippiCrossCorrValid_Norm_16u32f_C3R</td>
<td>+ippiCopy_16u_C3C1R +ippiCrossCorrNorm_16u32f_C1R +algType=ippiROIValid</td>
</tr>
</tbody>
</table>
### Removed from 9.0

<table>
<thead>
<tr>
<th>Function</th>
<th>Substitution or Workaround</th>
</tr>
</thead>
<tbody>
<tr>
<td>ippicrosscorrvalid_norm_16u32f_C4R</td>
<td>ippicopy_16u_C4C1R</td>
</tr>
<tr>
<td></td>
<td>+ ippicrosscorrnorm_16u32f_C1R</td>
</tr>
<tr>
<td></td>
<td>+ algType=ippiROIValu</td>
</tr>
<tr>
<td>ippicrosscorrvalid_norm_32f_ac4r</td>
<td>ippicopy_32f_C4C1R</td>
</tr>
<tr>
<td></td>
<td>+ ippicrosscorrnorm_32f_C1R</td>
</tr>
<tr>
<td></td>
<td>+ algType=ippiROIValu</td>
</tr>
<tr>
<td>ippicrosscorrvalid_norm_32f_C1r</td>
<td>ippicopy_32f_C3C1r</td>
</tr>
<tr>
<td></td>
<td>+ ippicrosscorrnorm_32f_C1r</td>
</tr>
<tr>
<td></td>
<td>+ algType=ippiROIValu</td>
</tr>
<tr>
<td>ippicrosscorrvalid_norm_32f_C4r</td>
<td>ippicopy_32f_C4C1r</td>
</tr>
<tr>
<td></td>
<td>+ ippicrosscorrnorm_32f_C1r</td>
</tr>
<tr>
<td></td>
<td>+ algType=ippiROIValu</td>
</tr>
<tr>
<td>ippicrosscorrvalid_norm_8s32f_ac4r</td>
<td>ippicopy_32f_C4C1r</td>
</tr>
<tr>
<td></td>
<td>+ ippicrosscorrnorm_32f_C1r</td>
</tr>
<tr>
<td></td>
<td>+ algType=ippiROIValu</td>
</tr>
<tr>
<td>ippicrosscorrvalid_norm_8s32f_c1r</td>
<td>ippicopy_32f_C3C1r</td>
</tr>
<tr>
<td></td>
<td>+ ippicrosscorrnorm_32f_C1r</td>
</tr>
<tr>
<td></td>
<td>+ algType=ippiROIValu</td>
</tr>
<tr>
<td>ippicrosscorrvalid_norm_8s32f_c4r</td>
<td>ippicopy_32f_C4C1r</td>
</tr>
<tr>
<td></td>
<td>+ ippicrosscorrnorm_32f_C1r</td>
</tr>
<tr>
<td></td>
<td>+ algType=ippiROIValu</td>
</tr>
<tr>
<td>ippicrosscorrvalid_norm_8u32f_ac4r</td>
<td>ippicopy_8u_C4C1r</td>
</tr>
<tr>
<td></td>
<td>+ ippicrosscorrnorm_8u32f_C1r</td>
</tr>
<tr>
<td></td>
<td>+ algType=ippiROIValu</td>
</tr>
<tr>
<td>ippicrosscorrvalid_norm_8u32f_c1r</td>
<td>ippicopy_8u_C3C1r</td>
</tr>
<tr>
<td></td>
<td>+ ippicrosscorrnorm_8u32f_C1r</td>
</tr>
<tr>
<td></td>
<td>+ algType=ippiROIValu</td>
</tr>
<tr>
<td>ippicrosscorrvalid_norm_8u32f_c3r</td>
<td>ippicopy_8u_C4C1r</td>
</tr>
<tr>
<td></td>
<td>+ ippicrosscorrnorm_8u32f_C1r</td>
</tr>
<tr>
<td></td>
<td>+ algType=ippiROIValu</td>
</tr>
<tr>
<td>ippicrosscorrvalid_norm_8u32f_c4r</td>
<td>ippicopy_8u_C4C1r</td>
</tr>
<tr>
<td></td>
<td>+ ippicrosscorrnorm_8u32f_C1r</td>
</tr>
<tr>
<td></td>
<td>+ algType=ippiROIValu</td>
</tr>
<tr>
<td>ippicrosscorrvalid_norm_8u_ac4rsfs</td>
<td>ippicopy_8u_C4C1r</td>
</tr>
<tr>
<td></td>
<td>+ ippicrosscorrnorm_8u_C1R</td>
</tr>
<tr>
<td></td>
<td>+ algType=ippiROIValu</td>
</tr>
<tr>
<td>Removed from 9.0</td>
<td>Substitution or Workaround</td>
</tr>
<tr>
<td>--------------------------------------------------------------------------------</td>
<td>-------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>ippiCrossCorrValid_Norm_8u_C1RSfs</td>
<td>ippiCrossCorrNorm_8u_C1RSfs +algType=ippiROIValid</td>
</tr>
<tr>
<td>ippiCrossCorrValid_Norm_8u_C3RSfs</td>
<td>ippiCopy_8u_C3C1R +ippiCrossCorrNorm_8u_C1RSfs +algType=ippiROIValid</td>
</tr>
<tr>
<td>ippiCrossCorrValid_Norm_8u_C4RSfs</td>
<td>ippiCopy_8u_C4C1R +ippiCrossCorrNorm_8u_C1RSfs +algType=ippiROIValid</td>
</tr>
<tr>
<td>ippiDCTFwdFree_32f</td>
<td>ippiFree</td>
</tr>
<tr>
<td>ippiDCTFwdGetBufSize_32f</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiDCTFwdInitAlloc_32f</td>
<td>ippiDCTGetSize+ippiMalloc+ippiDCTInit</td>
</tr>
<tr>
<td>ippiDCTInvFree_32f</td>
<td>ippiFree</td>
</tr>
<tr>
<td>ippiDCTInvGetBufSize_32f</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiDCTInvInitAlloc_32f</td>
<td>ippiDCTGetSize+ippiMalloc+ippiDCTInit</td>
</tr>
<tr>
<td>ippiDFTFree_C_32fc</td>
<td>ippiFree</td>
</tr>
<tr>
<td>ippiDFTFree_R_32f</td>
<td>ippiFree</td>
</tr>
<tr>
<td>ippiDFTFree_R_32s</td>
<td></td>
</tr>
<tr>
<td>ippiDFTFwd_RToPack_8u32s_Ac4RSfs</td>
<td>ippiConvert_8u32f+ippiDFTFwd_RToPack_32f +ippiDFTFwd_RToPack_32f</td>
</tr>
<tr>
<td>ippiDFTFwd_RToPack_8u32s_C1RSfs</td>
<td>ippiConvert_8u32f+ippiDFTFwd_RToPack_32f +ippiDFTFwd_RToPack_32f</td>
</tr>
<tr>
<td>ippiDFTFwd_RToPack_8u32s_C3RSfs</td>
<td>ippiConvert_8u32f+ippiDFTFwd_RToPack_32f +ippiDFTFwd_RToPack_32f</td>
</tr>
<tr>
<td>ippiDFTFwd_RToPack_8u32s_C4RSfs</td>
<td></td>
</tr>
<tr>
<td>ippiDFTGetBufSize_C_32fc</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiDFTGetBufSize_R_32f</td>
<td></td>
</tr>
<tr>
<td>ippiDFTGetBufSize_R_32s</td>
<td></td>
</tr>
<tr>
<td>ippiDFTInitAlloc_C_32fc</td>
<td></td>
</tr>
<tr>
<td>ippiDFTInitAlloc_R_32f</td>
<td></td>
</tr>
<tr>
<td>ippiDFTInitAlloc_R_32s</td>
<td></td>
</tr>
<tr>
<td>ippiDFTInv_PackToR_32s8u_Ac4RSfs</td>
<td></td>
</tr>
<tr>
<td>ippiDFTInv_PackToR_32s8u_C1RSfs</td>
<td></td>
</tr>
<tr>
<td>ippiDFTInv_PackToR_32s8u_C3RSfs</td>
<td></td>
</tr>
<tr>
<td>ippiDFTInv_PackToR_32s8u_C4RSfs</td>
<td></td>
</tr>
<tr>
<td>Removed from 9.0</td>
<td>Substitution or Workaround</td>
</tr>
<tr>
<td>------------------------------------------------------</td>
<td>---------------------------------------</td>
</tr>
<tr>
<td>ippiDeconvFFTFree_32f_C1R</td>
<td>ippFree</td>
</tr>
<tr>
<td>ippiDeconvFFTFree_32f_C3R</td>
<td>ippFree</td>
</tr>
<tr>
<td>ippiDeconvFFTInitAlloc_32f_C1R</td>
<td>ippiDeconvFFTGetSize_32f+ippMalloc_8u</td>
</tr>
<tr>
<td></td>
<td>+ippiDeconvFFTInit_32f_C1R</td>
</tr>
<tr>
<td>ippiDeconvFFTInitAlloc_32f_C3R</td>
<td>ippiDeconvFFTGetSize_32f+ippMalloc_8u</td>
</tr>
<tr>
<td></td>
<td>+ippiDeconvFFTInit_32f_C3R</td>
</tr>
<tr>
<td>ippiDeconvLRFree_32f_C1R</td>
<td>ippFree</td>
</tr>
<tr>
<td>ippiDeconvLRFree_32f_C3R</td>
<td>ippFree</td>
</tr>
<tr>
<td>ippiDeconvLRInitAlloc_32f_C1R</td>
<td>ippiDeconvLRGetSize_32f+ippMalloc_8u</td>
</tr>
<tr>
<td></td>
<td>+ippiDeconvLRInit_32f_C1R</td>
</tr>
<tr>
<td>ippiDeconvLRInitAlloc_32f_C3R</td>
<td>ippiDeconvLRGetSize_32f+ippMalloc_8u</td>
</tr>
<tr>
<td></td>
<td>+ippiDeconvLRInit_32f_C3R</td>
</tr>
<tr>
<td>ippiDilate3x3_16u_AC4IR</td>
<td>Use not-in-place flavor</td>
</tr>
<tr>
<td>ippiDilate3x3_16u_AC4R</td>
<td></td>
</tr>
<tr>
<td>ippiDilate3x3_16u_C1IR</td>
<td>Use not-in-place flavor</td>
</tr>
<tr>
<td>ippiDilate3x3_16u_C1R</td>
<td></td>
</tr>
<tr>
<td>ippiDilate3x3_16u_C3IR</td>
<td>Use not-in-place flavor</td>
</tr>
<tr>
<td>ippiDilate3x3_16u_C3R</td>
<td></td>
</tr>
<tr>
<td>ippiDilate3x3_16u_C4IR</td>
<td>Use not-in-place flavor</td>
</tr>
<tr>
<td>ippiDilate3x3_16u_C4R</td>
<td></td>
</tr>
<tr>
<td>ippiDilate3x3_32f_AC4IR</td>
<td>Use not-in-place flavor</td>
</tr>
<tr>
<td>ippiDilate3x3_32f_AC4R</td>
<td></td>
</tr>
<tr>
<td>ippiDilate3x3_32f_C1IR</td>
<td>Use not-in-place flavor</td>
</tr>
<tr>
<td>ippiDilate3x3_32f_C1R</td>
<td></td>
</tr>
<tr>
<td>ippiDilate3x3_32f_C3IR</td>
<td>Use not-in-place flavor</td>
</tr>
<tr>
<td>ippiDilate3x3_32f_C3R</td>
<td></td>
</tr>
<tr>
<td>ippiDilate3x3_32f_C4IR</td>
<td>Use not-in-place flavor</td>
</tr>
<tr>
<td>ippiDilate3x3_32f_C4R</td>
<td></td>
</tr>
<tr>
<td>ippiDilate3x3_8u_AC4IR</td>
<td>Use not-in-place flavor</td>
</tr>
<tr>
<td>ippiDilate3x3_8u_AC4R</td>
<td></td>
</tr>
<tr>
<td>ippiDilate3x3_8u_C1IR</td>
<td>Use not-in-place flavor</td>
</tr>
<tr>
<td>ippiDilate3x3_8u_C1R</td>
<td></td>
</tr>
<tr>
<td>ippiDilate3x3_8u_C3IR</td>
<td>Use not-in-place flavor</td>
</tr>
<tr>
<td>ippiDilate3x3_8u_C3R</td>
<td></td>
</tr>
<tr>
<td>ippiDilate3x3_8u_C4IR</td>
<td>Use not-in-place flavor</td>
</tr>
<tr>
<td>ippiDilate3x3_8u_C4R</td>
<td></td>
</tr>
<tr>
<td>Removed from 9.0</td>
<td>Substitution or Workaround</td>
</tr>
<tr>
<td>----------------------------------</td>
<td>-----------------------------------------------------------------</td>
</tr>
<tr>
<td>ippiDilate3x3_8u_C3IR</td>
<td>Use not-in-place flavor</td>
</tr>
<tr>
<td>ippiDilate3x3_8u_C3R</td>
<td>ippiDilateBorder_8u_C3R with 3x3 mask</td>
</tr>
<tr>
<td>ippiDilate3x3_8u_C4IR</td>
<td>Use not-in-place flavor</td>
</tr>
<tr>
<td>ippiDilate3x3_8u_C4R</td>
<td>ippiDilateBorder_8u_C4R with 3x3 mask</td>
</tr>
<tr>
<td>ippiDilate_16u_AC4IR</td>
<td>Use not-in-place flavor</td>
</tr>
<tr>
<td>ippiDilate_16u_AC4R</td>
<td>ippiDilateBorder_16u_C4R</td>
</tr>
<tr>
<td>ippiDilate_16u_C1IR</td>
<td>Use not-in-place flavor</td>
</tr>
<tr>
<td>ippiDilate_16u_C1R</td>
<td>ippiDilateBorder_16u_C1R</td>
</tr>
<tr>
<td>ippiDilate_16u_C3IR</td>
<td>Use not-in-place flavor</td>
</tr>
<tr>
<td>ippiDilate_16u_C3R</td>
<td>ippiDilateBorder_16u_C3R</td>
</tr>
<tr>
<td>ippiDilate_16u_C4R</td>
<td>Use not-in-place flavor</td>
</tr>
<tr>
<td>ippiDilate_16u_C4R</td>
<td>ippiDilateBorder_16u_C4R</td>
</tr>
<tr>
<td>ippiDilate_32f_AC4IR</td>
<td>Use not-in-place flavor</td>
</tr>
<tr>
<td>ippiDilate_32f_AC4R</td>
<td>ippiDilateBorder_32f_C4R</td>
</tr>
<tr>
<td>ippiDilate_32f_C1IR</td>
<td>Use not-in-place flavor</td>
</tr>
<tr>
<td>ippiDilate_32f_C1R</td>
<td>ippiDilateBorder_32f_C1R</td>
</tr>
<tr>
<td>ippiDilate_32f_C3IR</td>
<td>Use not-in-place flavor</td>
</tr>
<tr>
<td>ippiDilate_32f_C3R</td>
<td>ippiDilateBorder_32f_C3R</td>
</tr>
<tr>
<td>ippiDilate_32f_C4R</td>
<td>Use not-in-place flavor</td>
</tr>
<tr>
<td>ippiDilate_32f_C4R</td>
<td>ippiDilateBorder_32f_C4R</td>
</tr>
<tr>
<td>ippiDilate_8u_AC4IR</td>
<td>Use not-in-place flavor</td>
</tr>
<tr>
<td>ippiDilate_8u_AC4R</td>
<td>ippiDilateBorder_8u_C4R</td>
</tr>
<tr>
<td>ippiDilate_8u_C1IR</td>
<td>Use not-in-place flavor</td>
</tr>
<tr>
<td>ippiDilate_8u_C1R</td>
<td>ippiDilateBorder_8u_C1R</td>
</tr>
<tr>
<td>ippiDilate_8u_C3IR</td>
<td>Use not-in-place flavor</td>
</tr>
<tr>
<td>ippiDilate_8u_C3R</td>
<td>ippiDilateBorder_8u_C3R</td>
</tr>
<tr>
<td>ippiDilate_8u_C4R</td>
<td>Use not-in-place flavor</td>
</tr>
<tr>
<td>ippiDilate_8u_C4R</td>
<td>ippiDilateBorder_8u_C4R</td>
</tr>
<tr>
<td>ippiDivC_16sc_AC4IRSfs</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiDivC_16sc_AC4RSfs</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiDivC_16sc_C1IRSfs</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiDivC_16sc_C1RSfs</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiDivC_16sc_C3IRSfs</td>
<td>N/A</td>
</tr>
<tr>
<td>Removed from 9.0</td>
<td>Substitution or Workaround</td>
</tr>
<tr>
<td>------------------</td>
<td>-----------------------------</td>
</tr>
<tr>
<td>ippiDivC_16sc_C3RSfs</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiDivC_32fc_AC4IR</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiDivC_32fc_AC4R</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiDivC_32fc_C1IR</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiDivC_32fc_C1R</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiDivC_32fc_C3IR</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiDivC_32fc_C3R</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiDivC_32fc_C3IRSfs</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiDivC_32sc_AC4IRSfs</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiDivC_32sc_AC4RSfs</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiDivC_32sc_C1IRSfs</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiDivC_32sc_C1RSfs</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiDivC_32sc_C3IRSfs</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiDivC_32sc_C3RSfs</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiDiv_16sc_AC4IRSfs</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiDiv_16sc_AC4RSfs</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiDiv_16sc_C1IRSfs</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiDiv_16sc_C1RSfs</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiDiv_16sc_C3IRSfs</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiDiv_16sc_C3RSfs</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiDiv_32fc_AC4IR</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiDiv_32fc_AC4R</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiDiv_32fc_C1IR</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiDiv_32fc_C1R</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiDiv_32fc_C3IR</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiDiv_32fc_C3R</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiDiv_32fc_C3IRSfs</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiDiv_32sc_AC4IRSfs</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiDiv_32sc_AC4RSfs</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiDiv_32sc_C1IRSfs</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiDiv_32sc_C1RSfs</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiDiv_32sc_C3IRSfs</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiDiv_32sc_C3RSfs</td>
<td>N/A</td>
</tr>
<tr>
<td>Removed from 9.0</td>
<td>Substitution or Workaround</td>
</tr>
<tr>
<td>--------------------------------------</td>
<td>----------------------------------------------------------------</td>
</tr>
<tr>
<td>ippiDotProd_8s64f_AC4R</td>
<td>ippiConvert_8s16s+ippiDotProd_16s64f</td>
</tr>
<tr>
<td>ippiDotProd_8s64f_C1R</td>
<td>ippiConvert_8s16s+ippiDotProd_16s64f</td>
</tr>
<tr>
<td>ippiDotProd_8s64f_C3R</td>
<td>ippiConvert_8s16s+ippiDotProd_16s64f</td>
</tr>
<tr>
<td>ippiDotProd_8s64f_C4R</td>
<td>ippiConvert_8s16s+ippiDotProd_16s64f</td>
</tr>
<tr>
<td>ippiErode3x3_16u_AC4IR</td>
<td>Use not-in-place flavor</td>
</tr>
<tr>
<td>ippiErode3x3_16u_AC4R</td>
<td>ippiErodeBorder_16u_C4R with 3x3 mask</td>
</tr>
<tr>
<td>ippiErode3x3_16u_C1IR</td>
<td>Use not-in-place flavor</td>
</tr>
<tr>
<td>ippiErode3x3_16u_C1R</td>
<td>ippiErodeBorder_16u_C1R with 3x3 mask</td>
</tr>
<tr>
<td>ippiErode3x3_16u_C3R</td>
<td>Use not-in-place flavor</td>
</tr>
<tr>
<td>ippiErode3x3_16u_C3R</td>
<td>ippiErodeBorder_16u_C3R with 3x3 mask</td>
</tr>
<tr>
<td>ippiErode3x3_16u_C4R</td>
<td>Use not-in-place flavor</td>
</tr>
<tr>
<td>ippiErode3x3_32f_AC4IR</td>
<td>Use not-in-place flavor</td>
</tr>
<tr>
<td>ippiErode3x3_32f_AC4R</td>
<td>ippiErodeBorder_32f_C4R with 3x3 mask</td>
</tr>
<tr>
<td>ippiErode3x3_32f_C1IR</td>
<td>Use not-in-place flavor</td>
</tr>
<tr>
<td>ippiErode3x3_32f_C1R</td>
<td>ippiErodeBorder_32f_C1R with 3x3 mask</td>
</tr>
<tr>
<td>ippiErode3x3_32f_C3IR</td>
<td>Use not-in-place flavor</td>
</tr>
<tr>
<td>ippiErode3x3_32f_C3R</td>
<td>ippiErodeBorder_32f_C3R with 3x3 mask</td>
</tr>
<tr>
<td>ippiErode3x3_32f_C4IR</td>
<td>Use not-in-place flavor</td>
</tr>
<tr>
<td>ippiErode3x3_32f_C4R</td>
<td>ippiErodeBorder_32f_C4R with 3x3 mask</td>
</tr>
<tr>
<td>ippiErode3x3_8u_AC4IR</td>
<td>Use not-in-place flavor</td>
</tr>
<tr>
<td>ippiErode3x3_8u_AC4R</td>
<td>ippiErodeBorder_8u_C4R with 3x3 mask</td>
</tr>
<tr>
<td>ippiErode3x3_8u_C1IR</td>
<td>Use not-in-place flavor</td>
</tr>
<tr>
<td>ippiErode3x3_8u_C1R</td>
<td>ippiErodeBorder_8u_C1R with 3x3 mask</td>
</tr>
<tr>
<td>ippiErode3x3_8u_C3IR</td>
<td>Use not-in-place flavor</td>
</tr>
<tr>
<td>ippiErode3x3_8u_C3R</td>
<td>ippiErodeBorder_8u_C3R with 3x3 mask</td>
</tr>
<tr>
<td>ippiErode3x3_8u_C4IR</td>
<td>Use not-in-place flavor</td>
</tr>
<tr>
<td>ippiErode3x3_8u_C4R</td>
<td>ippiErodeBorder_8u_C4R with 3x3 mask</td>
</tr>
<tr>
<td>ippiErode_16u_AC4IR</td>
<td>Use not-in-place flavor</td>
</tr>
<tr>
<td>ippiErode_16u_AC4R</td>
<td>ippiErodeBorder_16u_C4R with 3x3 mask</td>
</tr>
<tr>
<td>Removed from 9.0</td>
<td>Substitution or Workaround</td>
</tr>
<tr>
<td>-----------------------------</td>
<td>---------------------------------------------</td>
</tr>
<tr>
<td>ippiErode_16u_C1IR</td>
<td>Use not-in-place flavor</td>
</tr>
<tr>
<td>ippiErode_16u_C1R</td>
<td>ippiErodeBorder_16u_C1R</td>
</tr>
<tr>
<td>ippiErode_16u_C3IR</td>
<td>Use not-in-place flavor</td>
</tr>
<tr>
<td>ippiErode_16u_C3R</td>
<td>ippiErodeBorder_16u_C3R</td>
</tr>
<tr>
<td>ippiErode_16u_C4R</td>
<td>ippiErodeBorder_16u_C4R</td>
</tr>
<tr>
<td>ippiErode_32f_AC4IR</td>
<td>Use not-in-place flavor</td>
</tr>
<tr>
<td>ippiErode_32f_AC4R</td>
<td>ippiErodeBorder_32f_C4R</td>
</tr>
<tr>
<td>ippiErode_32f_C1IR</td>
<td>Use not-in-place flavor</td>
</tr>
<tr>
<td>ippiErode_32f_C1R</td>
<td>ippiErodeBorder_32f_C1R</td>
</tr>
<tr>
<td>ippiErode_32f_C3IR</td>
<td>Use not-in-place flavor</td>
</tr>
<tr>
<td>ippiErode_32f_C3R</td>
<td>ippiErodeBorder_32f_C3R</td>
</tr>
<tr>
<td>ippiErode_32f_C4R</td>
<td>ippiErodeBorder_32f_C4R</td>
</tr>
<tr>
<td>ippiErode_8u_AC4IR</td>
<td>Use not-in-place flavor</td>
</tr>
<tr>
<td>ippiErode_8u_AC4R</td>
<td>ippiErodeBorder_8u_C4R</td>
</tr>
<tr>
<td>ippiErode_8u_C1IR</td>
<td>Use not-in-place flavor</td>
</tr>
<tr>
<td>ippiErode_8u_C1R</td>
<td>ippiErodeBorder_8u_C1R</td>
</tr>
<tr>
<td>ippiErode_8u_C3IR</td>
<td>Use not-in-place flavor</td>
</tr>
<tr>
<td>ippiErode_8u_C3R</td>
<td>ippiErodeBorder_8u_C3R</td>
</tr>
<tr>
<td>ippiErode_8u_C4R</td>
<td>ippiErodeBorder_8u_C4R</td>
</tr>
<tr>
<td>ippiFFTFree_C_32fc</td>
<td>ippiFree</td>
</tr>
<tr>
<td>ippiFFTFree_R_32f</td>
<td>ippiFree</td>
</tr>
<tr>
<td>ippiFFTFree_R_32s</td>
<td>ippiFree</td>
</tr>
<tr>
<td>ippiFFTFwd_RToPack_8u32s_AC4RSfs</td>
<td>ippiConvert_8u32f+ippiFFTFwd_RToPack_32f</td>
</tr>
<tr>
<td>ippiFFTFwd_RToPack_8u32s_C1RSfs</td>
<td>ippiConvert_8u32f+ippiFFTFwd_RToPack_32f</td>
</tr>
<tr>
<td>ippiFFTFwd_RToPack_8u32s_C3RSfs</td>
<td>ippiConvert_8u32f+ippiFFTFwd_RToPack_32f</td>
</tr>
<tr>
<td>ippiFFTFwd_RToPack_8u32s_C4RSfs</td>
<td>ippiConvert_8u32f+ippiFFTFwd_RToPack_32f</td>
</tr>
<tr>
<td>ippiFFTGetBufSize_C_32fc</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiFFTGetBufSize_R_32f</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiFFTGetBufSize_R_32s</td>
<td>N/A</td>
</tr>
</tbody>
</table>
| ippiFFTInitAlloc_C_32fc    |ippiFFTGetSize+ippiMalloc+ippiFFTInit
<table>
<thead>
<tr>
<th>Removed from 9.0</th>
<th>Substitution or Workaround</th>
</tr>
</thead>
<tbody>
<tr>
<td>ippiFFTInitAlloc_R_32f</td>
<td>ippiFFTGetSize+ippiMalloc+ippiFFTInit</td>
</tr>
<tr>
<td>ippiFFTInitAlloc_R_32s</td>
<td>ippConvert_32s32f(or to 64f)+ippiFFTGetSize+ippiMalloc+ippiFFTInit</td>
</tr>
<tr>
<td>ippiFFTInv_PackToR_32s8u_AC4RSfs</td>
<td>ippiFFTFwd_PACKToR_32f+ippiConvert_32f8u</td>
</tr>
<tr>
<td>ippiFFTInv_PackToR_32s8u_C1RSfs</td>
<td>ippiFFTFwd_PACKToR_32f+ippiConvert_32f8u</td>
</tr>
<tr>
<td>ippiFFTInv_PackToR_32s8u_C3RSfs</td>
<td>ippiFFTFwd_PACKToR_32f+ippiConvert_32f8u</td>
</tr>
<tr>
<td>ippiFFTInv_PackToR_32s8u_C4RSfs</td>
<td>ippiFFTFwd_PACKToR_32f+ippiConvert_32f8u</td>
</tr>
<tr>
<td>ippiFilter32f_16s_AC4R</td>
<td>ippiFilterBorderInit_32f+ippiFilterBorder_16s_C4R</td>
</tr>
<tr>
<td>ippipFilter32f_16s_C1R</td>
<td></td>
</tr>
<tr>
<td>ippipFilter32f_16s_C1R</td>
<td></td>
</tr>
<tr>
<td>ippipFilter32f_16s_C3R</td>
<td></td>
</tr>
<tr>
<td>ippipFilter32f_16s_C3R</td>
<td></td>
</tr>
<tr>
<td>ippipFilter32f_16s_C4R</td>
<td></td>
</tr>
<tr>
<td>ippipFilter32f_16s_C4R</td>
<td></td>
</tr>
</tbody>
</table>

**NOTE** Starting from Intel IPP 9.0, kernel coefficients in the ippiFilterBorder functions are used in direct order, as opposed to the inverse order used in lower versions. Make sure to change the filter order when replacing removed functions with ippiFilterBorder.
### Appendix C: Removed Functions for Image and Video Processing

<table>
<thead>
<tr>
<th>Removed from 9.0</th>
<th>Substitution or Workaround</th>
</tr>
</thead>
<tbody>
<tr>
<td>ippiFilter32f_16u_AC4R</td>
<td><strong>NOTE</strong> Starting from Intel IPP 9.0, kernel coefficients in the <code>ippiFilterBorder</code> functions are used in direct order, as opposed to the inverse order used in lower versions. Make sure to change the filter order when replacing removed functions with <code>ippiFilterBorder</code>.</td>
</tr>
<tr>
<td>ippiFilter32f_16u_C1R</td>
<td><code>ippiFilterBorderInit_32f</code>  +<code>ippiFilterBorder_16u_C1R</code></td>
</tr>
<tr>
<td>ippiFilter32f_16u_C3R</td>
<td><code>ippiFilterBorderInit_32f</code>  +<code>ippiFilterBorder_16u_C3R</code></td>
</tr>
<tr>
<td>ippiFilter32f_16u_C4R</td>
<td><code>ippiFilterBorderInit_32f</code>  +<code>ippiFilterBorder_16u_C4R</code></td>
</tr>
<tr>
<td>Removed from 9.0</td>
<td>Substitution or Workaround</td>
</tr>
<tr>
<td>----------------------------------</td>
<td>------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td><strong>NOTE</strong> Starting from Intel IPP 9.0, kernel coefficients in the <code>ippiFilterBorder</code> functions are used in direct order, as opposed to the inverse order used in lower versions. Make sure to change the filter order when replacing removed functions with <code>ippiFilterBorder</code>.</td>
<td></td>
</tr>
<tr>
<td><code>ippiFilter32f_32s_C1R</code></td>
<td><code>ippiFilterBorderInit_32f</code> + <code>ippiConvert_32s32f</code> + <code>ippiFilterBorder_32f_C1R</code></td>
</tr>
<tr>
<td><strong>NOTE</strong> Starting from Intel IPP 9.0, kernel coefficients in the <code>ippiFilterBorder</code> functions are used in direct order, as opposed to the inverse order used in lower versions. Make sure to change the filter order when replacing removed functions with <code>ippiFilterBorder</code>.</td>
<td></td>
</tr>
<tr>
<td><code>ippiFilter32f_32s_C3R</code></td>
<td><code>ippiFilterBorderInit_32f</code> + <code>ippiConvert_32s32f</code> + <code>ippiFilterBorder_32f_C3R</code></td>
</tr>
<tr>
<td><strong>NOTE</strong> Starting from Intel IPP 9.0, kernel coefficients in the <code>ippiFilterBorder</code> functions are used in direct order, as opposed to the inverse order used in lower versions. Make sure to change the filter order when replacing removed functions with <code>ippiFilterBorder</code>.</td>
<td></td>
</tr>
<tr>
<td><code>ippiFilter32f_32s_C4R</code></td>
<td><code>ippiFilterBorderInit_32f</code> + <code>ippiConvert_32s32f</code> + <code>ippiFilterBorder_32f_C4R</code></td>
</tr>
<tr>
<td><strong>NOTE</strong> Starting from Intel IPP 9.0, kernel coefficients in the <code>ippiFilterBorder</code> functions are used in direct order, as opposed to the inverse order used in lower versions. Make sure to change the filter order when replacing removed functions with <code>ippiFilterBorder</code>.</td>
<td></td>
</tr>
<tr>
<td><code>ippiFilter32f_8s16s_C1R</code></td>
<td><code>ippiFilterBorderInit_32f</code> + <code>ippiConvert_8s16s</code> + <code>ippiFilterBorder_16s_C1R</code></td>
</tr>
<tr>
<td>Removed from 9.0</td>
<td>Substitution or Workaround</td>
</tr>
<tr>
<td>------------------</td>
<td>-----------------------------</td>
</tr>
<tr>
<td><strong>NOTE</strong> Starting from Intel IPP 9.0, kernel coefficients in the <code>ippiFilterBorder</code> functions are used in direct order, as opposed to the inverse order used in lower versions. Make sure to change the filter order when replacing removed functions with <code>ippiFilterBorder</code>.</td>
<td></td>
</tr>
</tbody>
</table>
| `ippiFilter32f_8s16s_C3R` | `ippiFilterBorderInit_32f`  
+`ippiConvert_8s16s`  
+`ippiFilterBorder_16s_C3R` |
| **NOTE** Starting from Intel IPP 9.0, kernel coefficients in the `ippiFilterBorder` functions are used in direct order, as opposed to the inverse order used in lower versions. Make sure to change the filter order when replacing removed functions with `ippiFilterBorder`. |
| `ippiFilter32f_8s16s_C4R` | `ippiFilterBorderInit_32f`  
+`ippiConvert_8s16s`  
+`ippiFilterBorder_16s_C4R` |
| **NOTE** Starting from Intel IPP 9.0, kernel coefficients in the `ippiFilterBorder` functions are used in direct order, as opposed to the inverse order used in lower versions. Make sure to change the filter order when replacing removed functions with `ippiFilterBorder`. |
| `ippiFilter32f_8s_C1R` | `ippiFilterBorderInit_32f`  
+`ippiConvert_8s16s`  
+`ippiFilterBorder_16s_C1R` |
| **NOTE** Starting from Intel IPP 9.0, kernel coefficients in the `ippiFilterBorder` functions are used in direct order, as opposed to the inverse order used in lower versions. Make sure to change the filter order when replacing removed functions with `ippiFilterBorder`. |
| `ippiFilter32f_8s_C3R` | `ippiFilterBorderInit_32f`  
+`ippiConvert_8s16s`  
+`ippiFilterBorder_16s_C3R` |
<table>
<thead>
<tr>
<th>Removed from 9.0</th>
<th>Substitution or Workaround</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>NOTE</strong> Starting from Intel IPP 9.0, kernel coefficients in the <em>ippiFilterBorder</em> functions are used in direct order, as opposed to the inverse order used in lower versions. Make sure to change the filter order when replacing removed functions with <em>ippiFilterBorder</em>.</td>
<td></td>
</tr>
<tr>
<td>ippiFilter32f_8s_C4R</td>
<td>ippiFilterBorderInit_32f +ippiConvert_8s16s +ippiFilterBorder_16s_C4R</td>
</tr>
<tr>
<td><strong>NOTE</strong> Starting from Intel IPP 9.0, kernel coefficients in the <em>ippiFilterBorder</em> functions are used in direct order, as opposed to the inverse order used in lower versions. Make sure to change the filter order when replacing removed functions with <em>ippiFilterBorder</em>.</td>
<td></td>
</tr>
<tr>
<td>ippiFilter32f_8u16s_C1R</td>
<td>ippiFilterBorderInit_32f +ippiConvert_8u16s +ippiFilterBorder_16s_C1R</td>
</tr>
<tr>
<td><strong>NOTE</strong> Starting from Intel IPP 9.0, kernel coefficients in the <em>ippiFilterBorder</em> functions are used in direct order, as opposed to the inverse order used in lower versions. Make sure to change the filter order when replacing removed functions with <em>ippiFilterBorder</em>.</td>
<td></td>
</tr>
<tr>
<td>ippiFilter32f_8u16s_C3R</td>
<td>ippiFilterBorderInit_32f +ippiConvert_8u16s +ippiFilterBorder_16s_C3R</td>
</tr>
<tr>
<td><strong>NOTE</strong> Starting from Intel IPP 9.0, kernel coefficients in the <em>ippiFilterBorder</em> functions are used in direct order, as opposed to the inverse order used in lower versions. Make sure to change the filter order when replacing removed functions with <em>ippiFilterBorder</em>.</td>
<td></td>
</tr>
<tr>
<td>ippiFilter32f_8u16s_C4R</td>
<td>ippiFilterBorderInit_32f +ippiConvert_8u16s +ippiFilterBorder_16s_C4R</td>
</tr>
</tbody>
</table>
## Removed from 9.0

### Substitution or Workaround

**NOTE** Starting from Intel IPP 9.0, kernel coefficients in the `ippiFilterBorder` functions are used in direct order, as opposed to the inverse order used in lower versions. Make sure to change the filter order when replacing removed functions with `ippiFilterBorder`.

