Using OpenCL™ 2.0 sRGB Image Format

Tutorial

*Intel® SDK for OpenCL™ Applications - Tutorial*
<table>
<thead>
<tr>
<th>Contents</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Contents</td>
<td>2</td>
</tr>
<tr>
<td>Legal Information</td>
<td>3</td>
</tr>
<tr>
<td>Introduction</td>
<td>4</td>
</tr>
<tr>
<td>Difference between RGB and sRGB</td>
<td>4</td>
</tr>
<tr>
<td>Creating sRGB Images with OpenCL™ 2.0</td>
<td>6</td>
</tr>
<tr>
<td>Reading sRGB Images with OpenCL 2.0</td>
<td>8</td>
</tr>
<tr>
<td>Sample Prerequisites</td>
<td>8</td>
</tr>
<tr>
<td>Sample Implementation</td>
<td>8</td>
</tr>
<tr>
<td>Running And Controlling the Sample</td>
<td>9</td>
</tr>
<tr>
<td>Key Takeaways</td>
<td>9</td>
</tr>
<tr>
<td>Limitations</td>
<td>10</td>
</tr>
<tr>
<td>References</td>
<td>10</td>
</tr>
</tbody>
</table>
By using this document, in addition to any agreements you have with Intel, you accept the terms set forth below.

You may not use or facilitate the use of this document in connection with any infringement or other legal analysis concerning Intel products described herein. You agree to grant Intel a non-exclusive, royalty-free license to any patent claim thereafter drafted which includes subject matter disclosed herein.

INFORMATION IN THIS DOCUMENT IS PROVIDED IN CONNECTION WITH INTEL PRODUCTS. NO LICENSE, EXPRESS OR IMPLIED, BY ESTOPPEL OR OTHERWISE, TO ANY INTELLECTUAL PROPERTY RIGHTS IS GRANTED BY THIS DOCUMENT. EXCEPT AS PROVIDED IN INTEL'S TERMS AND CONDITIONS OF SALE FOR SUCH PRODUCTS, INTEL ASSUMES NO LIABILITY WHATSOEVER AND INTEL DISCLAIMS ANY EXPRESS OR IMPLIED WARRANTY, RELATING TO SALE AND/OR USE OF INTEL PRODUCTS INCLUDING LIABILITY OR WARRANTIES RELATING TO FITNESS FOR A PARTICULAR PURPOSE, MERCHANTABILITY, OR INFRINGEMENT OF ANY PATENT, COPYRIGHT OR OTHER INTELLECTUAL PROPERTY RIGHT.

A "Mission Critical Application" is any application in which failure of the Intel Product could result, directly or indirectly, in personal injury or death. SHOULD YOU PURCHASE OR USE INTEL'S PRODUCTS FOR ANY SUCH MISSION CRITICAL APPLICATION, YOU SHALL INDEMNIFY AND HOLD INTEL AND ITS SUBSIDIARIES, SUBCONTRACTORS AND AFFILIATES, AND THE DIRECTORS, OFFICERS, AND EMPLOYEES OF EACH, HARMLESS AGAINST ALL CLAIMS COSTS, DAMAGES, AND EXPENSES AND REASONABLE ATTORNEYS' FEES ARISING OUT OF, DIRECTLY OR INDIRECTLY, ANY CLAIM OF PRODUCT LIABILITY, PERSONAL INJURY, OR DEATH ARISING IN ANY WAY OUT OF SUCH MISSION CRITICAL APPLICATION, WHETHER OR NOT INTEL OR ITS SUBCONTRACTOR WAS NEGLIGENT IN THE DESIGN, MANUFACTURE, OR WARNING OF THE INTEL PRODUCT OR ANY OF ITS PARTS.

Intel may make changes to specifications and product descriptions at any time, without notice. Designers must not rely on the absence or characteristics of any features or instructions marked "reserved" or "undefined". Intel reserves these for future definition and shall have no responsibility whatsoever for conflicts or incompatibilities arising from future changes to them. The information here is subject to change without notice. Do not finalize a design with this information.

The products described in this document may contain design defects or errors known as errata which may cause the product to deviate from published specifications. Current characterized errata are available on request. Contact your local Intel sales office or your distributor to obtain the latest specifications and before placing your product order.

Copies of documents which have an order number and are referenced in this document, or other Intel literature, may be obtained by calling 1-800-548-4725, or go to:

Intel processor numbers are not a measure of performance. Processor numbers differentiate features within each processor family, not across different processor families. Go to:

Software and workloads used in performance tests may have been optimized for performance only on Intel microprocessors. Performance tests, such as SYSmark and MobileMark, are measured using specific computer systems, components, software, operations and functions. Any change to any of those factors may cause the results to vary. You should consult other information and performance tests to assist you in fully evaluating your contemplated purchases, including the performance of that product when combined with other products.

