Practical Morphological Anti-Aliasing on the GPU

SIGGRAPH 2010 Talk

Venceslas BIRI, Adrien HERUBEL & Stéphane DEVERLY

27 mai 2010
Univ Paris Est - LIGM & Duran Duboi
Motivation

Geometrical antialiasing

- Without antialiasing
- Supersampling
- Multisampling [Akeley 93] and Coverage sampling [Nvidia]

Limitations

- Not trivial with deferred shading
- Memory consumption. Geometrical dependency

© Martin Korndörfer, 2005
Motivation

Geometrical antialiasing

- Without antialiasing
- Supersampling
- Multisampling [Akeley 93] and Coverage sampling [Nvidia]

Limitations

- Not trivial with deferred shading
- Memory consumption. Geometrical dependency

© Martin Korndörfer, 2005
Motivation

Geometrical antialiasing

- Without antialiasing
- Supersampling
- Multisampling [Akeley 93] and Coverage sampling [Nvidia]

Limitations

- Not trivial with deferred shading
- Memory consumption. Geometrical dependency

© Martin Korndörfer, 2005
Motivation

Geometrical antialiasing

- Without antialiasing
- Supersampling
- Multisampling [Akeley 93] and Coverage sampling [Nvidia]

Limitations

- Not trivial with deferred shading
- Memory consumption. Geometrical dependency

© Martin Korndörfer, 2005
Motivation

Post processing antialiasing

MorphoLogical AntiAliasing (MLAA) [Reshetov, HPG 09]
Similar technique used in God of War III and Saboteur (PS3)

Limitations

- Full CPU implementation, low rendering time on 1024x768 image
- Costly data transfer between GPU & CPU

Objectives

- Adapt MLAA to GPU
- Keep it as a post process technique
Motivation

Post processing antialiasing

MorphoLogical AntiAliasing (MLAA) [Reshetov, HPG 09]
Similar technique used in God of War III and Saboteur (PS3)

Limitations

- Full CPU implementation, low rendering time on 1024x768 image
- Costly data transfer between GPU & CPU

Objectives

- Adapt MLAA to GPU
- Keep it as a post process technique
**Motivation**

**Post processing antialiasing**

MorphoLogical AntiAliasing (MLAA) [Reshetov, HPG 09]
Similar technique used in God of War III and Saboteur (PS3)

**Limitations**

- Full CPU implementation, low rendering time on 1024x768 image
- Costly data transfer between GPU & CPU

**Objectives**

- Adapt MLAA to GPU
- Keep it as a post process technique
Morphological Antialiasing

Antialiasing as a post process

From that ...
Antialiasing as a post process

... we want that
Morphological Antialiasing

Antialiasing as a post process

Border pixels must blend with their neighbor ...
Morphological Antialiasing

Antialiasing as a post process

... along ‘L’ shapes
Morphological Antialiasing

Antialiasing as a post process

... along ‘L’ shapes

\[ C_{new} = C_{old} \times (1 - \alpha) + \alpha C_{opposite} \]
Morphological Antialiasing

Antialiasing as a post process

MLAA algorithm on the CPU

1. Detect antialiased lines:
   - Detect ‘L’, ‘U’ and ‘Z’ shapes
   - ‘U’ or ‘Z’ are split into two ‘L’

2. Blend color along the ‘L’ shapes
   - Blend pixels inside the ‘L’ shape with their opposite neighbor
   - Weighted by the trapeze area
Morphological Antialiasing

Antialiasing as a post process

MLAA algorithm on the CPU

1. Detect antialiased lines:
   - Detect ‘L’, ‘U’ and ‘Z’ shapes
   - ‘U’ or ‘Z’ are split into two ‘L’

2. Blend color along the ‘L’ shapes
   - Blend pixels inside the ‘L’ shape with their opposite neighbor
   - Weighted by the trapeze area
Morphological Antialiasing

Antialiasing as a post process

MLAA algorithm on the CPU

1. Detect antialiased lines:
   - Detect ‘L’, ‘U’ and ‘Z’ shapes
   - ‘U’ or ‘Z’ are split into two ‘L’

2. Blend color along the ‘L’ shapes
   - Blend pixels inside the ‘L’ shape with their opposite neighbor
   - Weighted by the trapeze area
Morphological Antialiasing