<table>
<thead>
<tr>
<th>Removed Function</th>
<th>Substitution and Workaround</th>
</tr>
</thead>
</table>
| `ippiFilter32f_8u_AC4R` | `ippiFilterBorderInit_32f`  
+ `ippiFilterBorder_8u_C4R` |
| `ippiFilter32f_8u_C1R` | `ippiFilterBorderInit_32f`  
+ `ippiFilterBorder_8u_C1R` |
| `ippiFilter32f_8u_C3R` | `ippiFilterBorderInit_32f`  
+ `ippiFilterBorder_8u_C3R` |
| `ippiFilter32f_8u_C4R` | `ippiFilterBorderInit_32f`  
+ `ippiFilterBorder_8u_C4R` |
<table>
<thead>
<tr>
<th>Removed from 9.0</th>
<th>Substitution or Workaround</th>
</tr>
</thead>
<tbody>
<tr>
<td>ippiFilterBilateralGetBufSize_8u_C1R</td>
<td>ippiFilterBilateralBorderGetBufferSize</td>
</tr>
<tr>
<td>ippiFilterBilateralInit_8u_C1R</td>
<td>ippiFilterBilateralBorderInit</td>
</tr>
<tr>
<td>ippiFilterBilateral_8u_C1R</td>
<td>ippiFilterBilateralBorder_8u_C1R</td>
</tr>
<tr>
<td>ippiFilterBox_16s_AC4IR</td>
<td>Use ippiFilterBoxBorder</td>
</tr>
<tr>
<td>ippiFilterBox_16s_AC4R</td>
<td>Use ippiFilterBoxBorder</td>
</tr>
<tr>
<td>ippiFilterBox_16s_C1IR</td>
<td>Use ippiFilterBoxBorder</td>
</tr>
<tr>
<td>ippiFilterBox_16s_C1R</td>
<td>Use ippiFilterBoxBorder</td>
</tr>
<tr>
<td>ippiFilterBox_16s_C3IR</td>
<td>Use ippiFilterBoxBorder</td>
</tr>
<tr>
<td>ippiFilterBox_16s_C3R</td>
<td>Use ippiFilterBoxBorder</td>
</tr>
<tr>
<td>ippiFilterBox_16s_C4IR</td>
<td>Use ippiFilterBoxBorder</td>
</tr>
<tr>
<td>ippiFilterBox_16s_C4R</td>
<td>Use ippiFilterBoxBorder</td>
</tr>
<tr>
<td>ippiFilterBox_16u_AC4IR</td>
<td>Use ippiFilterBoxBorder</td>
</tr>
<tr>
<td>ippiFilterBox_16u_AC4R</td>
<td>Use ippiFilterBoxBorder</td>
</tr>
<tr>
<td>ippiFilterBox_16u_C1IR</td>
<td>Use ippiFilterBoxBorder</td>
</tr>
<tr>
<td>ippiFilterBox_16u_C1R</td>
<td>Use ippiFilterBoxBorder</td>
</tr>
<tr>
<td>ippiFilterBox_16u_C3IR</td>
<td>Use ippiFilterBoxBorder</td>
</tr>
<tr>
<td>ippiFilterBox_16u_C3R</td>
<td>Use ippiFilterBoxBorder</td>
</tr>
<tr>
<td>ippiFilterBox_16u_C4IR</td>
<td>Use ippiFilterBoxBorder</td>
</tr>
<tr>
<td>ippiFilterBox_16u_C4R</td>
<td>Use ippiFilterBoxBorder</td>
</tr>
<tr>
<td>ippiFilterBox_32f_AC4IR</td>
<td>Use ippiFilterBoxBorder</td>
</tr>
<tr>
<td>ippiFilterBox_32f_AC4R</td>
<td>Use ippiFilterBoxBorder</td>
</tr>
<tr>
<td>ippiFilterBox_32f_C1IR</td>
<td>Use ippiFilterBoxBorder</td>
</tr>
<tr>
<td>ippiFilterBox_32f_C1R</td>
<td>Use ippiFilterBoxBorder</td>
</tr>
<tr>
<td>ippiFilterBox_32f_C3IR</td>
<td>Use ippiFilterBoxBorder</td>
</tr>
<tr>
<td>ippiFilterBox_32f_C3R</td>
<td>Use ippiFilterBoxBorder</td>
</tr>
<tr>
<td>ippiFilterBox_32f_C4IR</td>
<td>Use ippiFilterBoxBorder</td>
</tr>
</tbody>
</table>

**NOTE** Starting from Intel IPP 9.0, kernel coefficients in the ippiFilterBorder functions are used in direct order, as opposed to the inverse order used in lower versions. Make sure to change the filter order when replacing removed functions with ippiFilterBorder.
### Removed from 9.0

<table>
<thead>
<tr>
<th>Function</th>
<th>Substitution or Workaround</th>
</tr>
</thead>
<tbody>
<tr>
<td>ippiFilterBox_32f_C3R</td>
<td>Use ippiFilterBoxBorder</td>
</tr>
<tr>
<td>ippiFilterBox_32f_C4IR</td>
<td>Use ippiFilterBoxBorder</td>
</tr>
<tr>
<td>ippiFilterBox_32f_C4R</td>
<td>Use ippiFilterBoxBorder</td>
</tr>
<tr>
<td>ippiFilterBox_8u_AC4IR</td>
<td>Use ippiFilterBoxBorder</td>
</tr>
<tr>
<td>ippiFilterBox_8u_AC4R</td>
<td>Use ippiFilterBoxBorder</td>
</tr>
<tr>
<td>ippiFilterBox_8u_C1IR</td>
<td>Use ippiFilterBoxBorder</td>
</tr>
<tr>
<td>ippiFilterBox_8u_C1R</td>
<td>Use ippiFilterBoxBorder</td>
</tr>
<tr>
<td>ippiFilterBox_8u_C3R</td>
<td>Use ippiFilterBoxBorder</td>
</tr>
<tr>
<td>ippiFilterBox_8u_C3R</td>
<td>Use ippiFilterBoxBorder</td>
</tr>
<tr>
<td>ippiFilterBox_8u_C4R</td>
<td>Use ippiFilterBoxBorder</td>
</tr>
<tr>
<td>ippiFilterBox_8u_C4R</td>
<td>Use ippiFilterBoxBorder</td>
</tr>
<tr>
<td>ippiFilterColumn_64f_C1R</td>
<td>Use ippiFilterBorder_64f_C1R (with filter.width=1)</td>
</tr>
<tr>
<td>ippiFilterColumn32f_16s_AC4R</td>
<td>Use ippiFilterBorderInit_32f</td>
</tr>
<tr>
<td>ippiFilterColumn32f_16s_AC4R</td>
<td>+ippiFilterBorder_16s_C4R (with filter.width=1)</td>
</tr>
<tr>
<td>ippiFilterColumn32f_16s_C1R</td>
<td>+ippiFilterBorder_16s_C1R (with filter.width=1)</td>
</tr>
<tr>
<td>ippiFilterColumn32f_16s_C3R</td>
<td>+ippiFilterBorder_16s_C3R (with filter.width=1)</td>
</tr>
</tbody>
</table>

**NOTE** Starting from Intel IPP 9.0, kernel coefficients in the ippiFilterBorder functions are used in direct order, as opposed to the inverse order used in lower versions. Make sure to change the filter order when replacing removed functions with ippiFilterBorder.
### Removed from 9.0

<table>
<thead>
<tr>
<th>Removed from 9.0</th>
<th>Substitution or Workaround</th>
</tr>
</thead>
<tbody>
<tr>
<td>ippifilterColumn32f_16s_C4R</td>
<td><strong>NOTE</strong> Starting from Intel IPP 9.0, kernel coefficients in the ippifilterBorder functions are used in direct order, as opposed to the inverse order used in lower versions. Make sure to change the filter order when replacing removed functions with ippifilterBorder. ippifilterBorderInit_32f + ippifilterBorder_16s_C4R (with filter.width=1)</td>
</tr>
<tr>
<td>ippifilterColumn32f_16u_AC4R</td>
<td><strong>NOTE</strong> Starting from Intel IPP 9.0, kernel coefficients in the ippifilterBorder functions are used in direct order, as opposed to the inverse order used in lower versions. Make sure to change the filter order when replacing removed functions with ippifilterBorder. ippifilterBorderInit_32f + ippifilterBorder_16u_C4R (with filter.width=1)</td>
</tr>
<tr>
<td>ippifilterColumn32f_16u_C1R</td>
<td><strong>NOTE</strong> Starting from Intel IPP 9.0, kernel coefficients in the ippifilterBorder functions are used in direct order, as opposed to the inverse order used in lower versions. Make sure to change the filter order when replacing removed functions with ippifilterBorder. ippifilterBorderInit_32f + ippifilterBorder_16u_C1R (with filter.width=1)</td>
</tr>
<tr>
<td>ippifilterColumn32f_16u_C3R</td>
<td><strong>NOTE</strong> Starting from Intel IPP 9.0, kernel coefficients in the ippifilterBorder functions are used in direct order, as opposed to the inverse order used in lower versions. Make sure to change the filter order when replacing removed functions with ippifilterBorder. ippifilterBorderInit_32f + ippifilterBorder_16u_C3R (with filter.width=1)</td>
</tr>
</tbody>
</table>
### Removed from 9.0 | Substitution or Workaround

<table>
<thead>
<tr>
<th>Removed Function</th>
<th>Substitution</th>
</tr>
</thead>
<tbody>
<tr>
<td>ippFilterColumn32f_16u_C4R</td>
<td>ippFilterBorderInit_32f ippFilterBorder_16u_C4R (with filter.width=1)</td>
</tr>
<tr>
<td>ippFilterColumn32f_8u_AC4R</td>
<td>ippFilterBorderInit_32f ippFilterBorder_8u_C4R (with filter.width=1)</td>
</tr>
<tr>
<td>ippFilterColumn32f_8u_C1R</td>
<td>ippFilterBorderInit_32f ippFilterBorder_8u_C1R (with filter.width=1)</td>
</tr>
<tr>
<td>ippFilterColumn32f_8u_C3R</td>
<td>ippFilterBorderInit_32f ippFilterBorder_8u_C3R (with filter.width=1)</td>
</tr>
</tbody>
</table>

**NOTE** Starting from Intel IPP 9.0, kernel coefficients in the ippFilterBorder functions are used in direct order, as opposed to the inverse order used in lower versions. Make sure to change the filter order when replacing removed functions with ippFilterBorder.
### Removed from 9.0

<table>
<thead>
<tr>
<th>Function</th>
<th>Substitution or Workaround</th>
</tr>
</thead>
<tbody>
<tr>
<td>ippFilterColumn32f_8u_C4R</td>
<td></td>
</tr>
<tr>
<td>ippFilterBorderInit_32f</td>
<td></td>
</tr>
<tr>
<td>+ippFilterBorder_8u_C4R (with filter.width=1)</td>
<td></td>
</tr>
<tr>
<td><strong>NOTE</strong> Starting from Intel IPP 9.0, kernel coefficients in the ippFilterBorder functions are used in direct order, as opposed to the inverse order used in lower versions. Make sure to change the filter order when replacing removed functions with ippFilterBorder.</td>
<td></td>
</tr>
<tr>
<td>ippFilterColumn16s_AC4R</td>
<td></td>
</tr>
<tr>
<td>ippFilterBorderInit_16s</td>
<td></td>
</tr>
<tr>
<td>+ippFilterBorder_16s_C4R (with filter.width=1)</td>
<td></td>
</tr>
<tr>
<td><strong>NOTE</strong> Starting from Intel IPP 9.0, kernel coefficients in the ippFilterBorder functions are used in direct order, as opposed to the inverse order used in lower versions. Make sure to change the filter order when replacing removed functions with ippFilterBorder.</td>
<td></td>
</tr>
<tr>
<td>ippFilterColumn16s_C1R</td>
<td></td>
</tr>
<tr>
<td>ippFilterBorderInit_16s</td>
<td></td>
</tr>
<tr>
<td>+ippFilterBorder_16s_C1R (with filter.width=1)</td>
<td></td>
</tr>
<tr>
<td><strong>NOTE</strong> Starting from Intel IPP 9.0, kernel coefficients in the ippFilterBorder functions are used in direct order, as opposed to the inverse order used in lower versions. Make sure to change the filter order when replacing removed functions with ippFilterBorder.</td>
<td></td>
</tr>
<tr>
<td>ippFilterColumn16s_C3R</td>
<td></td>
</tr>
<tr>
<td>ippFilterBorderInit_16s</td>
<td></td>
</tr>
<tr>
<td>+ippFilterBorder_16s_C3R (with filter.width=1)</td>
<td></td>
</tr>
<tr>
<td><strong>NOTE</strong> Starting from Intel IPP 9.0, kernel coefficients in the ippFilterBorder functions are used in direct order, as opposed to the inverse order used in lower versions. Make sure to change the filter order when replacing removed functions with ippFilterBorder.</td>
<td></td>
</tr>
<tr>
<td>Removed from 9.0</td>
<td>Substitution or Workaround</td>
</tr>
<tr>
<td>----------------------------------</td>
<td>---------------------------------------------------------------------------------------------</td>
</tr>
</tbody>
</table>
| ippiFilterColumn_16s_C4R         | NOTE Starting from Intel IPP 9.0, kernel coefficients in the `ippiFilterBorder` functions are used in direct order, as opposed to the inverse order used in lower versions. Make sure to change the filter order when replacing removed functions with `ippiFilterBorder`.  
|                                  | `ippiFilterBorderInit_16s`  
|                                  | +`ippiFilterBorder_16s_C4R` (with `filter.width`=1)                                        |
| ippiFilterColumn_16u_AC4R        | NOTE Starting from Intel IPP 9.0, kernel coefficients in the `ippiFilterBorder` functions are used in direct order, as opposed to the inverse order used in lower versions. Make sure to change the filter order when replacing removed functions with `ippiFilterBorder`.  
|                                  | `ippiFilterBorderInit_16s`  
|                                  | +`ippiFilterBorder_16u_C4R` (with `filter.width`=1)                                        |
| ippiFilterColumn_16u_C1R         | NOTE Starting from Intel IPP 9.0, kernel coefficients in the `ippiFilterBorder` functions are used in direct order, as opposed to the inverse order used in lower versions. Make sure to change the filter order when replacing removed functions with `ippiFilterBorder`.  
|                                  | `ippiFilterBorderInit_16s`  
|                                  | +`ippiFilterBorder_16u_C1R` (with `filter.width`=1)                                        |
| ippiFilterColumn_16u_C3R         | NOTE Starting from Intel IPP 9.0, kernel coefficients in the `ippiFilterBorder` functions are used in direct order, as opposed to the inverse order used in lower versions. Make sure to change the filter order when replacing removed functions with `ippiFilterBorder`.  
|                                  | `ippiFilterBorderInit_16s`  
<p>|                                  | +<code>ippiFilterBorder_16u_C3R</code> (with <code>filter.width</code>=1)                                        |</p>
<table>
<thead>
<tr>
<th>Removed from 9.0</th>
<th>Substitution or Workaround</th>
</tr>
</thead>
<tbody>
<tr>
<td>ippifilterColumn_16u_C4R</td>
<td>NOTE Starting from Intel IPP 9.0, kernel coefficients in the ippiFilterBorder functions are used in direct order, as opposed to the inverse order used in lower versions. Make sure to change the filter order when replacing removed functions with ippiFilterBorder.</td>
</tr>
<tr>
<td></td>
<td>ippiFilterBorderInit_16s</td>
</tr>
<tr>
<td></td>
<td>+ippiFilterBorder_16u_C4R (with filter.width=1)</td>
</tr>
<tr>
<td>ippifilterColumn_32f_AC4R</td>
<td>NOTE Starting from Intel IPP 9.0, kernel coefficients in the ippiFilterBorder functions are used in direct order, as opposed to the inverse order used in lower versions. Make sure to change the filter order when replacing removed functions with ippiFilterBorder.</td>
</tr>
<tr>
<td></td>
<td>ippiFilterBorderInit_32f</td>
</tr>
<tr>
<td></td>
<td>+ippiFilterBorder_32f_C4R (with filter.width=1)</td>
</tr>
<tr>
<td>ippifilterColumn_32f_C1R</td>
<td>NOTE Starting from Intel IPP 9.0, kernel coefficients in the ippiFilterBorder functions are used in direct order, as opposed to the inverse order used in lower versions. Make sure to change the filter order when replacing removed functions with ippiFilterBorder.</td>
</tr>
<tr>
<td></td>
<td>ippiFilterBorderInit_32f</td>
</tr>
<tr>
<td></td>
<td>+ippiFilterBorder_32f_C1R (with filter.width=1)</td>
</tr>
<tr>
<td>ippifilterColumn_32f_C3R</td>
<td>NOTE Starting from Intel IPP 9.0, kernel coefficients in the ippiFilterBorder functions are used in direct order, as opposed to the inverse order used in lower versions. Make sure to change the filter order when replacing removed functions with ippiFilterBorder.</td>
</tr>
<tr>
<td></td>
<td>ippiFilterBorderInit_32f</td>
</tr>
<tr>
<td></td>
<td>+ippiFilterBorder_32f_C3R (with filter.width=1)</td>
</tr>
<tr>
<td>Removed from 9.0</td>
<td>Substitution or Workaround</td>
</tr>
<tr>
<td>------------------</td>
<td>-----------------------------</td>
</tr>
<tr>
<td>ippiFilterColumn_32f_C4R</td>
<td><strong>NOTE</strong> Starting from Intel IPP 9.0, kernel coefficients in the <code>ippiFilterBorder</code> functions are used in direct order, as opposed to the inverse order used in lower versions. Make sure to change the filter order when replacing removed functions with <code>ippiFilterBorder</code>.</td>
</tr>
</tbody>
</table>
| ippiFilterColumn\_32f\_C4R | `ippiFilterBorderInit\_32f`  
+`ippiFilterBorder\_32f\_C4R` (with `filter.width=1`) |
| ippiFilterColumn\_8u\_AC4R | **NOTE** Starting from Intel IPP 9.0, kernel coefficients in the `ippiFilterBorder` functions are used in direct order, as opposed to the inverse order used in lower versions. Make sure to change the filter order when replacing removed functions with `ippiFilterBorder`. |
| ippiFilterColumn\_8u\_AC4R | `ippiFilterBorderInit\_16s`  
+`ippiFilterBorder\_8u\_C4R` (with `filter.width=1`) |
| ippiFilterColumn\_8u\_CI\_R | **NOTE** Starting from Intel IPP 9.0, kernel coefficients in the `ippiFilterBorder` functions are used in direct order, as opposed to the inverse order used in lower versions. Make sure to change the filter order when replacing removed functions with `ippiFilterBorder`. |
| ippiFilterColumn\_8u\_CI\_R | `ippiFilterBorderInit\_16s`  
+`ippiFilterBorder\_8u\_C1R` (with `filter.width=1`) |
| ippiFilterColumn\_8u\_C3\_R | **NOTE** Starting from Intel IPP 9.0, kernel coefficients in the `ippiFilterBorder` functions are used in direct order, as opposed to the inverse order used in lower versions. Make sure to change the filter order when replacing removed functions with `ippiFilterBorder`. |
| ippiFilterColumn\_8u\_C3\_R | `ippiFilterBorderInit\_16s`  
+`ippiFilterBorder\_8u\_C3R` (with `filter.width=1`) |
<table>
<thead>
<tr>
<th>Removed from 9.0</th>
<th>Substitution or Workaround</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>NOTE</strong> Starting from Intel IPP 9.0, kernel coefficients in the <code>ippiFilterBorder</code> functions are used in direct order, as opposed to the inverse order used in lower versions. Make sure to change the filter order when replacing removed functions with <code>ippiFilterBorder</code>.</td>
<td></td>
</tr>
<tr>
<td><code>ippiFilterColumn_8u_C4R</code></td>
<td><code>ippiFilterBorderInit_16s</code> +<code>ippiFilterBorder_8u_C4R (with filter.width=1)</code></td>
</tr>
<tr>
<td><strong>NOTE</strong> Starting from Intel IPP 9.0, kernel coefficients in the <code>ippiFilterBorder</code> functions are used in direct order, as opposed to the inverse order used in lower versions. Make sure to change the filter order when replacing removed functions with <code>ippiFilterBorder</code>.</td>
<td></td>
</tr>
<tr>
<td><code>ippiFilterGauss_16s_AC4R</code></td>
<td><code>ippiCopy_C4C1R</code> +<code>ippiFilterGaussianBorder_16s_C1R</code></td>
</tr>
<tr>
<td><code>ippiFilterGauss_16s_C1R</code></td>
<td><code>ippiFilterGaussianBorder_16s_C1R</code></td>
</tr>
<tr>
<td><code>ippiFilterGauss_16s_C3R</code></td>
<td><code>ippiFilterGaussianBorder_16s_C3R</code></td>
</tr>
<tr>
<td><code>ippiFilterGauss_16s_C4R</code></td>
<td><code>ippiCopy_C4C1R</code> +<code>ippiFilterGaussianBorder_16s_C1R</code></td>
</tr>
<tr>
<td><code>ippiFilterGauss_16u_AC4R</code></td>
<td><code>ippiCopy_C4C1R</code> +<code>ippiFilterGaussianBorder_16u_C1R</code></td>
</tr>
<tr>
<td><code>ippiFilterGauss_16u_C1R</code></td>
<td><code>ippiFilterGaussianBorder_16u_C1R</code></td>
</tr>
<tr>
<td><code>ippiFilterGauss_16u_C3R</code></td>
<td><code>ippiFilterGaussianBorder_16u_C3R</code></td>
</tr>
<tr>
<td><code>ippiFilterGauss_16u_C4R</code></td>
<td><code>ippiCopy_C4C1R</code> +<code>ippiFilterGaussianBorder_16u_C1R</code></td>
</tr>
<tr>
<td><code>ippiFilterGauss_32f_AC4R</code></td>
<td><code>ippiCopy_C4C1R</code> +<code>ippiFilterGaussianBorder_32f_C1R</code></td>
</tr>
<tr>
<td><code>ippiFilterGauss_32f_C1R</code></td>
<td><code>ippiFilterGaussianBorder_32f_C1R</code></td>
</tr>
<tr>
<td><code>ippiFilterGauss_32f_C3R</code></td>
<td><code>ippiFilterGaussianBorder_32f_C3R</code></td>
</tr>
<tr>
<td><code>ippiFilterGauss_32f_C4R</code></td>
<td><code>ippiCopy_C4C1R</code> +<code>ippiFilterGaussianBorder_32f_C1R</code></td>
</tr>
<tr>
<td><code>ippiFilterGauss_8u_AC4R</code></td>
<td><code>ippiCopy_C4C1R</code> +<code>ippiFilterGaussianBorder_8u_C1R</code></td>
</tr>
<tr>
<td><code>ippiFilterGauss_8u_C1R</code></td>
<td><code>ippiFilterGaussianBorder_8u_C1R</code></td>
</tr>
</tbody>
</table>
### Removed from 9.0

<table>
<thead>
<tr>
<th>Removed from 9.0</th>
<th>Substitution or Workaround</th>
</tr>
</thead>
<tbody>
<tr>
<td>ippiFilterGauss_8u_C3R</td>
<td>ippiFilterGaussianBorder_8u_C3R</td>
</tr>
<tr>
<td>ippiFilterGauss_8u_C4R</td>
<td>ippiCopy_C4C1R +ippiFilterGaussianBorder_8u_C1R</td>
</tr>
<tr>
<td>ippiFilterHipass_16s_AC4R</td>
<td>ippiFilterHipassBorder_16s_AC4R</td>
</tr>
<tr>
<td>ippiFilterHipass_16s_C1R</td>
<td>ippiFilterHipassBorder_16s_C1R</td>
</tr>
<tr>
<td>ippiFilterHipass_16s_C3R</td>
<td>ippiFilterHipassBorder_16s_C3R</td>
</tr>
<tr>
<td>ippiFilterHipass_16s_C4R</td>
<td>ippiFilterHipassBorder_16s_C4R</td>
</tr>
<tr>
<td>ippiFilterHipass_32f_AC4R</td>
<td>ippiFilterHipassBorder_32f_AC4R</td>
</tr>
<tr>
<td>ippiFilterHipass_32f_C1R</td>
<td>ippiFilterHipassBorder_32f_C1R</td>
</tr>
<tr>
<td>ippiFilterHipass_32f_C3R</td>
<td>ippiFilterHipassBorder_32f_C3R</td>
</tr>
<tr>
<td>ippiFilterHipass_32f_C4R</td>
<td>ippiFilterHipassBorder_32f_C4R</td>
</tr>
<tr>
<td>ippiFilterHipass_8u_AC4R</td>
<td>ippiFilterHipassBorder_8u_AC4R</td>
</tr>
<tr>
<td>ippiFilterHipass_8u_C1R</td>
<td>ippiFilterHipassBorder_8u_C1R</td>
</tr>
<tr>
<td>ippiFilterHipass_8u_C3R</td>
<td>ippiFilterHipassBorder_8u_C3R</td>
</tr>
<tr>
<td>ippiFilterHipass_8u_C4R</td>
<td>ippiFilterHipassBorder_8u_C4R</td>
</tr>
<tr>
<td>ippiFilterLaplace_16s_AC4R</td>
<td>ippiFilterLaplaceBorder_16s_AC4R</td>
</tr>
<tr>
<td>ippiFilterLaplace_16s_C1R</td>
<td>ippiFilterLaplaceBorder_16s_C1R</td>
</tr>
<tr>
<td>ippiFilterLaplace_16s_C3R</td>
<td>ippiFilterLaplaceBorder_16s_C3R</td>
</tr>
<tr>
<td>ippiFilterLaplace_16s_C4R</td>
<td>ippiFilterLaplaceBorder_16s_C4R</td>
</tr>
<tr>
<td>ippiFilterLaplace_32f_AC4R</td>
<td>ippiFilterLaplaceBorder_32f_AC4R</td>
</tr>
<tr>
<td>ippiFilterLaplace_32f_C1R</td>
<td>ippiFilterLaplaceBorder_32f_C1R</td>
</tr>
<tr>
<td>ippiFilterLaplace_32f_C3R</td>
<td>ippiFilterLaplaceBorder_32f_C3R</td>
</tr>
<tr>
<td>ippiFilterLaplace_32f_C4R</td>
<td>ippiFilterLaplaceBorder_32f_C4R</td>
</tr>
<tr>
<td>ippiFilterLaplace_8s16s_C1R</td>
<td>ippiConvert_8s16s +ippiFilterLaplaceBorder_16s</td>
</tr>
<tr>
<td>ippiFilterLaplace_8u16s_C1R</td>
<td>ippiConvert_8u16s +ippiFilterLaplaceBorder_16s</td>
</tr>
<tr>
<td>ippiFilterLaplace_8u_AC4R</td>
<td>ippiFilterLaplaceBorder_8u_AC4R</td>
</tr>
<tr>
<td>ippiFilterLaplace_8u_C1R</td>
<td>ippiFilterLaplaceBorder_8u_C1R</td>
</tr>
<tr>
<td>ippiFilterLaplace_8u_C3R</td>
<td>ippiFilterLaplaceBorder_8u_C3R</td>
</tr>
<tr>
<td>ippiFilterLaplace_8u_C4R</td>
<td>ippiFilterLaplaceBorder_8u_C4R</td>
</tr>
<tr>
<td>Removed from 9.0</td>
<td>Substitution or Workaround</td>
</tr>
<tr>
<td>----------------------------------</td>
<td>-------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>ippiFilterLowpass_16s_AC4R</td>
<td>ippiFilterBoxBorder_16s_AC4R</td>
</tr>
<tr>
<td>ippiFilterLowpass_16s_C1R</td>
<td>ippiFilterBoxBorder_16s_C1R</td>
</tr>
<tr>
<td>ippiFilterLowpass_16s_C3R</td>
<td>ippiFilterBoxBorder_16s_C3R</td>
</tr>
<tr>
<td>ippiFilterLowpass_16u_AC4R</td>
<td>ippiFilterBoxBorder_16u_AC4R</td>
</tr>
<tr>
<td>ippiFilterLowpass_16u_C1R</td>
<td>ippiFilterBoxBorder_16u_C1R</td>
</tr>
<tr>
<td>ippiFilterLowpass_16u_C3R</td>
<td>ippiFilterBoxBorder_16u_C3R</td>
</tr>
<tr>
<td>ippiFilterLowpass_32f_AC4R</td>
<td>ippiFilterBoxBorder_32f_AC4R</td>
</tr>
<tr>
<td>ippiFilterLowpass_32f_C1R</td>
<td>ippiFilterBoxBorder_32f_C1R</td>
</tr>
<tr>
<td>ippiFilterLowpass_32f_C3R</td>
<td>ippiFilterBoxBorder_32f_C3R</td>
</tr>
<tr>
<td>ippiFilterLowpass_8u_AC4R</td>
<td>ippiFilterBoxBorder_8u_AC4R</td>
</tr>
<tr>
<td>ippiFilterLowpass_8u_C1R</td>
<td>ippiFilterBoxBorder_8u_C1R</td>
</tr>
<tr>
<td>ippiFilterLowpass_8u_C3R</td>
<td>ippiFilterBoxBorder_8u_C3R</td>
</tr>
<tr>
<td>ippiFilterMax_16s_AC4R</td>
<td>Use ippiFilterMaxBorder</td>
</tr>
<tr>
<td>ippiFilterMax_16s_C1R</td>
<td>Use ippiFilterMaxBorder</td>
</tr>
<tr>
<td>ippiFilterMax_16s_C3R</td>
<td>Use ippiFilterMaxBorder</td>
</tr>
<tr>
<td>ippiFilterMax_16u_AC4R</td>
<td>Use ippiFilterMaxBorder</td>
</tr>
<tr>
<td>ippiFilterMax_16u_C1R</td>
<td>Use ippiFilterMaxBorder</td>
</tr>
<tr>
<td>ippiFilterMax_16u_C3R</td>
<td>Use ippiFilterMaxBorder</td>
</tr>
<tr>
<td>ippiFilterMax_16u_C4R</td>
<td>Use ippiFilterMaxBorder</td>
</tr>
<tr>
<td>ippiFilterMax_32f_AC4R</td>
<td>Use ippiFilterMaxBorder</td>
</tr>
<tr>
<td>ippiFilterMax_32f_C1R</td>
<td>Use ippiFilterMaxBorder</td>
</tr>
<tr>
<td>ippiFilterMax_32f_C3R</td>
<td>Use ippiFilterMaxBorder</td>
</tr>
<tr>
<td>ippiFilterMax_32f_C4R</td>
<td>Use ippiFilterMaxBorder</td>
</tr>
<tr>
<td>ippiFilterMax_8u_AC4R</td>
<td>Use ippiFilterMaxBorder</td>
</tr>
<tr>
<td>ippiFilterMax_8u_C1R</td>
<td>Use ippiFilterMaxBorder</td>
</tr>
<tr>
<td>ippiFilterMax_8u_C3R</td>
<td>Use ippiFilterMaxBorder</td>
</tr>
<tr>
<td>ippiFilterMax_8u_C4R</td>
<td>Use ippiFilterMaxBorder</td>
</tr>
<tr>
<td>ippiFilterMedianHoriz_16s_AC4R</td>
<td>Use ippiFilterMedianBorder with horiz mask (mask.height=1)</td>
</tr>
<tr>
<td>Removed from 9.0</td>
<td>Substitution or Workaround</td>
</tr>
<tr>
<td>-----------------</td>
<td>---------------------------</td>
</tr>
<tr>
<td>ippiFilterMedianHoriz_16s_C1R</td>
<td>Use ippiFilterMedianBorder with horiz mask (mask.height=1)</td>
</tr>
<tr>
<td>ippiFilterMedianHoriz_16s_C3R</td>
<td>Use ippiFilterMedianBorder with horiz mask (mask.height=1)</td>
</tr>
<tr>
<td>ippiFilterMedianHoriz_16s_C4R</td>
<td>Use ippiFilterMedianBorder with horiz mask (mask.height=1)</td>
</tr>
<tr>
<td>ippiFilterMedianHoriz_16u_AC4R</td>
<td>Use ippiFilterMedianBorder with horiz mask (mask.height=1)</td>
</tr>
<tr>
<td>ippiFilterMedianHoriz_16u_C1R</td>
<td>Use ippiFilterMedianBorder with horiz mask (mask.height=1)</td>
</tr>
<tr>
<td>ippiFilterMedianHoriz_16u_C3R</td>
<td>Use ippiFilterMedianBorder with horiz mask (mask.height=1)</td>
</tr>
<tr>
<td>ippiFilterMedianHoriz_16u_C4R</td>
<td>Use ippiFilterMedianBorder with horiz mask (mask.height=1)</td>
</tr>
<tr>
<td>ippiFilterMedianHoriz_8u_AC4R</td>
<td>Use ippiFilterMedianBorder with horiz mask (mask.height=1)</td>
</tr>
<tr>
<td>ippiFilterMedianHoriz_8u_C1R</td>
<td>Use ippiFilterMedianBorder with horiz mask (mask.height=1)</td>
</tr>
<tr>
<td>ippiFilterMedianHoriz_8u_C3R</td>
<td>Use ippiFilterMedianBorder with horiz mask (mask.height=1)</td>
</tr>
<tr>
<td>ippiFilterMedianHoriz_8u_C4R</td>
<td>Use ippiFilterMedianBorder with horiz mask (mask.height=1)</td>
</tr>
<tr>
<td>ippiFilterMedianVert_16s_AC4R</td>
<td>Use ippiFilterMedianBorder with vertical mask (mask.width=1)</td>
</tr>
<tr>
<td>ippiFilterMedianVert_16s_C1R</td>
<td>Use ippiFilterMedianBorder with vertical mask (mask.width=1)</td>
</tr>
<tr>
<td>ippiFilterMedianVert_16s_C3R</td>
<td>Use ippiFilterMedianBorder with vertical mask (mask.width=1)</td>
</tr>
<tr>
<td>ippiFilterMedianVert_16s_C4R</td>
<td>Use ippiFilterMedianBorder with vertical mask (mask.width=1)</td>
</tr>
<tr>
<td>ippiFilterMedianVert_16u_AC4R</td>
<td>Use ippiFilterMedianBorder with vertical mask (mask.width=1)</td>
</tr>
<tr>
<td>ippiFilterMedianVert_16u_C1R</td>
<td>Use ippiFilterMedianBorder with vertical mask (mask.width=1)</td>
</tr>
<tr>
<td>ippiFilterMedianVert_16u_C3R</td>
<td>Use ippiFilterMedianBorder with vertical mask (mask.width=1)</td>
</tr>
<tr>
<td>ippiFilterMedianVert_16u_C4R</td>
<td>Use ippiFilterMedianBorder with vertical mask (mask.width=1)</td>
</tr>
<tr>
<td>Removed from 9.0</td>
<td>Substitution or Workaround</td>
</tr>
<tr>
<td>-------------------------------------</td>
<td>-----------------------------------------------------------------</td>
</tr>
<tr>
<td>ippiFilterMedianVert_8u_AC4R</td>
<td>Use ippiFilterMedianBorder with vertical mask (mask.width=1)</td>
</tr>
<tr>
<td>ippiFilterMedianVert_8u_C1R</td>
<td>Use ippiFilterMedianBorder with vertical mask (mask.width=1)</td>
</tr>
<tr>
<td>ippiFilterMedianVert_8u_C3R</td>
<td>Use ippiFilterMedianBorder with vertical mask (mask.width=1)</td>
</tr>
<tr>
<td>ippiFilterMedianVert_8u_C4R</td>
<td>Use ippiFilterMedianBorder with vertical mask (mask.width=1)</td>
</tr>
<tr>
<td>ippiFilterMedian_16s_AC4R</td>
<td>Use ippiFilterMedianBorder</td>
</tr>
<tr>
<td>ippiFilterMedian_16s_C1R</td>
<td>Use ippiFilterMedianBorder</td>
</tr>
<tr>
<td>ippiFilterMedian_16s_C3R</td>
<td>Use ippiFilterMedianBorder</td>
</tr>
<tr>
<td>ippiFilterMedian_16s_C4R</td>
<td>Use ippiFilterMedianBorder</td>
</tr>
<tr>
<td>ippiFilterMedian_16u_AC4R</td>
<td>Use ippiFilterMedianBorder</td>
</tr>
<tr>
<td>ippiFilterMedian_16u_C1R</td>
<td>Use ippiFilterMedianBorder</td>
</tr>
<tr>
<td>ippiFilterMedian_16u_C3R</td>
<td>Use ippiFilterMedianBorder</td>
</tr>
<tr>
<td>ippiFilterMedian_16u_C4R</td>
<td>Use ippiFilterMedianBorder</td>
</tr>
<tr>
<td>ippiFilterMedian_32f_C1R</td>
<td>Use ippiFilterMedianBorder</td>
</tr>
<tr>
<td>ippiFilterMedian_8u_AC4R</td>
<td>Use ippiFilterMedianBorder</td>
</tr>
<tr>
<td>ippiFilterMedian_8u_C1R</td>
<td>Use ippiFilterMedianBorder</td>
</tr>
<tr>
<td>ippiFilterMedian_8u_C3R</td>
<td>Use ippiFilterMedianBorder</td>
</tr>
<tr>
<td>ippiFilterMedian_8u_C4R</td>
<td>Use ippiFilterMedianBorder</td>
</tr>
<tr>
<td>ippiFilterMin_16s_AC4R</td>
<td>Use ippiFilterMinBorder</td>
</tr>
<tr>
<td>ippiFilterMin_16s_C1R</td>
<td>Use ippiFilterMinBorder</td>
</tr>
<tr>
<td>ippiFilterMin_16s_C3R</td>
<td>Use ippiFilterMinBorder</td>
</tr>
<tr>
<td>ippiFilterMin_16s_C4R</td>
<td>Use ippiFilterMinBorder</td>
</tr>
<tr>
<td>ippiFilterMin_16u_AC4R</td>
<td>Use ippiFilterMinBorder</td>
</tr>
<tr>
<td>ippiFilterMin_16u_C1R</td>
<td>Use ippiFilterMinBorder</td>
</tr>
<tr>
<td>ippiFilterMin_16u_C3R</td>
<td>Use ippiFilterMinBorder</td>
</tr>
<tr>
<td>ippiFilterMin_16u_C4R</td>
<td>Use ippiFilterMinBorder</td>
</tr>
<tr>
<td>ippiFilterMin_32f_AC4R</td>
<td>Use ippiFilterMinBorder</td>
</tr>
<tr>
<td>ippiFilterMin_32f_C1R</td>
<td>Use ippiFilterMinBorder</td>
</tr>
</tbody>
</table>
### Appendix C: Removed Functions for Image and Video Processing