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

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

OpenCL and the OpenCL logo are trademarks of Apple Inc. used by permission from Khronos.

Microsoft product screen shot(s) reprinted with permission from Microsoft Corporation.

Copyright © 2014 Intel Corporation. All rights reserved.

<table>
<thead>
<tr>
<th>Optimization Notice</th>
</tr>
</thead>
<tbody>
<tr>
<td>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.</td>
</tr>
<tr>
<td>Notice revision #20110804</td>
</tr>
</tbody>
</table>
**Introduction**

sRGB is a standard RGB color space that appeared when CRT monitors were widely used. The sRGB uses the same color primaries and gamma correction that is close to CRT monitors gamma-correction, which means that the sRGB images could be displayed on CRT monitors directly, without any processing. That’s the valuable reason why the sRGB image formats became very popular in the time of CRT monitors. Today sRGB files are still widely used and the sRGB is a default format for internet files, consumer photo camera files, and so on.

The pixel intensity on CRT monitors has nonlinear dependency from the input voltage, which is the similar dependency the common RGB values have from nonlinear sRGB values. Most of image processing algorithms suppose that input data is linear RGB values. This makes sRGB to RGB conversion necessary for image processing or realistic lighting calculation during rendering.

OpenCL™ 2.0 standard provides embedded sRGB image format support. The new feature handles conversion from sRGB into RGB values and speeds up both the development time and the kernel performance. Now you don’t need to implement the conversion algorithm in your kernel.

**Difference between RGB and sRGB**

In the case of image processing, you have to perform sRGB image conversion into RGB image prior to processing the image with any other image processing algorithm. The sRGB color space resembles linear RGB, as the pixel color is also defined by Red, Green, and Blue components, but it differs from linear RGB in that color component values are not proportional to the intensity.

Assume that input and output values are in the [0:1] range. The graph below shows the nonlinear dependency between linear RGB values and sRGB values.

![Figure 1: Conversion from sRGB space into linear RGB space.
Each color component (Red, Green, and Blue) is converted independently using the same transformation](image-url)
To get linear RGB values for further processing, you need to make a nonlinear transformation from the sRGB: each component (Red, Green, or Blue) is converted independently using this formula

\[
C_{\text{RGB}} = \begin{cases} 
\frac{C_{\text{sRGB}}}{12.92} & \text{for } C_{\text{sRGB}} \leq 0.04045 \\
\left(\frac{C_{\text{sRGB}} + 0.055}{1.055}\right)^{2.4} & \text{for } C_{\text{sRGB}} > 0.04045
\end{cases}
\]

Here \(C_{\text{RGB}}\) is the Red, Green, or Blue linear RGB component value, and \(C_{\text{sRGB}}\) is the corresponding sRGB component value.

To see the importance of processing in linear RGB space, consider a blend operation for black and white images. Black pixel is represented as 0, and white pixel is represented as 1. Blend operation makes simple average of 2 images: 0.5*white + 0.5*black.

First, see the blend operation with using the linear RGB pixel values.

\[
\text{Blend: } 0.5 \times (\text{src1}) + 0.5 \times (\text{src2})
\]

\[
\begin{array}{c}
\text{black picture} \\
\text{RGB=0} \\
\text{(src1)} \\
\end{array} \quad \begin{array}{c}
\text{white picture} \\
\text{RGB=1} \\
\text{(src2)} \\
\end{array}
\]

\[
\begin{array}{c}
\text{gray picture} \\
\text{RGB=0.5} \\
\end{array}
\]

*Figure 2: Blend operation with using linear RGB input image format.*

After blending black (0) and white (1) linear RGB pixels the result is 0.5. As the input and the output data is linear RGB pixels that represent intensity, the real intensity displayed on a monitor is also 0.5 as expected.

Now consider the same operation but in the sRGB color space.
Because the arithmetic for the operation are the same, the output sRGB pixel value is 0.5. But sRGB pixel value is not proportional to the intensity that we see on a monitor and to get the real intensity value, you should convert it into a linear RGB. The linear RGB result, for example, is 0.2 and this is not the result that you might expect to see on the monitor.

With the advent of the OpenCL 2.0 standard, all the conversion mechanisms become embedded into runtimes, so you don’t need to implement the sRGB->RGB operations described above in your kernel explicitly.

Creating sRGB Images with OpenCL™ 2.0

OpenCL™ 2.0 standard introduces sRGB image format support. It means that you can create an OpenCL image object with CL_sRGB, CL_sRGBx, CL_sRGBA, or CL_sBGRA channel order flag. Note that CL_sRGB, CL_sRGBx, and CL_sBGRA are optional formats and could not be supported by some devices. For such image objects a read_image function converts sRGB pixel values into linear RGB values on the fly without any additional effort. So, you don’t need to care about the input image format in your OpenCL kernel and the same kernel can process both sRGB and RGB images in the same linear RGB space. For both image types read_image function returns linear RGB values.

