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

SIGGRAPH2010

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

SIGGRAPH2010

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

SIGGRAPH2010

Geometrical antialiasing

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

Limitations

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

SIGGRAPH2010

Geometrical antialiasing

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

© Martin Korndörfer, 2005

Limitations

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

SIGGRAPH2010

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

SIGGRAPH2010

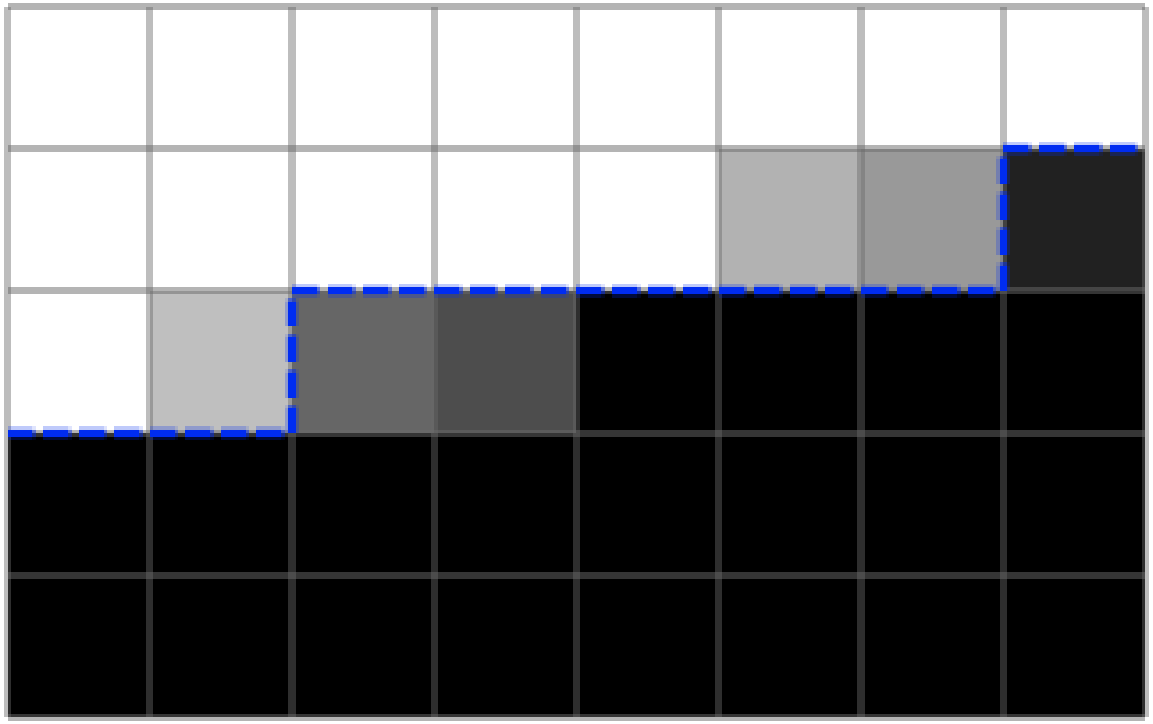
Antialiasing as a post process

From that ...