<table>
<thead>
<tr>
<th>Removed from 9.0</th>
<th>Substitution or Workaround</th>
</tr>
</thead>
<tbody>
<tr>
<td>ippiFilterMin_32f_C3R</td>
<td>Use ippiFilterMin_Border</td>
</tr>
<tr>
<td>ippiFilterMin_32f_C4R</td>
<td>Use ippiFilterMin_Border</td>
</tr>
<tr>
<td>ippiFilterMin_8u_AC4R</td>
<td>Use ippiFilterMin_Border</td>
</tr>
<tr>
<td>ippiFilterMin_8u_C1R</td>
<td>Use ippiFilterMin_Border</td>
</tr>
<tr>
<td>ippiFilterMin_8u_C3R</td>
<td>Use ippiFilterMin_Border</td>
</tr>
<tr>
<td>ippiFilterMin_8u_C4R</td>
<td>Use ippiFilterMin_Border</td>
</tr>
<tr>
<td>ippiFilterPrewittHoriz_16s_AC4R</td>
<td><strong>Use ippiCopy_16s_C4C1R</strong></td>
</tr>
<tr>
<td>ippiFilterPrewittHoriz_16s_C1R</td>
<td>++ippiFilterPrewittHoriz_Border_16s_C1R</td>
</tr>
<tr>
<td>ippiFilterPrewittHoriz_16s_C3R</td>
<td><strong>Use ippiCopy_16s_C3C1R</strong></td>
</tr>
<tr>
<td>ippiFilterPrewittHoriz_16s_C4R</td>
<td>++ippiFilterPrewittHoriz_Border_16s_C1R</td>
</tr>
<tr>
<td>ippiFilterPrewittHoriz_32f_AC4R</td>
<td><strong>Use ippiCopy_32f_C4C1R</strong></td>
</tr>
<tr>
<td>ippiFilterPrewittHoriz_32f_C1R</td>
<td>++ippiFilterPrewittHoriz_Border_32f_C1R</td>
</tr>
<tr>
<td>ippiFilterPrewittHoriz_32f_C3R</td>
<td><strong>Use ippiCopy_32f_C3C1R</strong></td>
</tr>
<tr>
<td>ippiFilterPrewittHoriz_32f_C4R</td>
<td>++ippiFilterPrewittHoriz_Border_32f_C1R</td>
</tr>
<tr>
<td>ippiFilterPrewittHoriz_8u_AC4R</td>
<td><strong>Use ippiCopy_8u_C4C1R</strong></td>
</tr>
<tr>
<td>ippiFilterPrewittHoriz_8u_C1R</td>
<td>++ippiFilterPrewittHoriz_Border_8u16s_C1R</td>
</tr>
<tr>
<td>ippiFilterPrewittHoriz_8u_C3R</td>
<td><strong>Use ippiCopy_8u_C3C1R</strong></td>
</tr>
<tr>
<td>ippiFilterPrewittHoriz_8u_C4R</td>
<td>++ippiFilterPrewittHoriz_Border_8u16s_C1R</td>
</tr>
<tr>
<td>ippiFilterPrewittVert_16s_AC4R</td>
<td><strong>Use ippiCopy_16s_C4C1R</strong></td>
</tr>
<tr>
<td>ippiFilterPrewittVert_16s_C1R</td>
<td>++ippiFilterPrewittVert_Border_16s_C1R</td>
</tr>
<tr>
<td>ippiFilterPrewittVert_16s_C3R</td>
<td><strong>Use ippiCopy_16s_C3C1R</strong></td>
</tr>
<tr>
<td>ippiFilterPrewittVert_16s_C4R</td>
<td>++ippiFilterPrewittVert_Border_16s_C1R</td>
</tr>
<tr>
<td>ippiFilterPrewittVert_16s_C4R</td>
<td><strong>Use ippiCopy_16s_C4C1R</strong></td>
</tr>
<tr>
<td>ippiFilterPrewittVert_16s_C4R</td>
<td>++ippiFilterPrewittVert_Border_16s_C1R</td>
</tr>
<tr>
<td>ippiFilterPrewittVert_16s_C4R</td>
<td><strong>Use ippiCopy_16s_C4C1R</strong></td>
</tr>
<tr>
<td>ippiFilterPrewittVert_16s_C4R</td>
<td>++ippiFilterPrewittVert_Border_16s_C1R</td>
</tr>
<tr>
<td>ippiFilterPrewittVert_16s_C4R</td>
<td><strong>Use ippiCopy_16s_C4C1R</strong></td>
</tr>
<tr>
<td>ippiFilterPrewittVert_16s_C4R</td>
<td>++ippiFilterPrewittVert_Border_16s_C1R</td>
</tr>
<tr>
<td>Removed from 9.0</td>
<td>Substitution or Workaround</td>
</tr>
<tr>
<td>----------------------------------</td>
<td>----------------------------------------------------------------</td>
</tr>
<tr>
<td>ippiFilterPrewittVert_32f_AC4R</td>
<td>Use ippiCopy_32f_C4C1R +ippiFilterPrewittVertBorder_32f_C1R</td>
</tr>
<tr>
<td>ippiFilterPrewittVert_32f_C1R</td>
<td>ippiFilterPrewittVertBorder_32f_C1R</td>
</tr>
<tr>
<td>ippiFilterPrewittVert_32f_C3R</td>
<td>Use ippiCopy_32f_C3C1R +ippiFilterPrewittVertBorder_32f_C1R</td>
</tr>
<tr>
<td>ippiFilterPrewittVert_32f_C4R</td>
<td>Use ippiCopy_32f_C4C1R +ippiFilterPrewittVertBorder_32f_C1R</td>
</tr>
<tr>
<td>ippiFilterPrewittVert_8u_AC4R</td>
<td>Use ippiCopy_8u_C4C1R +ippiFilterPrewittVertBorder_8u16s_C1R</td>
</tr>
<tr>
<td>ippiFilterPrewittVert_8u_C1R</td>
<td>ippiFilterPrewittVertBorder_8u16s_C1R</td>
</tr>
<tr>
<td>ippiFilterPrewittVert_8u_C3R</td>
<td>Use ippiCopy_8u_C3C1R +ippiFilterPrewittVertBorder_8u16s_C1R</td>
</tr>
<tr>
<td>ippiFilterPrewittVert_8u_C4R</td>
<td>Use ippiCopy_8u_C4C1R +ippiFilterPrewittVertBorder_8u16s_C1R</td>
</tr>
<tr>
<td>ippiFilterRobertsDown_16s_AC4R</td>
<td>Use ippiCopy_16s_C4C1R +ippiFilterRobertsDownBorder_16s_C1R</td>
</tr>
<tr>
<td>ippiFilterRobertsDown_16s_C1R</td>
<td>ippiFilterRobertsDownBorder_16s_C1R</td>
</tr>
<tr>
<td>ippiFilterRobertsDown_16s_C3R</td>
<td>Use ippiCopy_16s_C3C1R +ippiFilterRobertsDownBorder_16s_C1R</td>
</tr>
<tr>
<td>ippiFilterRobertsDown_32f_AC4R</td>
<td>Use ippiCopy_32f_C4C1R +ippiFilterRobertsDownBorder_32f_C1R</td>
</tr>
<tr>
<td>ippiFilterRobertsDown_32f_C1R</td>
<td>ippiFilterRobertsDownBorder_32f_C1R</td>
</tr>
<tr>
<td>ippiFilterRobertsDown_32f_C3R</td>
<td>Use ippiCopy_32f_C3C1R +ippiFilterRobertsDownBorder_32f_C1R</td>
</tr>
<tr>
<td>ippiFilterRobertsDown_8u_AC4R</td>
<td>Use ippiCopy_8u_C4C1R +ippiFilterRobertsDownBorder_8u16s_C1R</td>
</tr>
<tr>
<td>ippiFilterRobertsDown_8u_C1R</td>
<td>ippiFilterRobertsDownBorder_8u16s_C1R</td>
</tr>
<tr>
<td>ippiFilterRobertsDown_8u_C3R</td>
<td>Use ippiCopy_8u_C3C1R +ippiFilterRobertsDownBorder_8u16s_C1R</td>
</tr>
<tr>
<td>ippiFilterRobertsUp_16s_AC4R</td>
<td>Use ippiCopy_16s_C4C1R +ippiFilterRobertsUpBorder_16s_C1R</td>
</tr>
<tr>
<td>ippiFilterRobertsUp_16s_C1R</td>
<td>ippiFilterRobertsUpBorder_16s_C1R</td>
</tr>
<tr>
<td>ippiFilterRobertsUp_16s_C3R</td>
<td>Use ippiCopy_16s_C3C1R +ippiFilterRobertsUpBorder_16s_C1R</td>
</tr>
<tr>
<td>ippiFilterRobertsUp_32f_AC4R</td>
<td>Use ippiCopy_32f_C4C1R +ippiFilterRobertsUpBorder_32f_C1R</td>
</tr>
<tr>
<td>Removed from 9.0</td>
<td>Substitution or Workaround</td>
</tr>
<tr>
<td>-------------------------------------------------------</td>
<td>----------------------------------------------------------------</td>
</tr>
<tr>
<td>ippifilterRobertsUp_32f_C1R</td>
<td>ippifilterRobertsUpBorder_32f_C1R</td>
</tr>
<tr>
<td>ippifilterRobertsUp_32f_C3R</td>
<td>Use ippiCopy_32f_C3C1R + ippifilterRobertsUpBorder_32f_C1R</td>
</tr>
<tr>
<td>ippifilterRobertsUp_8u_AC4R</td>
<td>Use ippiCopy_8u_C4C1R + ippifilterRobertsUpBorder_8u16s_C1R</td>
</tr>
<tr>
<td>ippifilterRobertsUp_8u_C1R</td>
<td>ippifilterRobertsUpBorder_8u16s_C1R</td>
</tr>
<tr>
<td>ippifilterRobertsUp_8u_C3R</td>
<td>Use ippiCopy_8u_C3C1R + ippifilterRobertsUpBorder_8u16s_C1R</td>
</tr>
<tr>
<td>ippifilterRoundGetBufSize16s_8u_AC4R</td>
<td>ippifilterBorderGetSize</td>
</tr>
<tr>
<td>ippifilterRoundGetBufSize16s_8u_C1R</td>
<td>ippifilterBorderGetSize</td>
</tr>
<tr>
<td>ippifilterRoundGetBufSize16s_8u_C3R</td>
<td>ippifilterBorderGetSize</td>
</tr>
<tr>
<td>ippifilterRoundGetBufSize16s_8u_C4R</td>
<td>ippifilterBorderGetSize</td>
</tr>
<tr>
<td>ippifilterRoundGetBufSize32f_16s_AC4R</td>
<td>ippifilterBorderGetSize</td>
</tr>
<tr>
<td>ippifilterRoundGetBufSize32f_16s_C1R</td>
<td>ippifilterBorderGetSize</td>
</tr>
<tr>
<td>ippifilterRoundGetBufSize32f_16s_C3R</td>
<td>ippifilterBorderGetSize</td>
</tr>
<tr>
<td>ippifilterRoundGetBufSize32f_16s_C4R</td>
<td>ippifilterBorderGetSize</td>
</tr>
<tr>
<td>ippifilterRoundGetBufSize32f_16u_AC4R</td>
<td>ippifilterBorderGetSize</td>
</tr>
<tr>
<td>ippifilterRoundGetBufSize32f_16u_C1R</td>
<td>ippifilterBorderGetSize</td>
</tr>
<tr>
<td>ippifilterRoundGetBufSize32f_16u_C3R</td>
<td>ippifilterBorderGetSize</td>
</tr>
<tr>
<td>ippifilterRoundGetBufSize32f_16u_C4R</td>
<td>ippifilterBorderGetSize</td>
</tr>
<tr>
<td>ippifilterRoundGetBufSize32f_8u_AC4R</td>
<td>ippifilterBorderGetSize</td>
</tr>
<tr>
<td>ippifilterRoundGetBufSize32f_8u_C1R</td>
<td>ippifilterBorderGetSize</td>
</tr>
<tr>
<td>ippifilterRoundGetBufSize32f_8u_C3R</td>
<td>ippifilterBorderGetSize</td>
</tr>
<tr>
<td>ippifilterRoundGetBufSize32f_8u_C4R</td>
<td>ippifilterBorderGetSize</td>
</tr>
<tr>
<td>ippifilterRoundGetBufSize32s_16s_AC4R</td>
<td>ippifilterBorderGetSize</td>
</tr>
<tr>
<td>ippifilterRoundGetBufSize32s_16s_C1R</td>
<td>ippifilterBorderGetSize</td>
</tr>
<tr>
<td>ippifilterRoundGetBufSize32s_16s_C3R</td>
<td>ippifilterBorderGetSize</td>
</tr>
<tr>
<td>ippifilterRoundGetBufSize32s_16s_C4R</td>
<td>ippifilterBorderGetSize</td>
</tr>
<tr>
<td>ippifilterRoundGetBufSize32s_16u_AC4R</td>
<td>ippifilterBorderGetSize</td>
</tr>
<tr>
<td>ippifilterRoundGetBufSize32s_16u_C1R</td>
<td>ippifilterBorderGetSize</td>
</tr>
<tr>
<td>ippifilterRoundGetBufSize32s_16u_C3R</td>
<td>ippifilterBorderGetSize</td>
</tr>
<tr>
<td>ippifilterRoundGetBufSize32s_16u_C4R</td>
<td>ippifilterBorderGetSize</td>
</tr>
<tr>
<td>ippifilterRoundGetBufSize32s_16u16s_AC4R</td>
<td>ippifilterBorderGetSize</td>
</tr>
<tr>
<td>ippifilterRoundGetBufSize32s_16u16s_C1R</td>
<td>ippifilterBorderGetSize</td>
</tr>
<tr>
<td>ippifilterRoundGetBufSize32s_16u16s_C3R</td>
<td>ippifilterBorderGetSize</td>
</tr>
<tr>
<td>ippifilterRoundGetBufSize32s_16u16s_C4R</td>
<td>ippifilterBorderGetSize</td>
</tr>
<tr>
<td>ippifilterRoundGetBufSize32s_8u16s_AC4R</td>
<td>ippifilterBorderGetSize</td>
</tr>
<tr>
<td>ippifilterRoundGetBufSize32s_8u16s_C1R</td>
<td>ippifilterBorderGetSize</td>
</tr>
<tr>
<td>ippifilterRoundGetBufSize32s_8u16s_C3R</td>
<td>ippifilterBorderGetSize</td>
</tr>
<tr>
<td>ippifilterRoundGetBufSize32s_8u16s_C4R</td>
<td>ippifilterBorderGetSize</td>
</tr>
</tbody>
</table>
### Removed from 9.0

<table>
<thead>
<tr>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>ippiFilterRoundGetBufSize32s_16u_C4R</td>
</tr>
<tr>
<td>ippiFilterRow_64f_C1R</td>
</tr>
<tr>
<td>ippiFilterRow32f_16s_AC4R</td>
</tr>
<tr>
<td>ippiFilterRow32f_16s_C3R</td>
</tr>
<tr>
<td>ippiFilterRow32f_16s_C4R</td>
</tr>
</tbody>
</table>

### Substitution or Workaround

<table>
<thead>
<tr>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>ippiFilterBorderGetSize</td>
</tr>
<tr>
<td>ippiFilter_64f_C1R</td>
</tr>
<tr>
<td>ippiFilterBorderInit_32f +ippiFilterBorder_16s_C4R (with filter.height=1)</td>
</tr>
<tr>
<td>ippiFilterBorderInit_32f +ippiFilterBorder_16s_C1R (with filter.height=1)</td>
</tr>
<tr>
<td>ippiFilterBorderInit_32f +ippiFilterBorder_16s_C3R (with filter.height=1)</td>
</tr>
</tbody>
</table>

**NOTE** Starting from Intel IPP 9.0, kernel coefficients in the `ippiFilterBorder` functions are used in direct order, as opposed to the inverse order used in lower versions. Make sure to change the filter order when replacing removed functions with `ippiFilterBorder`.
<table>
<thead>
<tr>
<th>Removed from 9.0</th>
<th>Substitution or Workaround</th>
</tr>
</thead>
</table>
| ippiFilterRow32f_16u_AC4R | **NOTE** Starting from Intel IPP 9.0, kernel coefficients in the `ippiFilterBorder` functions are used in direct order, as opposed to the inverse order used in lower versions. Make sure to change the filter order when replacing removed functions with `ippiFilterBorder`.  
  `ippiFilterBorderInit_32f`  
  `+ippiFilterBorder_16u_C4R` (with `filter.height=1`) |
| ippiFilterRow32f_16u_C1R | **NOTE** Starting from Intel IPP 9.0, kernel coefficients in the `ippiFilterBorder` functions are used in direct order, as opposed to the inverse order used in lower versions. Make sure to change the filter order when replacing removed functions with `ippiFilterBorder`.  
  `ippiFilterBorderInit_32f`  
  `+ippiFilterBorder_16u_C1R` (with `filter.height=1`) |
| ippiFilterRow32f_16u_C3R | **NOTE** Starting from Intel IPP 9.0, kernel coefficients in the `ippiFilterBorder` functions are used in direct order, as opposed to the inverse order used in lower versions. Make sure to change the filter order when replacing removed functions with `ippiFilterBorder`.  
  `ippiFilterBorderInit_32f`  
  `+ippiFilterBorder_16u_C3R` (with `filter.height=1`) |
| ippiFilterRow32f_16u_C4R | **NOTE** Starting from Intel IPP 9.0, kernel coefficients in the `ippiFilterBorder` functions are used in direct order, as opposed to the inverse order used in lower versions. Make sure to change the filter order when replacing removed functions with `ippiFilterBorder`.  
  `ippiFilterBorderInit_32f`  
  `+ippiFilterBorder_16u_C4R` (with `filter.height=1`) |
<table>
<thead>
<tr>
<th>Removed from 9.0</th>
<th>Substitution or Workaround</th>
</tr>
</thead>
<tbody>
<tr>
<td>ippiFilterRow32f_8u_AC4R</td>
<td><strong>NOTE</strong> Starting from Intel IPP 9.0, kernel coefficients in the ippiFilterBorder functions are used in direct order, as opposed to the inverse order used in lower versions. Make sure to change the filter order when replacing removed functions with ippiFilterBorder.</td>
</tr>
</tbody>
</table>
|                          | ippiFilterBorderInit_32f
|                          | +ippiFilterBorder_8u_C4R (with filter.height=1)                                         |
| ippiFilterRow32f_8u_C1R  | **NOTE** Starting from Intel IPP 9.0, kernel coefficients in the ippiFilterBorder functions are used in direct order, as opposed to the inverse order used in lower versions. Make sure to change the filter order when replacing removed functions with ippiFilterBorder. |
|                          | ippiFilterBorderInit_32f
|                          | +ippiFilterBorder_8u_C1R (with filter.height=1)                                         |
| ippiFilterRow32f_8u_C3R  | **NOTE** Starting from Intel IPP 9.0, kernel coefficients in the ippiFilterBorder functions are used in direct order, as opposed to the inverse order used in lower versions. Make sure to change the filter order when replacing removed functions with ippiFilterBorder. |
|                          | ippiFilterBorderInit_32f
|                          | +ippiFilterBorder_8u_C3R (with filter.height=1)                                         |
| ippiFilterRow32f_8u_C4R  | **NOTE** Starting from Intel IPP 9.0, kernel coefficients in the ippiFilterBorder functions are used in direct order, as opposed to the inverse order used in lower versions. Make sure to change the filter order when replacing removed functions with ippiFilterBorder. |
|                          | ippiFilterBorderInit_32f
<p>|                          | +ippiFilterBorder_8u_C4R (with filter.height=1)                                         |</p>
<table>
<thead>
<tr>
<th>Removed from 9.0</th>
<th>Substitution or Workaround</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td><strong>NOTE</strong> Starting from Intel IPP 9.0, kernel coefficients in the <code>ippiFilterBorder</code> functions are used in direct order, as opposed to the inverse order used in lower versions. Make sure to change the filter order when replacing removed functions with <code>ippiFilterBorder</code>.</td>
</tr>
<tr>
<td><code>ippiFilterRow_16s_AC4R</code></td>
<td><code>ippiFilterBorderInit_16s</code></td>
</tr>
<tr>
<td></td>
<td>+<code>ippiFilterBorder_16s_AC4R</code> (with <code>filter.height</code>=1)</td>
</tr>
<tr>
<td><code>ippiFilterRow_16s_C1R</code></td>
<td><code>ippiFilterBorderInit_16s</code></td>
</tr>
<tr>
<td></td>
<td>+<code>ippiFilterBorder_16s_C1R</code> (with <code>filter.height</code>=1)</td>
</tr>
<tr>
<td><code>ippiFilterRow_16s_C3R</code></td>
<td><code>ippiFilterBorderInit_16s</code></td>
</tr>
<tr>
<td></td>
<td>+<code>ippiFilterBorder_16s_C3R</code> (with <code>filter.height</code>=1)</td>
</tr>
<tr>
<td><code>ippiFilterRow_16s_C4R</code></td>
<td><code>ippiFilterBorderInit_16s</code></td>
</tr>
<tr>
<td></td>
<td>+<code>ippiFilterBorder_16s_C4R</code> (with <code>filter.height</code>=1)</td>
</tr>
<tr>
<td>Removed from 9.0</td>
<td>Substitution or Workaround</td>
</tr>
<tr>
<td>------------------</td>
<td>---------------------------</td>
</tr>
<tr>
<td><strong>NOTE</strong> Starting from Intel IPP 9.0, kernel coefficients in the <code>ippiFilterBorder</code> functions are used in direct order, as opposed to the inverse order used in lower versions. Make sure to change the filter order when replacing removed functions with <code>ippiFilterBorder</code>.</td>
<td></td>
</tr>
<tr>
<td><code>ippiFilterRow_16u_AC4R</code></td>
<td><code>ippiFilterBorderInit_16s</code> + <code>ippiFilterBorder_16u_C4R</code> (with <code>filter.height=1</code>)</td>
</tr>
<tr>
<td><strong>NOTE</strong> Starting from Intel IPP 9.0, kernel coefficients in the <code>ippiFilterBorder</code> functions are used in direct order, as opposed to the inverse order used in lower versions. Make sure to change the filter order when replacing removed functions with <code>ippiFilterBorder</code>.</td>
<td></td>
</tr>
<tr>
<td><code>ippiFilterRow_16u_C1R</code></td>
<td><code>ippiFilterBorderInit_16s</code> + <code>ippiFilterBorder_16u_C1R</code> (with <code>filter.height=1</code>)</td>
</tr>
<tr>
<td><strong>NOTE</strong> Starting from Intel IPP 9.0, kernel coefficients in the <code>ippiFilterBorder</code> functions are used in direct order, as opposed to the inverse order used in lower versions. Make sure to change the filter order when replacing removed functions with <code>ippiFilterBorder</code>.</td>
<td></td>
</tr>
<tr>
<td><code>ippiFilterRow_16u_C3R</code></td>
<td><code>ippiFilterBorderInit_16s</code> + <code>ippiFilterBorder_16u_C3R</code> (with <code>filter.height=1</code>)</td>
</tr>
<tr>
<td><strong>NOTE</strong> Starting from Intel IPP 9.0, kernel coefficients in the <code>ippiFilterBorder</code> functions are used in direct order, as opposed to the inverse order used in lower versions. Make sure to change the filter order when replacing removed functions with <code>ippiFilterBorder</code>.</td>
<td></td>
</tr>
<tr>
<td><code>ippiFilterRow_16u_C4R</code></td>
<td><code>ippiFilterBorderInit_16s</code> + <code>ippiFilterBorder_16u_C4R</code> (with <code>filter.height=1</code>)</td>
</tr>
</tbody>
</table>
### Removed from 9.0

<table>
<thead>
<tr>
<th>Removed Function</th>
<th>Substitution or Workaround</th>
</tr>
</thead>
<tbody>
<tr>
<td>ippiFilterRow_32f_AC4R</td>
<td>ippiFilterBorderInit_32f +ippiFilterBorder_32f_C4R (with filter.height=1)</td>
</tr>
<tr>
<td>ippiFilterRow_32f_C1R</td>
<td>ippiFilterBorderInit_32f +ippiFilterBorder_32f_C1R (with filter.height=1)</td>
</tr>
<tr>
<td>ippiFilterRow_32f_C3R</td>
<td>ippiFilterBorderInit_32f +ippiFilterBorder_32f_C3R (with filter.height=1)</td>
</tr>
<tr>
<td>ippiFilterRow_32f_C4R</td>
<td>ippiFilterBorderInit_32f +ippiFilterBorder_32f_C4R (with filter.height=1)</td>
</tr>
</tbody>
</table>

**NOTE** Starting from Intel IPP 9.0, kernel coefficients in the ippiFilterBorder functions are used in direct order, as opposed to the inverse order used in lower versions. Make sure to change the filter order when replacing removed functions with ippiFilterBorder.
<table>
<thead>
<tr>
<th>Removed from 9.0</th>
<th>Substitution or Workaround</th>
</tr>
</thead>
</table>
| ippiFilterRow_8u_AC4R | **NOTE** Starting from Intel IPP 9.0, kernel coefficients in the `ippiFilterBorder` functions are used in direct order, as opposed to the inverse order used in lower versions. Make sure to change the filter order when replacing removed functions with `ippiFilterBorder`.  
ippiFilterBorderInit_16s  
+ippiFilterBorder_8u_C4R (with filter.height=1) |
| ippiFilterRow_8u_C1R  | **NOTE** Starting from Intel IPP 9.0, kernel coefficients in the `ippiFilterBorder` functions are used in direct order, as opposed to the inverse order used in lower versions. Make sure to change the filter order when replacing removed functions with `ippiFilterBorder`.  
ippiFilterBorderInit_16s  
+ippiFilterBorder_8u_C1R (with filter.height=1) |
| ippiFilterRow_8u_C3R  | **NOTE** Starting from Intel IPP 9.0, kernel coefficients in the `ippiFilterBorder` functions are used in direct order, as opposed to the inverse order used in lower versions. Make sure to change the filter order when replacing removed functions with `ippiFilterBorder`.  
ippiFilterBorderInit_16s  
+ippiFilterBorder_8u_C3R (with filter.height=1) |
| ippiFilterRow_8u_C4R  | **NOTE** Starting from Intel IPP 9.0, kernel coefficients in the `ippiFilterBorder` functions are used in direct order, as opposed to the inverse order used in lower versions. Make sure to change the filter order when replacing removed functions with `ippiFilterBorder`.  
ippiFilterBorderInit_16s  
+ippiFilterBorder_8u_C4R (with filter.height=1) |
<table>
<thead>
<tr>
<th>Removed from 9.0</th>
<th>Substitution or Workaround</th>
</tr>
</thead>
<tbody>
<tr>
<td>ippiFilterScharrHoriz_32f_C1R</td>
<td>ippiFilterScharrHorizMaskBorder_32f_C1R</td>
</tr>
<tr>
<td>ippiFilterScharrHoriz_8u16s_C1R</td>
<td>ippiConvert_8s16s</td>
</tr>
<tr>
<td></td>
<td>+ippiFilterScharrHorizMaskBorder_16s_C1R</td>
</tr>
<tr>
<td>ippiFilterScharrHoriz_8u16s_C1R</td>
<td>ippiFilterScharrHorizMaskBorder_8u16s_C1R</td>
</tr>
<tr>
<td>ippiFilterScharrVert_32f_C1R</td>
<td>ippiFilterScharrVertMaskBorder_32f_C1R</td>
</tr>
<tr>
<td>ippiFilterScharrVert_8s16s_C1R</td>
<td>ippiConvert_8s16s</td>
</tr>
<tr>
<td></td>
<td>+ippiFilterScharrVertMaskBorder_16s_C1R</td>
</tr>
<tr>
<td>ippiFilterScharrVert_8u16s_C1R</td>
<td>ippiFilterScharrVertMaskBorder_8u16s_C1R</td>
</tr>
<tr>
<td>ippiFilterSharpen_16s_AC4R</td>
<td>ippiFilterSharpenBorder_16s_AC4R</td>
</tr>
<tr>
<td>ippiFilterSharpen_16s_C1R</td>
<td>ippiFilterSharpenBorder_16s_C1R</td>
</tr>
<tr>
<td>ippiFilterSharpen_16s_C3R</td>
<td>ippiFilterSharpenBorder_16s_C3R</td>
</tr>
<tr>
<td>ippiFilterSharpen_16s_C4R</td>
<td>ippiFilterSharpenBorder_16s_C4R</td>
</tr>
<tr>
<td>ippiFilterSharpen_32f_AC4R</td>
<td>ippiFilterSharpenBorder_32f_AC4R</td>
</tr>
<tr>
<td>ippiFilterSharpen_32f_C1R</td>
<td>ippiFilterSharpenBorder_32f_C1R</td>
</tr>
<tr>
<td>ippiFilterSharpen_32f_C3R</td>
<td>ippiFilterSharpenBorder_32f_C3R</td>
</tr>
<tr>
<td>ippiFilterSharpen_32f_C4R</td>
<td>ippiFilterSharpenBorder_32f_C4R</td>
</tr>
<tr>
<td>ippiFilterSharpen_8u_AC4R</td>
<td>ippiFilterSharpenBorder_8u_AC4R</td>
</tr>
<tr>
<td>ippiFilterSharpen_8u_C1R</td>
<td>ippiFilterSharpenBorder_8u_C1R</td>
</tr>
<tr>
<td>ippiFilterSharpen_8u_C3R</td>
<td>ippiFilterSharpenBorder_8u_C3R</td>
</tr>
<tr>
<td>ippiFilterSharpen_8u_C4R</td>
<td>ippiFilterSharpenBorder_8u_C4R</td>
</tr>
<tr>
<td>ippiFilterSobelCross_32f_C1R</td>
<td>ippiFilterSobelCrossBorder_32f_C1R</td>
</tr>
<tr>
<td>ippiFilterSobelCross_8s16s_C1R</td>
<td>ippiConvert_8s32f</td>
</tr>
<tr>
<td></td>
<td>+ippiFilterSobelCrossBorder_32f_C1R</td>
</tr>
<tr>
<td>ippiFilterSobelCross_8u16s_C1R</td>
<td>ippiFilterSobelCrossBorder_8u16s_C1R</td>
</tr>
<tr>
<td>ippiFilterSobelHorizGetBufferSize_32f_C1R</td>
<td>ippiFilterSobelHorizBorderGetBufferSize</td>
</tr>
<tr>
<td>ippiFilterSobelHorizGetBufferSize_8u16s_C1R</td>
<td>ippiFilterSobelHorizBorderGetBufferSize</td>
</tr>
<tr>
<td>Removed from 9.0</td>
<td>Substitution or Workaround</td>
</tr>
<tr>
<td>------------------------------------------------------</td>
<td>-------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>ippiFilterSobelHorizMask_32f_C1R</td>
<td>ippiFilterSobelHorizBorder_32f_C1R</td>
</tr>
<tr>
<td>ippiFilterSobelHorizSecond_32f_C1R</td>
<td>ippiFilterSobelHorizSecondBorder_32f_C1R</td>
</tr>
<tr>
<td>ippiFilterSobelHorizSecond_8s16s_C1R</td>
<td>ippiConvert_8s32f +ippiFilterSobelHorizSecondBorder_32f_C1R</td>
</tr>
<tr>
<td>ippiFilterSobelHorizSecond_8u16s_C1R</td>
<td>ippiFilterSobelHorizSecondBorder_8u16s_C1R</td>
</tr>
<tr>
<td>ippiFilterSobelHoriz_16s_AC4R</td>
<td>Use ippiCopy_16s_C4C1R +ippiFilterSobelHorizBorder_16s_C1R</td>
</tr>
<tr>
<td>ippiFilterSobelHoriz_16s_C1R</td>
<td>ippiFilterSobelHorizBorder_16s_C1R</td>
</tr>
<tr>
<td>ippiFilterSobelHoriz_16s_C3R</td>
<td>Use ippiCopy_16s_C3C1R +ippiFilterSobelHorizBorder_16s_C1R</td>
</tr>
<tr>
<td>ippiFilterSobelHoriz_16s_C4R</td>
<td>Use ippiCopy_16s_C4C1R +ippiFilterSobelHorizBorder_16s_C1R</td>
</tr>
<tr>
<td>ippiFilterSobelHoriz_32f_AC4R</td>
<td>Use ippiCopy_32f_C4C1R +ippiFilterSobelHorizBorder_32f_C1R</td>
</tr>
<tr>
<td>ippiFilterSobelHoriz_32f_C1R</td>
<td>ippiFilterSobelHorizBorder_32f_C1R</td>
</tr>
<tr>
<td>ippiFilterSobelHoriz_32f_C3R</td>
<td>Use ippiCopy_32f_C3C1R +ippiFilterSobelHorizBorder_32f_C1R</td>
</tr>
<tr>
<td>ippiFilterSobelHoriz_32f_C4R</td>
<td>Use ippiCopy_32f_C4C1R +ippiFilterSobelHorizBorder_32f_C1R</td>
</tr>
<tr>
<td>ippiFilterSobelHoriz_8s16s_C1R</td>
<td>Use ippiCopy_8s_C4C1R +ippiFilterSobelHorizBorder_8s16s_C1R</td>
</tr>
<tr>
<td>ippiFilterSobelHoriz_8u16s_C1R</td>
<td>ippiConvert_8u16s +ippiFilterSobelHorizBorder_8u16s_C1R</td>
</tr>
<tr>
<td>ippiFilterSobelHoriz_8u_C1R</td>
<td>Use ippiCopy_8u_C4C1R +ippiFilterSobelHorizBorder_8u16s_C1R</td>
</tr>
<tr>
<td>ippiFilterSobelHoriz_8u_C3R</td>
<td>Use ippiCopy_8u_C3C1R +ippiFilterSobelHorizBorder_8u16s_C1R</td>
</tr>
<tr>
<td>ippiFilterSobelHoriz_8u_C4R</td>
<td>Use ippiCopy_8u_C4C1R +ippiFilterSobelHorizBorder_8u16s_C1R</td>
</tr>
<tr>
<td>ippiFilterSobelVertGetBufferSize_32f_C1R</td>
<td>ippiFilterSobelVertBorderGetBufferSize</td>
</tr>
<tr>
<td>ippiFilterSobelVertGetBufferSize_8u16s_C1R</td>
<td>ippiFilterSobelVertBorderGetBufferSize</td>
</tr>
<tr>
<td>ippiFilterSobelVertMask_32f_C1R</td>
<td>ippiFilterSobelVertBorder_32f_C1R</td>
</tr>
<tr>
<td>ippiFilterSobelVertSecond_32f_C1R</td>
<td>ippiFilterSobelVertSecondBorder_32f_C1R</td>
</tr>
<tr>
<td>Removed from 9.0</td>
<td>Substitution or Workaround</td>
</tr>
<tr>
<td>------------------</td>
<td>-----------------------------</td>
</tr>
<tr>
<td>ippiFilterSobelVertSecond_8s16s_C1R</td>
<td>ippiConvert_8s32f +ippiFilterSobelVertSecondBorder_32f_C1R</td>
</tr>
<tr>
<td>ippiFilterSobelVertSecond_8u16s_C1R</td>
<td>ippiFilterSobelVertSecondBorder_8u16s_C1R</td>
</tr>
<tr>
<td>ippiFilterSobelVert_16s_AC4R</td>
<td>Use ippiCopy_16s_C4C1R +ippiFilterSobelVertBorder_16s_C1R</td>
</tr>
<tr>
<td>ippiFilterSobelVert_16s_C1R</td>
<td>ippiFilterSobelVertBorder_16s_C1R</td>
</tr>
<tr>
<td>ippiFilterSobelVert_16s_C3R</td>
<td>Use ippiCopy_16s_C3C1R +ippiFilterSobelVertBorder_16s_C1R</td>
</tr>
<tr>
<td>ippiFilterSobelVert_16s_C4R</td>
<td>Use ippiCopy_16s_C4C1R +ippiFilterSobelVertBorder_16s_C1R</td>
</tr>
<tr>
<td>ippiFilterSobelVert_32f_AC4R</td>
<td>Use ippiCopy_32f_C4C1R +ippiFilterSobelVertBorder_32f_C1R</td>
</tr>
<tr>
<td>ippiFilterSobelVert_32f_C1R</td>
<td>ippiFilterSobelVertBorder_32f_C1R</td>
</tr>
<tr>
<td>ippiFilterSobelVert_32f_C3R</td>
<td>Use ippiCopy_32f_C3C1R +ippiFilterSobelVertBorder_32f_C1R</td>
</tr>
<tr>
<td>ippiFilterSobelVert_32f_C4R</td>
<td>Use ippiCopy_32f_C4C1R +ippiFilterSobelVertBorder_32f_C1R</td>
</tr>
<tr>
<td>ippiFilterSobelVert_8s16s_C1R</td>
<td>ippiConvert_8s16s +ippiFilterSobelVertBorder_16s_C1R</td>
</tr>
<tr>
<td>ippiFilterSobelVert_8u16s_C1R</td>
<td>ippiFilterSobelVertBorder_8u16s_C1R</td>
</tr>
<tr>
<td>ippiFilterSobelVert_8u_AC4R</td>
<td>Use ippiCopy_8u_C4C1R +ippiFilterSobelVertBorder_8u16s_C1R</td>
</tr>
<tr>
<td>ippiFilterSobelVert_8u_C1R</td>
<td>ippiFilterSobelVertBorder_8u16s_C1R</td>
</tr>
<tr>
<td>ippiFilterSobelVert_8u_C3R</td>
<td>Use ippiCopy_8u_C3C1R +ippiFilterSobelVertBorder_8u16s_C1R</td>
</tr>
<tr>
<td>ippiFilterSobelVert_8u_C4R</td>
<td>Use ippiCopy_8u_C4C1R +ippiFilterSobelVertBorder_8u16s_C1R</td>
</tr>
<tr>
<td>ippiFilter_16s_AC4R</td>
<td>ippiFilterBorderInit_16s +ippiFilterBorder_16s_C4R</td>
</tr>
</tbody>
</table>

