Efficient antialiasing on Intel HD graphics

Alexander Reshetov
Talk Outline

• Goal
  – To present efficient post-processing antialiasing on Intel HD Graphics

• Level
  – Introductory

• Intended Audience
Intended Audience

• Practitioners (game developers)
  – Provide pluggable image-based anti-aliasing module which improves performance by 2.5X on Intel hardware (src code)

• Researchers
  – State-of-the-art in post-processing AA

• Practitioners & Researchers
  – What works on Intel hardware
What We’re Talking About

this one is antialiased ➩

← this one is not

(if you can read it, you can see it)
Substantive Outcome

• Sample code (C++ plus HLSL) of SAA
  – *Or google for “Intel MLAA”*
  – Both CPU (MLAA) and HD Graphics (SAA) code

• Separable antialiasing (SAA): the same advantages (and disadvantages) as state-of-the-art GPU MLAA
  – image-based ➔ can be used anywhere
  – “good enough” quality

• Runs at ~ 1 ms
Why It is Important

• Previous best MLAA result on Intel HD ~ 2.5 ms
  – But it is only for the top-of-the-line desktop machine
  – 5 ms on this notebook ➔ barely acceptable for games

• Can the new version be used on other hardware?
  – Sure. But we wouldn't advise it...
  – Dual support via DirectX techniques
Why Releasing the Source Code

• Helps with application customization
• Avoids some artifacts (small text)
• Could be deeply integrated into post-processing pipeline
• Makes the app independent from the driver
• Could be reused for other problems
Legal Disclaimer and Optimization Notice

INFORMATION IN THIS DOCUMENT IS PROVIDED “AS IS”. NO LICENSE, EXPRESS OR IMPLIED, BY ESTOPPEL OR OTHERWISE, TO ANY INTELLECTUAL PROPERTY RIGHTS IS GRANTED BY THIS DOCUMENT. INTEL ASSUMES NO LIABILITY WHATSOEVER AND INTEL DISCLAIMS ANY EXPRESS OR IMPLIED WARRANTY, RELATING TO THIS INFORMATION INCLUDING LIABILITY OR WARRANTIES RELATING TO FITNESS FOR A PARTICULAR PURPOSE, MERCHANTABILITY, OR INFRINGEMENT OF ANY PATENT, COPYRIGHT OR OTHER INTELLECTUAL PROPERTY RIGHT.

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.

Copyright © , Intel Corporation. All rights reserved. Intel, the Intel logo, Xeon, Core, VTune, and Cilk are trademarks of Intel Corporation in the U.S. and other countries.

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
Research Agenda

• What is antialiasing (and aliasing)
• Review of existing AA techniques
• In the context of direct and deferred shading
• Deep dive into morphological methods
  – Existing GPU implementations
  – Modification for Intel HD graphics
Graphics Pipeline

1. Scene description
2. Graphics pipeline
3. 2D image
Computer Vision Pipeline

- scene description
- data processing
- 2D image
What is Important

• https://www.google.com/search?q=computer+vision+silhouettes
  ~ 1 230 000 hits
• https://www.google.com/search?q=human+vision+silhouettes
  ~ 1 090 000 hits
Why It is Important
Aliasing Does Not Exist in Nature
## A Case Study

<table>
<thead>
<tr>
<th>Device</th>
<th>Intrinsic antialiasing</th>
</tr>
</thead>
<tbody>
<tr>
<td><img src="image" alt="Eye" /></td>
<td>✔️</td>
</tr>
<tr>
<td><img src="image" alt="Camcorder" /></td>
<td>✔️</td>
</tr>
<tr>
<td><img src="image" alt="Computer" /></td>
<td>❌</td>
</tr>
</tbody>
</table>
Resolution is Important

Scene from Call of Duty: Word at War® courtesy of Activision
Resolution is Important
Artistic Value of Coarseness

Georges-Pierre Seurat’s *La Parade* (from [WIKIPEDIA](https://en.wikipedia.org)): “The tiny juxtaposed dots of multi-colored paint allow the viewer's eye to blend colors optically, rather than having the colors blended on the canvas or pre-blended as a material pigment.”
A Single Pixel
Kazimir Malevich’s *Black Square*, 1915, Oil on Canvas
At What Resolution We Could Forget about AA?