SIGGRAPH2010

Antialiasing as a post process

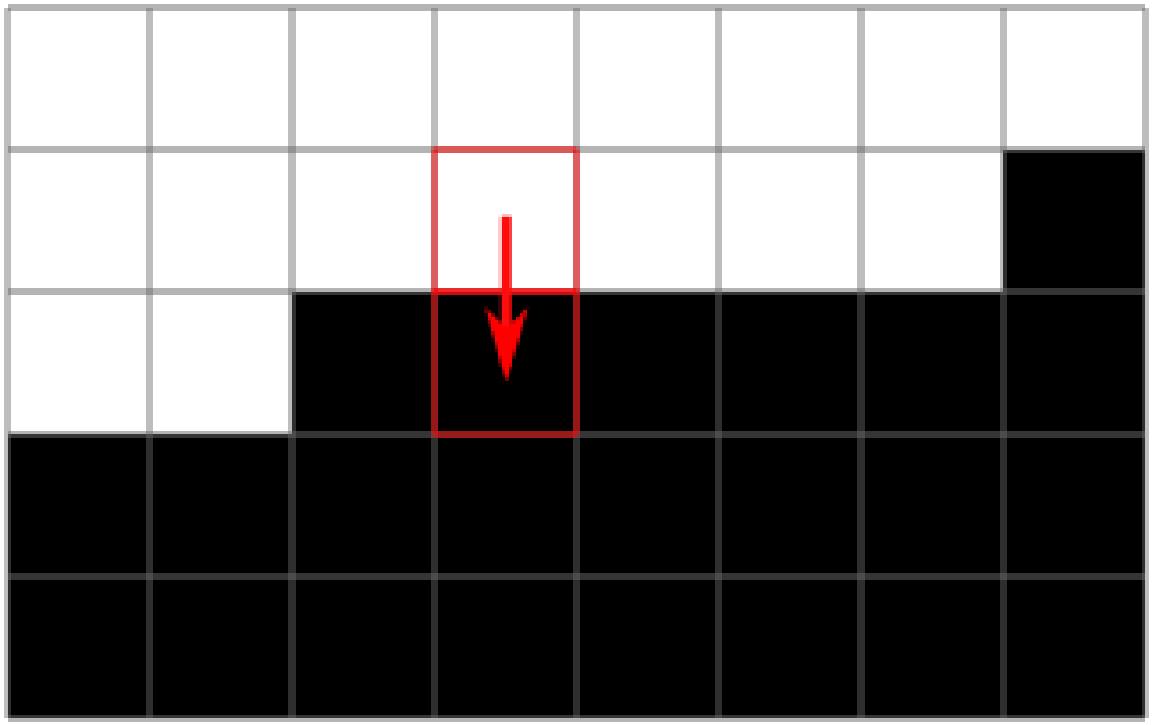
... we want that



SIGGRAPH2010

Antialiasing as a post process

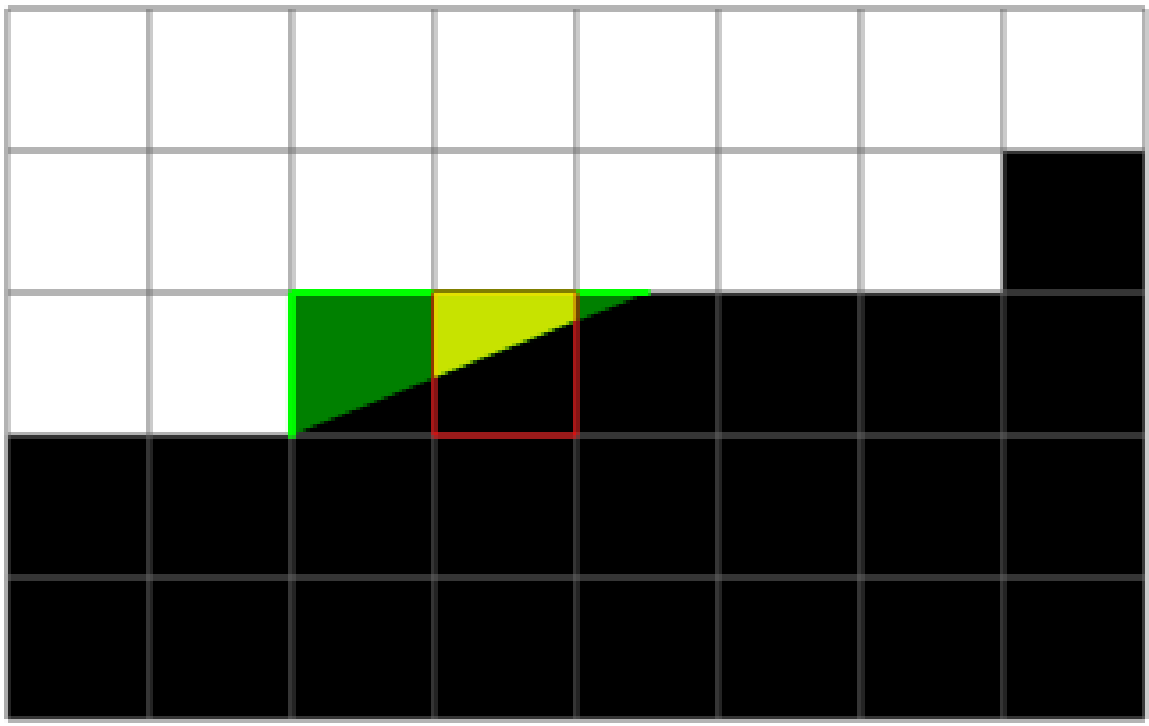
Border pixels must blend with their neighbor ...