**NOTE** Starting from Intel IPP 9.0, kernel coefficients in the ippiFilterBorder functions are used in direct order, as opposed to the inverse order used in lower versions. Make sure to change the filter order when replacing removed functions with ippiFilterBorder.
<table>
<thead>
<tr>
<th>Removed from 9.0</th>
<th>Substitution or Workaround</th>
</tr>
</thead>
<tbody>
<tr>
<td>ippiFilter_16s_C1R</td>
<td>ippiFilterBorderInit_16s&lt;br&gt;+ippiFilterBorder_16s_C1R&lt;br&gt;<strong>NOTE</strong> Starting from Intel IPP 9.0, kernel coefficients in the ippiFilterBorder functions are used in direct order, as opposed to the inverse order used in lower versions. Make sure to change the filter order when replacing removed functions with ippiFilterBorder.</td>
</tr>
<tr>
<td>ippiFilter_16s_C3R</td>
<td>ippiFilterBorderInit_16s&lt;br&gt;+ippiFilterBorder_16s_C3R&lt;br&gt;<strong>NOTE</strong> Starting from Intel IPP 9.0, kernel coefficients in the ippiFilterBorder functions are used in direct order, as opposed to the inverse order used in lower versions. Make sure to change the filter order when replacing removed functions with ippiFilterBorder.</td>
</tr>
<tr>
<td>ippiFilter_16s_C4R</td>
<td>ippiFilterBorderInit_16s&lt;br&gt;+ippiFilterBorder_16s_C4R&lt;br&gt;<strong>NOTE</strong> Starting from Intel IPP 9.0, kernel coefficients in the ippiFilterBorder functions are used in direct order, as opposed to the inverse order used in lower versions. Make sure to change the filter order when replacing removed functions with ippiFilterBorder.</td>
</tr>
<tr>
<td>ippiFilter_16u_AC4R</td>
<td>ippiFilterBorderInit_16s&lt;br&gt;+ippiFilterBorder_16u_C4R&lt;br&gt;<strong>NOTE</strong> Starting from Intel IPP 9.0, kernel coefficients in the ippiFilterBorder functions are used in direct order, as opposed to the inverse order used in lower versions. Make sure to change the filter order when replacing removed functions with ippiFilterBorder.</td>
</tr>
<tr>
<td>ippiFilter_16u_C1R</td>
<td>ippiFilterBorderInit_16s&lt;br&gt;+ippiFilterBorder_16u_C1R&lt;br&gt;<strong>NOTE</strong> Starting from Intel IPP 9.0, kernel coefficients in the ippiFilterBorder functions are used in direct order, as opposed to the inverse order used in lower versions. Make sure to change the filter order when replacing removed functions with ippiFilterBorder.</td>
</tr>
<tr>
<td>Removed from 9.0</td>
<td>Substitution or Workaround</td>
</tr>
<tr>
<td>------------------</td>
<td>-----------------------------</td>
</tr>
<tr>
<td><strong>NOTE</strong> Starting from Intel IPP 9.0, kernel coefficients in the <em>ippiFilterBorder</em> functions are used in direct order, as opposed to the inverse order used in lower versions. Make sure to change the filter order when replacing removed functions with <em>ippiFilterBorder</em>.</td>
<td></td>
</tr>
<tr>
<td><code>ippiFilter_16u_C3R</code></td>
<td><code>ippiFilterBorderInit_16s</code></td>
</tr>
<tr>
<td></td>
<td><code>ippiFilterBorder_16u_C3R</code></td>
</tr>
<tr>
<td><code>ippiFilter_16u_C4R</code></td>
<td><code>ippiFilterBorderInit_16s</code></td>
</tr>
<tr>
<td></td>
<td><code>ippiFilterBorder_16u_C4R</code></td>
</tr>
<tr>
<td><code>ippiFilter_32f_AC4R</code></td>
<td><code>ippiFilterBorderInit_32f</code></td>
</tr>
<tr>
<td></td>
<td><code>ippiFilterBorder_32f_C4R</code></td>
</tr>
<tr>
<td><code>ippiFilter_32f_C1R</code></td>
<td><code>ippiFilterBorderInit_32f</code></td>
</tr>
<tr>
<td></td>
<td><code>ippiFilterBorder_32f_C1R</code></td>
</tr>
</tbody>
</table>
### Removed from 9.0

**ippiFilter_32f_C3R**

**NOTE** Starting from Intel IPP 9.0, kernel coefficients in the `ippiFilterBorder` functions are used in direct order, as opposed to the inverse order used in lower versions. Make sure to change the filter order when replacing removed functions with `ippiFilterBorder`.

+ `ippiFilterBorderInit_32f`
+ `ippiFilterBorder_32f_C3R`

**ippiFilter_32f_C4R**

+ `ippiFilterBorderInit_32f`
+ `ippiFilterBorder_32f_C4R`

**NOTE** Starting from Intel IPP 9.0, kernel coefficients in the `ippiFilterBorder` functions are used in direct order, as opposed to the inverse order used in lower versions. Make sure to change the filter order when replacing removed functions with `ippiFilterBorder`.

**ippiFilter_8u_AC4R**

+ `ippiFilterBorderInit_16s`
+ `ippiFilterBorder_8u_C4R`

**NOTE** Starting from Intel IPP 9.0, kernel coefficients in the `ippiFilterBorder` functions are used in direct order, as opposed to the inverse order used in lower versions. Make sure to change the filter order when replacing removed functions with `ippiFilterBorder`.

**ippiFilter_8u_C1R**

+ `ippiFilterBorderInit_16s`
+ `ippiFilterBorder_8u_C1R`

**NOTE** Starting from Intel IPP 9.0, kernel coefficients in the `ippiFilterBorder` functions are used in direct order, as opposed to the inverse order used in lower versions. Make sure to change the filter order when replacing removed functions with `ippiFilterBorder`. 
<table>
<thead>
<tr>
<th>Removed from 9.0</th>
<th>Substitution or Workaround</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>NOTE</strong> Starting from Intel IPP 9.0, kernel coefficients in the <code>ippiFilterBorder</code> functions are used in direct order, as opposed to the inverse order used in lower versions. Make sure to change the filter order when replacing removed functions with <code>ippiFilterBorder</code>.</td>
<td></td>
</tr>
<tr>
<td><code>ippiFilter_8u_C3R</code></td>
<td><code>ippiFilterBorderInit_16s</code></td>
</tr>
<tr>
<td></td>
<td><code>+ippiFilterBorder_8u_C3R</code></td>
</tr>
<tr>
<td><code>ippiFilter_8u_C4R</code></td>
<td><code>ippiFilterBorderInit_16s</code></td>
</tr>
<tr>
<td></td>
<td><code>+ippiFilterBorder_8u_C4R</code></td>
</tr>
<tr>
<td><code>ippiFilter_Round16s_8u_AC4R</code></td>
<td><code>ippiFilterBorderInit_16s</code></td>
</tr>
<tr>
<td></td>
<td><code>+ippiFilterBorder_8u_C4R</code></td>
</tr>
<tr>
<td><code>ippiFilter_Round16s_8u_C1R</code></td>
<td><code>ippiFilterBorderInit_16s</code></td>
</tr>
<tr>
<td></td>
<td><code>+ippiFilterBorder_8u_C1R</code></td>
</tr>
<tr>
<td>Removed from 9.0</td>
<td>Substitution or Workaround</td>
</tr>
<tr>
<td>----------------------------------</td>
<td>-------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td><strong>NOTE</strong> Starting from Intel IPP 9.0, kernel coefficients in theippiFilterBorder functions are used in direct order, as opposed to the inverse order used in lower versions. Make sure to change the filter order when replacing removed functions with ippiFilterBorder.</td>
<td></td>
</tr>
<tr>
<td>ippiFilter_Round16s_8u_C3R</td>
<td>ippiFilterBorderInit_16s</td>
</tr>
<tr>
<td></td>
<td>+ippiFilterBorder_8u_C3R</td>
</tr>
<tr>
<td>ippiFilter_Round16s_8u_C4R</td>
<td>ippiFilterBorderInit_16s</td>
</tr>
<tr>
<td></td>
<td>+ippiFilterBorder_8u_C4R</td>
</tr>
<tr>
<td>ippiFilter_Round32f_16s_AC4R</td>
<td>ippiFilterBorderInit_32f</td>
</tr>
<tr>
<td></td>
<td>+ippiFilterBorder_16s_C4R</td>
</tr>
<tr>
<td>ippiFilter_Round32f_16s_C1R</td>
<td>ippiFilterBorderInit_32f</td>
</tr>
<tr>
<td></td>
<td>+ippiFilterBorder_16s_C1R</td>
</tr>
<tr>
<td>Removed from 9.0</td>
<td>Substitution or Workaround</td>
</tr>
<tr>
<td>------------------</td>
<td>-----------------------------</td>
</tr>
<tr>
<td>ippiFilter_Round32f_16s_C3R</td>
<td>NOTE Starting from Intel IPP 9.0, kernel coefficients in the ippiFilterBorder functions are used in direct order, as opposed to the inverse order used in lower versions. Make sure to change the filter order when replacing removed functions with ippiFilterBorder.</td>
</tr>
<tr>
<td></td>
<td>ippiFilterBorderInit_32f +ippiFilterBorder_16s_C3R</td>
</tr>
<tr>
<td>ippiFilter_Round32f_16s_C4R</td>
<td>NOTE Starting from Intel IPP 9.0, kernel coefficients in the ippiFilterBorder functions are used in direct order, as opposed to the inverse order used in lower versions. Make sure to change the filter order when replacing removed functions with ippiFilterBorder.</td>
</tr>
<tr>
<td></td>
<td>ippiFilterBorderInit_32f +ippiFilterBorder_16s_C4R</td>
</tr>
<tr>
<td>ippiFilter_Round32f_16u_AC4R</td>
<td>NOTE Starting from Intel IPP 9.0, kernel coefficients in the ippiFilterBorder functions are used in direct order, as opposed to the inverse order used in lower versions. Make sure to change the filter order when replacing removed functions with ippiFilterBorder.</td>
</tr>
<tr>
<td></td>
<td>ippiFilterBorderInit_32f +ippiFilterBorder_16u_C4R</td>
</tr>
<tr>
<td>ippiFilter_Round32f_16u_C1R</td>
<td>NOTE Starting from Intel IPP 9.0, kernel coefficients in the ippiFilterBorder functions are used in direct order, as opposed to the inverse order used in lower versions. Make sure to change the filter order when replacing removed functions with ippiFilterBorder.</td>
</tr>
<tr>
<td></td>
<td>ippiFilterBorderInit_32f +ippiFilterBorder_16u_C1R</td>
</tr>
<tr>
<td>Removed from 9.0</td>
<td>Substitution or Workaround</td>
</tr>
<tr>
<td>------------------------------------------------------</td>
<td>----------------------------------------------------------------</td>
</tr>
<tr>
<td><strong>NOTE</strong> Starting from Intel IPP 9.0, kernel</td>
<td></td>
</tr>
<tr>
<td>coefficients in the <code>ippiFilterBorder</code> functions</td>
<td></td>
</tr>
<tr>
<td>are used in direct order, as opposed to the</td>
<td></td>
</tr>
<tr>
<td>inverse order used in lower versions. Make sure</td>
<td></td>
</tr>
<tr>
<td>to change the filter order when replacing</td>
<td></td>
</tr>
<tr>
<td>removed functions with <code>ippiFilterBorder</code>.</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td><code>ippiFilter_Round32f_16u_C3R</code></td>
<td><code>ippiFilterBorderInit_32f</code></td>
</tr>
<tr>
<td></td>
<td><code>ippiFilterBorder_16u_C3R</code></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>NOTE</strong> Starting from Intel IPP 9.0, kernel</td>
<td></td>
</tr>
<tr>
<td>coefficients in the <code>ippiFilterBorder</code> functions</td>
<td></td>
</tr>
<tr>
<td>are used in direct order, as opposed to the</td>
<td></td>
</tr>
<tr>
<td>inverse order used in lower versions. Make sure</td>
<td></td>
</tr>
<tr>
<td>to change the filter order when replacing</td>
<td></td>
</tr>
<tr>
<td>removed functions with <code>ippiFilterBorder</code>.</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td><code>ippiFilter_Round32f_16u_C4R</code></td>
<td><code>ippiFilterBorderInit_32f</code></td>
</tr>
<tr>
<td></td>
<td><code>ippiFilterBorder_16u_C4R</code></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>NOTE</strong> Starting from Intel IPP 9.0, kernel</td>
<td></td>
</tr>
<tr>
<td>coefficients in the <code>ippiFilterBorder</code> functions</td>
<td></td>
</tr>
<tr>
<td>are used in direct order, as opposed to the</td>
<td></td>
</tr>
<tr>
<td>inverse order used in lower versions. Make sure</td>
<td></td>
</tr>
<tr>
<td>to change the filter order when replacing</td>
<td></td>
</tr>
<tr>
<td>removed functions with <code>ippiFilterBorder</code>.</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td><code>ippiFilter_Round32f_8u_AC4R</code></td>
<td><code>ippiFilterBorderInit_32f</code></td>
</tr>
<tr>
<td></td>
<td><code>ippiFilterBorder_8u_C4R</code></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>NOTE</strong> Starting from Intel IPP 9.0, kernel</td>
<td></td>
</tr>
<tr>
<td>coefficients in the <code>ippiFilterBorder</code> functions</td>
<td></td>
</tr>
<tr>
<td>are used in direct order, as opposed to the</td>
<td></td>
</tr>
<tr>
<td>inverse order used in lower versions. Make sure</td>
<td></td>
</tr>
<tr>
<td>to change the filter order when replacing</td>
<td></td>
</tr>
<tr>
<td>removed functions with <code>ippiFilterBorder</code>.</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td><code>ippiFilter_Round32f_8u_C1R</code></td>
<td><code>ippiFilterBorderInit_32f</code></td>
</tr>
<tr>
<td></td>
<td><code>ippiFilterBorder_8u_C1R</code></td>
</tr>
<tr>
<td>Removed from 9.0</td>
<td>Substitution or Workaround</td>
</tr>
<tr>
<td>----------------------------------------</td>
<td>-------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td><strong>NOTE</strong> Starting from Intel IPP 9.0, kernel coefficients in the ippiFilterBorder functions are used in direct order, as opposed to the inverse order used in lower versions. Make sure to change the filter order when replacing removed functions with ippiFilterBorder.</td>
<td></td>
</tr>
</tbody>
</table>
|ippiFilter_Round32f_8u_C3R             |ippiFilterBorderInit_32f  
+ippiFilterBorder_8u_C3R                      |
| **NOTE** Starting from Intel IPP 9.0, kernel coefficients in the ippiFilterBorder functions are used in direct order, as opposed to the inverse order used in lower versions. Make sure to change the filter order when replacing removed functions with ippiFilterBorder. |
|ippiFilter_Round32f_8u_C4R             |ippiFilterBorderInit_32f  
+ippiFilterBorder_8u_C4R                      |
| **NOTE** Starting from Intel IPP 9.0, kernel coefficients in the ippiFilterBorder functions are used in direct order, as opposed to the inverse order used in lower versions. Make sure to change the filter order when replacing removed functions with ippiFilterBorder. |
|ippiFilter_Round32s_16s_AC4R           |ippiFilterBorderInit_16s  
+ippiFilterBorder_16s_C4R                      |
| **NOTE** Starting from Intel IPP 9.0, kernel coefficients in the ippiFilterBorder functions are used in direct order, as opposed to the inverse order used in lower versions. Make sure to change the filter order when replacing removed functions with ippiFilterBorder. |
|ippiFilter_Round32s_16s_C1R            |ippiFilterBorderInit_16s  
+ippiFilterBorder_16s_C1R                      |
<table>
<thead>
<tr>
<th>Removed from 9.0</th>
<th>Substitution or Workaround</th>
</tr>
</thead>
</table>
| ippiFilter_Round32s_16s_C3R | **NOTE** Starting from Intel IPP 9.0, kernel coefficients in the `ippiFilterBorder` functions are used in direct order, as opposed to the inverse order used in lower versions. Make sure to change the filter order when replacing removed functions with `ippiFilterBorder`.  
`ippiFilterBorderInit_16s`  
`+ippiFilterBorder_16s_C3R` |
| ippiFilter_Round32s_16s_C4R | **NOTE** Starting from Intel IPP 9.0, kernel coefficients in the `ippiFilterBorder` functions are used in direct order, as opposed to the inverse order used in lower versions. Make sure to change the filter order when replacing removed functions with `ippiFilterBorder`.  
`ippiFilterBorderInit_16s`  
`+ippiFilterBorder_16s_C4R` |
| ippiFilter_Round32s_16u_AC4R | **NOTE** Starting from Intel IPP 9.0, kernel coefficients in the `ippiFilterBorder` functions are used in direct order, as opposed to the inverse order used in lower versions. Make sure to change the filter order when replacing removed functions with `ippiFilterBorder`.  
`ippiFilterBorderInit_16s`  
`+ippiFilterBorder_16u_C4R` |
| ippiFilter_Round32s_16u_C1R | **NOTE** Starting from Intel IPP 9.0, kernel coefficients in the `ippiFilterBorder` functions are used in direct order, as opposed to the inverse order used in lower versions. Make sure to change the filter order when replacing removed functions with `ippiFilterBorder`.  
`ippiFilterBorderInit_16s`  
`+ippiFilterBorder_16u_C1R` |
<table>
<thead>
<tr>
<th>Removed from 9.0</th>
<th>Substitution or Workaround</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>NOTE</strong> Starting from Intel IPP 9.0, kernel coefficients in the <code>ippiFilterBorder</code> functions are used in direct order, as opposed to the inverse order used in lower versions. Make sure to change the filter order when replacing removed functions with <code>ippiFilterBorder</code>.</td>
<td></td>
</tr>
</tbody>
</table>
| `ippiFilter_Round32s_16u_C3R`                      | `ippiFilterBorderInit_16s`  
+`ippiFilterBorder_16u_C3R` |
| `ippiFilter_Round32s_16u_C4R`                      | `ippiFilterBorderInit_16s`  
+`ippiFilterBorder_16u_C4R` |
<p>| <strong>NOTE</strong> Starting from Intel IPP 9.0, kernel coefficients in the <code>ippiFilterBorder</code> functions are used in direct order, as opposed to the inverse order used in lower versions. Make sure to change the filter order when replacing removed functions with <code>ippiFilterBorder</code>. |
| <code>ippiGetCentralMoment_64s</code>                         | Use Moments_64f |
| <code>ippiGetHuMoments_64s</code>                              | Use Moments_64f |
| <code>ippiGetNormalizedCentralMoment_64s</code>               | Use Moments_64f |
| <code>ippiGetNormalizedSpatialMoment_64s</code>               | Use Moments_64f |
| <code>ippiGetRotateBound</code>                                | N/A |
| <code>ippiGetRotateQuad</code>                                 | N/A |
| <code>ippiGetShearBound</code>                                 | N/A |
| <code>ippiGetShearQuad</code>                                  | N/A |
| <code>ippiGetSpatialMoment_64s</code>                          | Use Moments_64f |
| <code>ippiHistogramEven_16s_AC4R</code>                        | <code>ippiHistogram_16s_C4R</code> |
| <code>ippiHistogramEven_16s_C1R</code>                         | <code>ippiHistogram_16s_C1R</code> |
| <code>ippiHistogramEven_16s_C3R</code>                         | <code>ippiHistogram_16s_C3R</code> |</p>
<table>
<thead>
<tr>
<th>Removed from 9.0</th>
<th>Substitution or Workaround</th>
</tr>
</thead>
<tbody>
<tr>
<td>ippiHistogramEven_16s_C4R</td>
<td>ippiHistogram_16s_C4R</td>
</tr>
<tr>
<td>ippiHistogramEven_16u_AC4R</td>
<td>ippiHistogram_16u_C4R</td>
</tr>
<tr>
<td>ippiHistogramEven_16u_C1R</td>
<td>ippiHistogram_16u_C1R</td>
</tr>
<tr>
<td>ippiHistogramEven_16u_C3R</td>
<td>ippiHistogram_16u_C3R</td>
</tr>
<tr>
<td>ippiHistogramEven_16u_C4R</td>
<td>ippiHistogram_16u_C4R</td>
</tr>
<tr>
<td>ippiHistogramEven_8u_AC4R</td>
<td>ippiHistogram_8u_C4R</td>
</tr>
<tr>
<td>ippiHistogramEven_8u_C1R</td>
<td>ippiHistogram_8u_C1R</td>
</tr>
<tr>
<td>ippiHistogramEven_8u_C3R</td>
<td>ippiHistogram_8u_C3R</td>
</tr>
<tr>
<td>ippiHistogramEven_8u_C4R</td>
<td>ippiHistogram_8u_C4R</td>
</tr>
<tr>
<td>ippiHistogramRange_16s_AC4R</td>
<td>ippiHistogram_16s_C4R</td>
</tr>
<tr>
<td>ippiHistogramRange_16s_C1R</td>
<td>ippiHistogram_16s_C1R</td>
</tr>
<tr>
<td>ippiHistogramRange_16s_C3R</td>
<td>ippiHistogram_16s_C3R</td>
</tr>
<tr>
<td>ippiHistogramRange_16s_C4R</td>
<td>ippiHistogram_16s_C4R</td>
</tr>
<tr>
<td>ippiHistogramRange_16u_AC4R</td>
<td>ippiHistogram_16u_C4R</td>
</tr>
<tr>
<td>ippiHistogramRange_16u_C1R</td>
<td>ippiHistogram_16u_C1R</td>
</tr>
<tr>
<td>ippiHistogramRange_16u_C3R</td>
<td>ippiHistogram_16u_C3R</td>
</tr>
<tr>
<td>ippiHistogramRange_16u_C4R</td>
<td>ippiHistogram_16u_C4R</td>
</tr>
<tr>
<td>ippiHistogramRange_32f_AC4R</td>
<td>ippiHistogram_32f_C4R</td>
</tr>
<tr>
<td>ippiHistogramRange_32f_C1R</td>
<td>ippiHistogram_32f_C1R</td>
</tr>
<tr>
<td>ippiHistogramRange_32f_C3R</td>
<td>ippiHistogram_32f_C3R</td>
</tr>
<tr>
<td>ippiHistogramRange_32f_C4R</td>
<td>ippiHistogram_32f_C4R</td>
</tr>
<tr>
<td>ippiHistogramRange_8u_AC4R</td>
<td>ippiHistogram_8u_C4R</td>
</tr>
<tr>
<td>ippiHistogramRange_8u_C1R</td>
<td>ippiHistogram_8u_C1R</td>
</tr>
<tr>
<td>ippiHistogramRange_8u_C3R</td>
<td>ippiHistogram_8u_C3R</td>
</tr>
<tr>
<td>ippiHistogramRange_8u_C4R</td>
<td>ippiHistogram_8u_C4R</td>
</tr>
<tr>
<td>ippiImageJaehne_32s_AC4R</td>
<td>ippiConvert_32f32s</td>
</tr>
<tr>
<td>ippiImageJaehne_32s_C1R</td>
<td>ippiConvert_32f32s</td>
</tr>
<tr>
<td>ippiImageJaehne_32s_C3R</td>
<td>ippiConvert_32f32s</td>
</tr>
</tbody>
</table>
## Appendix C: Removed Functions for Image and Video Processing

<table>
<thead>
<tr>
<th>Removed from 9.0</th>
<th>Substitution or Workaround</th>
</tr>
</thead>
<tbody>
<tr>
<td>ippiImageJaehne_32s_C4R</td>
<td>ippiImageJaehne_32f_C4R +ippiConvert_32f32s</td>
</tr>
<tr>
<td>ippiImageJaehne_8s_AC4R</td>
<td>ippiImageJaehne_16s_C4RippiConvert_16s8s</td>
</tr>
<tr>
<td>ippiImageJaehne_8s_C1R</td>
<td>ippiImageJaehne_16s_C1RippiConvert_16s8s</td>
</tr>
<tr>
<td>ippiImageJaehne_8s_C3R</td>
<td>ippiImageJaehne_16s_C3RippiConvert_16s8s</td>
</tr>
<tr>
<td>ippiImageJaehne_8s_C4R</td>
<td>ippiImageJaehne_16s_C4RippiConvert_16s8s</td>
</tr>
<tr>
<td>ippiImageRamp_32s_AC4R</td>
<td>ippiImageRamp_32f_C1RippiConvert_32f32s</td>
</tr>
<tr>
<td>ippiImageRamp_32s_C1R</td>
<td>ippiImageJaehne_32f_C1R +ippiConvert_32f32s</td>
</tr>
<tr>
<td>ippiImageRamp_32s_C3R</td>
<td>ippiImageJaehne_32f_C3R +ippiConvert_32f32s</td>
</tr>
<tr>
<td>ippiImageRamp_32s_C4R</td>
<td>ippiImageJaehne_32f_C4R +ippiConvert_32f32s</td>
</tr>
<tr>
<td>ippiImageRamp_8s_AC4R</td>
<td>ippiImageRamp_16s_C4RippiConvert_16s8s</td>
</tr>
<tr>
<td>ippiImageRamp_8s_C1R</td>
<td>ippiImageRamp_16s_C1RippiConvert_16s8s</td>
</tr>
<tr>
<td>ippiImageRamp_8s_C3R</td>
<td>ippiImageRamp_16s_C3RippiConvert_16s8s</td>
</tr>
<tr>
<td>ippiImageRamp_8s_C4R</td>
<td>ippiImageRamp_16s_C4RippiConvert_16s8s</td>
</tr>
<tr>
<td>ippiLBPImage3x3_32f8u_C1R</td>
<td>ippiLBPImageMode3x3_32f8u_C1R</td>
</tr>
<tr>
<td>ippiLBPImage3x3_8u_C1R</td>
<td>ippiLBPImageMode3x3_8u_C1R</td>
</tr>
<tr>
<td>ippiLBPImage5x5_32f16u_C1R</td>
<td>ippiLBPImageMode5x5_32f16u_C1R</td>
</tr>
<tr>
<td>ippiLBPImage5x5_32f8u_C1R</td>
<td>ippiLBPImageMode5x5_32f8u_C1R</td>
</tr>
<tr>
<td>ippiLBPImage5x5_8u16u_C1R</td>
<td>ippiLBPImageMode5x5_8u16u_C1R</td>
</tr>
<tr>
<td>ippiLBPImage5x5_8u_C1R</td>
<td>ippiLBPImageMode5x5_8u_C1R</td>
</tr>
<tr>
<td>ippiLUTPaletteSwap_16u_C3A0C4R</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiLUTPaletteSwap_8u_C3A0C4R</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiLUTPalette_16u_C3A0C4R</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiLUTPalette_8u_C3A0C4R</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiLUT_Cubic</td>
<td>Use ippiLUT</td>
</tr>
<tr>
<td>ippiLUT_Linear</td>
<td>Use ippiLUT</td>
</tr>
<tr>
<td>ippiMagnitudePack_16s_C1RSfs</td>
<td>ippiConvert_16s32fippiMagnitudePack_32f</td>
</tr>
<tr>
<td>ippiMagnitudePack_16s_C3RSfs</td>
<td>ippiConvert_16s32fippiMagnitudePack_32f</td>
</tr>
<tr>
<td>Removed from 9.0</td>
<td>Substitution or Workaround</td>
</tr>
<tr>
<td>----------------------------------------------</td>
<td>------------------------------------------------------------------</td>
</tr>
<tr>
<td>ippiMagnitudePack_32s_C1RSfs</td>
<td>ippiConvert_32s32f+ippiMagnitudePack_32f</td>
</tr>
<tr>
<td>ippiMagnitudePack_32s_C3RSfs</td>
<td>ippiConvert_32s32f+ippiMagnitudePack_32f</td>
</tr>
<tr>
<td>ippiMagnitude_16sc16s_C1RSfs</td>
<td>ippiConvert_16s32f+ippiMagnitude_32f</td>
</tr>
<tr>
<td>ippiMagnitude_16sc16s_C3RSfs</td>
<td>ippiConvert_16s32f+ippiMagnitude_32f</td>
</tr>
<tr>
<td>ippiMagnitude_16uc16u_C1RSfs</td>
<td>ippiConvert_16u32f+ippiMagnitude_32f</td>
</tr>
<tr>
<td>ippiMagnitude_16uc16u_C3RSfs</td>
<td>ippiConvert_16u32f+ippiMagnitude_32f</td>
</tr>
<tr>
<td>ippiMagnitude_32sc32s_C1RSfs</td>
<td>ippiConvert_32s32f+ippiMagnitude_32f</td>
</tr>
<tr>
<td>ippiMagnitude_32sc32s_C3RSfs</td>
<td>ippiConvert_32s32f+ippiMagnitude_32f</td>
</tr>
<tr>
<td>ippiMean_16s_AC4R</td>
<td>ippiMean_16s_C4R</td>
</tr>
<tr>
<td>ippiMean_16u_AC4R</td>
<td>ippiMean_16u_C4R</td>
</tr>
<tr>
<td>ippiMean_32f_AC4R</td>
<td>ippiMean_32f_C4R</td>
</tr>
<tr>
<td>ippiMean_8u_AC4R</td>
<td>ippiMean_8u_C4R</td>
</tr>
<tr>
<td>ippiMomentFree_64f</td>
<td>ippiFree</td>
</tr>
<tr>
<td>ippiMomentFree_64s</td>
<td>Use Moments_64f</td>
</tr>
<tr>
<td>ippiMomentGetStateSize_64s</td>
<td>Use Moments_64f</td>
</tr>
<tr>
<td>ippiMomentInitAlloc_64f</td>
<td>ippiMomentsGetSize+ippiMalloc+ippiMomentsInit</td>
</tr>
<tr>
<td>ippiMomentInitAlloc_64s</td>
<td>Use Moments_64f</td>
</tr>
<tr>
<td>ippiMomentInit_64s</td>
<td>Use Moments_64f</td>
</tr>
<tr>
<td>ippiMoments64s_16u_AC4R</td>
<td>Use Moments_64f</td>
</tr>
<tr>
<td>ippiMoments64s_16u_C1R</td>
<td>Use Moments_64f</td>
</tr>
<tr>
<td>ippiMoments64s_16u_C3R</td>
<td>Use Moments_64f</td>
</tr>
<tr>
<td>ippiMoments64s_8u_AC4R</td>
<td>Use Moments_64f</td>
</tr>
<tr>
<td>ippiMoments64s_8u_C1R</td>
<td>Use Moments_64f</td>
</tr>
<tr>
<td>ippiMoments64s_8u_C3R</td>
<td>Use Moments_64f</td>
</tr>
<tr>
<td>ippiMulC_16sc_AC4IRSfs</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiMulC_16sc_AC4RSfs</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiMulC_16sc_C1IRSfs</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiMulC_16sc_C1RSfs</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiMulC_16sc_C3IRSfs</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiMulC_16sc_C3RSfs</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiMulC_16sc_C3RSfs</td>
<td>N/A</td>
</tr>
<tr>
<td>Removed from 9.0</td>
<td>Substitution or Workaround</td>
</tr>
<tr>
<td>----------------------------------------</td>
<td>----------------------------</td>
</tr>
<tr>
<td>ippiMulC_32fc_AC4IR</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiMulC_32fc_AC4R</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiMulC_32fc_C1IR</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiMulC_32fc_C1R</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiMulC_32fc_C3IR</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiMulC_32fc_C3R</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiMulC_32sc_AC4IRSfs</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiMulC_32sc_AC4RSfs</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiMulC_32sc_C1IRSfs</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiMulC_32sc_C1RSfs</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiMulC_32sc_C3IRSfs</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiMulC_32sc_C3RSfs</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiMulPack_16s_AC4IRSfs</td>
<td>ippiConvert_16s32f+ippiMulPack_32f</td>
</tr>
<tr>
<td>ippiMulPack_16s_AC4RSfs</td>
<td>ippiConvert_16s32f+ippiMulPack_32f</td>
</tr>
<tr>
<td>ippiMulPack_16s_C1IRSfs</td>
<td>ippiConvert_16s32f+ippiMulPack_32f</td>
</tr>
<tr>
<td>ippiMulPack_16s_C1RSfs</td>
<td>ippiConvert_16s32f+ippiMulPack_32f</td>
</tr>
<tr>
<td>ippiMulPack_16s_C3IRSfs</td>
<td>ippiConvert_16s32f+ippiMulPack_32f</td>
</tr>
<tr>
<td>ippiMulPack_16s_C3RSfs</td>
<td>ippiConvert_16s32f+ippiMulPack_32f</td>
</tr>
<tr>
<td>ippiMulPack_16s_C4IRSfs</td>
<td>ippiConvert_16s32f+ippiMulPack_32f</td>
</tr>
<tr>
<td>ippiMulPack_16s_C4RSfs</td>
<td>ippiConvert_16s32f+ippiMulPack_32f</td>
</tr>
<tr>
<td>ippi MulPack_32s_AC4IRSfs</td>
<td>ippiConvert_16s32f+ippiMulPack_32f</td>
</tr>
<tr>
<td>ippiMulPack_32s_AC4RSfs</td>
<td>ippiConvert_16s32f+ippiMulPack_32f</td>
</tr>
<tr>
<td>ippiMulPack_32s_C1IRSfs</td>
<td>ippiConvert_16s32f+ippiMulPack_32f</td>
</tr>
<tr>
<td>ippiMulPack_32s_C1RSfs</td>
<td>ippiConvert_16s32f+ippiMulPack_32f</td>
</tr>
<tr>
<td>ippiMulPack_32s_C3IRSfs</td>
<td>ippiConvert_16s32f+ippiMulPack_32f</td>
</tr>
<tr>
<td>ippiMulPack_32s_C3RSfs</td>
<td>ippiConvert_16s32f+ippiMulPack_32f</td>
</tr>
<tr>
<td>ippiMulPack_32s_C4IRSfs</td>
<td>ippiConvert_16s32f+ippiMulPack_32f</td>
</tr>
<tr>
<td>ippiMulPack_32s_C4RSfs</td>
<td>ippiConvert_16s32f+ippiMulPack_32f</td>
</tr>
<tr>
<td>ippiMul_16sc_AC4IRSfs</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiMul_16sc_AC4RSfs</td>
<td>N/A</td>
</tr>
<tr>
<td>Removed from 9.0</td>
<td>Substitution or Workaround</td>
</tr>
<tr>
<td>----------------------------------</td>
<td>-----------------------------</td>
</tr>
<tr>
<td>ippiMul_16sc_C1IRSfs</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiMul_16sc_C1RSfs</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiMul_16sc_C3IRSfs</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiMul_16sc_C3RSfs</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiMul_32fc_AC4IR</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiMul_32fc_AC4R</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiMul_32fc_C1IR</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiMul_32fc_C1R</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiMul_32fc_C3IR</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiMul_32fc_C3R</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiMul_32sc_AC4IRSfs</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiMul_32sc_AC4RSfs</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiMul_32sc_C1IRSfs</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiMul_32sc_C1RSfs</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiMul_32sc_C3IRSfs</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiMul_32sc_C3RSfs</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiNormDiff_Inf_16s_AC4R</td>
<td>ippiNormDiff_Inf_16s_C4R</td>
</tr>
<tr>
<td>ippiNormDiff_Inf_16u_AC4R</td>
<td>ippiNormDiff_Inf_16u_C4R</td>
</tr>
<tr>
<td>ippiNormDiff_Inf_32f_AC4R</td>
<td>ippiNormDiff_Inf_32f_C4R</td>
</tr>
<tr>
<td>ippiNormDiff_Inf_8u_AC4R</td>
<td>ippiNormDiff_Inf_8u_C4R</td>
</tr>
<tr>
<td>ippiNormDiff_L1_16s_AC4R</td>
<td>ippiNormDiff_L1_16s_C4R</td>
</tr>
<tr>
<td>ippiNormDiff_L1_16u_AC4R</td>
<td>ippiNormDiff_L1_16u_C4R</td>
</tr>
<tr>
<td>ippiNormDiff_L1_32f_AC4R</td>
<td>ippiNormDiff_L1_32f_C4R</td>
</tr>
<tr>
<td>ippiNormDiff_L1_8u_AC4R</td>
<td>ippiNormDiff_L1_8u_C4R</td>
</tr>
<tr>
<td>ippiNormDiff_L2_16s_AC4R</td>
<td>ippiNormDiff_L2_16s_C4R</td>
</tr>
<tr>
<td>ippiNormDiff_L2_16u_AC4R</td>
<td>ippiNormDiff_L2_16u_C4R</td>
</tr>
<tr>
<td>ippiNormDiff_L2_32f_AC4R</td>
<td>ippiNormDiff_L2_32f_C4R</td>
</tr>
<tr>
<td>ippiNormDiff_L2_8u_AC4R</td>
<td>ippiNormDiff_L2_8u_C4R</td>
</tr>
<tr>
<td>ippiNormRel_Inf_16s_AC4R</td>
<td>ippiNormRel_Inf_16s_C4R</td>
</tr>
<tr>
<td>ippiNormRel_Inf_16u_AC4R</td>
<td>ippiNormRel_Inf_16u_C4R</td>
</tr>
</tbody>
</table>
## Removed Functions for Image and Video Processing