• 300 dpi helps but aliasing is still noticeable...due to hyperacuity
  – John Hable’s blog
  – David Luebke’s The Ultimate Display

• It is possible to process at lower resolution and upscale latter
  – Saves energy
  – Scalable Smart Displays by Turner Whitted

• Morphological methods naturally reconstruct continuous images ➔ possibilities
Research Agenda

• What is antialiasing (*and* aliasing)
• **Review of existing AA techniques**
• In the context of direct and deferred shading
• Deep dive into morphological methods
  – Existing GPU implementations
  – Modification for Intel HD graphics
No Antialiasing ➔ Aliasing

This is a pixel ➔

all values are sampled @ the center
(depth, geometry, textures, etc)
Supersampling = Numerical Integration

values are sampled at multiple locations
(and resulting colors are averaged)
MultiSampling

It works because:

1. Geometry & shading values are correlated
2. ∃ hardware for bilinear interpolation of shading values

- Depth is sampled @ subsamples
- Everything else @ the center
MultiSampling Resolve
What is Deferred Shading?

- Scene description
- Deferred shading pipeline
- 2D image
Advantages of Deferred Shading/Lighting

• Shading only visible pixels
• more lights (each light is processed only for affected pixels)
• Pipeline simplification
• Post effects
MSAA + Deferred Shading
MSAA + Deferred Shading
Alternative Techniques

• Goal: a reasonable quality while reducing
  – Bandwidth and/or
  – Memory footprint
  – # of computations

• ~30 entries on AA Naming Guide on ANANDTECH

Achieved by MLAA + orthogonal to the pipeline + universal
# Filtering Approaches for Real-Time Anti-Aliasing

<table>
<thead>
<tr>
<th>Time</th>
<th>Title</th>
<th>Presenter</th>
</tr>
</thead>
<tbody>
<tr>
<td>2:00</td>
<td>Introduction</td>
<td>Diego Gutierrez</td>
</tr>
<tr>
<td>2:05</td>
<td>A Directionally Adaptive Edge Anti-Aliasing Filter</td>
<td>Jason Yang</td>
</tr>
<tr>
<td>2:20</td>
<td>Morphological Anti-Aliasing (MLAA)</td>
<td>Alexander Reshetov</td>
</tr>
<tr>
<td>2:35</td>
<td>Jimenez's MLAA &amp; SMAA (Subpixel Morphological Anti-Aliasing)</td>
<td>Jorge Jimenez</td>
</tr>
<tr>
<td>2:50</td>
<td>Hybrid CPU/GPU MLAA on the Xbox-360</td>
<td>Pete Demoreuille</td>
</tr>
<tr>
<td>3:05</td>
<td>MLAA on the PS3</td>
<td>Cedric Perthuis</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Tobias Berghoff</td>
</tr>
<tr>
<td>3:35</td>
<td>The Saboteur Anti-Aliasing (SPUAA)</td>
<td>Henry Yu</td>
</tr>
<tr>
<td>4:00</td>
<td>Subpixel Reconstruction Antialiasing (SRAA)</td>
<td>Morgan McGuire</td>
</tr>
<tr>
<td>4:15</td>
<td>FXAA 3.11 in 15 Slides</td>
<td>Timothy Lottes</td>
</tr>
<tr>
<td>4:30</td>
<td>Distance-to-edge Anti-Aliasing (DEAA)</td>
<td>Hugh Malan</td>
</tr>
<tr>
<td>4:45</td>
<td>Geometry Buffer Antialiasing (GBAA)</td>
<td>Emil Persson</td>
</tr>
<tr>
<td>4:55</td>
<td>Directionally Localized Anti-Aliasing (DLAA)</td>
<td>Dmitry Andreev</td>
</tr>
<tr>
<td>5:10</td>
<td>Anti-Aliasing Methods in CryENGINE 3</td>
<td>Tiago Sousa</td>
</tr>
</tbody>
</table>
# AA Taxonomy