SIGGRAPH2010

Antialiasing as a post process

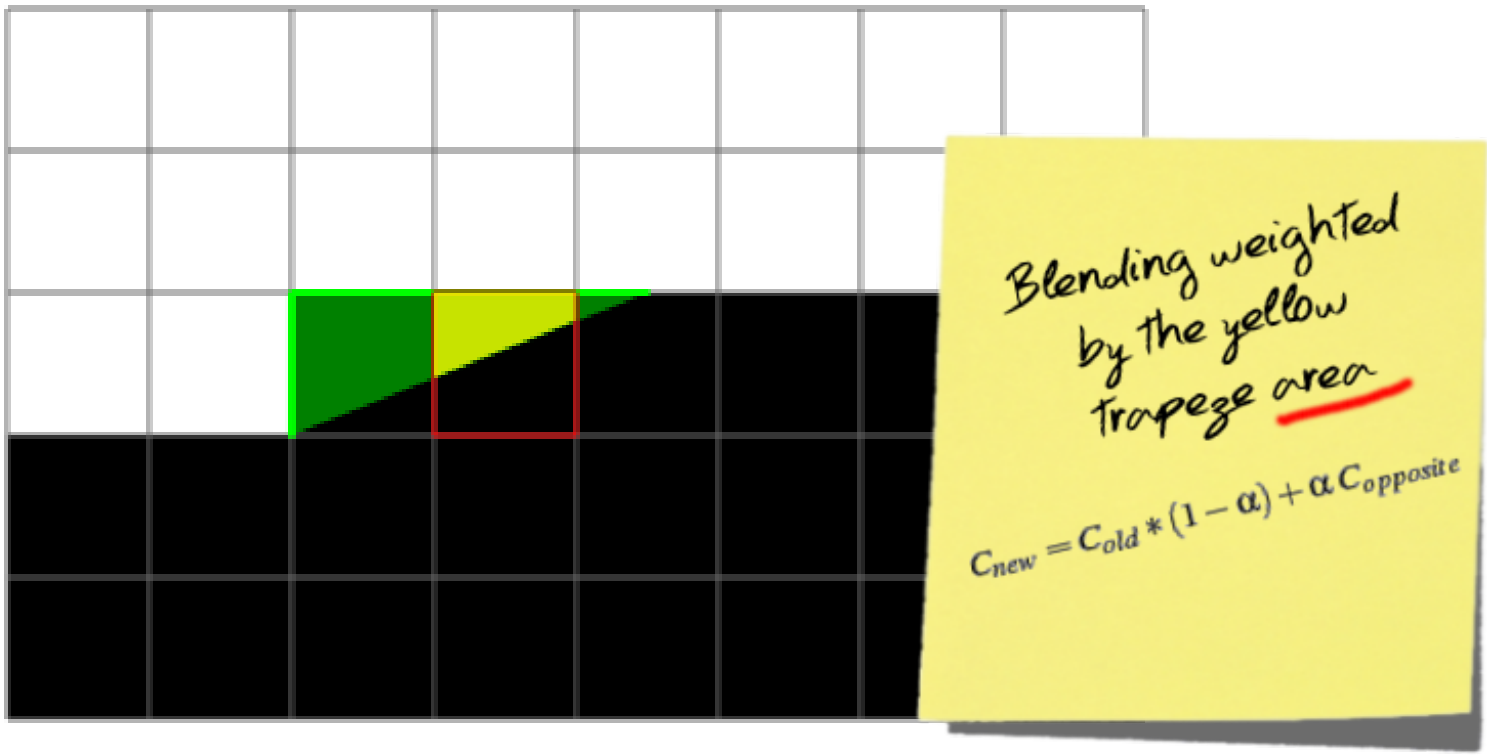
... along ‘L’ shapes



SIGGRAPH2010

Antialiasing as a post process

... along ‘L’ shapes



SIGGRAPH2010

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

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

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

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

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

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

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

SIGGRAPH2010

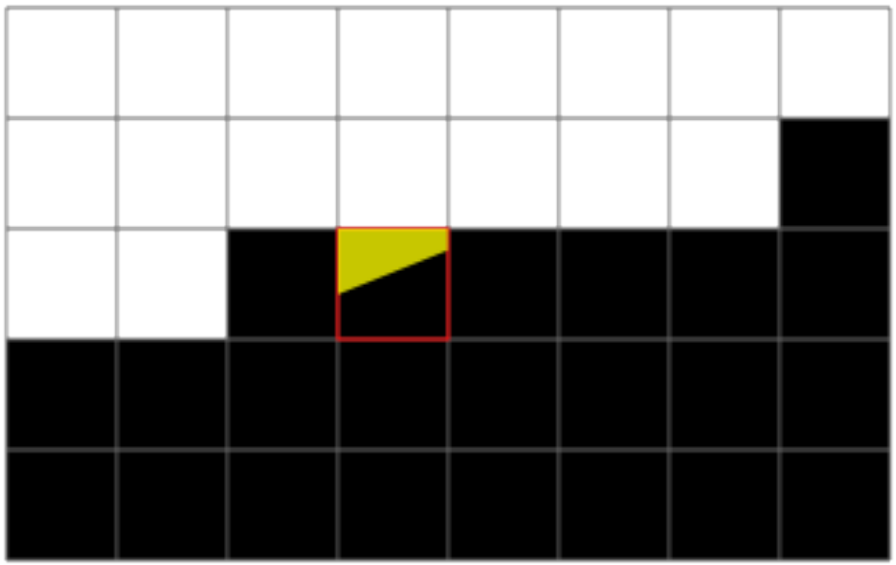
GPU Implementation