<table>
<thead>
<tr>
<th>Removed from 9.0</th>
<th>Substitution or Workaround</th>
</tr>
</thead>
<tbody>
<tr>
<td>ippiNormRel_Inf_32f_AC4R</td>
<td>ippiNormRel_Inf_32f_C4R</td>
</tr>
<tr>
<td>ippiNormRel_Inf_8u_AC4R</td>
<td>ippiNormRel_Inf_8u_C4R</td>
</tr>
<tr>
<td>ippiNormRel_L1_16s_AC4R</td>
<td>ippiNormRel_L1_16s_C4R</td>
</tr>
<tr>
<td>ippiNormRel_L1_16u_AC4R</td>
<td>ippiNormRel_L1_16u_C4R</td>
</tr>
<tr>
<td>ippiNormRel_L1_32f_AC4R</td>
<td>ippiNormRel_L1_32f_C4R</td>
</tr>
<tr>
<td>ippiNormRel_L1_8u_AC4R</td>
<td>ippiNormRel_L1_8u_C4R</td>
</tr>
<tr>
<td>ippiNormRel_L2_16s_AC4R</td>
<td>ippiNormRel_L2_16s_C4R</td>
</tr>
<tr>
<td>ippiNormRel_L2_16u_AC4R</td>
<td>ippiNormRel_L2_16u_C4R</td>
</tr>
<tr>
<td>ippiNormRel_L2_32f_AC4R</td>
<td>ippiNormRel_L2_32f_C4R</td>
</tr>
<tr>
<td>ippiNormRel_L2_8u_AC4R</td>
<td>ippiNormRel_L2_8u_C4R</td>
</tr>
<tr>
<td>ippiNorm_Inf_16s_AC4R</td>
<td>ippiNorm_Inf_16s_C4R</td>
</tr>
<tr>
<td>ippiNorm_Inf_16u_AC4R</td>
<td>ippiNorm_Inf_16u_C4R</td>
</tr>
<tr>
<td>ippiNorm_Inf_32f_AC4R</td>
<td>ippiNorm_Inf_32f_C4R</td>
</tr>
<tr>
<td>ippiNorm_Inf_32s_C1R</td>
<td>ippiConvert_32s32f+ippiNorm_Inf_32f</td>
</tr>
<tr>
<td>ippiNorm_Inf_8u_AC4R</td>
<td>ippiNorm_Inf_8u_C4R</td>
</tr>
<tr>
<td>ippiNorm_L1_16s_AC4R</td>
<td>ippiNorm_L1_16s_C4R</td>
</tr>
<tr>
<td>ippiNorm_L1_16u_AC4R</td>
<td>ippiNorm_L1_16u_C4R</td>
</tr>
<tr>
<td>ippiNorm_L1_32f_AC4R</td>
<td>ippiNorm_L1_32f_C4R</td>
</tr>
<tr>
<td>ippiNorm_L1_8u_AC4R</td>
<td>ippiNorm_L1_8u_C4R</td>
</tr>
<tr>
<td>ippiNorm_L2_16s_AC4R</td>
<td>ippiNorm_L2_16s_C4R</td>
</tr>
<tr>
<td>ippiNorm_L2_16u_AC4R</td>
<td>ippiNorm_L2_16u_C4R</td>
</tr>
<tr>
<td>ippiNorm_L2_32f_AC4R</td>
<td>ippiNorm_L2_32f_C4R</td>
</tr>
<tr>
<td>ippiNorm_L2_8u_AC4R</td>
<td>ippiNorm_L2_8u_C4R</td>
</tr>
<tr>
<td>ippiPackToCplxExtend_32s32sc_C1R</td>
<td>ippiConvert_32s32f+ippiPackToCplxExtend_32f</td>
</tr>
<tr>
<td>ippiPhasePack_16s_C1RSfs</td>
<td>ippiConvert_16s32f+ippiPhasePack_32f</td>
</tr>
<tr>
<td>ippiPhasePack_16s_C3RSfs</td>
<td>ippiConvert_16s32f+ippiPhasePack_32f</td>
</tr>
<tr>
<td>ippiPhasePack_32s_C1RSfs</td>
<td>ippiConvert_32s32f+ippiPhasePack_32f</td>
</tr>
<tr>
<td>ippiPhasePack_32s_C3RSfs</td>
<td>ippiConvert_32s32f+ippiPhasePack_32f</td>
</tr>
<tr>
<td>ippiPhase_16sc16s_C1RSfs</td>
<td>ippiConvert_16s32f+ippiPhase_32f</td>
</tr>
<tr>
<td>Removed from 9.0</td>
<td>Substitution or Workaround</td>
</tr>
<tr>
<td>------------------------------------------------------</td>
<td>----------------------------------------------------------------</td>
</tr>
<tr>
<td>ippiPhase_16sc16s_C3RSfs</td>
<td>ippiConvert_16s32f+ippiPhase_32f</td>
</tr>
<tr>
<td>ippiPhase_16uc16u_C1RSfs</td>
<td>ippiConvert_16u32f+ippiPhase_32f</td>
</tr>
<tr>
<td>ippiPhase_16uc16u_C3RSfs</td>
<td>ippiConvert_16u32f+ippiPhase_32f</td>
</tr>
<tr>
<td>ippiPhase_32sc32s_C1RSfs</td>
<td>ippiConvert_32s32f+ippiPhase_32f</td>
</tr>
<tr>
<td>ippiPhase_32sc32s_C3RSfs</td>
<td>ippiConvert_32s32f+ippiPhase_32f</td>
</tr>
<tr>
<td>ippiPolarToCart_16sc_C1R</td>
<td></td>
</tr>
<tr>
<td>ippiPolarToCart_16sc_C3R</td>
<td></td>
</tr>
<tr>
<td>ippiPolarToCart_32f32fc_P2C1R</td>
<td>Use ippsCplxToReal_32fc with loop by row</td>
</tr>
<tr>
<td>ippiPolarToCart_32sc_C1R</td>
<td></td>
</tr>
<tr>
<td>ippiPolarToCart_32sc_C3R</td>
<td></td>
</tr>
<tr>
<td>ippiRShiftC_8s_AC4IR</td>
<td>Convert to other data type</td>
</tr>
<tr>
<td>ippiRShiftC_8s_AC4R</td>
<td>Convert to other data type</td>
</tr>
<tr>
<td>ippiRShiftC_8s_C1IR</td>
<td>Convert to other data type</td>
</tr>
<tr>
<td>ippiRShiftC_8s_C1R</td>
<td>Convert to other data type</td>
</tr>
<tr>
<td>ippiRShiftC_8s_C3IR</td>
<td>Convert to other data type</td>
</tr>
<tr>
<td>ippiRShiftC_8s_C3R</td>
<td>Convert to other data type</td>
</tr>
<tr>
<td>ippiRShiftC_8s_C4IR</td>
<td>Convert to other data type</td>
</tr>
<tr>
<td>ippiRShiftC_8s_C4R</td>
<td>Convert to other data type</td>
</tr>
<tr>
<td>ippiRemap_16s_P3R</td>
<td>Use ippiRemap_XX_C1R for each plane</td>
</tr>
<tr>
<td>ippiRemap_16s_P4R</td>
<td>Use ippiRemap_XX_C1R for each plane</td>
</tr>
<tr>
<td>ippiRemap_16u_P3R</td>
<td>Use ippiRemap_XX_C1R for each plane</td>
</tr>
<tr>
<td>ippiRemap_16u_P4R</td>
<td>Use ippiRemap_XX_C1R for each plane</td>
</tr>
<tr>
<td>ippiRemap_32f_P3R</td>
<td>Use ippiRemap_XX_C1R for each plane</td>
</tr>
<tr>
<td>ippiRemap_32f_P4R</td>
<td>Use ippiRemap_XX_C1R for each plane</td>
</tr>
<tr>
<td>ippiRemap_64f_P3R</td>
<td>Use ippiRemap_XX_C1R for each plane</td>
</tr>
<tr>
<td>ippiRemap_64f_P4R</td>
<td>Use ippiRemap_XX_C1R for each plane</td>
</tr>
<tr>
<td>ippiRemap_8u_P3R</td>
<td>Use ippiRemap_XX_C1R for each plane</td>
</tr>
<tr>
<td>Removed from 9.0</td>
<td>Substitution or Workaround</td>
</tr>
<tr>
<td>----------------------------------------</td>
<td>----------------------------------------------------------------</td>
</tr>
<tr>
<td>ippiRemap_8u_P4R</td>
<td>Use ippiRemap_{XX}_C1R for each plane</td>
</tr>
<tr>
<td>ippiResampleRowGetBorderWidth_32f</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiResampleRowGetSize_32f</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiResampleRowInit_32f</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiResampleRowReplicateBorder_32f_C1R</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiResampleRow_32f_C1</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiResizeGetBufSize</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiResizeGetBufSize_64f</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiResizeSqrPixel_16s_AC4R</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiResizeSqrPixel_16s_C1R</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiResizeSqrPixel_16s_C3R</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiResizeSqrPixel_16s_C4R</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiResizeSqrPixel_16s_P3R</td>
<td>Use C1R flavor for each plane</td>
</tr>
<tr>
<td>ippiResizeSqrPixel_16s_P4R</td>
<td>Use C1R flavor for each plane</td>
</tr>
<tr>
<td>ippiResizeSqrPixel_16u_AC4R</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiResizeSqrPixel_16u_C1R</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiResizeSqrPixel_16u_C3R</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiResizeSqrPixel_16u_C4R</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiResizeSqrPixel_16u_P3R</td>
<td>Use C1R flavor for each plane</td>
</tr>
<tr>
<td>ippiResizeSqrPixel_16u_P4R</td>
<td>Use C1R flavor for each plane</td>
</tr>
<tr>
<td>ippiResizeSqrPixel_32f_AC4R</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiResizeSqrPixel_32f_C1R</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiResizeSqrPixel_32f_C3R</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiResizeSqrPixel_32f_C4R</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiResizeSqrPixel_32f_P3R</td>
<td>Use C1R flavor for each plane</td>
</tr>
<tr>
<td>ippiResizeSqrPixel_32f_P4R</td>
<td>Use C1R flavor for each plane</td>
</tr>
<tr>
<td>ippiResizeSqrPixel_64f_AC4R</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiResizeSqrPixel_64f_C1R</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiResizeSqrPixel_64f_C3R</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiResizeSqrPixel_64f_C4R</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiResizeSqrPixel_64f_P3R</td>
<td>Use C1R flavor for each plane</td>
</tr>
</tbody>
</table>
### Removed from 9.0

<table>
<thead>
<tr>
<th>Function</th>
<th>Substitution or Workaround</th>
</tr>
</thead>
<tbody>
<tr>
<td>ippiResizeSqrPixel_64f_P4R</td>
<td>Use C1R flavor for each plane</td>
</tr>
<tr>
<td>ippiResizeSqrPixel_8u_AC4R</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiResizeSqrPixel_8u_C1R</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiResizeSqrPixel_8u_C3R</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiResizeSqrPixel_8u_C4R</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiResizeSqrPixel_8u_P3R</td>
<td>Use C1R flavor for each plane</td>
</tr>
<tr>
<td>ippiResizeSqrPixel_8u_P4R</td>
<td>Use C1R flavor for each plane</td>
</tr>
<tr>
<td>ippiResizeYUV420GetBufSize</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiResizeYUV420_8u_P2R</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiResizeYUV422_8u_C2R</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiRotateCenter_16u_AC4R</td>
<td>See the RotateCenter.c example.</td>
</tr>
<tr>
<td>ippiRotateCenter_16u_C1R</td>
<td></td>
</tr>
<tr>
<td>ippiRotateCenter_16u_C3R</td>
<td></td>
</tr>
<tr>
<td>ippiRotateCenter_16u_C4R</td>
<td></td>
</tr>
<tr>
<td>ippiRotateCenter_16u_P3R</td>
<td></td>
</tr>
<tr>
<td>ippiRotateCenter_16u_P4R</td>
<td></td>
</tr>
<tr>
<td>ippiRotateCenter_32f_AC4R</td>
<td></td>
</tr>
<tr>
<td>ippiRotateCenter_32f_C1R</td>
<td></td>
</tr>
<tr>
<td>ippiRotateCenter_32f_C3R</td>
<td></td>
</tr>
<tr>
<td>ippiRotateCenter_32f_C4R</td>
<td></td>
</tr>
<tr>
<td>ippiRotateCenter_32f_P3R</td>
<td></td>
</tr>
<tr>
<td>ippiRotateCenter_32f_P4R</td>
<td></td>
</tr>
<tr>
<td>ippiRotateCenter_64f_AC4R</td>
<td></td>
</tr>
<tr>
<td>ippiRotateCenter_64f_C1R</td>
<td></td>
</tr>
<tr>
<td>ippiRotateCenter_64f_C3R</td>
<td></td>
</tr>
<tr>
<td>ippiRotateCenter_64f_C4R</td>
<td></td>
</tr>
<tr>
<td>ippiRotateCenter_64f_P3R</td>
<td></td>
</tr>
<tr>
<td>ippiRotateCenter_64f_P4R</td>
<td></td>
</tr>
<tr>
<td>ippiRotateCenter_8u_AC4R</td>
<td></td>
</tr>
<tr>
<td>ippiRotateCenter_8u_C1R</td>
<td></td>
</tr>
<tr>
<td>ippiRotateCenter_8u_C3R</td>
<td></td>
</tr>
<tr>
<td>ippiRotateCenter_8u_C4R</td>
<td></td>
</tr>
<tr>
<td>ippiRotateCenter_8u_P3R</td>
<td></td>
</tr>
<tr>
<td>ippiRotateCenter_8u_P4R</td>
<td></td>
</tr>
<tr>
<td>ippiRotate_16u_AC4R</td>
<td>See the Rotate.c example.</td>
</tr>
<tr>
<td>ippiRotate_16u_C1R</td>
<td></td>
</tr>
<tr>
<td>ippiRotate_16u_C3R</td>
<td></td>
</tr>
<tr>
<td>ippiRotate_16u_C4R</td>
<td></td>
</tr>
</tbody>
</table>
## Appendix C: Removed Functions for Image and Video Processing

<table>
<thead>
<tr>
<th>Removed from 9.0</th>
<th>Substitution or Workaround</th>
</tr>
</thead>
<tbody>
<tr>
<td>ippiRotate_16u_P3R</td>
<td></td>
</tr>
<tr>
<td>ippiRotate_16u_P4R</td>
<td></td>
</tr>
<tr>
<td>ippiRotate_32f_AC4R</td>
<td></td>
</tr>
<tr>
<td>ippiRotate_32f_C1R</td>
<td></td>
</tr>
<tr>
<td>ippiRotate_32f_C3R</td>
<td></td>
</tr>
<tr>
<td>ippiRotate_32f_C4R</td>
<td></td>
</tr>
<tr>
<td>ippiRotate_32f_P3R</td>
<td></td>
</tr>
<tr>
<td>ippiRotate_32f_P4R</td>
<td></td>
</tr>
<tr>
<td>ippiRotate_64f_AC4R</td>
<td></td>
</tr>
<tr>
<td>ippiRotate_64f_C1R</td>
<td></td>
</tr>
<tr>
<td>ippiRotate_64f_C3R</td>
<td></td>
</tr>
<tr>
<td>ippiRotate_64f_C4R</td>
<td></td>
</tr>
<tr>
<td>ippiRotate_64f_P3R</td>
<td></td>
</tr>
<tr>
<td>ippiRotate_64f_P4R</td>
<td></td>
</tr>
<tr>
<td>ippiRotate_8u_AC4R</td>
<td></td>
</tr>
<tr>
<td>ippiRotate_8u_C1R</td>
<td></td>
</tr>
<tr>
<td>ippiRotate_8u_C3R</td>
<td></td>
</tr>
<tr>
<td>ippiRotate_8u_C4R</td>
<td></td>
</tr>
<tr>
<td>ippiRotate_8u_P3R</td>
<td></td>
</tr>
<tr>
<td>ippiRotate_8u_P4R</td>
<td></td>
</tr>
<tr>
<td>ippiSSIM_32f_C1R</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiShear_16u_AC4R</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiShear_16u_C1R</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiShear_16u_C3R</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiShear_16u_C4R</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiShear_16u_P3R</td>
<td>Use C1R flavor for each plane</td>
</tr>
<tr>
<td>ippiShear_16u_P4R</td>
<td>Use C1R flavor for each plane</td>
</tr>
<tr>
<td>ippiShear_32f_AC4R</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiShear_32f_C1R</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiShear_32f_C3R</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiShear_32f_C4R</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiShear_32f_P3R</td>
<td>Use C1R flavor for each plane</td>
</tr>
<tr>
<td>ippiShear_32f_P4R</td>
<td>Use C1R flavor for each plane</td>
</tr>
<tr>
<td>ippiShear_64f_AC4R</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiShear_64f_C1R</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiShear_64f_C3R</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiSSIM_32f_C1R</td>
<td>N/A</td>
</tr>
<tr>
<td>Removed from 9.0</td>
<td>Substitution or Workaround</td>
</tr>
<tr>
<td>------------------------------------------------------</td>
<td>------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>ippiShear_64f_C4R</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiShear_64f_P3R</td>
<td>Use C1R flavor for each plane</td>
</tr>
<tr>
<td>ippiShear_64f_P4R</td>
<td>Use C1R flavor for each plane</td>
</tr>
<tr>
<td>ippiShear_8u_AC4R</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiShear_8u_C1R</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiShear_8u_C3R</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiShear_8u_C4R</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiShear_8u_P3R</td>
<td>Use C1R flavor for each plane</td>
</tr>
<tr>
<td>ippiShear_8u_P4R</td>
<td>Use C1R flavor for each plane</td>
</tr>
<tr>
<td>ippiSqrDistanceFull_Norm_16u32f_AC4R</td>
<td>ippiCopy_16u_C4C1R +ippiSqrDistanceNorm_16u32f_C1R +algType=ippiROIValid</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>ippiSqrDistanceFull_Norm_16u32f_C1R</td>
<td>ippiCopy_16u_C3C1R +ippiSqrDistanceNorm_16u32f_C1R +algType=ippiROIValid</td>
</tr>
<tr>
<td>ippiSqrDistanceFull_Norm_16u32f_C3R</td>
<td>ippiCopy_16u_C4C1R +ippiSqrDistanceNorm_16u32f_C1R +algType=ippiROIValid</td>
</tr>
<tr>
<td>ippiSqrDistanceFull_Norm_16u32f_C4R</td>
<td>ippiCopy_16u_C4C1R +ippiSqrDistanceNorm_16u32f_C1R +algType=ippiROIValid</td>
</tr>
<tr>
<td>ippiSqrDistanceFull_Norm_32f_AC4R</td>
<td>ippiCopy_32f_C4C1R +ippiSqrDistanceNorm_32f_C1R +algType=ippiROIValid</td>
</tr>
<tr>
<td>ippiSqrDistanceFull_Norm_32f_C1R</td>
<td>ippiCopy_32f_C3C1R +ippiSqrDistanceNorm_32f_C1R +algType=ippiROIValid</td>
</tr>
<tr>
<td>ippiSqrDistanceFull_Norm_32f_C3R</td>
<td>ippiCopy_32f_C4C1R +ippiSqrDistanceNorm_32f_C1R +algType=ippiROIValid</td>
</tr>
<tr>
<td>ippiSqrDistanceFull_Norm_32f_C4R</td>
<td>ippiCopy_32f_C4C1R +ippiSqrDistanceNorm_32f_C1R +algType=ippiROIValid</td>
</tr>
<tr>
<td>ippiSqrDistanceFull_Norm_8s32f_AC4R</td>
<td>ippiCopy_8s32f+ippiCopy_32f_C4C1R +ippiSqrDistanceNorm_32f_C1R +algType=ippiROIValid</td>
</tr>
<tr>
<td>ippiSqrDistanceFull_Norm_8s32f_C1R</td>
<td>ippiCopy_8s32f +ippiSqrDistanceNorm_32f_C1R +algType=ippiROIValid</td>
</tr>
</tbody>
</table>
## Removed from 9.0

<table>
<thead>
<tr>
<th>Function</th>
<th>Substitution or Workaround</th>
</tr>
</thead>
<tbody>
<tr>
<td>ippiSqrDistanceFull_Norm_8s32f_C3R</td>
<td>ippiConvert_8s32f+ IPPICopy_32f_C3C1R</td>
</tr>
<tr>
<td></td>
<td>+ algType= ippiROIvalid</td>
</tr>
<tr>
<td>ippiSqrDistanceFull_Norm_8s32f_C4R</td>
<td>ippiConvert_8s32f+ IPPICopy_32f_C4C1R</td>
</tr>
<tr>
<td></td>
<td>+ algType= ippiROIvalid</td>
</tr>
<tr>
<td>ippiSqrDistanceFull_Norm_8u32f_AC4R</td>
<td>IPPICopy_8u_C4C1R</td>
</tr>
<tr>
<td></td>
<td>+ algType= ippiROIvalid</td>
</tr>
<tr>
<td>ippiSqrDistanceFull_Norm_8u32f_C4R</td>
<td>IPPICopy_8u_C4C1R</td>
</tr>
<tr>
<td></td>
<td>+ algType= ippiROIvalid</td>
</tr>
<tr>
<td>ippiSqrDistanceFull_Norm_8u32f_CF4</td>
<td>IPPICopy_8u_C3C1R</td>
</tr>
<tr>
<td></td>
<td>+ algType= ippiROIvalid</td>
</tr>
<tr>
<td>ippiSqrDistanceFull_Norm_8u32f_C4R</td>
<td>IPPICopy_8u_C4C1R</td>
</tr>
<tr>
<td></td>
<td>+ algType= ippiROIvalid</td>
</tr>
<tr>
<td>ippiSqrDistanceFull_Norm_8u_AC4RSfs</td>
<td>IPPICopy_8u_C4C1R</td>
</tr>
<tr>
<td></td>
<td>+ algType= ippiROIvalid</td>
</tr>
<tr>
<td>ippiSqrDistanceFull_Norm_8u_C1RSfs</td>
<td>IPPICopy_8u_C3C1R</td>
</tr>
<tr>
<td></td>
<td>+ algType= ippiROIvalid</td>
</tr>
<tr>
<td>ippiSqrDistanceFull_Norm_8u_C3RSfs</td>
<td>IPPICopy_8u_C4C1R</td>
</tr>
<tr>
<td></td>
<td>+ algType= ippiROIvalid</td>
</tr>
<tr>
<td>ippiSqrDistanceSame_Norm_16u32f_AC4R</td>
<td>IPPICopy_16u_C4C1R</td>
</tr>
<tr>
<td></td>
<td>+ algType= ippiROIvalid</td>
</tr>
<tr>
<td>ippiSqrDistanceSame_Norm_16u32f_C1R</td>
<td>IPPICopy_16u_C3C1R</td>
</tr>
<tr>
<td></td>
<td>+ algType= ippiROIvalid</td>
</tr>
<tr>
<td>ippiSqrDistanceSame_Norm_16u32f_C3R</td>
<td>IPPICopy_16u_C3C1R</td>
</tr>
<tr>
<td></td>
<td>+ algType= ippiROIvalid</td>
</tr>
<tr>
<td>ippiSqrDistanceSame_Norm_16u32f_C4R</td>
<td>IPPICopy_16u_C4C1R</td>
</tr>
<tr>
<td></td>
<td>+ algType= ippiROIvalid</td>
</tr>
<tr>
<td>Removed from 9.0</td>
<td>Substitution or Workaround</td>
</tr>
<tr>
<td>--------------------------------------------------------------------------------</td>
<td>----------------------------------------------------------------</td>
</tr>
<tr>
<td>ippiSqrDistanceSame_Norm_32f_AC4R</td>
<td>ippiCopy_32f_C4C1R</td>
</tr>
<tr>
<td></td>
<td>+ippiSqrDistanceNorm_32f_C1R</td>
</tr>
<tr>
<td></td>
<td>+<strong>algType</strong>=ippiROISame</td>
</tr>
<tr>
<td>ippiSqrDistanceSame_Norm_32f_C1R</td>
<td>ippiSqrDistanceNorm_32f_C1R</td>
</tr>
<tr>
<td></td>
<td>+<strong>algType</strong>=ippiROISame</td>
</tr>
<tr>
<td>ippiSqrDistanceSame_Norm_32f_C3R</td>
<td>ippiCopy_32f_C3C1R</td>
</tr>
<tr>
<td></td>
<td>+ippiSqrDistanceNorm_32f_C1R</td>
</tr>
<tr>
<td></td>
<td>+<strong>algType</strong>=ippiROISame</td>
</tr>
<tr>
<td>ippiSqrDistanceSame_Norm_32f_C4R</td>
<td>ippiCopy_32f_C4C1R</td>
</tr>
<tr>
<td></td>
<td>+ippiSqrDistanceNorm_32f_C1R</td>
</tr>
<tr>
<td></td>
<td>+<strong>algType</strong>=ippiROISame</td>
</tr>
<tr>
<td>ippiSqrDistanceSame_Norm_8s32f_AC4R</td>
<td>ippiConvert_8s32f+ippiCopy_32f_C4C1R</td>
</tr>
<tr>
<td></td>
<td>+ippiSqrDistanceNorm_32f_C1R</td>
</tr>
<tr>
<td></td>
<td>+<strong>algType</strong>=ippiROISame</td>
</tr>
<tr>
<td>ippiSqrDistanceSame_Norm_8s32f_C1R</td>
<td>ippiConvert_8s32f</td>
</tr>
<tr>
<td></td>
<td>+ippiSqrDistanceNorm_32f_C1R</td>
</tr>
<tr>
<td></td>
<td>+<strong>algType</strong>=ippiROISame</td>
</tr>
<tr>
<td>ippiSqrDistanceSame_Norm_8s32f_C3R</td>
<td>ippiConvert_8s32f+ippiCopy_32f_C3C1R</td>
</tr>
<tr>
<td></td>
<td>+ippiSqrDistanceNorm_32f_C1R</td>
</tr>
<tr>
<td></td>
<td>+<strong>algType</strong>=ippiROISame</td>
</tr>
<tr>
<td>ippiSqrDistanceSame_Norm_8s32f_C4R</td>
<td>ippiConvert_8s32f+ippiCopy_32f_C4C1R</td>
</tr>
<tr>
<td></td>
<td>+ippiSqrDistanceNorm_32f_C1R</td>
</tr>
<tr>
<td></td>
<td>+<strong>algType</strong>=ippiROISame</td>
</tr>
<tr>
<td>ippiSqrDistanceSame_Norm_8u32f_AC4R</td>
<td>ippiCopy_8u_C4C1R</td>
</tr>
<tr>
<td></td>
<td>+ippiSqrDistanceNorm_8u32f_C1R</td>
</tr>
<tr>
<td></td>
<td>+<strong>algType</strong>=ippiROISame</td>
</tr>
<tr>
<td>ippiSqrDistanceSame_Norm_8u32f_C1R</td>
<td>ippiSqrDistanceNorm_8u32f_C1R</td>
</tr>
<tr>
<td></td>
<td>+<strong>algType</strong>=ippiROISame</td>
</tr>
<tr>
<td>ippiSqrDistanceSame_Norm_8u32f_C3R</td>
<td>ippiCopy_8u_C3C1R</td>
</tr>
<tr>
<td></td>
<td>+ippiSqrDistanceNorm_8u32f_C1R</td>
</tr>
<tr>
<td></td>
<td>+<strong>algType</strong>=ippiROISame</td>
</tr>
<tr>
<td>ippiSqrDistanceSame_Norm_8u32f_C4R</td>
<td>ippiCopy_8u_C4C1R</td>
</tr>
<tr>
<td></td>
<td>+ippiSqrDistanceNorm_8u32f_C1R</td>
</tr>
<tr>
<td></td>
<td>+<strong>algType</strong>=ippiROISame</td>
</tr>
<tr>
<td>ippiSqrDistanceSame_Norm_8u_AC4RSfs</td>
<td>ippiCopy_8u_C4C1R</td>
</tr>
<tr>
<td></td>
<td>+ippiSqrDistanceNorm_8u_C1RSfs</td>
</tr>
<tr>
<td></td>
<td>+<strong>algType</strong>=ippiROISame</td>
</tr>
<tr>
<td>ippiSqrDistanceSame_Norm_8u_C1RSfs</td>
<td>ippiSqrDistanceNorm_8u_C1RSfs</td>
</tr>
<tr>
<td></td>
<td>+<strong>algType</strong>=ippiROISame</td>
</tr>
<tr>
<td>Removed from 9.0</td>
<td>Substitution or Workaround</td>
</tr>
<tr>
<td>------------------------------------------------------</td>
<td>-------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>ippiSqrDistanceSame_Norm_8u_C3RSfs</td>
<td>ippiCopy_8u_C3C1R</td>
</tr>
<tr>
<td></td>
<td>+ippiSqrDistanceNorm_8u_C1RSfs</td>
</tr>
<tr>
<td></td>
<td>+algType=ippiROIValid</td>
</tr>
<tr>
<td>ippiSqrDistanceSame_Norm_8u_C4RSfs</td>
<td>ippiCopy_8u_C4C1R</td>
</tr>
<tr>
<td></td>
<td>+ippiSqrDistanceNorm_8u_C1RSfs</td>
</tr>
<tr>
<td></td>
<td>+algType=ippiROIValid</td>
</tr>
<tr>
<td>ippiSqrDistanceValid_Norm_16u32f_AC4R</td>
<td>ippiCopy_16u_C4C1R</td>
</tr>
<tr>
<td></td>
<td>+ippiSqrDistanceNorm_16u32f_C1R</td>
</tr>
<tr>
<td></td>
<td>+algType=ippiROIValid</td>
</tr>
<tr>
<td>ippiSqrDistanceValid_Norm_16u32f_C1R</td>
<td>ippiSqrDistanceNorm_16u32f_C1R</td>
</tr>
<tr>
<td></td>
<td>+algType=ippiROIValid</td>
</tr>
<tr>
<td>ippiSqrDistanceValid_Norm_16u32f_C3R</td>
<td>ippiCopy_16u_C3C1R</td>
</tr>
<tr>
<td></td>
<td>+ippiSqrDistanceNorm_16u32f_C1R</td>
</tr>
<tr>
<td></td>
<td>+algType=ippiROIValid</td>
</tr>
<tr>
<td>ippiSqrDistanceValid_Norm_16u32f_C4R</td>
<td>ippiCopy_16u_C4C1R</td>
</tr>
<tr>
<td></td>
<td>+ippiSqrDistanceNorm_16u32f_C1R</td>
</tr>
<tr>
<td></td>
<td>+algType=ippiROIValid</td>
</tr>
<tr>
<td>ippiSqrDistanceValid_Norm_32f_AC4R</td>
<td>ippiCopy_32f_C4C1R</td>
</tr>
<tr>
<td></td>
<td>+ippiSqrDistanceNorm_32f_C1R</td>
</tr>
<tr>
<td></td>
<td>+algType=ippiROIValid</td>
</tr>
<tr>
<td>ippiSqrDistanceValid_Norm_32f_C1R</td>
<td>ippiSqrDistanceNorm_32f_C1R</td>
</tr>
<tr>
<td></td>
<td>+algType=ippiROIValid</td>
</tr>
<tr>
<td>ippiSqrDistanceValid_Norm_32f_C3R</td>
<td>ippiCopy_32f_C3C1R</td>
</tr>
<tr>
<td></td>
<td>+ippiSqrDistanceNorm_32f_C1R</td>
</tr>
<tr>
<td></td>
<td>+algType=ippiROIValid</td>
</tr>
<tr>
<td>ippiSqrDistanceValid_Norm_32f_C4R</td>
<td>ippiCopy_32f_C4C1R</td>
</tr>
<tr>
<td></td>
<td>+ippiSqrDistanceNorm_32f_C1R</td>
</tr>
<tr>
<td></td>
<td>+algType=ippiROIValid</td>
</tr>
<tr>
<td>ippiSqrDistanceValid_Norm_8s32f_AC4R</td>
<td>ippiConvert_8s32f+ippiCopy_32f_C4C1R</td>
</tr>
<tr>
<td></td>
<td>+ippiSqrDistanceNorm_32f_C1R</td>
</tr>
<tr>
<td></td>
<td>+algType=ippiROIValid</td>
</tr>
<tr>
<td>ippiSqrDistanceValid_Norm_8s32f_C1R</td>
<td>ippiConvert_8s32f</td>
</tr>
<tr>
<td></td>
<td>+ippiSqrDistanceNorm_32f_C1R</td>
</tr>
<tr>
<td></td>
<td>+algType=ippiROIValid</td>
</tr>
<tr>
<td>ippiSqrDistanceValid_Norm_8s32f_C3R</td>
<td>ippiConvert_8s32f+ippiCopy_32f_C3C1R</td>
</tr>
<tr>
<td></td>
<td>+ippiSqrDistanceNorm_32f_C1R</td>
</tr>
<tr>
<td></td>
<td>+algType=ippiROIValid</td>
</tr>
<tr>
<td>ippiSqrDistanceValid_Norm_8s32f_C4R</td>
<td>ippiConvert_8s32f+ippiCopy_32f_C4C1R</td>
</tr>
<tr>
<td></td>
<td>+ippiSqrDistanceNorm_32f_C1R</td>
</tr>
<tr>
<td></td>
<td>+algType=ippiROIValid</td>
</tr>
<tr>
<td>Removed from 9.0</td>
<td>Substitution or Workaround</td>
</tr>
<tr>
<td>------------------</td>
<td>---------------------------</td>
</tr>
</tbody>
</table>
| ippiSqrDistanceValid_Norm_8u32f_AC4R | ippiCopy_8u_C4C1R  
+ippiSqrDistanceNorm_8u32f_C1R  
+algType=ippiROIValid|ippiNorm  |
| ippiSqrDistanceValid_Norm_8u32f_C1R | ippiSqrDistanceNorm_8u32f_C1R  
+algType=ippiROIValid|ippiNorm  |
| ippiSqrDistanceValid_Norm_8u32f_C3R | ippiCopy_8u_C3C1R  
+ippiSqrDistanceNorm_8u32f_C1R  
+algType=ippiROIValid|ippiNorm  |
| ippiSqrDistanceValid_Norm_8u32f_C4R | ippiCopy_8u_C4C1R  
+ippiSqrDistanceNorm_8u32f_C1R  
+algType=ippiROIValid|ippiNorm  |
| ippiSqrDistanceValid_Norm_8u_AC4RSfs | ippiCopy_8u_C4C1R  
+ippiSqrDistanceNorm_8u_C1RSfs  
+algType=ippiROIValid|ippiNorm  |
| ippiSqrDistanceValid_Norm_8u_C1RSfs | ippiSqrDistanceNorm_8u_C1RSfs  
+algType=ippiROIValid|ippiNorm  |
| ippiSqrDistanceValid_Norm_8u_C3RSfs | ippiCopy_8u_C3C1R  
+ippiSqrDistanceNorm_8u_C1RSfs  
+algType=ippiROIValid|ippiNorm  |
| ippiSqrDistanceValid_Norm_8u_C4RSfs | ippiCopy_8u_C4C1R  
+ippiSqrDistanceNorm_8u_C1RSfs  
+algType=ippiROIValid|ippiNorm  |
| ippiSubC_16sc_AC4IRSfs | N/A  |
| ippiSubC_16sc_AC4RSfs | N/A  |
| ippiSubC_16sc_C1IRSfs | N/A  |
| ippiSubC_16sc_C1RSfs | N/A  |
| ippiSubC_16sc_C3IRSfs | N/A  |
| ippiSubC_16sc_C3RSfs | N/A  |
| ippiSubC_32fc_AC4IR | N/A  |
| ippiSubC_32fc_AC4R | N/A  |
| ippiSubC_32fc_C1IR | N/A  |
| ippiSubC_32fc_C1R | N/A  |
| ippiSubC_32fc_C3IR | N/A  |
| ippiSubC_32fc_C3R | N/A  |
| ippiSubC_32sc_AC4IRSfs | N/A  |
| ippiSubC_32sc_AC4RSfs | N/A  |
## Removed from 9.0