To create sRGB image you can use the standard clCreateImage call with cl_image_format.image_channel_order initialized in one of the supported sRGB formats. In the code below CL_sRGBA is used:

```c
// structure that describes format of the created image
cl_image_format format;
format.image_channel_data_type = CL_UNORM_INT8;
// choose sRGBA or RGBA image format
format.image_channel_order = CL_sRGBA;
// allocate Input image and init it by sRGB pixels values
input = clCreateImage(
    context, 
    CL_MEM_READ_ONLY | CL_MEM_COPY_HOST_PTR, 
    &format, 
    &desc, 
    // buffer with s RGBA pixel values for image initialization
    // for example pixels from BMP file
```
psRGBValues, &err);

Note that the OpenCL 2.0 specification declares that OpenCL 2.0 device must support channel order CL_sRGBA with the CL_UNORM_INT8 channel data type combination only. All other combinations of the sRGB channel orders and channel data types are optional. Use the clGetSupportedImageFormats function to get list of supported formats and then to check whether a specific combination is supported by the device or not.

```c
cl_uint num_image_formats = 0;
// get number of supported formats
clGetSupportedImageFormats(
    context,
    // pass the same flag that will be used in clCreateImage
    CL_MEM_READ_ONLY | CL_MEM_COPY_HOST_PTR,
    CL_MEM_OBJECT_IMAGE2D,
    0,
    NULL,
    &num_image_formats);
// The next line allocates continuous memory region with the
// num_image_formats*sizeof(cl_image_format) size.  
// For our case it is an analog of
// malloc(num_image_formats*sizeof(cl_image_format)).
// The free() will be automatically called when
// the list objects are destroyed.
vector<cl_image_format> list(num_image_formats);
// fill the allocated buffer with the list of supported formats
clGetSupportedImageFormats(
    context,
    // pass the same flag that will be used in clCreateImage
    CL_MEM_READ_ONLY | CL_MEM_COPY_HOST_PTR,
    CL_MEM_OBJECT_IMAGE2D,
    num_image_formats,
    // pass the pointer to the first element
    // that is the pointer to the whole continuous region
    &list[0],
    NULL);
```

Another option to get sRGB image is image object creating based on existed RGB image object. For example, if you already have an RGBA image object, then you can create a new sRGBA image that shares the same data as the existing RGBA image but views pixels as sRGBA. So, you can do the implicit conversion of sRGB into RGB via the read_image function for such a new image object. To do this, the cl_image_desc structure passed to clCreateImage has to be used accordingly. The cl_image_desc.mem_object has to be set to the existing CL_RGBA image object and cl_image_desc.image_type set to CL_MEM_OBJECT_IMAGE2D. Be sure that all other values specified in cl_image_desc match the values in image descriptor for the existed RGB image object. Another restriction is that the channel data type in cl_image_format for the new image object must be the same as the channel data for the existing image object.

```c
// structure that describes format of created image
cl_image_format format;
// initialize channel data type by the same value as for the existing
imageRGBA
format.image_channel_data_type = CL_UNORM_INT8;
// choose the sRGBA channel order
format.image_channel_order = CL_sRGBA;
// image description structure
// initialized by description for existing inputRGBA image object
cl_image_desc desc = desc_inputRGB;
// initialize mem_object and image_type
desc.mem_object = inputRGBA;
desc.image_type = CL_MEM_OBJECT_IMAGE2D;
```
Reading sRGB Images with OpenCL 2.0

After an sRGB image object is created, the pixel values can be read transparently by the `read_image` call. This function explicitly converts sRGB values into linear RGB format and you get linear RGB values ready for processing. You do not need to convert sRGB into RGB in the kernel explicitly in case the sRGB format is supported by the device.

```c
kernel void process(
    __read_only image2d_t   texture) // sRGBA image object
{
    int x = (int)get_global_id(0);
    int y = (int)get_global_id(1);

    // read RGBA pixel values
    // sRGBA pixels are implicitly converted from sRGBA into RGBA space
    float4  inp = read_imagef(texture, smp, (int2)(x, y));
    // process linear RGB values
    ...
}
```

Sample Prerequisites

Before you start with the sample code, make sure your system meets the following requirements.

To build and run the sample application, you need

- A processor based on Intel® microarchitecture code name Broadwell. See the list of supported processors in the [SDK release notes](#).
- Intel OpenCL Driver for Intel Graphics available at the [OpenCL Drivers and Runtimes for Intel® Architecture](#) page.
- Intel® SDK for OpenCL™ Applications 2014 and higher, available at the [SDK main page](#).
- Microsoft Visual Studio® 2010 and higher.