Implementation on GPU issues

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

SIGGRAPH2010

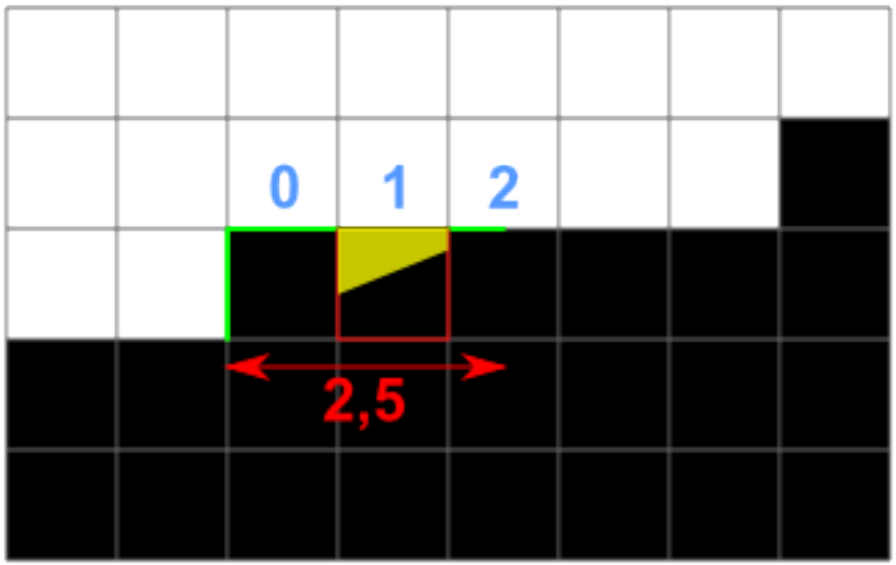
GPU Implementation



To blend the red pixel, we need to compute the yellow area

SIGGRAPH2010

GPU Implementation

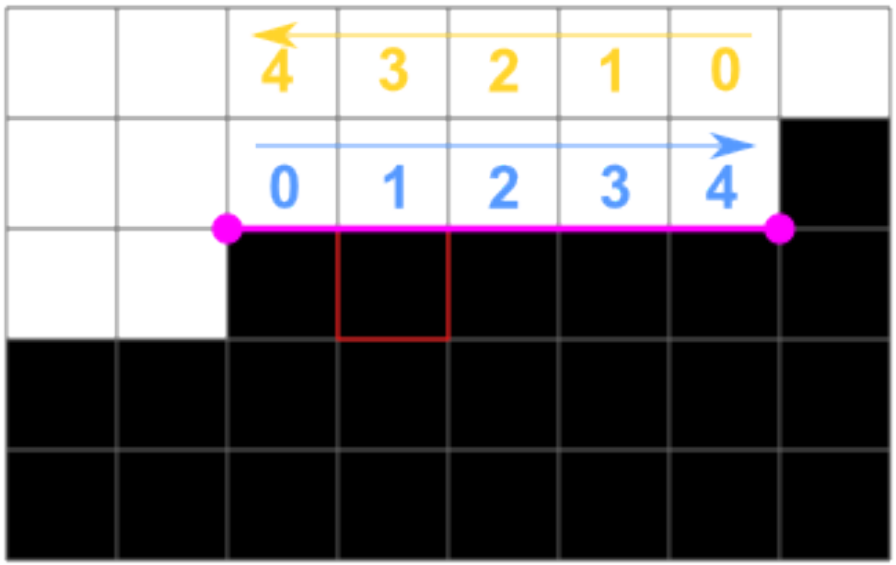


This can be obtained using the pixel's relative position p related to the L shape and its length $L/2$

$$\alpha = \left(1 + \frac{\frac{L}{2} - p - 1}{\frac{L}{2}}\right) / 4.0$$

SIGGRAPH2010

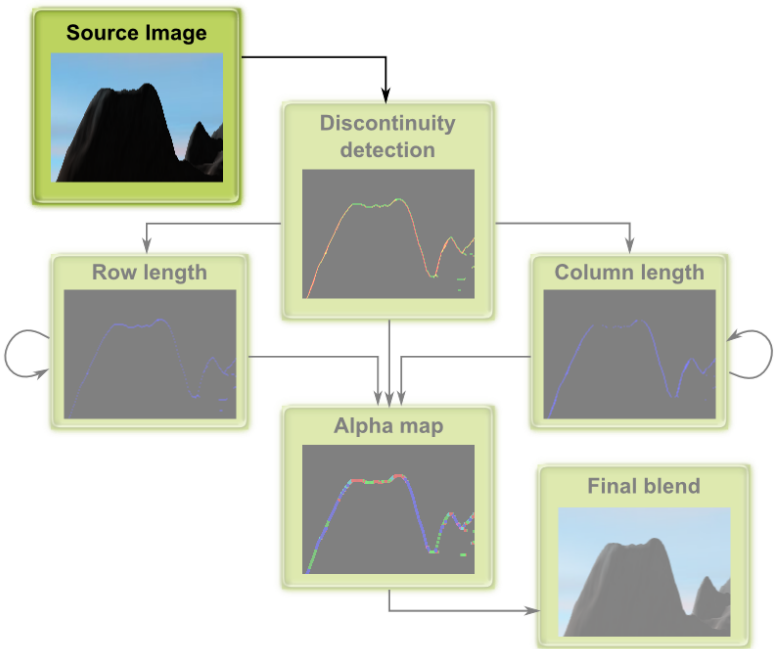
GPU Implementation



The red pixel position is the minimum of the distance to the two discontinuity line extremities