<table>
<thead>
<tr>
<th>Function</th>
<th>Substitution or Workaround</th>
</tr>
</thead>
<tbody>
<tr>
<td>ippiSubC_32sc_C1IRSfs</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiSubC_32sc_C1RSfs</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiSubC_32sc_C3IRSfs</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiSubC_32sc_C3RSfs</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiSub_16sc_AC4IRSfs</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiSub_16sc_AC4RSfs</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiSub_16sc_C1IRSfs</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiSub_16sc_C1RSfs</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiSub_16sc_C3IRSfs</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiSub_16sc_C3RSfs</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiSub_32fc_AC4IR</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiSub_32fc_AC4R</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiSub_32fc_C1IR</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiSub_32fc_C1R</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiSub_32fc_C3IR</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiSub_32fc_C3R</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiSub_32sc_AC4IRSfs</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiSub_32sc_AC4RSfs</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiSub_32sc_C1IRSfs</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiSub_32sc_C1RSfs</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiSub_32sc_C3IRSfs</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiSub_32sc_C3RSfs</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiSum_16s_AC4R</td>
<td>ippiSum_16s_C4R</td>
</tr>
<tr>
<td>ippiSum_16u_AC4R</td>
<td>ippiSum_16u_C4R</td>
</tr>
<tr>
<td>ippiSum_32f_AC4R</td>
<td>ippiSum_32f_C4R</td>
</tr>
<tr>
<td>ippiSum_8u_AC4R</td>
<td>ippiSum_8u_C4R</td>
</tr>
<tr>
<td>ippiSuperSamplingGetBufSize</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiSuperSampling_16s_AC4R</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiSuperSampling_16s_C1R</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiSuperSampling_16s_C3R</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiSuperSampling_16s_C4R</td>
<td>N/A</td>
</tr>
<tr>
<td>Removed from 9.0</td>
<td>Substitution or Workaround</td>
</tr>
<tr>
<td>-------------------------------------</td>
<td>------------------------------------------------</td>
</tr>
<tr>
<td>ippiSuperSampling_16s_P3R</td>
<td>Use C1R flavor for each plane</td>
</tr>
<tr>
<td>ippiSuperSampling_16s_P4R</td>
<td>Use C1R flavor for each plane</td>
</tr>
<tr>
<td>ippiSuperSampling_16u_AC4R</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiSuperSampling_16u_C1R</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiSuperSampling_16u_C3R</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiSuperSampling_16u_C4R</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiSuperSampling_16u_P3R</td>
<td>Use C1R flavor for each plane</td>
</tr>
<tr>
<td>ippiSuperSampling_16u_P4R</td>
<td>Use C1R flavor for each plane</td>
</tr>
<tr>
<td>ippiSuperSampling_32f_AC4R</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiSuperSampling_32f_C1R</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiSuperSampling_32f_C3R</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiSuperSampling_32f_C4R</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiSuperSampling_32f_P3R</td>
<td>Use C1R flavor for each plane</td>
</tr>
<tr>
<td>ippiSuperSampling_32f_P4R</td>
<td>Use C1R flavor for each plane</td>
</tr>
<tr>
<td>ippiSuperSampling_8u_AC4R</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiSuperSampling_8u_C1R</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiSuperSampling_8u_C3R</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiSuperSampling_8u_C4R</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiSuperSampling_8u_P3R</td>
<td>Use C1R flavor for each plane</td>
</tr>
<tr>
<td>ippiSuperSampling_8u_P4R</td>
<td>Use C1R flavor for each plane</td>
</tr>
<tr>
<td>ippiWTFwdFree_32f_C1R</td>
<td>ippiFree</td>
</tr>
<tr>
<td>ippiWTFwdFree_32f_C3R</td>
<td>ippiFree</td>
</tr>
<tr>
<td>ippiWTFwdGetSize_C1R</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiWTFwdGetSize_C3R</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiWTFwdInitAlloc_32f_C1R</td>
<td>ippiWTFwdGetSize+ippiMalloc+ippiWTFwdInit</td>
</tr>
<tr>
<td>ippiWTFwdInitAlloc_32f_C3R</td>
<td>ippiWTFwdGetSize+ippiMalloc+ippiWTFwdInit</td>
</tr>
<tr>
<td>ippiWTInvFree_32f_C1R</td>
<td>ippiFree</td>
</tr>
<tr>
<td>ippiWTInvFree_32f_C3R</td>
<td>ippiFree</td>
</tr>
<tr>
<td>ippiWTInvGetBufSize_C1R</td>
<td>N/A</td>
</tr>
</tbody>
</table>
## Removed from 9.0

<table>
<thead>
<tr>
<th>Function</th>
<th>Substitution or Workaround</th>
</tr>
</thead>
<tbody>
<tr>
<td>ippiWTInvGetBufSize_C3R</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiWTInvInitAlloc_32f_C1R</td>
<td>ippiWTInvGetSize+ippiMalloc+ippiWTInvInit</td>
</tr>
<tr>
<td>ippiWTInvInitAlloc_32f_C3R</td>
<td>ippiWTInvGetSize+ippiMalloc+ippiWTInvInit</td>
</tr>
<tr>
<td>ippiWarpAffineBack_16u_AC4R</td>
<td>Use new WarpAffine functions. For an example on how to replace the old API with the new one refer to <a href="https://software.intel.com/en-us/articles/code-examples-for-new-apis-and-sample-how-to-replace-deprecated-api-with-them">https://software.intel.com/en-us/articles/code-examples-for-new-apis-and-sample-how-to-replace-deprecated-api-with-them</a>.</td>
</tr>
<tr>
<td>ippiWarpAffineBack_16u_C1R</td>
<td>Use new WarpAffine functions. For an example on how to replace the old API with the new one refer to <a href="https://software.intel.com/en-us/articles/code-examples-for-new-apis-and-sample-how-to-replace-deprecated-api-with-them">https://software.intel.com/en-us/articles/code-examples-for-new-apis-and-sample-how-to-replace-deprecated-api-with-them</a>.</td>
</tr>
<tr>
<td>ippiWarpAffineBack_32f_AC4R</td>
<td>Use new WarpAffine functions. For an example on how to replace the old API with the new one refer to <a href="https://software.intel.com/en-us/articles/code-examples-for-new-apis-and-sample-how-to-replace-deprecated-api-with-them">https://software.intel.com/en-us/articles/code-examples-for-new-apis-and-sample-how-to-replace-deprecated-api-with-them</a>.</td>
</tr>
<tr>
<td>Removed from 9.0</td>
<td>Substitution or Workaround</td>
</tr>
<tr>
<td>----------------------------------</td>
<td>---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>ippiWarpAffineBack_32f_C1R</td>
<td>Use new WarpAffine functions. For an example on how to replace the old API with the new one refer to <a href="https://software.intel.com/en-us/articles/code-examples-for-new-apis-and-sample-how-to-replace-deprecated-api-with-them">https://software.intel.com/en-us/articles/code-examples-for-new-apis-and-sample-how-to-replace-deprecated-api-with-them</a>.</td>
</tr>
<tr>
<td>ippiWarpAffineBack_32f_C3R</td>
<td>Use new WarpAffine functions. For an example on how to replace the old API with the new one refer to <a href="https://software.intel.com/en-us/articles/code-examples-for-new-apis-and-sample-how-to-replace-deprecated-api-with-them">https://software.intel.com/en-us/articles/code-examples-for-new-apis-and-sample-how-to-replace-deprecated-api-with-them</a>.</td>
</tr>
<tr>
<td>ippiWarpAffineBack_32f_C4R</td>
<td>Use new WarpAffine functions. For an example on how to replace the old API with the new one refer to <a href="https://software.intel.com/en-us/articles/code-examples-for-new-apis-and-sample-how-to-replace-deprecated-api-with-them">https://software.intel.com/en-us/articles/code-examples-for-new-apis-and-sample-how-to-replace-deprecated-api-with-them</a>.</td>
</tr>
<tr>
<td>ippiWarpAffineBack_32f_P3R</td>
<td>Use new WarpAffine functions. For an example on how to replace the old API with the new one refer to <a href="https://software.intel.com/en-us/articles/code-examples-for-new-apis-and-sample-how-to-replace-deprecated-api-with-them">https://software.intel.com/en-us/articles/code-examples-for-new-apis-and-sample-how-to-replace-deprecated-api-with-them</a>.</td>
</tr>
<tr>
<td>ippiWarpAffineBack_8u_AC4R</td>
<td>Use new WarpAffine functions. For an example on how to replace the old API with the new one refer to <a href="https://software.intel.com/en-us/articles/code-examples-for-new-apis-and-sample-how-to-replace-deprecated-api-with-them">https://software.intel.com/en-us/articles/code-examples-for-new-apis-and-sample-how-to-replace-deprecated-api-with-them</a>.</td>
</tr>
<tr>
<td>ippiWarpAffineBack_8u_C1R</td>
<td>Use new WarpAffine functions. For an example on how to replace the old API with the new one refer to <a href="https://software.intel.com/en-us/articles/code-examples-for-new-apis-and-sample-how-to-replace-deprecated-api-with-them">https://software.intel.com/en-us/articles/code-examples-for-new-apis-and-sample-how-to-replace-deprecated-api-with-them</a>.</td>
</tr>
<tr>
<td>ippiWarpAffineBack_8u_C3R</td>
<td>Use new WarpAffine functions. For an example on how to replace the old API with the new one refer to <a href="https://software.intel.com/en-us/articles/code-examples-for-new-apis-and-sample-how-to-replace-deprecated-api-with-them">https://software.intel.com/en-us/articles/code-examples-for-new-apis-and-sample-how-to-replace-deprecated-api-with-them</a>.</td>
</tr>
<tr>
<td>ippiWarpAffineBack_8u_C4R</td>
<td>Use new WarpAffine functions. For an example on how to replace the old API with the new one refer to <a href="https://software.intel.com/en-us/articles/code-examples-for-new-apis-and-sample-how-to-replace-deprecated-api-with-them">https://software.intel.com/en-us/articles/code-examples-for-new-apis-and-sample-how-to-replace-deprecated-api-with-them</a>.</td>
</tr>
</tbody>
</table>
## Removed from 9.0

<table>
<thead>
<tr>
<th>Function</th>
<th>Substitution or Workaround</th>
</tr>
</thead>
<tbody>
<tr>
<td>ippiWarpAffineBack_8u_P3R</td>
<td>Use new WarpAffine functions. For an example on how to replace the old API refer to the website.</td>
</tr>
<tr>
<td>ippiWarpAffineBack_8u_P4R</td>
<td>Use new WarpAffine functions. For an example on how to replace the old API refer to the website.</td>
</tr>
<tr>
<td>ippiWarpAffineQuad_16u_AC4R</td>
<td>Use new WarpQuad functions. For an example on how to replace the old API refer to the website.</td>
</tr>
<tr>
<td>ippiWarpAffineQuad_16u_C1R</td>
<td>Use new WarpQuad functions. For an example on how to replace the old API refer to the website.</td>
</tr>
<tr>
<td>ippiWarpAffineQuad_16u_C3R</td>
<td>Use new WarpQuad functions. For an example on how to replace the old API refer to the website.</td>
</tr>
<tr>
<td>ippiWarpAffineQuad_16u_C4R</td>
<td>Use new WarpQuad functions. For an example on how to replace the old API refer to the website.</td>
</tr>
<tr>
<td>ippiWarpAffineQuad_16u_P3R</td>
<td>Use new WarpQuad functions. For an example on how to replace the old API refer to the website.</td>
</tr>
<tr>
<td>ippiWarpAffineQuad_16u_P4R</td>
<td>Use new WarpQuad functions. For an example on how to replace the old API refer to the website.</td>
</tr>
<tr>
<td>ippiWarpAffineQuad_32f_AC4R</td>
<td>Use new WarpQuad functions. For an example on how to replace the old API refer to the website.</td>
</tr>
<tr>
<td>Removed from 9.0</td>
<td>Substitution or Workaround</td>
</tr>
<tr>
<td>-----------------------------</td>
<td>---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>ippiWarpAffineQuad_32f_C1R</td>
<td>Use new WarpQuad functions. For an example on how to replace the old API with the new one refer to <a href="https://software.intel.com/en-us/articles/code-examples-for-new-apis-and-sample-how-to-replace-deprecated-api-with-them">https://software.intel.com/en-us/articles/code-examples-for-new-apis-and-sample-how-to-replace-deprecated-api-with-them</a>.</td>
</tr>
<tr>
<td>ippiWarpAffineQuad_32f_C3R</td>
<td>Use new WarpQuad functions. For an example on how to replace the old API with the new one refer to <a href="https://software.intel.com/en-us/articles/code-examples-for-new-apis-and-sample-how-to-replace-deprecated-api-with-them">https://software.intel.com/en-us/articles/code-examples-for-new-apis-and-sample-how-to-replace-deprecated-api-with-them</a>.</td>
</tr>
<tr>
<td>ippiWarpAffineQuad_32f_C4R</td>
<td>Use new WarpQuad functions. For an example on how to replace the old API with the new one refer to <a href="https://software.intel.com/en-us/articles/code-examples-for-new-apis-and-sample-how-to-replace-deprecated-api-with-them">https://software.intel.com/en-us/articles/code-examples-for-new-apis-and-sample-how-to-replace-deprecated-api-with-them</a>.</td>
</tr>
<tr>
<td>ippiWarpAffineQuad_32f_P3R</td>
<td>Use new WarpQuad functions. For an example on how to replace the old API with the new one refer to <a href="https://software.intel.com/en-us/articles/code-examples-for-new-apis-and-sample-how-to-replace-deprecated-api-with-them">https://software.intel.com/en-us/articles/code-examples-for-new-apis-and-sample-how-to-replace-deprecated-api-with-them</a>.</td>
</tr>
<tr>
<td>ippiWarpAffineQuad_32f_P4R</td>
<td>Use new WarpQuad functions. For an example on how to replace the old API with the new one refer to <a href="https://software.intel.com/en-us/articles/code-examples-for-new-apis-and-sample-how-to-replace-deprecated-api-with-them">https://software.intel.com/en-us/articles/code-examples-for-new-apis-and-sample-how-to-replace-deprecated-api-with-them</a>.</td>
</tr>
<tr>
<td>ippiWarpAffineQuad_8u_AC4R</td>
<td>Use new WarpQuad functions. For an example on how to replace the old API with the new one refer to <a href="https://software.intel.com/en-us/articles/code-examples-for-new-apis-and-sample-how-to-replace-deprecated-api-with-them">https://software.intel.com/en-us/articles/code-examples-for-new-apis-and-sample-how-to-replace-deprecated-api-with-them</a>.</td>
</tr>
<tr>
<td>ippiWarpAffineQuad_8u_C1R</td>
<td>Use new WarpQuad functions. For an example on how to replace the old API with the new one refer to <a href="https://software.intel.com/en-us/articles/code-examples-for-new-apis-and-sample-how-to-replace-deprecated-api-with-them">https://software.intel.com/en-us/articles/code-examples-for-new-apis-and-sample-how-to-replace-deprecated-api-with-them</a>.</td>
</tr>
<tr>
<td>ippiWarpAffineQuad_8u_C3R</td>
<td>Use new WarpQuad functions. For an example on how to replace the old API with the new one refer to <a href="https://software.intel.com/en-us/articles/code-examples-for-new-apis-and-sample-how-to-replace-deprecated-api-with-them">https://software.intel.com/en-us/articles/code-examples-for-new-apis-and-sample-how-to-replace-deprecated-api-with-them</a>.</td>
</tr>
<tr>
<td>ippiWarpAffineQuad_8u_C4R</td>
<td>Use new WarpQuad functions. For an example on how to replace the old API with the new one refer to <a href="https://software.intel.com/en-us/articles/code-examples-for-new-apis-and-sample-how-to-replace-deprecated-api-with-them">https://software.intel.com/en-us/articles/code-examples-for-new-apis-and-sample-how-to-replace-deprecated-api-with-them</a>.</td>
</tr>
</tbody>
</table>
### Appendix C: Removed Functions for Image and Video Processing

<table>
<thead>
<tr>
<th>Removed from 9.0</th>
<th>Substitution or Workaround</th>
</tr>
</thead>
<tbody>
<tr>
<td>Removed from 9.0</td>
<td>Substitution or Workaround</td>
</tr>
<tr>
<td>---------------------------</td>
<td>------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>ippiWarpAffine_32f_C1R</td>
<td>Use new <code>WarpAffine</code> functions. For an example on how to replace the old API with the new one refer to <a href="https://software.intel.com/en-us/articles/code-examples-for-new-apis-and-sample-how-to-replace-deprecated-api-with-them">https://software.intel.com/en-us/articles/code-examples-for-new-apis-and-sample-how-to-replace-deprecated-api-with-them</a>.</td>
</tr>
<tr>
<td>ippiWarpAffine_32f_C3R</td>
<td>Use new <code>WarpAffine</code> functions. For an example on how to replace the old API with the new one refer to <a href="https://software.intel.com/en-us/articles/code-examples-for-new-apis-and-sample-how-to-replace-deprecated-api-with-them">https://software.intel.com/en-us/articles/code-examples-for-new-apis-and-sample-how-to-replace-deprecated-api-with-them</a>.</td>
</tr>
<tr>
<td>ippiWarpAffine_32f_C4R</td>
<td>Use new <code>WarpAffine</code> functions. For an example on how to replace the old API with the new one refer to <a href="https://software.intel.com/en-us/articles/code-examples-for-new-apis-and-sample-how-to-replace-deprecated-api-with-them">https://software.intel.com/en-us/articles/code-examples-for-new-apis-and-sample-how-to-replace-deprecated-api-with-them</a>.</td>
</tr>
<tr>
<td>ippiWarpAffine_32f_P3R</td>
<td>Use new <code>WarpAffine</code> functions. For an example on how to replace the old API with the new one refer to <a href="https://software.intel.com/en-us/articles/code-examples-for-new-apis-and-sample-how-to-replace-deprecated-api-with-them">https://software.intel.com/en-us/articles/code-examples-for-new-apis-and-sample-how-to-replace-deprecated-api-with-them</a>.</td>
</tr>
<tr>
<td>ippiWarpAffine_32f_P4R</td>
<td>Use new <code>WarpAffine</code> functions. For an example on how to replace the old API with the new one refer to <a href="https://software.intel.com/en-us/articles/code-examples-for-new-apis-and-sample-how-to-replace-deprecated-api-with-them">https://software.intel.com/en-us/articles/code-examples-for-new-apis-and-sample-how-to-replace-deprecated-api-with-them</a>.</td>
</tr>
<tr>
<td>ippiWarpAffine_64f_AC4R</td>
<td>Use new <code>WarpAffine</code> functions. For an example on how to replace the old API with the new one refer to <a href="https://software.intel.com/en-us/articles/code-examples-for-new-apis-and-sample-how-to-replace-deprecated-api-with-them">https://software.intel.com/en-us/articles/code-examples-for-new-apis-and-sample-how-to-replace-deprecated-api-with-them</a>.</td>
</tr>
<tr>
<td>ippiWarpAffine_64f_C1R</td>
<td>Use new <code>WarpAffine</code> functions. For an example on how to replace the old API with the new one refer to <a href="https://software.intel.com/en-us/articles/code-examples-for-new-apis-and-sample-how-to-replace-deprecated-api-with-them">https://software.intel.com/en-us/articles/code-examples-for-new-apis-and-sample-how-to-replace-deprecated-api-with-them</a>.</td>
</tr>
<tr>
<td>ippiWarpAffine_64f_C3R</td>
<td>Use new <code>WarpAffine</code> functions. For an example on how to replace the old API with the new one refer to <a href="https://software.intel.com/en-us/articles/code-examples-for-new-apis-and-sample-how-to-replace-deprecated-api-with-them">https://software.intel.com/en-us/articles/code-examples-for-new-apis-and-sample-how-to-replace-deprecated-api-with-them</a>.</td>
</tr>
<tr>
<td>ippiWarpAffine_64f_C4R</td>
<td>Use new <code>WarpAffine</code> functions. For an example on how to replace the old API with the new one refer to <a href="https://software.intel.com/en-us/articles/code-examples-for-new-apis-and-sample-how-to-replace-deprecated-api-with-them">https://software.intel.com/en-us/articles/code-examples-for-new-apis-and-sample-how-to-replace-deprecated-api-with-them</a>.</td>
</tr>
<tr>
<td>Removed from 9.0</td>
<td>Substitution or Workaround</td>
</tr>
<tr>
<td>-----------------</td>
<td>---------------------------</td>
</tr>
<tr>
<td>ippiWarpAffine_64f_P3R</td>
<td>Use new WarpAffine functions. For an example on how to replace the old API with the new one refer to <a href="https://software.intel.com/en-us/articles/code-examples-for-new-apis-and-sample-how-to-replace-deprecated-api-with-them">https://software.intel.com/en-us/articles/code-examples-for-new-apis-and-sample-how-to-replace-deprecated-api-with-them</a>.</td>
</tr>
<tr>
<td>ippiWarpAffine_64f_P4R</td>
<td>Use new WarpAffine functions. For an example on how to replace the old API with the new one refer to <a href="https://software.intel.com/en-us/articles/code-examples-for-new-apis-and-sample-how-to-replace-deprecated-api-with-them">https://software.intel.com/en-us/articles/code-examples-for-new-apis-and-sample-how-to-replace-deprecated-api-with-them</a>.</td>
</tr>
<tr>
<td>ippiWarpAffine_8u_AC4R</td>
<td>Use new WarpAffine functions. For an example on how to replace the old API with the new one refer to <a href="https://software.intel.com/en-us/articles/code-examples-for-new-apis-and-sample-how-to-replace-deprecated-api-with-them">https://software.intel.com/en-us/articles/code-examples-for-new-apis-and-sample-how-to-replace-deprecated-api-with-them</a>.</td>
</tr>
<tr>
<td>ippiWarpAffine_8u_C1R</td>
<td>Use new WarpAffine functions. For an example on how to replace the old API with the new one refer to <a href="https://software.intel.com/en-us/articles/code-examples-for-new-apis-and-sample-how-to-replace-deprecated-api-with-them">https://software.intel.com/en-us/articles/code-examples-for-new-apis-and-sample-how-to-replace-deprecated-api-with-them</a>.</td>
</tr>
<tr>
<td>ippiWarpAffine_8u_C3R</td>
<td>Use new WarpAffine functions. For an example on how to replace the old API with the new one refer to <a href="https://software.intel.com/en-us/articles/code-examples-for-new-apis-and-sample-how-to-replace-deprecated-api-with-them">https://software.intel.com/en-us/articles/code-examples-for-new-apis-and-sample-how-to-replace-deprecated-api-with-them</a>.</td>
</tr>
<tr>
<td>ippiWarpAffine_8u_C4R</td>
<td>Use new WarpAffine functions. For an example on how to replace the old API with the new one refer to <a href="https://software.intel.com/en-us/articles/code-examples-for-new-apis-and-sample-how-to-replace-deprecated-api-with-them">https://software.intel.com/en-us/articles/code-examples-for-new-apis-and-sample-how-to-replace-deprecated-api-with-them</a>.</td>
</tr>
<tr>
<td>ippiWarpAffine_8u_P3R</td>
<td>Use new WarpAffine functions. For an example on how to replace the old API with the new one refer to <a href="https://software.intel.com/en-us/articles/code-examples-for-new-apis-and-sample-how-to-replace-deprecated-api-with-them">https://software.intel.com/en-us/articles/code-examples-for-new-apis-and-sample-how-to-replace-deprecated-api-with-them</a>.</td>
</tr>
<tr>
<td>ippiWarpAffine_8u_P4R</td>
<td>Use new WarpAffine functions. For an example on how to replace the old API with the new one refer to <a href="https://software.intel.com/en-us/articles/code-examples-for-new-apis-and-sample-how-to-replace-deprecated-api-with-them">https://software.intel.com/en-us/articles/code-examples-for-new-apis-and-sample-how-to-replace-deprecated-api-with-them</a>.</td>
</tr>
<tr>
<td>ippiWarpBilinearBack_16u_AC4R</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiWarpBilinearBack_16u_P3R</td>
<td>Use C1R flavor for each plane</td>
</tr>
<tr>
<td>ippiWarpBilinearBack_16u_P4R</td>
<td>Use C1R flavor for each plane</td>
</tr>
<tr>
<td>ippiWarpBilinearBack_32f_AC4R</td>
<td>N/A</td>
</tr>
<tr>
<td>Removed from 9.0</td>
<td>Substitution or Workaround</td>
</tr>
<tr>
<td>----------------------------------------</td>
<td>-------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>ippiWarpBilinearBack_32f_P3R</td>
<td>Use C1R flavor for each plane</td>
</tr>
<tr>
<td>ippiWarpBilinearBack_32f_P4R</td>
<td>Use C1R flavor for each plane</td>
</tr>
<tr>
<td>ippiWarpBilinearBack_8u_AC4R</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiWarpBilinearBack_8u_P3R</td>
<td>Use C1R flavor for each plane</td>
</tr>
<tr>
<td>ippiWarpBilinearBack_8u_P4R</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiWarpBilinearQuad_16u_AC4R</td>
<td>Use new WarpQuad functions. For an example on how to replace the old API with the new one refer to <a href="https://software.intel.com/en-us/articles/code-examples-for-new-apis-and-sample-how-to-replace-deprecated-api-with-them">https://software.intel.com/en-us/articles/code-examples-for-new-apis-and-sample-how-to-replace-deprecated-api-with-them</a>.</td>
</tr>
<tr>
<td>ippiWarpBilinearQuad_16u_P3R</td>
<td>Use new WarpQuad functions. For an example on how to replace the old API with the new one refer to <a href="https://software.intel.com/en-us/articles/code-examples-for-new-apis-and-sample-how-to-replace-deprecated-api-with-them">https://software.intel.com/en-us/articles/code-examples-for-new-apis-and-sample-how-to-replace-deprecated-api-with-them</a>.</td>
</tr>
<tr>
<td>ippiWarpBilinearQuad_16u_P4R</td>
<td>Use new WarpQuad functions. For an example on how to replace the old API with the new one refer to <a href="https://software.intel.com/en-us/articles/code-examples-for-new-apis-and-sample-how-to-replace-deprecated-api-with-them">https://software.intel.com/en-us/articles/code-examples-for-new-apis-and-sample-how-to-replace-deprecated-api-with-them</a>.</td>
</tr>
<tr>
<td>ippiWarpBilinearQuad_32f_AC4R</td>
<td>Use new WarpQuad functions. For an example on how to replace the old API with the new one refer to <a href="https://software.intel.com/en-us/articles/code-examples-for-new-apis-and-sample-how-to-replace-deprecated-api-with-them">https://software.intel.com/en-us/articles/code-examples-for-new-apis-and-sample-how-to-replace-deprecated-api-with-them</a>.</td>
</tr>
<tr>
<td>ippiWarpBilinearQuad_32f_P3R</td>
<td>Use new WarpQuad functions. For an example on how to replace the old API with the new one refer to <a href="https://software.intel.com/en-us/articles/code-examples-for-new-apis-and-sample-how-to-replace-deprecated-api-with-them">https://software.intel.com/en-us/articles/code-examples-for-new-apis-and-sample-how-to-replace-deprecated-api-with-them</a>.</td>
</tr>
<tr>
<td>ippiWarpBilinearQuad_32f_P4R</td>
<td>Use new WarpQuad functions. For an example on how to replace the old API with the new one refer to <a href="https://software.intel.com/en-us/articles/code-examples-for-new-apis-and-sample-how-to-replace-deprecated-api-with-them">https://software.intel.com/en-us/articles/code-examples-for-new-apis-and-sample-how-to-replace-deprecated-api-with-them</a>.</td>
</tr>
<tr>
<td>ippiWarpBilinearQuad_8u_AC4R</td>
<td>Use new WarpQuad functions. For an example on how to replace the old API with the new one refer to <a href="https://software.intel.com/en-us/articles/code-examples-for-new-apis-and-sample-how-to-replace-deprecated-api-with-them">https://software.intel.com/en-us/articles/code-examples-for-new-apis-and-sample-how-to-replace-deprecated-api-with-them</a>.</td>
</tr>
</tbody>
</table>
## Appendix C: Removed Functions for Image and Video Processing

<table>
<thead>
<tr>
<th>Removed from 9.0</th>
<th>Substitution or Workaround</th>
</tr>
</thead>
<tbody>
<tr>
<td>ippiWarpBilinearQuad_8u_P3R</td>
<td>Use new WarpQuad functions. For an example on how to replace the old API with the new one refer to <a href="https://software.intel.com/en-us/articles/code-examples-for-new-apis-and-sample-how-to-replace-deprecated-api-with-them">https://software.intel.com/en-us/articles/code-examples-for-new-apis-and-sample-how-to-replace-deprecated-api-with-them</a>.</td>
</tr>
<tr>
<td>ippiWarpBilinearQuad_8u_P4R</td>
<td>Use new WarpQuad functions. For an example on how to replace the old API with the new one refer to <a href="https://software.intel.com/en-us/articles/code-examples-for-new-apis-and-sample-how-to-replace-deprecated-api-with-them">https://software.intel.com/en-us/articles/code-examples-for-new-apis-and-sample-how-to-replace-deprecated-api-with-them</a>.</td>
</tr>
<tr>
<td>ippiWarpBilinear_16u_AC4R</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiWarpBilinear_16u_P3R</td>
<td>Use C1R flavor for each plane</td>
</tr>
<tr>
<td>ippiWarpBilinear_16u_P4R</td>
<td>Use C1R flavor for each plane</td>
</tr>
<tr>
<td>ippiWarpBilinear_32f_AC4R</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiWarpBilinear_32f_P3R</td>
<td>Use C1R flavor for each plane</td>
</tr>
<tr>
<td>ippiWarpBilinear_32f_P4R</td>
<td>Use C1R flavor for each plane</td>
</tr>
<tr>
<td>ippiWarpBilinear_8u_AC4R</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiWarpBilinear_8u_P3R</td>
<td>Use C1R flavor for each plane</td>
</tr>
<tr>
<td>ippiWarpBilinear_8u_P4R</td>
<td>Use C1R flavor for each plane</td>
</tr>
<tr>
<td>ippiWarpPerspectiveBack_16u_AC4R</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiWarpPerspectiveBack_16u_C1R</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiWarpPerspectiveBack_16u_C3R</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiWarpPerspectiveBack_16u_C4R</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiWarpPerspectiveBack_16u_P3R</td>
<td>Use C1R flavor for each plane</td>
</tr>
<tr>
<td>ippiWarpPerspectiveBack_16u_P4R</td>
<td>Use C1R flavor for each plane</td>
</tr>
<tr>
<td>ippiWarpPerspectiveBack_32f_AC4R</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiWarpPerspectiveBack_32f_C1R</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiWarpPerspectiveBack_32f_C3R</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiWarpPerspectiveBack_32f_C4R</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiWarpPerspectiveBack_32f_P3R</td>
<td>Use C1R flavor for each plane</td>
</tr>
<tr>
<td>ippiWarpPerspectiveBack_32f_P4R</td>
<td>Use C1R flavor for each plane</td>
</tr>
<tr>
<td>ippiWarpPerspectiveBack_8u_AC4R</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiWarpPerspectiveBack_8u_C1R</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiWarpPerspectiveBack_8u_C3R</td>
<td>N/A</td>
</tr>
<tr>
<td>Removed from 9.0</td>
<td>Substitution or Workaround</td>
</tr>
<tr>
<td>------------------</td>
<td>-----------------------------</td>
</tr>
<tr>
<td>ippiWarpPerspectiveBack_8u_C4R</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiWarpPerspectiveBack_8u_P3R</td>
<td>Use C1R flavor for each plane</td>
</tr>
<tr>
<td>ippiWarpPerspectiveBack_8u_P4R</td>
<td>Use C1R flavor for each plane</td>
</tr>
<tr>
<td>ippiWarpPerspectiveQuad_16u_AC4R</td>
<td>Use new WarpQuad functions. For an example on how to replace the old API with the new one refer to <a href="https://software.intel.com/en-us/articles/code-examples-for-new-apis-and-sample-how-to-replace-deprecated-api-with-them">https://software.intel.com/en-us/articles/code-examples-for-new-apis-and-sample-how-to-replace-deprecated-api-with-them</a>.</td>
</tr>
<tr>
<td>ippiWarpPerspectiveQuad_16u_C1R</td>
<td>Use new WarpQuad functions. For an example on how to replace the old API with the new one refer to <a href="https://software.intel.com/en-us/articles/code-examples-for-new-apis-and-sample-how-to-replace-deprecated-api-with-them">https://software.intel.com/en-us/articles/code-examples-for-new-apis-and-sample-how-to-replace-deprecated-api-with-them</a>.</td>
</tr>
<tr>
<td>ippiWarpPerspectiveQuad_16u_C3R</td>
<td>Use new WarpQuad functions. For an example on how to replace the old API with the new one refer to <a href="https://software.intel.com/en-us/articles/code-examples-for-new-apis-and-sample-how-to-replace-deprecated-api-with-them">https://software.intel.com/en-us/articles/code-examples-for-new-apis-and-sample-how-to-replace-deprecated-api-with-them</a>.</td>
</tr>
<tr>
<td>ippiWarpPerspectiveQuad_16u_C4R</td>
<td>Use new WarpQuad functions. For an example on how to replace the old API with the new one refer to <a href="https://software.intel.com/en-us/articles/code-examples-for-new-apis-and-sample-how-to-replace-deprecated-api-with-them">https://software.intel.com/en-us/articles/code-examples-for-new-apis-and-sample-how-to-replace-deprecated-api-with-them</a>.</td>
</tr>
<tr>
<td>ippiWarpPerspectiveQuad_16u_P3R</td>
<td>Use new WarpQuad functions. For an example on how to replace the old API with the new one refer to <a href="https://software.intel.com/en-us/articles/code-examples-for-new-apis-and-sample-how-to-replace-deprecated-api-with-them">https://software.intel.com/en-us/articles/code-examples-for-new-apis-and-sample-how-to-replace-deprecated-api-with-them</a>.</td>
</tr>
<tr>
<td>ippiWarpPerspectiveQuad_16u_P4R</td>
<td>Use new WarpQuad functions. For an example on how to replace the old API with the new one refer to <a href="https://software.intel.com/en-us/articles/code-examples-for-new-apis-and-sample-how-to-replace-deprecated-api-with-them">https://software.intel.com/en-us/articles/code-examples-for-new-apis-and-sample-how-to-replace-deprecated-api-with-them</a>.</td>
</tr>
<tr>
<td>ippiWarpPerspectiveQuad_32f_AC4R</td>
<td>Use new WarpQuad functions. For an example on how to replace the old API with the new one refer to <a href="https://software.intel.com/en-us/articles/code-examples-for-new-apis-and-sample-how-to-replace-deprecated-api-with-them">https://software.intel.com/en-us/articles/code-examples-for-new-apis-and-sample-how-to-replace-deprecated-api-with-them</a>.</td>
</tr>
<tr>
<td>ippiWarpPerspectiveQuad_32f_C1R</td>
<td>Use new WarpQuad functions. For an example on how to replace the old API with the new one refer to <a href="https://software.intel.com/en-us/articles/code-examples-for-new-apis-and-sample-how-to-replace-deprecated-api-with-them">https://software.intel.com/en-us/articles/code-examples-for-new-apis-and-sample-how-to-replace-deprecated-api-with-them</a>.</td>
</tr>
</tbody>
</table>
## Appendix C: Removed Functions for Image and Video Processing