Antialiasing as a post process

MLAA algorithm on the CPU

1. Detect antialiased lines:
   - Detect ‘L’, ‘U’ and ‘Z’ shapes
   - ‘U’ or ‘Z’ are split into two ‘L’

2. Blend color along the ‘L’ shapes
   - Blend pixels inside the ‘L’ shape with their opposite neighbor
   - Weighted by the trapeze area
Morphological Antialiasing

Antialiasing as a post process

MLAA algorithm on the CPU

1. Detect antialiased lines:
   - Detect ‘L’, ‘U’ and ‘Z’ shapes
   - ‘U’ or ‘Z’ are split into two ‘L’

2. Blend color along the ‘L’ shapes
   - Blend pixels inside the ‘L’ shape with their opposite neighbor
   - Weighted by the trapeze area
Morphological Antialiasing

Antialiasing as a post process

### MLAA algorithm on the CPU

1. **Detect antialiased lines:**
   - Detect ‘L’, ‘U’ and ‘Z’ shapes
   - ‘U’ or ‘Z’ are split into two ‘L’

2. **Blend color along the ‘L’ shapes**
   - Blend pixels inside the ‘L’ shape with their opposite neighbor
   - *Weighted by the trapezoid area*
GPU Implementation

Implementation on GPU issues

- Non linear image processing involves deep branching
- Algorithm requires sparse memory access
- GPU is limited to pixel-wise operation
MLAA on GPU

GPU Implementation

Implementation on GPU issues

- Non linear image processing involves deep branching
- Algorithm requires sparse memory access
- GPU is limited to pixel-wise operation

How can we detect the L shapes?
To blend the red pixel, we need to compute the yellow area.
This can be obtained using the pixel’s relative position $p$ related to the L shape and its length $L/2$

$$\alpha = \left(1 + \frac{\frac{L}{2} - p}{\frac{L}{2}}\right) / 4.0$$
The red pixel position is the minimum of the distance to the two discontinuity line extremities.
MLAA on GPU

Algorithm overview

1. Detect discontinuity lines
2. Compute the positions
   - On rows and columns
   - Related to the two extremities
   - Using the “recursive doubling” algorithm
3. Compute the trapeze area for each pixel
4. Final blending
Algorithm overview

1. Detect discontinuity lines
2. Compute the positions
   - On rows and columns
   - Related to the two extremities
   - Using the “recursive doubling” algorithm
3. Compute the trapeze area for each pixel
4. Final blending
MLAA on GPU

Algorithm overview

1. Detect discontinuity lines
2. Compute the positions
   - On rows and columns
   - Related to the two extremities
   - Using the “recursive doubling” algorithm
3. Compute the trapeze area for each pixel
4. Final blending
Algorithm overview

1. Detect discontinuity lines
2. Compute the positions
   - On rows and columns
   - Related to the two extremities
   - Using the “recursive doubling” algorithm
3. Compute the trapeze area for each pixel
4. Final blending
MLAA on GPU

Algorithm overview

1. Detect discontinuity lines
2. Compute the positions
   - On rows and columns
   - Related to the two extremities
   - Using the “recursive doubling” algorithm
3. Compute the trapeze area for each pixel
4. Final blending
MLAA on GPU

Computation of discontinuity lines

1 - Detect discontinuities of color

- Using a simple distance on $L^* a^* b^*$ color space
- Result stored in a texture

channel R: discontinuity in pixel bottom
channel G: discontinuity in pixel right
1 - Detect discontinuities of color

- Using a simple distance on $L^* a^* b^*$ color space
- Result stored in a texture

channel R : discontinuity in pixel bottom
channel G : discontinuity in pixel right
2 - Computation of line length and relative position

On rows and columns and in both directions, using a ‘recursive doubling’ technique.

```
| 0 | 1 | 1 | 1 | 1 | 1 | 0 | 1 | 1 | 1 | 1 |
```

Discontinuity value

```
| X | 0 | 0 | 0 | 0 | 0 | 0 | X | 0 | 0 | 0 |
```

Distance to left edge

Stop value

INITIALIZATION
Line length computation

### 2 - Computation of line length and relative position

On rows and columns and in both directions, using a ‘recursive doubling’ technique.