SIGGRAPH2010

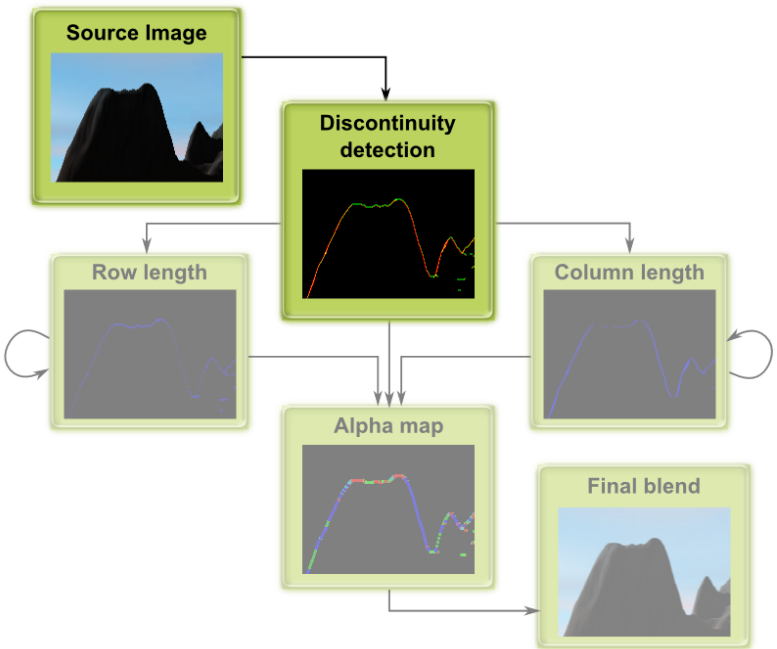
Algorithm overview



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

SIGGRAPH2010

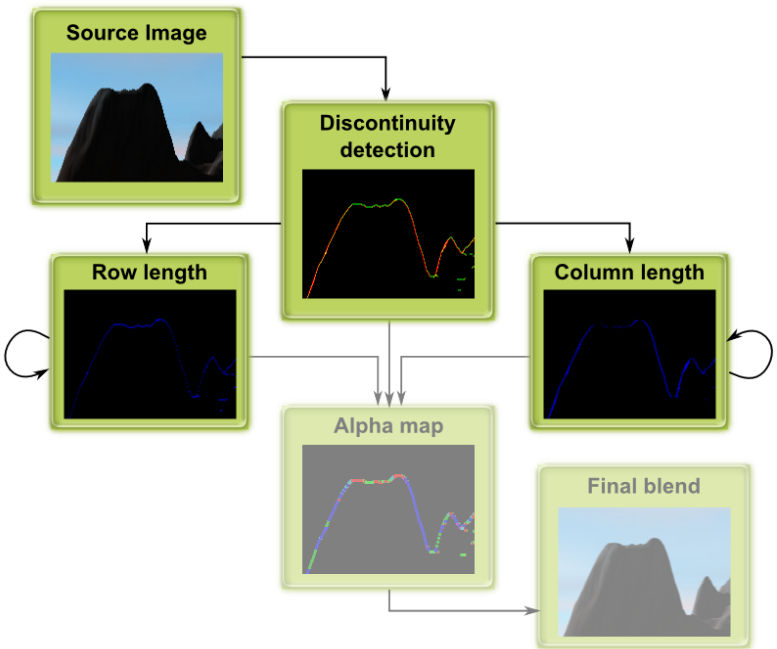
Algorithm overview



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

SIGGRAPH2010

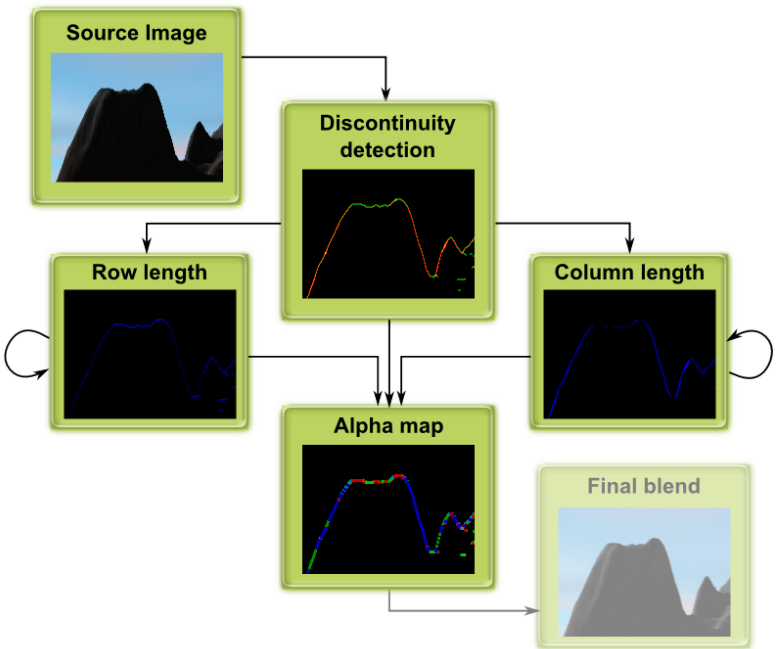
Algorithm overview



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

SIGGRAPH2010

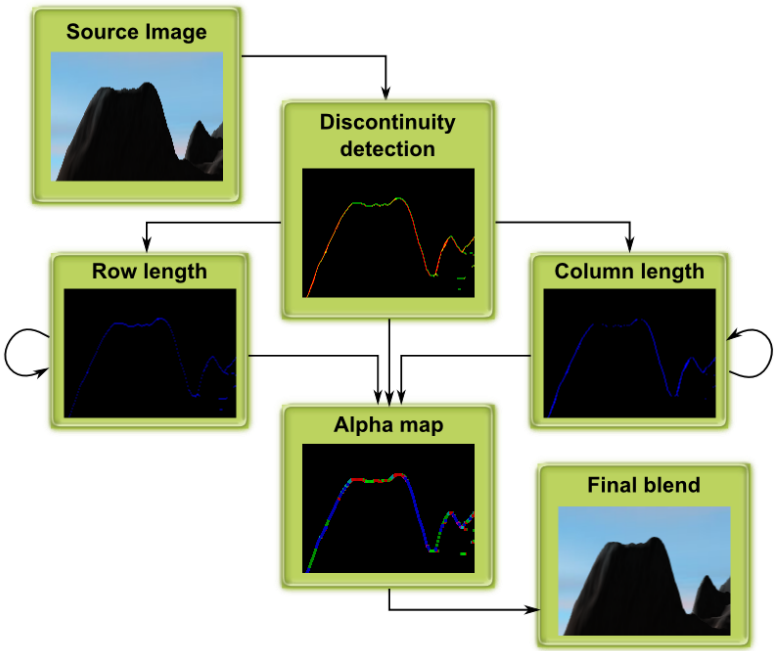
Algorithm overview



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

SIGGRAPH2010

Algorithm overview



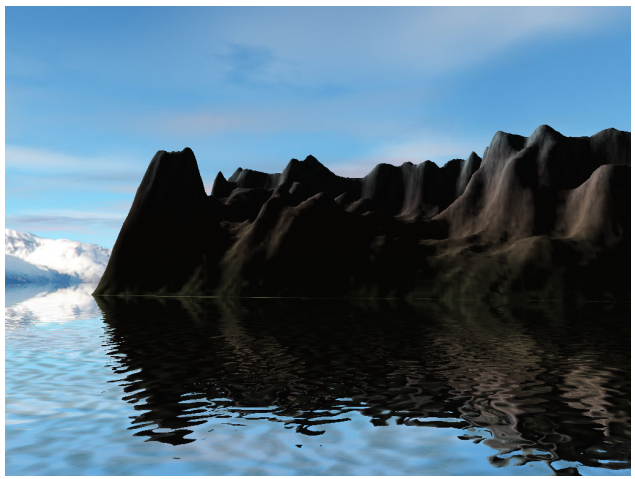
1. Detect discontinuity lines
2. Compute the positions
   - On rows and columns
   - Related to the two extremities
   - Using the “recursive doubling” algorithm
3. Compute the trapeze area for each pixel
4. Final blending