<table>
<thead>
<tr>
<th>antialiasing method</th>
<th>depth</th>
<th>coverage</th>
<th>geometry</th>
<th>shading</th>
<th>storage</th>
<th>BW</th>
</tr>
</thead>
<tbody>
<tr>
<td>no antialiasing</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>multisampling</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>antialiasing MSAA</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>[Ake93]</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>coverage sampling</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>antialiasing CSAA</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>[You66]</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>supersampling</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>antialiasing SSAA</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>[Lel80]</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>MLAA [Res09, BHD10, Per10, Bir11, JME’11, FXAA [Lot11], SMAA 1x [JES’12]</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>a directionally</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>adaptive edge</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>antialiasing</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>[YP09, Joh12]</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>geometric methods</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>[BWG03, CD05, Mal10, GG12, Per12]</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>edge blurring</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>directionally</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>localized DLAA [And11]</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>normal filter</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>NFAA screening</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>space SSAA</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>[Unr11]</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>temporal reproj</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ejection [NSL’07, YNS’09, Kap10]</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>spatial/temporal</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>supersampling</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>+ morphological</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>antialiasing SAA</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>[JES’12]</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>deferred MSAA</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>[Pet10]</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>subpixel recon</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>struction antialiasing</td>
<td>SRAA</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>[CML11]</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>surface based</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>antialiasing SBAA</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>[SV12]</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>resampling</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>antialiasing RSAA</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>[Res12]</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Legend:**
- **Sampling rate per pixel**
  - **X**: not used
  - **Once**
  - **Some**
  - **Many**
  - **All**
  - **∞**
It is All About Sampling Frequency

• No antialiasing: everything done once per pixel
• Supersampling: everything many times per pixel
• Multisampling: decoupling geometry and shading
  – Also in SRAA, deferred MSAA, and RSAA (all targeting deferred shading applications)
• Coverage sampling: decoupling area estimation
• Surface-based AA: decoupling storage
Geometric Antialiasing
Geometric Antialiasing

complexity will increase as $n^2$
Geometric Antialiasing

complexity will increase as $n$
Pro et Contra of Geometric Methods

+ Infinite resolution... when it works
- Difficult to find silhouettes (needs topology)
- There could be too many of them
- Some cases are not handled (or difficult): shadows, overlaps, textured objects...
- Requires pipeline modifications
- It is application specific
Existing Geometric Antialiasing Techniques

• Distance-to-edge Anti-Aliasing, Hugh Malan
  – Output all edges → use colors to decide what to use

• Geometry Buffer **Anti-Aliasing**, Emil Persson
  – Resolve fg/bg pixels by searching the neighborhood

• Smoothed Lines **Antialiasing**, Gjol Mikkel and Mark
  – Targeting mobile devices
Raison d’être for Morphological Methods

- Infinite resolution... when it works
- Difficult to find silhouettes (needs topology)
- There could be too many of them
- Some cases are not handled (or difficult): shadows, overlaps, textured objects...
- Requires pipeline modifications
- It is application specific
Research Agenda

• What is antialiasing (and aliasing)
• Review of existing AA techniques
• In the context of direct and deferred shading
• Deep dive into morphological methods
  – Existing GPU implementations
  – Modification for Intel HD graphics
Morphological Antialiasing

1. Somehow find silhouettes in images (and hope that it will correspond to real objects)
2. Blend (aka filter) colors around the silhouettes
MLAA in 3 Steps
1. Discontinuities
2. Silhouettes
3. Color Blending
How to Blend Colors
How to Blend Colors

\[
\text{pixel} \  \text{color} \ = \ \begin{cases} 
\text{red} \\
\text{blue}
\end{cases} \times \begin{cases} 
\text{left pixel} \\
\text{current pixel}
\end{cases} + \begin{cases} 
\text{green} \\
\text{blue}
\end{cases} \times \begin{cases} 
\text{left pixel} \\
\text{current pixel}
\end{cases}
\]

( \text{red} \quad \text{comes from the left pixel,} \quad \text{blue} \quad \text{— from the current one} )
### How to decide if pixels are different

<table>
<thead>
<tr>
<th>Method</th>
<th>Features</th>
</tr>
</thead>
</table>
| threshold for each color channel | ≠ human vision  
  • issues with illumination changes near silhouettes |
| luminosity [ITU-R BT. 709] | false negatives |
| Non-linear thresholding (in God of War 3 and SMAA) |  
  • good detection over the whole range  
  • requires artist’s adjustment |
| depth only |  
  • choosing a scale is difficult  
  • problems with corners |
| Local filter (3x3 Sobol in FXAA) |  
  • avoids spurious edges  
  • requires more computations |
MLAA Steps