Sample Implementation

The sRGB support provides benefit for many image processing kernels. To be specific a blur box linear filter was chosen in this tutorial.

**NOTE:** This tutorial focuses on sRGB support, so the efficient blur box implementation is out of scope. The blur box OpenCL kernel code in this sample has brute-force implementation and should not be treated as highly optimized version for OpenCL GPU.

The sample produces blur box filtering for the following cases:
1. You do not have CL_sRGBA support and OpenCl kernel should do explicit conversion of sRGBA into RGBA for each pixel.

2. CL_sRGBA image format is supported and there is no need to do the conversion from sRGBA into RGBA explicitly in the kernel as the read_image function does it implicitly on fly.

In both cases the single function, run_srgb_sample(..., bool use_srgb_image) is used with two different use_srgb_image flag values. For the first case (use_srgb_image=false) and for the second case (use_srgb_image=true) function does the following steps:

1. Build the OpenCL program with different options:
   a. In case of use_srgb_image==false the additional build parameter "-D ENABLE_sRGB2RGB_CONVERSION" is passed to enable explicit sRGB to RGB conversion inside kernel.
   b. In case of use_srgb_image==true there are no additional defines, and the explicit conversion code is not used in the kernel.

```c
float4 inp33 = read_imagef(texture, smp, (int2)(x+1, y+1));
#ifdef ENABLE_sRGB2RGB_CONVERSION
sRGB2RGB4(inp33);
#endif
```

2. Check supported format for images. In case of use_srgb_image==true we have to check that all devices in the context support CL_sRGBA format. To do this the clGetSupportedImageFormats call is used with the same flags that are used with the clCreateImage call (see <where>).

   The result of this call is the list of cl_image_format structures filled by the supported formats. Iterate through these structures and find the required CL_sRGBA format in the image_channel_data_type field. In the case when the CL_sRGBA format is not present in the list, the sample reports error and exits.

3. Create input image object and output buffer object. The input image is created as CL_sRGBA or CL_RGBA image, depending on the input use_srgb_image flag value (true or false).

4. Set input parameters for kernel and send the kernel to the command queue using clEnqueueNDRangeKernel. To accurately measure the kernel execution time, create the event object. See the Simple Optimization sample for information on how to measure kernel execution time. You need the kernel execution time info to compare the kernel performance between the following cases:
   a. use_srgb_image==false and explicit conversion is done in the kernel.
   b. use_srgb_image==true and implicit conversion is done inside read_image function.

5. The last step is to write the final data into a file and release memory objects.

### Running And Controlling the Sample

The sample executable is a console application without any input parameters. The sample starts by looking for GPU device on the Intel platform. If there is no such device, the sample exits with an error message. Otherwise the kernel executes for 2 cases (without and with sRGB support). For each case the kernel execution time is printed to console to demonstrate the performance benefit of using sRGB image support by the kernel running on the OpenCL device.

### Key Takeaways

The following are the main benefits that the sRGB support in OpenCL 2.0 provides:
• Single kernel for both the sRGB and the RGB input images. You do not have to write different versions of your kernel, one for processing sRGB pixels and another for processing RGB pixels.

• sRGB support enables performing the sRGB into RGB conversion implicitly using the hardware functionality. This OpenCL 2.0 feature may significantly improve kernel performance, comparing to the explicitly implemented conversion in an OpenCL kernel.

**Limitations**
The current sRGB support on the Intel OpenCL platform for the 5th Generation Intel® Core™ processors (previously Broadwell) has the following limitations:

- You can not write directly into sRGB image. This possibility is not a core OpenCL 2.0 feature but it is cl_khr_srgb_image_writes extension that is not supported by the Broadwell platform today. So, to write sRGB pixels in kernel, you have to make explicit conversion from linear RGB into sRGB and write the values to RGB image object. Use the following formula to make sRGB->RGB conversion for each channel:

\[
C_{sRGB} = \begin{cases} 
C_{RGB} + 12.92 & \text{for } C_{RGB} \leq 0.0031308 \\
1.055 \cdot (C_{RGB} + 0.055)^{\frac{1}{2}} & \text{for } C_{RGB} > 0.0031308
\end{cases}
\]

- Currently sRGB support is available only with CL sRGBA and CL sBGRA channel order, and CL_UNORM_INT8 data type. Anyway you should check supported sRGB formats using clGetSupportedImageFormats.

**References**

http://www.khronos.org/registry/cl/specs/opencl-2.0-openclc.pdf, chapter “8.3.5 Conversion rules for sRGBA and sBGRA images”

http://www.khronos.org/registry/cl/specs/opencl-2.0.pdf, chapter “5.3 Image Objects”