SIGGRAPH2010

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

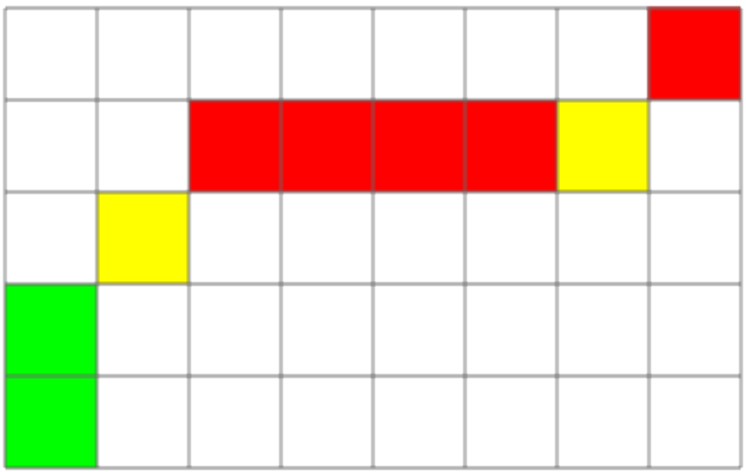
channel G : discontinuity in pixel right

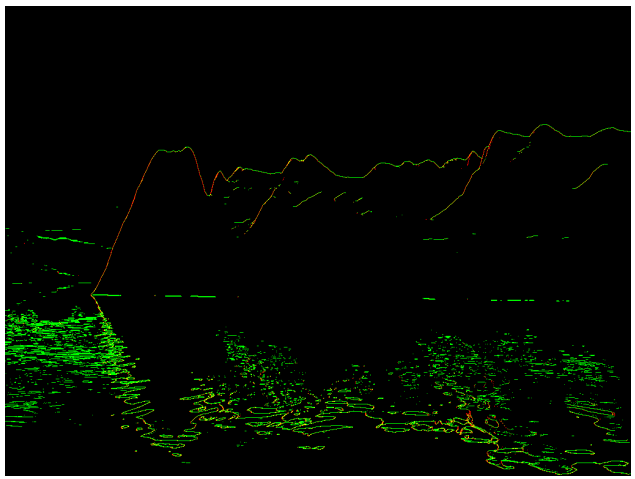
SIGGRAPH2010

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

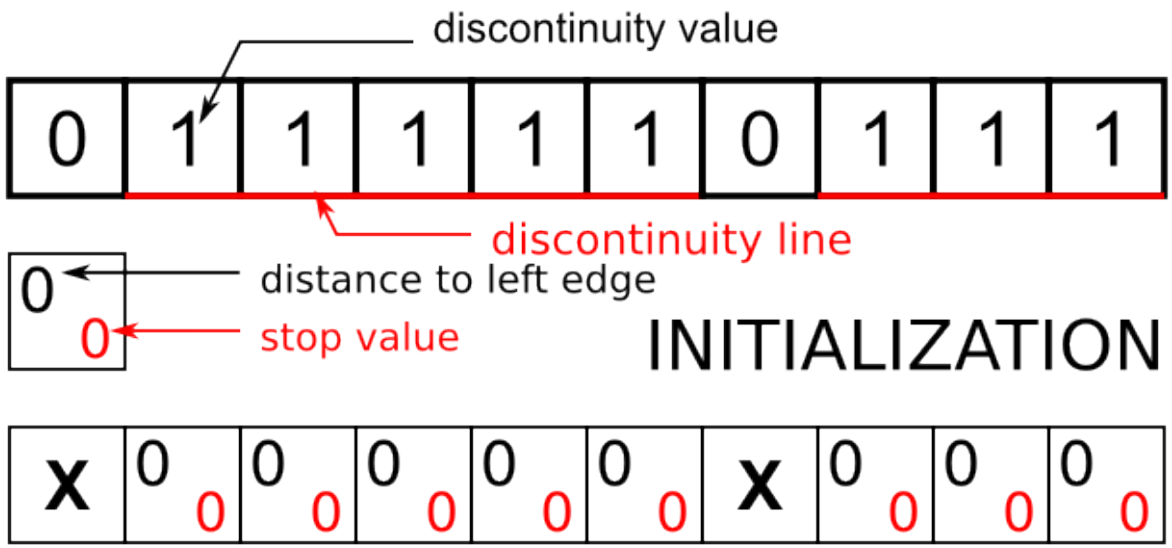
channel G : discontinuity in pixel right