1. Find discontinuities between pixels
2. Extract silhouettes
3. Blend colors
Pixel Discontinuities \(\Rightarrow\) Silhouettes

silhouette segments start/end at edges of pixels at which horizontal and vertical separation lines intersect
Silhouette Disambiguation

for each separation line

• look at all start/end points on adjacent orthogonal lines

• choose the longest segment
## MLAA’s Lineage

<table>
<thead>
<tr>
<th>Year</th>
<th>Feature</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>2009</td>
<td>original CPU implementation</td>
<td>proof of concept</td>
</tr>
<tr>
<td></td>
<td>Metro 2033, Saboteur</td>
<td>unpublished</td>
</tr>
<tr>
<td>2010</td>
<td>Practical Morphological Antialiasing on the GPU</td>
<td>recursive doubling</td>
</tr>
<tr>
<td></td>
<td>God of War 3 by Sony (reference implementation)</td>
<td>inside current frame</td>
</tr>
<tr>
<td></td>
<td>AMD driver (now MLAA 2.0)</td>
<td>allowing older games</td>
</tr>
<tr>
<td></td>
<td>Nvidia FXAA (Sobol; fractional supersampling)</td>
<td>many modes</td>
</tr>
<tr>
<td>2011</td>
<td>Jimenez’ MLAA @ GPU Pro 2</td>
<td>leveraging TS, src available</td>
</tr>
<tr>
<td></td>
<td>Hybrid CPU/GPU MLAA</td>
<td>Xbox 360</td>
</tr>
<tr>
<td></td>
<td>Siggraph course</td>
<td>CPU/GPU/consoles</td>
</tr>
<tr>
<td>2012</td>
<td>SMAA (from 1x to 4x supersampling)</td>
<td>many modes; improved</td>
</tr>
<tr>
<td></td>
<td>Nvidia TXAA</td>
<td>Kepler+</td>
</tr>
</tbody>
</table>
MLAA GPU

• Restrict neighborhood search \(\Rightarrow\) precompute everything and use textures
• Use texture sampler creatively
• Mask operations by using the stencil buffer
• Bottom line:
  – Avoid branches
  – Trade memory BW for computations

As implemented in Jorge Jimenez’ MLAA and SMAA
How to Train Your Texture Unit

• Texture stores 0.0 (no edge) and 1.0 (edge)
  – Sample at (0,0) – get back 0.0 or 1.0
  – Sample at offset position – learn 4 values at once

\[ f_{xy}(b,x,y) = f_x(b_1, b_2, x) \cdot y + f_x(b_3, b_4, x) \cdot (1 - y) \]
SMAA performance

• GTX 580, 1280x720 resolution
  – 0.35 ms for 1 color sample per pixel
  – 1 ms for 2 color samples
• Intel® HD 4000 Graphics
  – 2.5 ms (1 sample)
Research Agenda

• What is antialiasing (and aliasing)
• Review of existing AA techniques
• In the context of direct and deferred shading
• Deep dive into morphological methods
  – Existing GPU implementations
  – Modification for Intel HD graphics
What We Know about Intel HD 4000

• Well-balanced machine
• With flexible instruction set
• Does not have a specialized memory
• Texture unit is small
• Good integer performance (for GPU)

• SMAA characteristics are not particularly enticing
  – Avoid branches
  – Trade memory BW for computations
Reducing BW

- Use bits instead of floats
  - Addressable as bytes
  - Extracting bits with firstbitlow/firstbithigh (SL 5)
- One caveat: no support for 2D arrays of bits
- Solution: never mix horizontal and vertical operations

Let’s call the new algorithm Separable Anitialiasing
Separating Horizontal and Vertical Operations

don’t need blue vertical lines to deduce the end points for horizontal separation lines.
HLSL Edge-Detection Kernel