**STEP 1**

- **Distance to left edge**: 0
- **Stop value**: 0

```
0 1 1 1 1 1 1 0 1 1 1 1
```

```
X 0 0 0 0 0 0 X 0 0 0 0
```

```
0 0 0 0 0 0 0 0 0 0 0 0
```
2 - Computation of line length and relative position

On rows and columns and in both directions, using a ‘recursive doubling’ technique.
2 - Computation of line length and relative position

On rows and columns and in both directions, using a ‘recursive doubling’ technique.

STEP 2
MLAA on GPU

Line length computation

2 - Computation of line length and relative position

On rows and columns and in both directions, using a ‘recursive doubling’ technique.

```
0 1 1 1 1 0 1 1 1 1
```

Distance to left edge  
Stop value

```
X 0 1 2 3 0 0 X 0 1 1 2 1
```

STEP 2
Line length computation

2 - Computation of line length and relative position

On rows and columns and in both directions, using a ‘recursive doubling’ technique.

![Diagram showing computation process]

STEP 3
Line length computation

2 - Computation of line length and relative position

On rows and columns and in both directions, using a ‘recursive doubling’ technique.

![Diagram showing line length computation process and step 3 example](image)
2 - Computation of line length and relative position

Number of iteration for row = \( \log_2(\text{width}) \). Number of iteration for columns = \( \log_2(\text{height}) \). Results are stored in textures (one for the rows, one for the columns).
MLAA on GPU

**Blending map**

### 3 - Computing the blending factors

- Select L shape type of each pixel (if any)
- For each remaining pixel, compute the trapeze area
- Results stored in a blending map, in each direction

8 cases

![Diagram](image)
MLAA on GPU

Blending map

3 - Computing the blending factors

- Select L shape type of each pixel (if any)
- For each remaining pixel, compute the trapeze area
- Results stored in an blending map, in each direction

\[ \alpha = \left( 1 + \frac{\frac{L}{2} - p^{-1}}{\frac{L}{2}} \right) / 4.0 \]
3 - Computing the blending factors

- Select L shape type of each pixel (if any)
- For each remaining pixel, compute the trapeze area
- Results stored in an blending map, in each direction
Final rendering

4 - Final blending

- We blend each pixel with its 4 neighbors
- Using previously computed blending map

![Using MLAA](image1.png)

![Without Antialiasing](image2.png)
Results

**Computation times**

<table>
<thead>
<tr>
<th>Image size</th>
<th>1248x1024</th>
<th>1600x1200</th>
</tr>
</thead>
<tbody>
<tr>
<td>MLAA CPU</td>
<td>67ms</td>
<td>128ms</td>
</tr>
<tr>
<td>MLAA GPU</td>
<td>3.49ms</td>
<td>5.54ms</td>
</tr>
</tbody>
</table>

**TAB.:** Algorithm cost (in ms) using a GeForce 295 GTX and a Core2Duo 2.20GHz

<table>
<thead>
<tr>
<th>Step</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
</tr>
</thead>
<tbody>
<tr>
<td>Percent</td>
<td>16</td>
<td>64</td>
<td>13</td>
<td>7</td>
</tr>
</tbody>
</table>

**TAB.:** Detail of each step cost in the algorithm

**Used with ...**

... many effects like motion blur, bloom, cartoon shading, shadow mapping or parallax mapping.
Results

Result on bloom
Result on bloom
Results

Result on shadow mapping
Result on shadow mapping
Result on per pixel toonshading
Results

Result on per pixel toonshading
Results

Comparison

<table>
<thead>
<tr>
<th>noAA</th>
<th>MLAA</th>
<th>MSAA 8x</th>
</tr>
</thead>
<tbody>
<tr>
<td><img src="image1.jpg" alt="Image" /></td>
<td><img src="image2.jpg" alt="Image" /></td>
<td><img src="image3.jpg" alt="Image" /></td>
</tr>
</tbody>
</table>
Results

Comparison
Conclusion

Overview

- Antialiasing as a post process
- Implemented fully on the GPU, rendering engine friendly
- A new line length computation algorithm
- Around 4ms overhead with a high performance graphic card

Discussion & Future works

- No subpixel antialiasing available
- Packing and profiling
- Other topological configuration than the L?
Conclusion

Thanks
Duran Duboi R&D Team, Baptiste Malaga, Franck Letellier, Vincent Nozick, Nadine Domanget

Code available
in http://igm.univ-mlv.fr/~biri/mlaa-gpu/

Questions?
biri@univ-mlv.fr, aherubel@quintaindustries.com, sdeverly@quintaindustries.com