SIGGRAPH2010

Line length computation

2 - Computation of line length and relative position

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

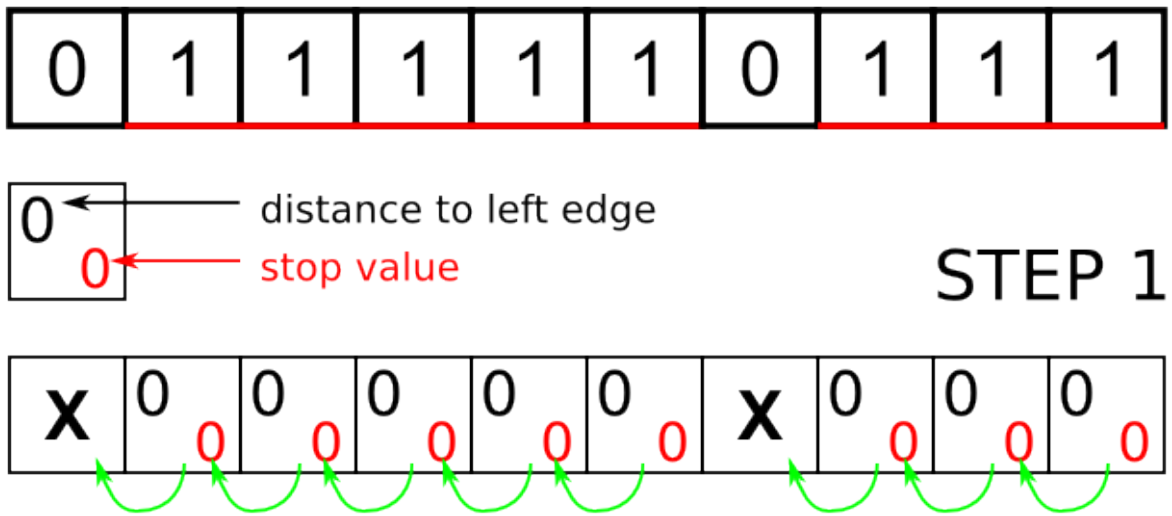


SIGGRAPH2010

Line length computation

2 - Computation of line length and relative position

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

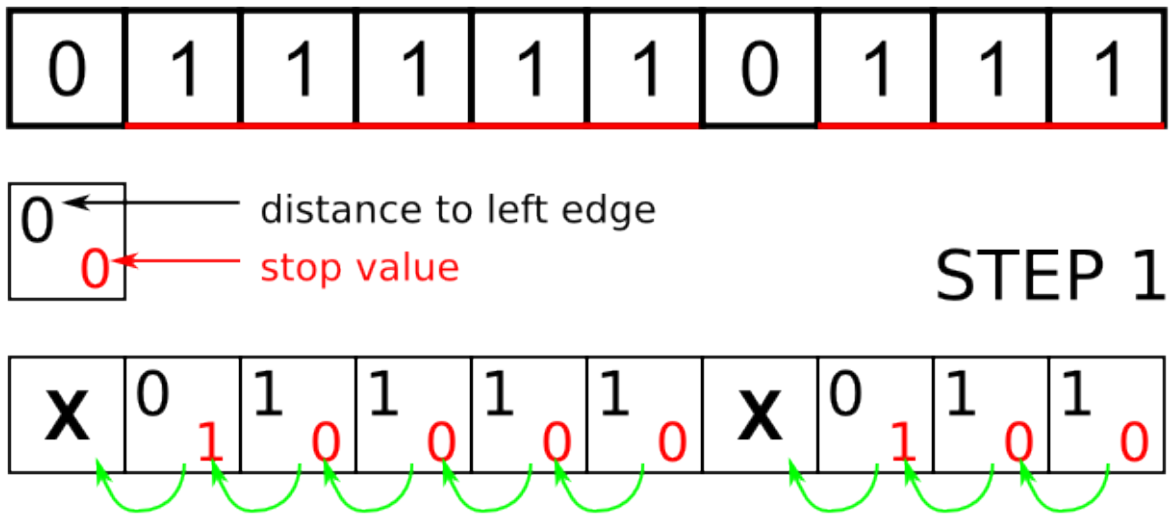


SIGGRAPH2010

Line length computation

2 - Computation of line length and relative position

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

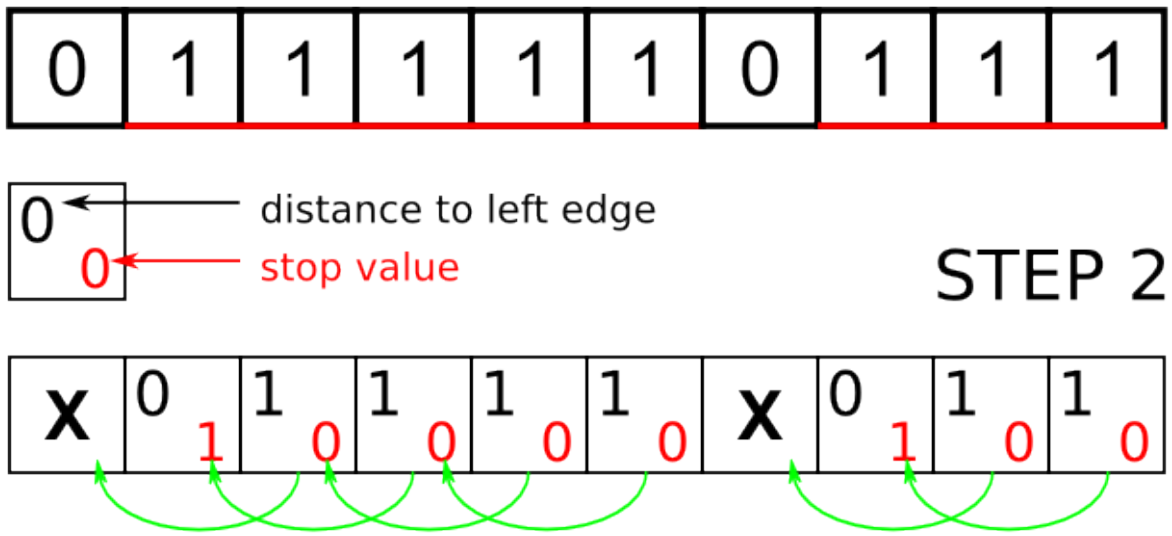


SIGGRAPH2010

Line length computation

2 - Computation of line length and relative position