<table>
<thead>
<tr>
<th>Removed from 9.0</th>
<th>Substitution or Workaround</th>
</tr>
</thead>
<tbody>
<tr>
<td>ippiWarpPerspectiveQuad_32f_C3R</td>
<td>Use new WarpQuad functions. For an example on how to replace the old API with the new one refer to <a href="https://software.intel.com/en-us/articles/code-examples-for-new-apis-and-sample-how-to-replace-deprecated-api-with-them">https://software.intel.com/en-us/articles/code-examples-for-new-apis-and-sample-how-to-replace-deprecated-api-with-them</a>.</td>
</tr>
<tr>
<td>ippiWarpPerspectiveQuad_32f_C4R</td>
<td>Use new WarpQuad functions. For an example on how to replace the old API with the new one refer to <a href="https://software.intel.com/en-us/articles/code-examples-for-new-apis-and-sample-how-to-replace-deprecated-api-with-them">https://software.intel.com/en-us/articles/code-examples-for-new-apis-and-sample-how-to-replace-deprecated-api-with-them</a>.</td>
</tr>
<tr>
<td>ippiWarpPerspectiveQuad_32f_P3R</td>
<td>Use new WarpQuad functions. For an example on how to replace the old API with the new one refer to <a href="https://software.intel.com/en-us/articles/code-examples-for-new-apis-and-sample-how-to-replace-deprecated-api-with-them">https://software.intel.com/en-us/articles/code-examples-for-new-apis-and-sample-how-to-replace-deprecated-api-with-them</a>.</td>
</tr>
<tr>
<td>ippiWarpPerspectiveQuad_32f_P4R</td>
<td>Use new WarpQuad functions. For an example on how to replace the old API with the new one refer to <a href="https://software.intel.com/en-us/articles/code-examples-for-new-apis-and-sample-how-to-replace-deprecated-api-with-them">https://software.intel.com/en-us/articles/code-examples-for-new-apis-and-sample-how-to-replace-deprecated-api-with-them</a>.</td>
</tr>
<tr>
<td>ippiWarpPerspectiveQuad_8u_AC4R</td>
<td>Use new WarpQuad functions. For an example on how to replace the old API with the new one refer to <a href="https://software.intel.com/en-us/articles/code-examples-for-new-apis-and-sample-how-to-replace-deprecated-api-with-them">https://software.intel.com/en-us/articles/code-examples-for-new-apis-and-sample-how-to-replace-deprecated-api-with-them</a>.</td>
</tr>
<tr>
<td>ippiWarpPerspectiveQuad_8u_C1R</td>
<td>Use new WarpQuad functions. For an example on how to replace the old API with the new one refer to <a href="https://software.intel.com/en-us/articles/code-examples-for-new-apis-and-sample-how-to-replace-deprecated-api-with-them">https://software.intel.com/en-us/articles/code-examples-for-new-apis-and-sample-how-to-replace-deprecated-api-with-them</a>.</td>
</tr>
<tr>
<td>ippiWarpPerspectiveQuad_8u_C3R</td>
<td>Use new WarpQuad functions. For an example on how to replace the old API with the new one refer to <a href="https://software.intel.com/en-us/articles/code-examples-for-new-apis-and-sample-how-to-replace-deprecated-api-with-them">https://software.intel.com/en-us/articles/code-examples-for-new-apis-and-sample-how-to-replace-deprecated-api-with-them</a>.</td>
</tr>
<tr>
<td>ippiWarpPerspectiveQuad_8u_C4R</td>
<td>Use new WarpQuad functions. For an example on how to replace the old API with the new one refer to <a href="https://software.intel.com/en-us/articles/code-examples-for-new-apis-and-sample-how-to-replace-deprecated-api-with-them">https://software.intel.com/en-us/articles/code-examples-for-new-apis-and-sample-how-to-replace-deprecated-api-with-them</a>.</td>
</tr>
<tr>
<td>ippiWarpPerspectiveQuad_8u_P3R</td>
<td>Use new WarpQuad functions. For an example on how to replace the old API with the new one refer to <a href="https://software.intel.com/en-us/articles/code-examples-for-new-apis-and-sample-how-to-replace-deprecated-api-with-them">https://software.intel.com/en-us/articles/code-examples-for-new-apis-and-sample-how-to-replace-deprecated-api-with-them</a>.</td>
</tr>
<tr>
<td>Removed from 9.0</td>
<td>Substitution or Workaround</td>
</tr>
<tr>
<td>------------------</td>
<td>-----------------------------</td>
</tr>
<tr>
<td>ippiWarpPerspectiveQuad_8u_P4R</td>
<td>Use new WarpQuad functions. For an example on how to replace the old API with the new one refer to <a href="https://software.intel.com/en-us/articles/code-examples-for-new-apis-and-sample-how-to-replace-deprecated-api-with-them">https://software.intel.com/en-us/articles/code-examples-for-new-apis-and-sample-how-to-replace-deprecated-api-with-them</a>.</td>
</tr>
<tr>
<td>ippiWarpPerspective_16u_AC4R</td>
<td>Use new WarpPerspective functions. For an example on how to replace the old API with the new one refer to <a href="https://software.intel.com/en-us/articles/code-examples-for-new-apis-and-sample-how-to-replace-deprecated-api-with-them">https://software.intel.com/en-us/articles/code-examples-for-new-apis-and-sample-how-to-replace-deprecated-api-with-them</a>.</td>
</tr>
<tr>
<td>ippiWarpPerspective_16u_C1R</td>
<td>Use new WarpPerspective functions. For an example on how to replace the old API with the new one refer to <a href="https://software.intel.com/en-us/articles/code-examples-for-new-apis-and-sample-how-to-replace-deprecated-api-with-them">https://software.intel.com/en-us/articles/code-examples-for-new-apis-and-sample-how-to-replace-deprecated-api-with-them</a>.</td>
</tr>
<tr>
<td>ippiWarpPerspective_16u_C3R</td>
<td>Use new WarpPerspective functions. For an example on how to replace the old API with the new one refer to <a href="https://software.intel.com/en-us/articles/code-examples-for-new-apis-and-sample-how-to-replace-deprecated-api-with-them">https://software.intel.com/en-us/articles/code-examples-for-new-apis-and-sample-how-to-replace-deprecated-api-with-them</a>.</td>
</tr>
<tr>
<td>ippiWarpPerspective_16u_C4R</td>
<td>Use new WarpPerspective functions. For an example on how to replace the old API with the new one refer to <a href="https://software.intel.com/en-us/articles/code-examples-for-new-apis-and-sample-how-to-replace-deprecated-api-with-them">https://software.intel.com/en-us/articles/code-examples-for-new-apis-and-sample-how-to-replace-deprecated-api-with-them</a>.</td>
</tr>
<tr>
<td>ippiWarpPerspective_16u_P3R</td>
<td>Use new WarpPerspective functions. For an example on how to replace the old API with the new one refer to <a href="https://software.intel.com/en-us/articles/code-examples-for-new-apis-and-sample-how-to-replace-deprecated-api-with-them">https://software.intel.com/en-us/articles/code-examples-for-new-apis-and-sample-how-to-replace-deprecated-api-with-them</a>.</td>
</tr>
<tr>
<td>ippiWarpPerspective_16u_P4R</td>
<td>Use new WarpPerspective functions. For an example on how to replace the old API with the new one refer to <a href="https://software.intel.com/en-us/articles/code-examples-for-new-apis-and-sample-how-to-replace-deprecated-api-with-them">https://software.intel.com/en-us/articles/code-examples-for-new-apis-and-sample-how-to-replace-deprecated-api-with-them</a>.</td>
</tr>
<tr>
<td>ippiWarpPerspective_32f_AC4R</td>
<td>Use new WarpPerspective functions. For an example on how to replace the old API with the new one refer to <a href="https://software.intel.com/en-us/articles/code-examples-for-new-apis-and-sample-how-to-replace-deprecated-api-with-them">https://software.intel.com/en-us/articles/code-examples-for-new-apis-and-sample-how-to-replace-deprecated-api-with-them</a>.</td>
</tr>
<tr>
<td>ippiWarpPerspective_32f_C1R</td>
<td>Use new WarpPerspective functions. For an example on how to replace the old API with the new one refer to <a href="https://software.intel.com/en-us/articles/code-examples-for-new-apis-and-sample-how-to-replace-deprecated-api-with-them">https://software.intel.com/en-us/articles/code-examples-for-new-apis-and-sample-how-to-replace-deprecated-api-with-them</a>.</td>
</tr>
<tr>
<td>Removed from 9.0</td>
<td>Substitution or Workaround</td>
</tr>
<tr>
<td>-----------------</td>
<td>-----------------------------</td>
</tr>
<tr>
<td>ippiWarpPerspective_32f_C3R</td>
<td>Use new WarpPerspective functions. For an example on how to replace the old API with the new one refer to <a href="https://software.intel.com/en-us/articles/code-examples-for-new-apis-and-sample-how-to-replace-deprecated-api-with-them">https://software.intel.com/en-us/articles/code-examples-for-new-apis-and-sample-how-to-replace-deprecated-api-with-them</a>.</td>
</tr>
<tr>
<td>ippiWarpPerspective_32f_C4R</td>
<td>Use new WarpPerspective functions. For an example on how to replace the old API with the new one refer to <a href="https://software.intel.com/en-us/articles/code-examples-for-new-apis-and-sample-how-to-replace-deprecated-api-with-them">https://software.intel.com/en-us/articles/code-examples-for-new-apis-and-sample-how-to-replace-deprecated-api-with-them</a>.</td>
</tr>
<tr>
<td>ippiWarpPerspective_32f_P3R</td>
<td>Use new WarpPerspective functions. For an example on how to replace the old API with the new one refer to <a href="https://software.intel.com/en-us/articles/code-examples-for-new-apis-and-sample-how-to-replace-deprecated-api-with-them">https://software.intel.com/en-us/articles/code-examples-for-new-apis-and-sample-how-to-replace-deprecated-api-with-them</a>.</td>
</tr>
<tr>
<td>ippiWarpPerspective_32f_P4R</td>
<td>Use new WarpPerspective functions. For an example on how to replace the old API with the new one refer to <a href="https://software.intel.com/en-us/articles/code-examples-for-new-apis-and-sample-how-to-replace-deprecated-api-with-them">https://software.intel.com/en-us/articles/code-examples-for-new-apis-and-sample-how-to-replace-deprecated-api-with-them</a>.</td>
</tr>
<tr>
<td>ippiWarpPerspective_8u_AC4R</td>
<td>Use new WarpPerspective functions. For an example on how to replace the old API with the new one refer to <a href="https://software.intel.com/en-us/articles/code-examples-for-new-apis-and-sample-how-to-replace-deprecated-api-with-them">https://software.intel.com/en-us/articles/code-examples-for-new-apis-and-sample-how-to-replace-deprecated-api-with-them</a>.</td>
</tr>
<tr>
<td>ippiWarpPerspective_8u_C1R</td>
<td>Use new WarpPerspective functions. For an example on how to replace the old API with the new one refer to <a href="https://software.intel.com/en-us/articles/code-examples-for-new-apis-and-sample-how-to-replace-deprecated-api-with-them">https://software.intel.com/en-us/articles/code-examples-for-new-apis-and-sample-how-to-replace-deprecated-api-with-them</a>.</td>
</tr>
<tr>
<td>ippiWarpPerspective_8u_C3R</td>
<td>Use new WarpPerspective functions. For an example on how to replace the old API with the new one refer to <a href="https://software.intel.com/en-us/articles/code-examples-for-new-apis-and-sample-how-to-replace-deprecated-api-with-them">https://software.intel.com/en-us/articles/code-examples-for-new-apis-and-sample-how-to-replace-deprecated-api-with-them</a>.</td>
</tr>
<tr>
<td>ippiWarpPerspective_8u_C4R</td>
<td>Use new WarpPerspective functions. For an example on how to replace the old API with the new one refer to <a href="https://software.intel.com/en-us/articles/code-examples-for-new-apis-and-sample-how-to-replace-deprecated-api-with-them">https://software.intel.com/en-us/articles/code-examples-for-new-apis-and-sample-how-to-replace-deprecated-api-with-them</a>.</td>
</tr>
<tr>
<td>ippiWarpPerspective_8u_P3R</td>
<td>Use new WarpPerspective functions. For an example on how to replace the old API with the new one refer to <a href="https://software.intel.com/en-us/articles/code-examples-for-new-apis-and-sample-how-to-replace-deprecated-api-with-them">https://software.intel.com/en-us/articles/code-examples-for-new-apis-and-sample-how-to-replace-deprecated-api-with-them</a>.</td>
</tr>
</tbody>
</table>
### Removed from 9.0

<table>
<thead>
<tr>
<th>Function</th>
<th>Substitution or Workaround</th>
</tr>
</thead>
<tbody>
<tr>
<td>ippiWarpPerspective_8u_P4R</td>
<td>Use new WarpPerspective functions. For an example on how to replace the old API with the new one refer to <a href="https://software.intel.com/en-us/articles/code-examples-for-new-apis-and-sample-how-to-replace-deprecated-api-with-them">https://software.intel.com/en-us/articles/code-examples-for-new-apis-and-sample-how-to-replace-deprecated-api-with-them</a>.</td>
</tr>
</tbody>
</table>

### ippj.h

<table>
<thead>
<tr>
<th>Removed from 9.0</th>
<th>Substitution or Workaround</th>
</tr>
</thead>
<tbody>
<tr>
<td>ippiAdd128_JPEG_16s8u_C1R</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiBGR555ToYCbCr411LS_MCU_16u16s_C3F3R</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiBGR555ToYCbCr422LS_MCU_16u16s_C3F3R</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiBGR555ToYCbCr444LS_MCU_16u16s_C3F3R</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiBGR555ToYCbCr_JPEG_16u8u_C3F3R</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiBGR565ToYCbCr411LS_MCU_16u16s_C3F3R</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiBGR565ToYCbCr422LS_MCU_16u16s_C3F3R</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiBGR565ToYCbCr444LS_MCU_16u16s_C3F3R</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiBGR565ToYCbCr_JPEG_16u8u_C3F3R</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiBGRToYCbCr411LS_MCU_8u16s_C3P3R</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiBGRToYCbCr411_JPEG_8u_C3P3R</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiBGRToYCbCr411_JPEG_8u_C4P3R</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiBGRToYCbCr422LS_MCU_8u16s_C3P3R</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiBGRToYCbCr422_JPEG_8u_C3P3R</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiBGRToYCbCr422_JPEG_8u_C4P3R</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiBGRToYCbCr444LS_MCU_8u16s_C3P3R</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiBGRToYCbCr_JPEG_8u_C3P3R</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiBGRToYCbCr_JPEG_8u_C4P3R</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiBGRToY_JPEG_8u_C3C1R</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiCMYKToYCCK411LS_MCU_8u16s_C4P4R</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiCMYKToYCCK422LS_MCU_8u16s_C4P4R</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiCMYKToYCCK444LS_MCU_8u16s_C4P4R</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiCMYKToYCCK_JPEG_8u_C4P4R</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiCMYKToYCCK_JPEG_8u_P4R</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiDCTQuantFwd8x8LS_JPEG_16u16s_C1R</td>
<td>N/A</td>
</tr>
<tr>
<td>Removed from 9.0</td>
<td>Substitution or Workaround</td>
</tr>
<tr>
<td>---------------------------------------------------------------</td>
<td>----------------------------</td>
</tr>
<tr>
<td>ippiDCTQuantFwd8x8LS_JPEG_8u16s_C1R</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiDCTQuantFwd8x8_JPEG_16s_C1</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiDCTQuantFwd8x8_JPEG_16s_C1I</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiDCTQuantInv8x8LS_1x1_JPEG_16s8u_C1R</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiDCTQuantInv8x8LS_2x2_JPEG_16s8u_C1R</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiDCTQuantInv8x8LS_4x4_JPEG_16s8u_C1R</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiDCTQuantInv8x8LS_JPEG_16s16u_C1R</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiDCTQuantInv8x8LS_JPEG_16s8u_C1R</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiDCTQuantInv8x8To2x2LS_JPEG_16s8u_C1R</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiDCTQuantInv8x8To4x4LS_JPEG_16s8u_C1R</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiDCTQuantInv8x8_JPEG_16s_C1</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiDCTQuantInv8x8_JPEG_16s_C1I</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiDecodeCBProgrAttach_JPEG2K_32s_C1R</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiDecodeCBProgrFree_JPEG2K</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiDecodeCBProgrGetCurBitPlaneNum_JPEG2K</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiDecodeCBProgrGetPassCounter_JPEG2K</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiDecodeCBProgrGetStateSize_JPEG2K</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiDecodeCBProgrInitAlloc_JPEG2K</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiDecodeCBProgrInit_JPEG2K</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiDecodeCBProgrSetPassCounter_JPEG2K</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiDecodeCBProgrStep_JPEG2K</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiDecodeCodeBlock_JPEG2K_1u32s_C1R</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiDecodeGetBufSize_JPEG2K</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiDecodeHuffman8x8_ACFirst_JPEG_1u16s_C1</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiDecodeHuffman8x8_ACrefine_JPEG_1u16s_C1</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiDecodeHuffman8x8_DCFirst_JPEG_1u16s_C1</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiDecodeHuffman8x8_DCrefine_JPEG_1u16s_C1</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiDecodeHuffman8x8_Direct_JPEG_1u16s_C1</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiDecodeHuffman8x8_JPEG_1u16s_C1</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiDecodeHuffmanOne_JPEG_1u16s_C1</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiDecodeHuffmanRow_JPEG_1u16s_C1P4</td>
<td>N/A</td>
</tr>
<tr>
<td>Removed from 9.0</td>
<td>Substitution or Workaround</td>
</tr>
<tr>
<td>------------------------------------------------------</td>
<td>----------------------------</td>
</tr>
<tr>
<td>ippiDecodeHuffmanSpecFree_JPEG_8u</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiDecodeHuffmanSpecGetBufSize_JPEG_8u</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiDecodeHuffmanSpecInitAlloc_JPEG_8u</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiDecodeHuffmanSpecInit_JPEG_8u</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiDecodeHuffmanStateFree_JPEG_8u</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiDecodeHuffmanStateGetBufSize_JPEG_8u</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiDecodeHuffmanStateInitAlloc_JPEG_8u</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiDecodeHuffmanStateInit_JPEG_8u</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiDiffPredFirstRow_JPEG_16s_C1</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiDiffPredRow_JPEG_16s_C1</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiEncodeFree_JPEG2K</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiEncodeGetDist_JPEG2K</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiEncodeGetRate_JPEG2K</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiEncodeGetTermPassLen_JPEG2K</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiEncodeHuffman8x8_ACFirst_JPEG_16s1u_C1</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiEncodeHuffman8x8_ACRefine_JPEG_16s1u_C1</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiEncodeHuffman8x8_DCFirst_JPEG_16s1u_C1</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiEncodeHuffman8x8_DCRefine_JPEG_16s1u_C1</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiEncodeHuffman8x8_Direct_JPEG_16s1u_C1</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiEncodeHuffman8x8_JPEG_16s1u_C1</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiEncodeHuffmanOne_JPEG_16s1u_C1</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiEncodeHuffmanRawTableInit_JPEG_8u</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiEncodeHuffmanRow_JPEG_16s1u_P4C1</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiEncodeHuffmanSpecFree_JPEG_8u</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiEncodeHuffmanSpecGetBufSize_JPEG_8u</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiEncodeHuffmanSpecInitAlloc_JPEG_8u</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiEncodeHuffmanSpecInit_JPEG_8u</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiEncodeHuffmanStateFree_JPEG_8u</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiEncodeHuffmanStateGetBufSize_JPEG_8u</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiEncodeHuffmanStateInitAlloc_JPEG_8u</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiEncodeHuffmanStateInit_JPEG_8u</td>
<td>N/A</td>
</tr>
</tbody>
</table>
## Removed from 9.0

<table>
<thead>
<tr>
<th>Function</th>
<th>Substitution or Workaround</th>
</tr>
</thead>
<tbody>
<tr>
<td>ippiEncodeInitAlloc_JPEG2K</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiEncodeLoadCodeBlock_JPEG2K_32s_C1R</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiEncodeStoreBits_JPEG2K_1u</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiFLCDecode4x4_JPEGXR_1u16s_C1R</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiFLCDecode4x4_JPEGXR_1u32s_C1R</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiFLCEncode4x4_JPEGXR_16s1u_C1R</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiFLCEncode4x4_JPEGXR_32s1u_C1R</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiGetHuffmanStatistics8x8_ACFirst_JPEG_16s_C1</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiGetHuffmanStatistics8x8_ACRefine_JPEG_16s_C1</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiGetHuffmanStatistics8x8_DCFirst_JPEG_16s_C1</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiGetHuffmanStatistics8x8_JPEG_16s_C1</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiGetHuffmanStatisticsOne_JPEG_16s_C1</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiICTFwd_JPEG2K_16s_P3IR</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiICTFwd_JPEG2K_32f_C3P3R</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiICTFwd_JPEG2K_32f_P3IR</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiICTFwd_JPEG2K_32s_P3IR</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiICTInv_JPEG2K_16s_P3IR</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiICTInv_JPEG2K_32f_P3C3R</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiICTInv_JPEG2K_32f_P3IR</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiICTInv_JPEG2K_32s_P3IR</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiJoin422LS_MCU_16s8u_P3C2R</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiJoinRow_TIFF_8u16u_C1</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiPCTFwd16x16_JPEGXR_16s_C1IR</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiPCTFwd16x16_JPEGXR_32f_C1IR</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiPCTFwd16x16_JPEGXR_32s_C1IR</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiPCTFwd8x16_JPEGXR_16s_C1IR</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiPCTFwd8x16_JPEGXR_32f_C1IR</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiPCTFwd8x16_JPEGXR_32s_C1IR</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiPCTFwd8x8_JPEGXR_16s_C1IR</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiPCTFwd8x8_JPEGXR_32f_C1IR</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiPCTFwd8x8_JPEGXR_32s_C1IR</td>
<td>N/A</td>
</tr>
<tr>
<td>Removed from 9.0</td>
<td>Substitution or Workaround</td>
</tr>
<tr>
<td>----------------------------------------</td>
<td>----------------------------</td>
</tr>
<tr>
<td>ippiPCTFwd_JPEGXR_16s_C1IR</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiPCTFwd_JPEGXR_32f_C1IR</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiPCTFwd_JPEGXR_32s_C1IR</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiPCTInv16x16_JPEGXR_16s_C1IR</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiPCTInv16x16_JPEGXR_32f_C1IR</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiPCTInv16x16_JPEGXR_32s_C1IR</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiPCTInv8x16_JPEGXR_16s_C1IR</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiPCTInv8x16_JPEGXR_32f_C1IR</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiPCTInv8x16_JPEGXR_32s_C1IR</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiPCTInv8x8_JPEGXR_16s_C1IR</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiPCTInv8x8_JPEGXR_32f_C1IR</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiPCTInv8x8_JPEGXR_32s_C1IR</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiPCTInv_JPEGXR_16s_C1IR</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiPCTInv_JPEGXR_32f_C1IR</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiPCTInv_JPEGXR_32s_C1IR</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiPackBitsRow_TIFF_8u_C1</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiQuantFwd8x8_JPEG_16s_C1I</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiQuantFwdRawTableInit_JPEG_8u</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiQuantFwdTableInit_JPEG_8u16u</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiQuantInv8x8_JPEG_16s_C1I</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiQuantTableInit_JPEG_8u16u</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiRCTFwd_JPEG2K_16s_C3P3R</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiRCTFwd_JPEG2K_16s_P3IR</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiRCTFwd_JPEG2K_32s_C3P3R</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiRCTFwd_JPEG2K_32s_P3IR</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiRCTInv_JPEG2K_16s_P3C3R</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiRCTInv_JPEG2K_16s_P3IR</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiRCTInv_JPEG2K_32s_P3C3R</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiRCTInv_JPEG2K_32s_P3IR</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiRGB555ToYCbCr_JPEG_16u8u_C3P3R</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiRGB565ToYCbCr_JPEG_16u8u_C3P3R</td>
<td>N/A</td>
</tr>
<tr>
<td>Removed from 9.0</td>
<td>Substitution or Workaround</td>
</tr>
<tr>
<td>--------------------------------------------------------------------------------</td>
<td>----------------------------</td>
</tr>
<tr>
<td>ippiRGBToYCbCr411LS_MCU_8u16s_C3P3R</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiRGBToYCbCr411_JPEG_8u_C3P3R</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiRGBToYCbCr411_JPEG_8u_C4P3R</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiRGBToYCbCr422LS_MCU_8u16s_C3P3R</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiRGBToYCbCr422_JPEG_8u_C3P3R</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiRGBToYCbCr422_JPEG_8u_C4P3R</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiRGBToYCbCr444LS_MCU_8u16s_C3P3R</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiRGBToYCr_JPEG_8u_C3P3R</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiRGBToYCr_JPEG_8u_C4P3R</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiRGBToYCr_JPEG_8u_P3R</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiRGBToY_JPEG_8u_C3C1R</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiRGBToY_JPEG_8u_P3C1R</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiReconstructPredFirstRow_JPEG_16s_C1</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiReconstructPredRow_JPEG_16s_C1</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiSampleDown411LS_MCU_8u16s_C3P3R</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiSampleDown422LS_MCU_8u16s_C3P3R</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiSampleDown444LS_MCU_8u16s_C3P3R</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiSampleDownH2V1_JPEG_8u_C1R</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiSampleDownH2V2_JPEG_8u_C1R</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiSampleDownRowH2V1_Box_JPEG_8u_C1</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiSampleDownRowH2V2_Box_JPEG_8u_C1</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiSampleUp411LS_MCU_16s8u_P3C3R</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiSampleUp422LS_MCU_16s8u_P3C3R</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiSampleUp444LS_MCU_16s8u_P3C3R</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiSampleUpH2V1_JPEG_8u_C1R</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiSampleUpH2V2_JPEG_8u_C1R</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiSampleUpRowH2V1_Triangle_JPEG_8u_C1</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiSampleUpRowH2V2_Triangle_JPEG_8u_C1</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiSplit422LS_MCU_8u16s_C2P3R</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiSplitRow_TIFF_16u8u_C1</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiSub128_JPEG_8u16s_C1</td>
<td>N/A</td>
</tr>
<tr>
<td>Removed from 9.0</td>
<td>Substitution or Workaround</td>
</tr>
<tr>
<td>--------------------------------------------------------------------------------</td>
<td>-----------------------------</td>
</tr>
<tr>
<td>ippiUnpackBitsRow_TIFF_8u_C1</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiVLCAdapt_JPEGXR</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiVLCDecode4x4_JPEGXR_1u16s_C1R</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiVLCDecode4x4_JPEGXR_1u32s_C1R</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiVLCDecodeDC420_JPEGXR_1u32s</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiVLCDecodeDC422_JPEGXR_1u32s</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiVLCDecodeDC444_JPEGXR_1u32s</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiVLCEncode4x4Flex_JPEGXR_16s1u_C1R</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiVLCEncode4x4Flex_JPEGXR_32s1u_C1R</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiVLCEncode4x4_JPEGXR_16s1u_C1R</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiVLCEncode4x4_JPEGXR_32s1u_C1R</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiVLCEncodeDC420_JPEGXR_32s1u</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiVLCEncodeDC422_JPEGXR_32s1u</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiVLCEncodeDC444_JPEGXR_32s1u</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiVLCGetStateSize_JPEGXR</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiVLCInit_JPEGXR</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiVLCScan4x4DC_JPEGXR_32s</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiVLCScanReset_JPEGXR</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiVLCScanSet_JPEGXR</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiWTFwdColLift_B53_JPEG2K_16s_C1</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiWTFwdColLift_B53_JPEG2K_32s_C1</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiWTFwdColLift_D97_JPEG2K_16s_C1</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiWTFwdColLift_D97_JPEG2K_32f_C1</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiWTFwdColLift_D97_JPEG2K_32s_C1</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiWTFwdColB53_JPEG2K_16s_C1</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiWTFwdColB53_JPEG2K_32s_C1</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiWTFwdCol_D97_JPEG2K_32f_C1</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiWTFwdRow_B53_JPEG2K_16s_C1</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiWTFwdRow_B53_JPEG2K_32s_C1</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiWTFwdRow_D97_JPEG2K_16s_C1</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiWTFwdRow_D97_JPEG2K_32f_C1</td>
<td>N/A</td>
</tr>
</tbody>
</table>
### Appendix C: Removed Functions for Image and Video Processing

<table>
<thead>
<tr>
<th>Removed from 9.0</th>
<th>Substitution or Workaround</th>
</tr>
</thead>
<tbody>
<tr>
<td>ippiWTFwdRow_D97_JPEG2K_32s_C1R</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiWTFwd_B53_JPEG2K_16s_C1IR</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiWTFwd_B53_JPEG2K_32s_C1R</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiWTFwd_B53_JPEG2K_32s_C1R</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiWTFwd_D97_JPEG2K_16s_C1IR</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiWTFwd_D97_JPEG2K_16s_C1R</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiWTFwd_D97_JPEG2K_32s_C1IR</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiWTFwd_D97_JPEG2K_32s_C1R</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiWTGetBufSize_B53_JPEG2K_16s_C1IR</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiWTGetBufSize_B53_JPEG2K_16s_C1IR</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiWTGetBufSize_B53_JPEG2K_32s_C1IR</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiWTGetBufSize_B53_JPEG2K_32s_C1IR</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiWTGetBufSize_D97_JPEG2K_16s_C1IR</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiWTGetBufSize_D97_JPEG2K_16s_C1IR</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiWTGetBufSize_D97_JPEG2K_32s_C1IR</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiWTGetBufSize_D97_JPEG2K_32s_C1IR</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiWTInvColLift_B53_JPEG2K_16s_C1</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiWTInvColLift_B53_JPEG2K_32s_C1</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiWTInvColLift_D97_JPEG2K_16s_C1</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiWTInvColLift_D97_JPEG2K_32f_C1</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiWTInvColLift_D97_JPEG2K_32s_C1</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiWTInvCol_B53_JPEG2K_16s_C1R</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiWTInvCol_B53_JPEG2K_32s_C1R</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiWTInvCol_D97_JPEG2K_32f_C1R</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiWTInvRow_B53_JPEG2K_16s_C1R</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiWTInvRow_B53_JPEG2K_32s_C1R</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiWTInvRow_D97_JPEG2K_16s_C1R</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiWTInvRow_D97_JPEG2K_32f_C1R</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiWTInvRow_D97_JPEG2K_32s_C1R</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiWTInv_B53_JPEG2K_16s_C1IR</td>
<td>N/A</td>
</tr>
<tr>
<td>Removed from 9.0</td>
<td>Substitution or Workaround</td>
</tr>
<tr>
<td>--------------------------------------------------------------------------------</td>
<td>---------------------------</td>
</tr>
<tr>
<td>ippiWIntv_B53_JPEG2K_16s_C1R</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiWIntv_B53_JPEG2K_32s_C1IR</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiWIntv_B53_JPEG2K_32s_C1R</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiWIntv_D97_JPEG2K_16s_C1IR</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiWIntv_D97_JPEG2K_16s_C1R</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiWIntv_D97_JPEG2K_32s_C1IR</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiWIntv_D97_JPEG2K_32s_C1R</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiYCCK411ToCMYKLS_MCU_16s8u_P4C4R</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiYCCK422ToCMYKLS_MCU_16s8u_P4C4R</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiYCCK444ToCMYKLS_MCU_16s8u_P4C4R</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiYCCKToCMYK_JPEG_8u_P4C4R</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiYCCKToCMYK_JPEG_8u_P4R</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiYCBcr411ToBGR555LS_MCU_16s16u_P3C3R</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiYCBcr411ToBGR565LS_MCU_16s16u_P3C3R</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiYCBcr411ToBGR565LS_MCU_16s16u_P3C3R</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiYCBcr411ToBGR565LS_MCU_16s16u_P3C3R</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiYCBcr411ToBGR565LS_MCU_16s16u_P3C3R</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiYCBcr422ToBGR_JPEG_8u_P3C3R</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiYCBcr422ToBGR_JPEG_8u_P3C3R</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiYCBcr422ToBGR_JPEG_8u_P3C3R</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiYCBcr422ToBGR_JPEG_8u_P3C3R</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiYCBcr422ToBGR_JPEG_8u_P3C3R</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiYCBcr422ToBGR_JPEG_8u_P3C3R</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiYCBcr422ToBGR_JPEG_8u_P3C3R</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiYCBcr444ToBGR555LS_MCU_16s16u_P3C3R</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiYCBcr444ToBGR565LS_MCU_16s16u_P3C3R</td>
<td>N/A</td>
</tr>
</tbody>
</table>
### Appendix C: Removed Functions for Image and Video Processing

<table>
<thead>
<tr>
<th>Removed from 9.0</th>
<th>Substitution or Workaround</th>
</tr>
</thead>
<tbody>
<tr>
<td>ippiYCbcCr444ToBGRLS_MCU_16s8u_P3C3R</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiYCbcCr444ToRGBLS_MCU_16s8u_P3C3R</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiYCbcCrToBGR555_JPEG_8u16u_P3C3R</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiYCbcCrToBGR565_JPEG_8u16u_P3C3R</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiYCbcCrToBGR_JPEG_8u_P3C3R</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiYCbcCrToBGR_JPEG_8u_P3C4R</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiYCbcCrToRGB555_JPEG_8u16u_P3C3R</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiYCbcCrToRGB565_JPEG_8u16u_P3C3R</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiYCbcCrToRGB_JPEG_8u_P3C3R</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiYCbcCrToRGB_JPEG_8u_P3C4R</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiYCbcCrToRGB_JPEG_8u_P3R</td>
<td>N/A</td>
</tr>
<tr>
<td>ippjGetLibVersion</td>
<td>N/A</td>
</tr>
</tbody>
</table>

**ippvc.h**

<table>
<thead>
<tr>
<th>Removed from 9.0</th>
<th>Substitution or Workaround</th>
</tr>
</thead>
<tbody>
<tr>
<td>ippiAdd8x8HF_16s8u_C1RS</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiAdd8x8_16s8u_C1IRS</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiAddBackPredFB_H263_8u_C1R</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiAddC8x8_16s8u_C1IR</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiAverage16x16_8u_C1IR</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiAverage16x16_8u_C1R</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiAverage8x8_8u_C1IR</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiAverage8x8_8u_C1R</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiBiDirWeightBlockImplicit_H264_8u_P2P1R</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiBiDirWeightBlockImplicit_H264_8u_P3P1R</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiBiDirWeightBlock_H264_8u_C2R</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiBiDirWeightBlock_H264_8u_P2P1R</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiBiDirWeightBlock_H264_8u_P3P1R</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiBidirWeightImplicit_H264_16u_P2P1R</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiBidirWeight_H264_16u_P2P1R</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiBidir_H264_16u_P2P1R</td>
<td>N/A</td>
</tr>
<tr>
<td>Removed from 9.0</td>
<td>Substitution or Workaround</td>
</tr>
<tr>
<td>-------------------------------------------------------</td>
<td>----------------------------</td>
</tr>
<tr>
<td>ippiCABACEncodeBinBypass_H264</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiCABACEncodeBin_H264</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiCABACEncodeResidualBlock_H264_16s</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiCABACEncodeResidualBlock_H264_32s</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiCABACFree_H264</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiCABACGetContexts_H264</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiCABACGetSize_H264</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiCABACGetStreamSize_H264</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiCABACInitAlloc_H264</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiCABACInit_H264</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiCABACSetStream_H264</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiCABACTerminateSlice_H264</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiCalcGlobalMV_MPEG4</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiCbYCr422ToYCbCr420 Rotate_8u_C2P3R</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiCbYCr422ToYCbCr420 Rotate_8u_P3R</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiChangeSpriteBrightness_MPEG4_8u_C1IR</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiCopy16x16HP_8u_C1R</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiCopy16x16QP_MPEG4_8u_C1R</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiCopy16x16_8u_C1R</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiCopy16x8HP_8u_C1R</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiCopy16x8QP_MPEG4_8u_C1R</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiCopy8x4HP_8u_C1R</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiCopy8x8HP_8u_C1R</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiCopy8x8QP_MPEG4_8u_C1R</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiCopy8x8_8u_C1R</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiCreateRLEncodeTable</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiDCT2x4x8Frw_16s_C1I</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiDCT2x4x8Inv_16s_C1I</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiDCT8x4x2To4x4Inv_DV_16s_C1I</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiDCT8x8Fwd_8u16s_C2P2</td>
<td>N/A</td>
</tr>
<tr>
<td>Removed from 9.0</td>
<td>Substitution or Workaround</td>
</tr>
<tr>
<td>--------------------------------------------------------------------------------</td>
<td>----------------------------</td>
</tr>
<tr>
<td>ippiDCT8x8InvOrSet_16s8u_P2C2</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiDCT8x8Inv_AANTransposed_16s8u_C1R</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiDCT8x8Inv_AANTransposed_16s8u_P2C2R</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiDCT8x8Inv_AANTransposed_16s_C1R</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiDCT8x8Inv_AANTransposed_16s_P2C2R</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiDecodeCAVLCChroma422DcCoeffs_H264_1u16s</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiDecodeCAVLCChroma422DcCoeffs_H264_1u32s</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiDecodeCAVLCChromaDcCoeffs_H264_1u16s</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiDecodeCAVLCChromaDcCoeffs_H264_1u32s</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiDecodeCAVLCCoeffsIdxs_H264_1u16s</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiDecodeCAVLCCoeffs_H264_1u16s</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiDecodeCAVLCCoeffs_H264_1u32s</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiDecodeChromaBlock_AVS_1u16s</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiDecodeCoeffsInterRVLCBack_MPEG4_1u16s</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiDecodeCoeffsInter_H261_1u16s</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiDecodeCoeffsInter_H263_1u16s</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiDecodeCoeffsInter_MPEG4_1u16s</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiDecodeCoeffsIntraRVLCBack_MPEG4_1u16s</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiDecodeCoeffsIntra_H261_1u16s</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiDecodeCoeffsIntra_H263_1u16s</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiDecodeCoeffsIntra_MPEG4_1u16s</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiDecodeDCIntra_H263_1u16s</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiDecodeDCIntra_MPEG4_1u16s</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiDecodeExpGolombOne_H264_1u16s</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiDecodeExpGolombOne_H264_1u32s</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiDecodeHuffmanOne_1u32s</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiDecodeHuffmanPair_1u16s</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiDecodeLumaBlockInter_AVS_1u16s</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiDecodeLumaBlockInter_AVS_1u16s</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiDeinterlaceBlendFree_8u_C1</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiDeinterlaceBlendFree_8u_C2</td>
<td>N/A</td>
</tr>
<tr>
<td>Removed from 9.0</td>
<td>Substitution or Workaround</td>
</tr>
<tr>
<td>------------------------------------------------------------</td>
<td>----------------------------</td>
</tr>
<tr>
<td>ippiDeinterlaceBlendGetSize_8u_C1</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiDeinterlaceBlendGetSize_8u_C2</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiDeinterlaceBlendInitAlloc_8u_C1</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiDeinterlaceBlendInitAlloc_8u_C2</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiDeinterlaceBlendInit_8u_C1</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiDeinterlaceBlendInit_8u_C2</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiDeinterlaceBlend_8u_C1</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiDeinterlaceBlend_8u_C2</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiDeinterlaceEdgeDetect_8u_C1R</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiDeinterlaceFilterTriangle_8u_C1R</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiDeinterlaceFilterTriangle_8u_C2R</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiDeinterlaceMedianThreshold_8u_C1R</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiDeinterlaceMotionAdaptive_8u_C1</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiDequantTransformResidualAndAdd_H264_16s-C1</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiDequantTransformResidual_H264_16s-C1I</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiDequantTransformResidual_SISP_H264_16s-C1I</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiDisassembleChroma420Intra_AVS_16s8u_C1R</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiDisassembleLumaIntra_AVS_16s8u_C1R</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiDownsampleFour16x16_H263_16s_C1R</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiDownsampleFour_H263_8u_C1R</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiEdgesDetect16x16_16u_C1R</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiEdgesDetect16x16_8u_C1R</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiEncodeChromaDcCoeffsCAVLChroma_H264_16s</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiEncodeCoeffsCAVLChromaDC2X2_H264_32s</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiEncodeCoeffsCAVLChromaDC2X4_H264_16s</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiEncodeCoeffsCAVLChromaDC2X4_H264_32s</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiEncodeCoeffsCAVLChroma_H264_16s</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiEncodeCoeffsCAVLChroma_H264_32s</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiEncodeCoeffsInter_H261_16s1u</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiEncodeCoeffsInter_H263_16s1u</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiEncodeCoeffsInter_MPEG4_16s1u</td>
<td>N/A</td>
</tr>
</tbody>
</table>
### Removed from 9.0