- 32 threads in a thread group
- Each thread handles a column of 32 pixels
- Horizontal: InterlockedAdd to avoid conflicts between different threads
- Vertical: Accumulate bits and write them at once

```c
// We are processing the color buffer in tiles of (32x32) pixels.
// Each thread processes a column of 32 pixels.
// Each thread group has 32 such threads, so corresponds to the processing of a tile.
// Therefore the thread-related system values are automatically assigned thus:
// SV_GroupID = tile coordinates (e.g. (3,2,0) = 3rd tile from the left, 2nd tile from top)
// SV_DispatchThreadID = coordinates of the column of 32 pixels of the color buffer we are processing
// (X direction: in pixels; Y direction: in tiles)
// SV_GroupThreadID = which column in tile we are processing (e.g. (27,0,0) = tile’s 28th column).

[numthreads(32, 1, 1)] // Each thread group is a grid of (32, 1, 1) = 32 threads
void CSMain(
uint3 TileCoords : SV_GroupID,
            uint3 ColumnCoords : SV_DispatchThreadID,
            uint3 TileColumnIndex : SV_GroupThreadID)
{
    float3 LumWeights = float3(0.2126f, 0.7152f, 0.0722f);
    float LumThreshold = 30.f / 256.f;

    // ColumnCoords.y is in tiles, so need to multiply by 32 to get proper coords for Load
    uint2 PixelCoords = uint2(ColumnCoords.x, ColumnCoords.y << 5);
    float4 Pixel = ColorBuffer.Load( float3(PixelCoords, 0) );
    uint EdgeVFlag = 0;

    [unroll] for ( int i = 0; i < 32; i++ ) {
        // Load bottom neighbor and right neighbor
        float4 Bottom = ColorBuffer.Load( uint3(PixelCoords.x, PixelCoords.y + 1, 0) );
        float4 Right = ColorBuffer.Load( uint3(PixelCoords.x + 1, PixelCoords.y, 0) );

        // Compute differences
        float DiffBottom = abs((Pixel.r - Bottom.r) * LumWeights.r +
                               (Pixel.g - Bottom.g) * LumWeights.g +
                               (Pixel.b - Bottom.b) * LumWeights.b);
        float DiffRight = abs((Pixel.r - Right.r) * LumWeights.r +
                               (Pixel.g - Right.g) * LumWeights.g +
                               (Pixel.b - Right.b) * LumWeights.b);

        if (DiffBottom > LumThreshold) {
            // We have a horizontal edge so update the horizontal edge buffer now.
            EdgeHBitArray.InterlockedAdd((TileCoords.x + ((ColumnCoords.y << 5) + i) * ColorBufferTileCount.x) << 2, 1 << (31 - TileColumnIndex.x));
        }
        if (DiffRight > LumThreshold) {
            // We have a vertical edge
            EdgeVFlag += 1 << (31 - i);
        }
        Pixel = Bottom;
        PixelCoords.y++;
    }
    EdgeVBitArray.Store((TileCoords.y + ColumnCoords.x * ColorBufferTileCount.y) << 2, EdgeVFlag);
}
```
Blending Pass

• Detect silhouettes ➔ blend colors (no reason to store the silhouettes)
• One horizontal and one vertical pass
• Also want to do it with tiles
• **Problem**: pixels @ borders could be blended twice
Solution: Consider Only Edges with Padding
Side Effect: Helps with Small Text

<table>
<thead>
<tr>
<th></th>
<th>separable antialiasing algorithm</th>
</tr>
</thead>
<tbody>
<tr>
<td>No AA</td>
<td></td>
</tr>
<tr>
<td>SAA</td>
<td></td>
</tr>
<tr>
<td>MLAA</td>
<td></td>
</tr>
</tbody>
</table>
Performance @ 1280x720

- 0.5 ms for edge detection
- 0.25 ms for each of (H/V) blending passes
- The same performance for OpenCL
- 20% faster with hand-coded optimization
  - Slicing register file
  - Using media reads/writes
Wrap-up: SAA characteristics (and how it is optimized for Intel HD Graphics)

- Lots of integer ops
- Branchy code benefits from short SIMD size (16)
- Cache-optimized layout
  - Processing of vertical edges allows efficient cache-friendly memory ops
  - InterlockedAdds for horizontal edges have small impact
Thank You!

The Team

Alexei Soupirov
OpenCL implementation
low-level optimization

Thomas F Raoux
HLSL implementation

Alexandre De Pereyra
CPU optimization
code samples