<table>
<thead>
<tr>
<th>Function</th>
<th>Substitution or Workaround</th>
</tr>
</thead>
<tbody>
<tr>
<td>ippiEncodeCoeffsIntra_H261_16s1u</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiEncodeCoeffsIntra_H263_16s1u</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiEncodeCoeffsIntra_MPEG4_16s1u</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiEncodeDCIntra_H263_16s1u</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiEncodeDCIntra_MPEG4_16s1u</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiExpandPlane_H264_8u_C1R</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiFilter8x8_H261_8u_C1R</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiFilterBlockBoundaryHorEdge_H263_8u_C1IR</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiFilterBlockBoundaryVerEdge_H263_8u_C1IR</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiFilterDeblocking16x16HorEdge_H263_8u_C1IR</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiFilterDeblocking16x16VerEdge_H263_8u_C1IR</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiFilterDeblocking8x8HorEdge_H263_8u_C1IR</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiFilterDeblocking8x8HorEdge_MPEG4_8u_C1IR</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiFilterDeblocking8x8VerEdge_H263_8u_C1IR</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiFilterDeblocking8x8VerEdge_MPEG4_8u_C1IR</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiFilterDeblockingChroma422HorEdge_H264_16u_C1IR</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiFilterDeblockingChroma422HorEdge_H264_8u_C1IR</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiFilterDeblockingChroma422VerEdge_H264_16u_C1IR</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiFilterDeblockingChroma422VerEdge_H264_8u_C1IR</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiFilterDeblockingChromaHorEdge_H264_16u_C1IR</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiFilterDeblockingChromaVerEdgeMBAFF_H264_16u_C1IR</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiFilterDeblockingChromaVerEdge_H264_16u_C1IR</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiFilterDeblockingChroma_HorEdge_AVS_8u_C1IR</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiFilterDeblockingChroma_HorEdge_H264_8u_C1IR</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiFilterDeblockingChroma_HorEdge_H264_8u_C2I</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiFilterDeblockingChroma_HorEdge_VC1_8u_C1IR</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiFilterDeblockingChroma_HorEdge_VC1_8u_C2I</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiFilterDeblockingChroma_VerEdge_AVS_8u_C1IR</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiFilterDeblockingChroma_VerEdge_H264_8u_C1IR</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiFilterDeblockingChroma_VerEdge_H264_8u_C2I</td>
<td>N/A</td>
</tr>
<tr>
<td>Removed from 9.0</td>
<td>Substitution or Workaround</td>
</tr>
<tr>
<td>------------------------------------------------------</td>
<td>-----------------------------</td>
</tr>
<tr>
<td>ippiFilterDeblockingChroma_VerEdge_H264_8u_C2IR</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiFilterDeblockingChroma_VerEdge_MBAFF_H264_8u_C1IR</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiFilterDeblockingChroma_VerEdge_VC1_8u_C1IR</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiFilterDeblockingChroma_VerEdge_VC1_8u_C2IR</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiFilterDeblockingLumaHorEdge_H264_16u_C1IR</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiFilterDeblockingLumaVerEdgeMBAFF_H264_16u_C1IR</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiFilterDeblockingLumaVerEdge_H264_16u_C1IR</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiFilterDeblockingLumaHorEdge_AVS_8u_C1IR</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiFilterDeblockingLumaHorEdge_H264_8u_C1IR</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiFilterDeblockingLumaHorEdge_VC1_8u_C1IR</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiFilterDeblockingLumaVerEdge_AVS_8u_C1IR</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiFilterDeblockingLumaVerEdge_H264_8u_C1IR</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiFilterDeblockingLumaVerEdge_H264_8u_C1IR</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiFilterDeblockingLumaVerEdge_VC1_8u_C1IR</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiFilterDenoiseAdaptiveFree_8u_C1</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiFilterDenoiseAdaptiveInitAlloc_8u_C1</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiFilterDenoiseAdaptive_8u_C1R</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiFilterDenoiseCASTInit</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiFilterDenoiseCASTYUV422_8u_C2R</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiFilterDenoiseCAST_8u_C1R</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiFilterDenoiseMosquitoFree_8u_C1</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiFilterDenoiseMosquitoInitAlloc_8u_C1</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiFilterDenoiseMosquito_8u_C1R</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiFilterDenoiseSmooth_8u_C1R</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiFilterDerangingSmooth8x8_MPEG4_8u_C1R</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiFilterDerangingThreshold_MPEG4_8u_P3R</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiFrameFieldSAD16x16_16s32s_C1R</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiFrameFieldSAD16x16_8u32s_C1R</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiFreeHuffmanTable_DV_32u</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiGenScaleLevel14x4_H264_8u16s_C1</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiGenScaleLevel18x8_H264_8u16s_D2</td>
<td>N/A</td>
</tr>
</tbody>
</table>
### Appendix C: Removed Functions for Image and Video Processing

<table>
<thead>
<tr>
<th>Removed from 9.0</th>
<th>Substitution or Workaround</th>
</tr>
</thead>
<tbody>
<tr>
<td>ippiGetDiff16x16B_8u16s_C1</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiGetDiff16x16_8u16s_C1</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiGetDiff16x8B_8u16s_C1</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiGetDiff16x8_8u16s_C1</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiGetDiff8x16B_8u16s_C1</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiGetDiff8x16_8u16s_C1</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiGetDiff8x4B_8u16s_C1</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiGetDiff8x4B_8u16s_C2P2</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiGetDiff8x4_8u16s_C1</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiGetDiff8x4_8u16s_C2P2</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiGetDiff8x8B_8u16s_C1</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiGetDiff8x8B_8u16s_C2P2</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiGetDiff8x8_8u16s_C1</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiGetDiff8x8_8u16s_C2P2</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiHadamard8x8Sum_VC1_16s</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiHadamard8x8Sum_VC1_8u16s</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiHuffmanDecodeSegmentOnePass_DV_8u16s</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiHuffmanDecodeSegment_DV100_8u16s</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiHuffmanDecodeSegment_DV_8u16s</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiHuffmanRunLevelTableInitAlloc_32s</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiHuffmanTableFree_32s</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiHuffmanTableInitAlloc_32s</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiInitAllocHuffmanTable_DV_32u</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiInterpolateAverage16x16_8u_C1IR</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiInterpolateAverage16x8_8u_C1IR</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiInterpolateAverage8x4_8u_C1IR</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiInterpolateAverage8x8_8u_C1IR</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiInterpolateBlock_H264_8u_P2P1R</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiInterpolateBlock_H264_8u_P3P1R</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiInterpolateChromaBlock_H264_16u_P2R</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiInterpolateChromaBlock_H264_8u_C2C2R</td>
<td>N/A</td>
</tr>
<tr>
<td>Removed from 9.0</td>
<td>Substitution or Workaround</td>
</tr>
<tr>
<td>-------------------------------------------------------</td>
<td>-----------------------------</td>
</tr>
<tr>
<td>ippiInterpolateChromaBlock_H264_8u_C2P2R</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiInterpolateChromaBlock_H264_8u_P2R</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiInterpolateChromaBottom_H264_16u_C1R</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiInterpolateChromaBottom_H264_8u_C1R</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiInterpolateChromaTop_H264_16u_C1R</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiInterpolateChromaTop_H264_8u_C1R</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiInterpolateChroma_H264_16u_C1R</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiInterpolateChroma_H264_8u_C1R</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiInterpolateChroma_H264_8u_C2P2R</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiInterpolateICBicubicBlock_VC1_8u_C1R</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiInterpolateICBilinearBlock_VC1_8u_C1R</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiInterpolateICBilinearBlock_VC1_8u_C2R</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiInterpolateLumaBlock_AVS_8u_P1R</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiInterpolateLumaBlock_H264_16u_P1R</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiInterpolateLumaBlock_H264_8u_P1R</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiInterpolateLumaBottom_H264_16u_C1R</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiInterpolateLumaBottom_H264_8u_C1R</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiInterpolateLumaTop_H264_16u_C1R</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiInterpolateLumaTop_H264_8u_C1R</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiInterpolateLuma_H264_16u_C1R</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiInterpolateLuma_H264_8u_C1R</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiInterpolateQBicubic_VC1_8u_C1R</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiInterpolateQBilinear_VC1_8u_C1R</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiInterpolateQBilinear_VC1_8u_C2R</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiMC16x16B_8u_C1</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiMC16x16_8u_C1</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiMC16x4B_8u_C1</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiMC16x4_8u_C1</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiMC16x8BUV_8u_C1</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiMC16x8B_8u_C1</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiMC16x8UV_8u_C1</td>
<td>N/A</td>
</tr>
<tr>
<td>Removed from 9.0</td>
<td>Substitution or Workaround</td>
</tr>
<tr>
<td>-----------------------------------------</td>
<td>----------------------------</td>
</tr>
<tr>
<td>ippiMC16x8_8u_C1</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiMC2x2B_8u_C1</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiMC2x2_8u_C1</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiMC2x4B_8u_C1</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiMC2x4_8u_C1</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiMC4x2B_8u_C1</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiMC4x2_8u_C1</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiMC4x4B_8u_C1</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiMC4x4_8u_C1</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiMC4x8B_8u_C1</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiMC4x8_8u_C1</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiMC8x16B_8u_C1</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiMC8x16_8u_C1</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiMC8x4B_16s8u_P2C2R</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiMC8x4B_8u_C1</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiMC8x4_16s8u_P2C2R</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiMC8x4_8u_C1</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiMC8x8B_16s8u_P2C2R</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiMC8x8B_8u_C1</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiMC8x8_16s8u_P2C2R</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiMC8x8_8u_C1</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiMeanAbsDev16x16_8u32s_C1R</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiMeanAbsDev8x8_8u32s_C1R</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiOBMC16x16HP_MPEG4_8u_C1R</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiOBMC8x8HP_MPEG4_8u_C1R</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiOBMC8x8QP_MPEG4_8u_C1R</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiPredictIntraChroma8x8_H264_8u_C1IR</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiPredictIntra_16x16_H264_8u_C1IR</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiPredictIntra_4x4_H264_8u_C1IR</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiPutIntraBlock</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiQuantInterGetSize_MPEG4</td>
<td>N/A</td>
</tr>
<tr>
<td>Removed from 9.0</td>
<td>Substitution or Workaround</td>
</tr>
<tr>
<td>------------------------------------------------------</td>
<td>----------------------------</td>
</tr>
<tr>
<td>ippiQuantInterInit_MPEG4</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiQuantInterNonuniform_VC1_16s_C1IR</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiQuantInterUniform_VC1_16s_C1IR</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiQuantInter_H263_16s_C1I</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiQuantInter_MPEG4_16s_C1I</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiQuantIntraGetSize_MPEG4</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiQuantIntraInit_MPEG4</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiQuantIntraNonuniform_VC1_16s_C1IR</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiQuantIntraUniform_VC1_16s_C1IR</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiQuantIntra_H263_16s_C1I</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiQuantIntra_MPEG2_16s_C1I</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiQuantIntra_MPEG4_16s_C1I</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiQuantInvInterGetSize_MPEG4</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiQuantInvInterInit_MPEG4</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiQuantInvInterNonuniform_VC1_16s_C1IR</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiQuantInvInterUniform_VC1_16s_C1IR</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiQuantInvInter_H263_16s_C1I</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiQuantInvInter_MPEG4_16s_C1I</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiQuantInvIntraGetSize_MPEG4</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiQuantInvIntraInit_MPEG4</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiQuantInvIntraNonuniform_VC1_16s_C1IR</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiQuantInvIntraUniform_VC1_16s_C1IR</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiQuantInvIntra_H263_16s_C1I</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiQuantInvIntra_MPEG2_16s_C1I</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiQuantInvLuma8x8_H264_32s_C1I</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiQuantInv_DV_16s_C1I</td>
<td>N/A</td>
</tr>
</tbody>
</table>
### Removed from 9.0

<table>
<thead>
<tr>
<th>Function</th>
<th>Substitution or Workaround</th>
</tr>
</thead>
<tbody>
<tr>
<td>ippiQuantInv_MPEG2_16s_C1I</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiQuantLuma8x8Inv_H264_16s_C1I</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiQuantLuma8x8_H264_16s_C1</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiQuantLuma8x8_H264_32s_C1</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiQuantWeightBlockInv_DV100_16s_C1I</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiQuantWeightBlockInv_DV_16s_C1</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiQuant_MPEG2_16s_C1I</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiQuantizeResidual4x4Fwd_H264_16s32s_C1</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiQuantizeResidual4x4Fwd_H264_16s_C1</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiQuantizeResidual4x4Fwd_H264_32s_C1</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiRangeMapping_VC1_8u_C1R</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiReconstructChroma422Inter4x4_H264High_16s8u_IP2R</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiReconstructChroma422Inter4x4_H264High_32s16u_IP2R</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiReconstructChroma422Intra4x4_H264High_16s8u_IP2R</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiReconstructChroma422Intra4x4_H264High_32s16u_IP2R</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiReconstructChroma422IntraHalf4x4_H264High_16s8u_IP2R</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiReconstructChroma422IntraHalf4x4_H264High_32s16u_IP2R</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiReconstructChroma422Inter4x4MB_H264_16s8u_C2R</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiReconstructChroma422Inter4x4MB_H264_16s8u_B2R</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiReconstructChroma422Inter4x4_H264High_32s16u_IP2R</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiReconstructChromaIntra4x4MB_H264_16s8u_C2R</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiReconstructChromaIntra4x4MB_H264_16s8u_P2R</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiReconstructChromaIntra_AVS_16s8u_C1R</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiReconstructChromaIntra4x4MB_H264_16s8u_C2R</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiReconstructChromaIntra4x4_H264High_32s16u_IP2R</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiReconstructChromaIntraHalf4x4_H264High_32s16u_IP2R</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiReconstructChromaIntraHalves4x4MB_H264_16s8u_P2R</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiReconstructChromaIntraHalvesMB_H264_16s8u_P2R</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiReconstructChromaIntraHalvesMB_H264_16s8u_C2R</td>
<td>N/A</td>
</tr>
<tr>
<td>Removed from 9.0</td>
<td>Substitution or Workaround</td>
</tr>
<tr>
<td>--------------------------------------------------------------------------------</td>
<td>----------------------------</td>
</tr>
<tr>
<td>ippireconstructChromaIntraHalvesMB_H264_16s8u_P2R</td>
<td>N/A</td>
</tr>
<tr>
<td>ippireconstructChromaIntraMB_H264_16s8u_C2R</td>
<td>N/A</td>
</tr>
<tr>
<td>ippireconstructChromaIntraMB_H264_16s8u_P2R</td>
<td>N/A</td>
</tr>
<tr>
<td>ippireconstructChromaIntra_AVS_16s8u_C1R</td>
<td>N/A</td>
</tr>
<tr>
<td>ippireconstructCoeffsInter_H261_1u16s</td>
<td>N/A</td>
</tr>
<tr>
<td>ippireconstructCoeffsInter_H263_1u16s</td>
<td>N/A</td>
</tr>
<tr>
<td>ippireconstructCoeffsInter_MPEG4_1u16s</td>
<td>N/A</td>
</tr>
<tr>
<td>ippireconstructCoeffsIntra_H261_1u16s</td>
<td>N/A</td>
</tr>
<tr>
<td>ippireconstructCoeffsIntra_H263_1u16s</td>
<td>N/A</td>
</tr>
<tr>
<td>ippireconstructDCTBlockIntra_MPEG1_32s</td>
<td>N/A</td>
</tr>
<tr>
<td>ippireconstructDCTBlockIntra_MPEG2_32s</td>
<td>N/A</td>
</tr>
<tr>
<td>ippireconstructDCTBlock_MPEG1_32s</td>
<td>N/A</td>
</tr>
<tr>
<td>ippireconstructDCTBlock_MPEG2_32s</td>
<td>N/A</td>
</tr>
<tr>
<td>ippireconstructLumaInter4x4MB_H264_16s8u_C1R</td>
<td>N/A</td>
</tr>
<tr>
<td>ippireconstructLumaInter4x4_H264High_32s16u_IP1R</td>
<td>N/A</td>
</tr>
<tr>
<td>ippireconstructLumaInter8x8MB_H264_16s8u_C1R</td>
<td>N/A</td>
</tr>
<tr>
<td>ippireconstructLumaInter8x8_H264High_32s16u_IP1R</td>
<td>N/A</td>
</tr>
<tr>
<td>ippireconstructLumaInterMB_H264_16s8u_C1R</td>
<td>N/A</td>
</tr>
<tr>
<td>ippireconstructLumaInter_AVS_16s8u_C1R</td>
<td>N/A</td>
</tr>
<tr>
<td>ippireconstructLumaIntra16x16MB_H264_16s8u_C1R</td>
<td>N/A</td>
</tr>
<tr>
<td>ippireconstructLumaIntra16x16_H264High_32s16u_IP1R</td>
<td>N/A</td>
</tr>
<tr>
<td>ippireconstructLumaIntra4x4MB_H264_16s8u_C1R</td>
<td>N/A</td>
</tr>
<tr>
<td>ippireconstructLumaIntra4x4_H264High_32s16u_IP1R</td>
<td>N/A</td>
</tr>
<tr>
<td>ippireconstructLumaIntra8x8MB_H264_16s8u_C1R</td>
<td>N/A</td>
</tr>
<tr>
<td>ippireconstructLumaIntra8x8_H264High_32s16u_IP1R</td>
<td>N/A</td>
</tr>
<tr>
<td>ippireconstructLumaIntraHalf4x4MB_H264_16s8u_C1R</td>
<td>N/A</td>
</tr>
<tr>
<td>ippireconstructLumaIntraHalf4x4_H264High_32s16u_IP1R</td>
<td>N/A</td>
</tr>
<tr>
<td>ippireconstructLumaIntraHalf8x8MB_H264_16s8u_C1R</td>
<td>N/A</td>
</tr>
<tr>
<td>ippireconstructLumaIntraHalf8x8_H264High_32s16u_IP1R</td>
<td>N/A</td>
</tr>
<tr>
<td>ippireconstructLumaIntraHalfMB_H264_16s8u_C1R</td>
<td>N/A</td>
</tr>
<tr>
<td>ippireconstructLumaIntraMB_H264_16s8u_C1R</td>
<td>N/A</td>
</tr>
<tr>
<td>Removed from 9.0</td>
<td>Substitution or Workaround</td>
</tr>
<tr>
<td>------------------------------------------------------</td>
<td>----------------------------</td>
</tr>
<tr>
<td>ippiReconstructLumaIntra_16x16MB_H264_16s8u_C1R</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiReconstructLumaIntra_AVS_16s8u_C1R</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiResample_H263_8u_P3R</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiResizeCCRotate_8u_C2R</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiSAD16x16Blocks4x4_16u32u_C1R</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiSAD16x16Blocks4x4_8u16u</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiSAD16x16Blocks8x8_16u32u_C1R</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiSAD16x16Blocks8x8_8u16u</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiSAD16x16_16u32s_C1R</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiSAD16x16_8u32s</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiSAD16x16xNI_8u16u_C1R</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiSAD16x16xN_8u16u_C1R</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiSAD16x8_8u32s_C1R</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiSAD2x2xN_8u16u_C1R</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiSAD4x4_16u32s_C1R</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiSAD4x4_8u32s</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiSAD4x4xNI_8u16u_C1R</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiSAD4x4xN_8u16u_C1R</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiSAD4x8_8u32s_C1R</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiSAD8x16_8u32s_C1R</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiSAD8x4_8u32s_C1R</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiSAD8x8_16u32s_C1R</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiSAD8x8_8u32s_C1R</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiSAD8x8_8u32s_C2R</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiSAD8x8xNI_8u16u_C1R</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiSAD8x8xN_8u16u_C1R</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiSAT8x8D_16u32s_C1R</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiSAT8x8D_8u32s_C1R</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiSATD16x16_8u32s_C1R</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiSATD16x8_8u32s_C1R</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiSATD4x4_8u32s_C1R</td>
<td>N/A</td>
</tr>
<tr>
<td>Removed from 9.0</td>
<td>Substitution or Workaround</td>
</tr>
<tr>
<td>--------------------------------------------------------------------------------</td>
<td>---------------------------</td>
</tr>
<tr>
<td>ippisATD4x8_8u32s_C1R</td>
<td>N/A</td>
</tr>
<tr>
<td>ippisATD8x16_8u32s_C1R</td>
<td>N/A</td>
</tr>
<tr>
<td>ippisATD8x4_8u32s_C1R</td>
<td>N/A</td>
</tr>
<tr>
<td>ippisATD8x8_8u32s_C1R</td>
<td>N/A</td>
</tr>
<tr>
<td>ippisSD4x4_8u32s_C1R</td>
<td>N/A</td>
</tr>
<tr>
<td>ippisSD8x8_8u32s_C1R</td>
<td>N/A</td>
</tr>
<tr>
<td>ippisScanFwd_16s_C1</td>
<td>N/A</td>
</tr>
<tr>
<td>ippisScanInv_16s_C1</td>
<td>N/A</td>
</tr>
<tr>
<td>ippisSmoothingChroma_HorEdge_VC1_16s8u_C1R</td>
<td>N/A</td>
</tr>
<tr>
<td>ippisSmoothingChroma_HorEdge_VC1_16s8u_P2C2R</td>
<td>N/A</td>
</tr>
<tr>
<td>ippisSmoothingChroma_VerEdge_VC1_16s8u_C1R</td>
<td>N/A</td>
</tr>
<tr>
<td>ippisSmoothingChroma_VerEdge_VC1_16s8u_P2C2R</td>
<td>N/A</td>
</tr>
<tr>
<td>ippisSmoothingLuma_HorEdge_VC1_16s8u_C1R</td>
<td>N/A</td>
</tr>
<tr>
<td>ippisSmoothingLuma_VerEdge_VC1_16s8u_C1R</td>
<td>N/A</td>
</tr>
<tr>
<td>ippisSpatialInterpolation_H263_8u_C1R</td>
<td>N/A</td>
</tr>
<tr>
<td>ippisSqrDiff16x16B_8u32s</td>
<td>N/A</td>
</tr>
<tr>
<td>ippisSqrDiff16x16_8u32s</td>
<td>N/A</td>
</tr>
<tr>
<td>ippisSub16x16_8u16s_C1R</td>
<td>N/A</td>
</tr>
<tr>
<td>ippisSub4x4_16u16s_C1R</td>
<td>N/A</td>
</tr>
<tr>
<td>ippisSub4x4_8u16s_C1R</td>
<td>N/A</td>
</tr>
<tr>
<td>ippisSub8x8_16u16s_C1R</td>
<td>N/A</td>
</tr>
<tr>
<td>ippisSub8x8_8u16s_C1R</td>
<td>N/A</td>
</tr>
<tr>
<td>ippisSubSAD8x8_8u16s_C1R</td>
<td>N/A</td>
</tr>
<tr>
<td>ippisSumsDiff16x16Blocks4x4_16u32s_C1R</td>
<td>N/A</td>
</tr>
<tr>
<td>ippisSumsDiff16x16Blocks4x4_8u16s_C1</td>
<td>N/A</td>
</tr>
<tr>
<td>ippisSumsDiff8x8Blocks4x4_16u32s_C1R</td>
<td>N/A</td>
</tr>
<tr>
<td>ippisSumsDiff8x8Blocks4x4_8u16s_C1</td>
<td>N/A</td>
</tr>
<tr>
<td>ippisSumsDiff8x8Blocks4x4_8u16s_C2P2</td>
<td>N/A</td>
</tr>
<tr>
<td>ippisTransform4x4Fwd_VC1_16s_C1R</td>
<td>N/A</td>
</tr>
<tr>
<td>ippisTransform4x4Fwd_VC1_16s_C1R</td>
<td>N/A</td>
</tr>
<tr>
<td>ippisTransform4x4Inv_VC1_16s_C1R</td>
<td>N/A</td>
</tr>
<tr>
<td>Removed from 9.0</td>
<td>Substitution or Workaround</td>
</tr>
<tr>
<td>------------------------------------------------------</td>
<td>----------------------------</td>
</tr>
<tr>
<td>ippiTransform4x4Inv_VC1_16s_C1R</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiTransform4x8Fwd_VC1_16s_C1IR</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiTransform4x8Fwd_VC1_16s_C1R</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiTransform4x8Inv_VC1_16s_C1IR</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiTransform4x8Inv_VC1_16s_C1R</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiTransform8x4Fwd_VC1_16s_C1IR</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiTransform8x4Fwd_VC1_16s_C1R</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiTransform8x4Inv_VC1_16s_C1IR</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiTransform8x4Inv_VC1_16s_C1R</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiTransform8x8Fwd_VC1_16s_C1IR</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiTransform8x8Fwd_VC1_16s_C1R</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiTransform8x8Inv_VC1_16s_C1IR</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiTransform8x8Inv_VC1_16s_C1R</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiTransformDequantChromaDC_H264_16s_C1I</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiTransformDequantChromaDC_SISP_H264_16s_C1I</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiTransformDequantLumaDC_H264_16s_C1I</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiTransformFwdLuma8x8_H264_16s32s_C1</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiTransformFwdLuma8x8_H264_16s_C1</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiTransformInvAddPredLuma8x8_H264_32s16u_C1R</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiTransformLuma8x8Fwd_H264_16s_C1I</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiTransformLuma8x8InvAddPred_H264_16s8u_C1R</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiTransformPrediction_H264_8u16s_C1</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiTransformQuant8x8Fwd_AVS_16s_C1</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiTransformQuantChromaDC_H264_16s_C1I</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiTransformQuantFwd4x4_H264_16s32s_C1</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiTransformQuantFwd4x4_H264_16s_C1</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiTransformQuantFwdChromaDC2x2_H264_16s_C1I</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiTransformQuantFwdChromaDC2x2_H264_32s_C1I</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiTransformQuantFwdChromaDC2x4_H264_16s_C1I</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiTransformQuantFwdChromaDC2x4_H264_32s_C1I</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiTransformQuantFwdLumaDC4x4_H264_16s_C1I</td>
<td>N/A</td>
</tr>
<tr>
<td>Removed from 9.0</td>
<td>Substitution or Workaround</td>
</tr>
<tr>
<td>------------------------------------------------------</td>
<td>-----------------------------</td>
</tr>
<tr>
<td>ippiTransformQuantFwdLumaDC4x4_H264_32s_C1I</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiTransformQuantInvAddPred4x4_H264_16s_C1IR</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiTransformQuantInvAddPred4x4_H264_32s_C1IR</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiTransformQuantInvChromaDC2x2_H264_16s_C1I</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiTransformQuantInvChromaDC2x2_H264_32s_C1I</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiTransformQuantInvChromaDC2x4_H264_16s_C1I</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiTransformQuantInvChromaDC2x4_H264_32s_C1I</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiTransformQuantLumaDC_H264_16s_C1I</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiTransformQuantResidual_H264_16s_C1I</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiTransformResidual4x4Fwd_H264_16s_C1</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiTransformResidual4x4Fwd_H264_32s_C1</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiTransformResidual4x4Inv_H264_16s_C1</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiTransformResidual4x4Inv_H264_32s_C1</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiUniDirWeightBlock_H264_8u_C1IR</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiUniDirWeightBlock_H264_8u_C1R</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiUniDirWeightBlock_H264_8u_C2R</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiUnidirWeight_H264_16u_IP2P1R</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiUpsampleFour8x8_H263_16s_C1R</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiUpsampleFour_H263_8u_C1R</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiVarMean8x8_16s32s_C1R</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiVarMean8x8_8u32s_C1R</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiVarMeanDiff16x16_8u32s_C1R</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiVarMeanDiff16x8_8u32s_C1R</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiVarSum8x8_16s32s_C1R</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiVarSum8x8_8u32s_C1R</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiVariance16x16_8u32s</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiWarpChroma_MPEG4_8u_P2R</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiWarpGetSize_MPEG4</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiWarpInit_MPEG4</td>
<td>N/A</td>
</tr>
</tbody>
</table>
### Appendix C: Removed Functions for Image and Video Processing

<table>
<thead>
<tr>
<th>Removed from 9.0</th>
<th>Substitution or Workaround</th>
</tr>
</thead>
<tbody>
<tr>
<td>ippiWarpLuma_MPEG4_8u_C1R</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiWeightPrediction_AVS_8u_C1R</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiWeightedAverage_H264_8u_C1IR</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiYCrCb411ToYCbCr422_16x4x5MB_DV_16s8u_P3C2R</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiYCrCb411ToYCbCr422_5MBDV_16s8u_P3C2R</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiYCrCb411ToYCbCr422_8x8MB_DV_16s8u_P3C2R</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiYCrCb411ToYCbCr422_EdgeDV_16s8u_P3C2R</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiYCrCb411ToYCbCr422_ZoomOut2_5MBDV_16s8u_P3C2R</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiYCrCb411ToYCbCr422_ZoomOut2_EdgeDV_16s8u_P3C2R</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiYCrCb411ToYCbCr422_ZoomOut4_5MBDV_16s8u_P3C2R</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiYCrCb411ToYCbCr422_ZoomOut8_5MBDV_16s8u_P3C2R</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiYCrCb411ToYCbCr422_ZoomOut8_EdgeDV_16s8u_P3C2R</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiYCrCb420ToYCbCr422_5MBDV_16s8u_P3C2R</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiYCrCb420ToYCbCr422_8x8x5MB_DV_16s8u_P3C2R</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiYCrCb420ToYCbCr422_ZoomOut2_5MBDV_16s8u_P3C2R</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiYCrCb420ToYCbCr422_ZoomOut4_5MBDV_16s8u_P3C2R</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiYCrCb420ToYCbCr422_ZoomOut8_5MBDV_16s8u_P3C2R</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiYCrCb422ToYCbCr422_10HalvesMB16x8_DV100_16s8u_P3C2R</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiYCrCb422ToYCbCr422_5MBDV_16s8u_P3C2R</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiYCrCb422ToYCbCr422_8x4x5MB_DV_16s8u_P3C2R</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiYCrCb422ToYCbCr422_ZoomOut2_5MBDV_16s8u_P3C2R</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiYCrCb422ToYCbCr422_ZoomOut4_5MBDV_16s8u_P3C2R</td>
<td>N/A</td>
</tr>
<tr>
<td>ippiYCrCb422ToYCbCr422_ZoomOut8_5MBDV_16s8u_P3C2R</td>
<td>N/A</td>
</tr>
<tr>
<td>ippvcGetLibVersion</td>
<td>N/A</td>
</tr>
</tbody>
</table>
Bibliography for Image Processing

This bibliography provides a list of publications that might be helpful to you in using the image processing subset of Intel IPP. This list is not complete; it serves only as a starting point. The books [Rog85], [Rog90], and [Foley90] are good resources of information on image processing and computer graphics, with mathematical formulas and code examples.


[SMPTE314M] SMPTE 314M-2005 for Television - Data Structure for DV-Based Audio, Data and Compressed Video - 25 and 50 Mb/s. The Society of Motion Picture and Television Engineers (09/05).

[SMPTE370M] SMPTE 370M-2002 for Television - Data Structure for DV-Based Audio, Data and Compressed Video at 100 Mb/s, 1080/60i, 1080/50i, 720/60p. The Society of Motion Picture and Television Engineers (07/02).

[SMPTE370M-06] SMPTE 370M-2006 for Television - Data Structure for DV-Based Audio, Data and Compressed Video at 100 Mb/s, 1080/60i, 1080/50i, 720/60p, 720/50p. The Society of Motion Picture and Television Engineers (04/06).


## Glossary

<table>
<thead>
<tr>
<th>Term</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>absolute colors</td>
<td>Colors specified by each pixel's coordinates in a color space. Intel Integrated Performance Primitives for image processing use images with absolute colors.</td>
</tr>
<tr>
<td>alpha channel</td>
<td>A color channel, also known as the opacity channel, that can be used in color models; for example, the RGBA model.</td>
</tr>
<tr>
<td>arithmetic operation</td>
<td>An operation that adds, subtracts, multiplies, divides, or squares the image pixel values.</td>
</tr>
<tr>
<td>color-twist matrix</td>
<td>A matrix used to multiply the pixel components in one color space for determining the components in another color space.</td>
</tr>
<tr>
<td>DCT</td>
<td>Acronym for the discrete cosine transform. See Discrete Cosine Transforms in Chapter 10 of this document.</td>
</tr>
<tr>
<td>dilation</td>
<td>A morphological operation that sets each output pixel to the minimum of the corresponding input pixel and its 8 neighbors.</td>
</tr>
<tr>
<td>dyadic operation</td>
<td>An operation that has two input images. It can have other input parameters as well.</td>
</tr>
<tr>
<td>element-wise operation</td>
<td>An element-wise operation performs the same operation on each element of a vector, or uses the elements of the same position in multiple vectors as inputs to the operation.</td>
</tr>
<tr>
<td>erosion</td>
<td>A morphological operation that sets each output pixel to the maximum of the corresponding input pixel and its 8 neighbors.</td>
</tr>
<tr>
<td>four-channel model</td>
<td>A color model that uses four color channels; for example, the RGBA color model.</td>
</tr>
<tr>
<td>gray scale image</td>
<td>An image characterized by a single intensity channel so that each intensity value corresponds to a certain shade of gray.</td>
</tr>
<tr>
<td>in-place operation</td>
<td>An operation whose output image is one of the input images.</td>
</tr>
<tr>
<td>linear filtering</td>
<td>In this document, 2D convolution operations.</td>
</tr>
<tr>
<td>linear image transforms</td>
<td>In this document, the discrete cosine transform (DCT).</td>
</tr>
<tr>
<td>MMX™ technology</td>
<td>An enhancement to the Intel® architecture aimed at better performance in multimedia and communications applications. The technology uses four additional data types, eight 64-bit MMX registers, and 57 additional instructions implementing the SIMD (single instruction, multiple data) technique.</td>
</tr>
<tr>
<td>monadic operation</td>
<td>An operation that has a single input image. It can have other input parameters as well.</td>
</tr>
<tr>
<td>morphological operation</td>
<td>An erosion, dilation, or their combinations.</td>
</tr>
<tr>
<td>not-in-place operation</td>
<td>An operation whose output is an image other than the input image(s). See in-place operation.</td>
</tr>
<tr>
<td>pixel depth</td>
<td>The number of bits determining each channel intensity for a single pixel in the image.</td>
</tr>
<tr>
<td>Term</td>
<td>Definition</td>
</tr>
<tr>
<td>-------------------------------</td>
<td>----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>pixel-oriented ordering</td>
<td>Storing the image information in such an order that the values of all color channels for each pixel are clustered; for example, RGBA.</td>
</tr>
<tr>
<td>planar-oriented ordering</td>
<td>Storing the image information so that all data of one color channel follow all data of another channel, thus forming a separate “plane” for each channel; for example, RRRGRRGGGBBBB....</td>
</tr>
<tr>
<td>region of interest</td>
<td>A rectangular image region on which an operation acts (or processing occurs).</td>
</tr>
<tr>
<td>RGB</td>
<td>Red-green-blue. A three-channel color model that uses red, green, and blue color channels.</td>
</tr>
<tr>
<td>RGBA</td>
<td>Red-green-blue-alpha. A four-channel color model that uses red, green, blue, and alpha (or opacity) channels.</td>
</tr>
<tr>
<td>ROI</td>
<td>See identity matrix.</td>
</tr>
<tr>
<td>row-major order</td>
<td>The default storage method for arrays in C. Memory representation is such that the rows of an array are stored contiguously. For example, for the array a[3][4], the element a[1][0] immediately follows a[0][3].</td>
</tr>
<tr>
<td>saturation</td>
<td>Using saturation arithmetic, when a number exceeds the data-range limit for its data type, it saturates to the upper data-range limit. For example, a signed word greater than 7FFFh saturates to 7FFFh. When a number is less than the lower data-range limit, it saturates to the lower data-range. For example, a signed word less than 8000h saturates to 8000h.</td>
</tr>
<tr>
<td>Intel® Streaming SIMD Extensions</td>
<td>The enhancement to the Intel architecture instruction set for the next generation processors. It incorporates a group of general-purpose floating-point instructions operating on packed data, additional packed integer instructions, together with cacheability control and state management instructions. These instructions significantly improve performance of applications using compute-intensive processing of floating-point and integer data.</td>
</tr>
<tr>
<td>three-channel model</td>
<td>A color model that uses three color channels; for example, the RGB color model.</td>
</tr>
</tbody>
</table>