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

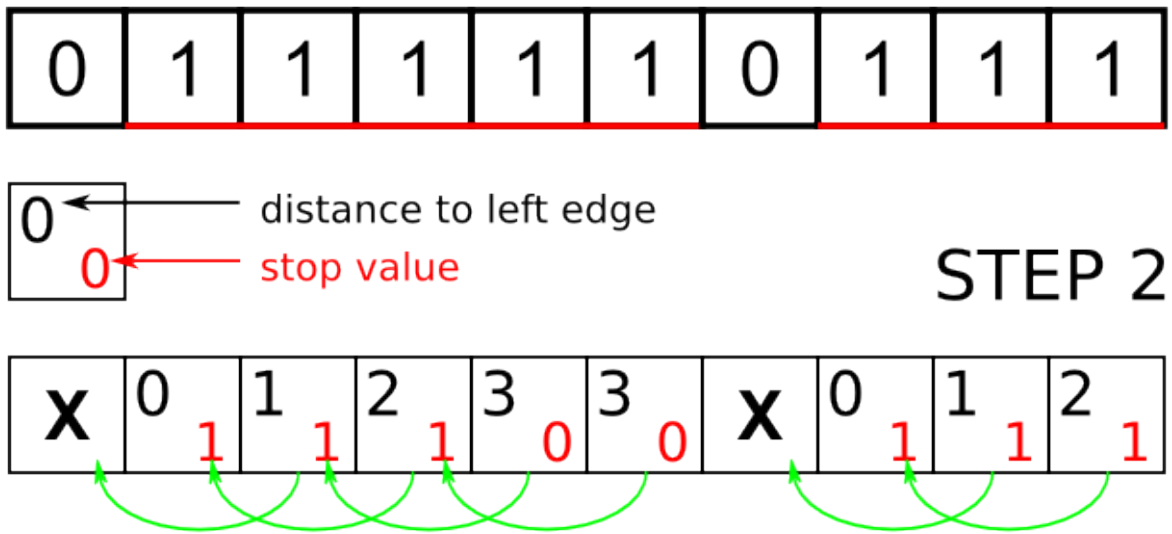


SIGGRAPH2010

Line length computation

2 - Computation of line length and relative position

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

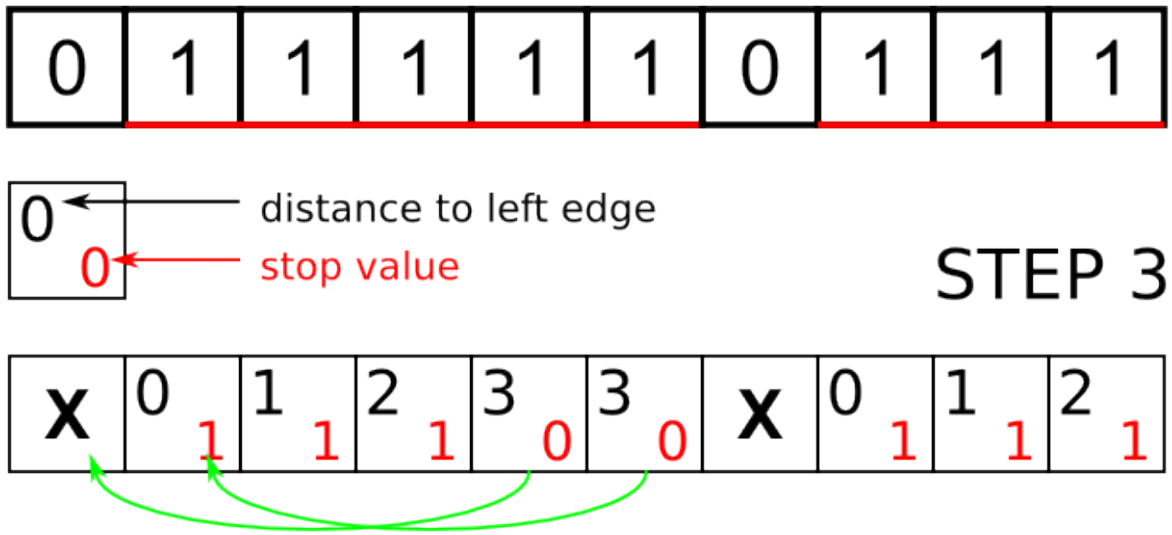


SIGGRAPH2010

Line length computation

2 - Computation of line length and relative position

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

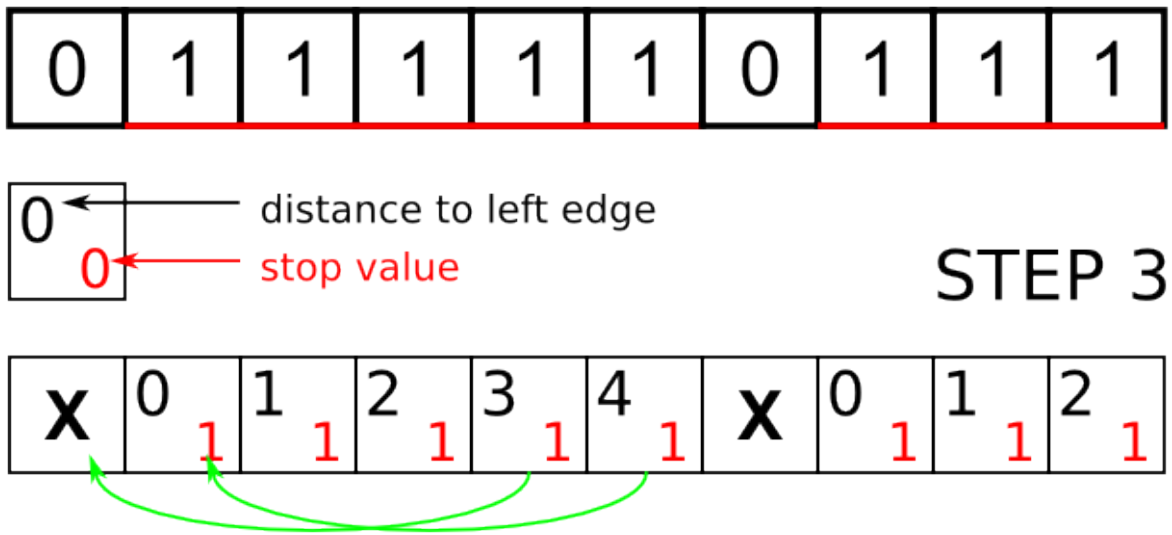


SIGGRAPH2010

Line length computation

2 - Computation of line length and relative position

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

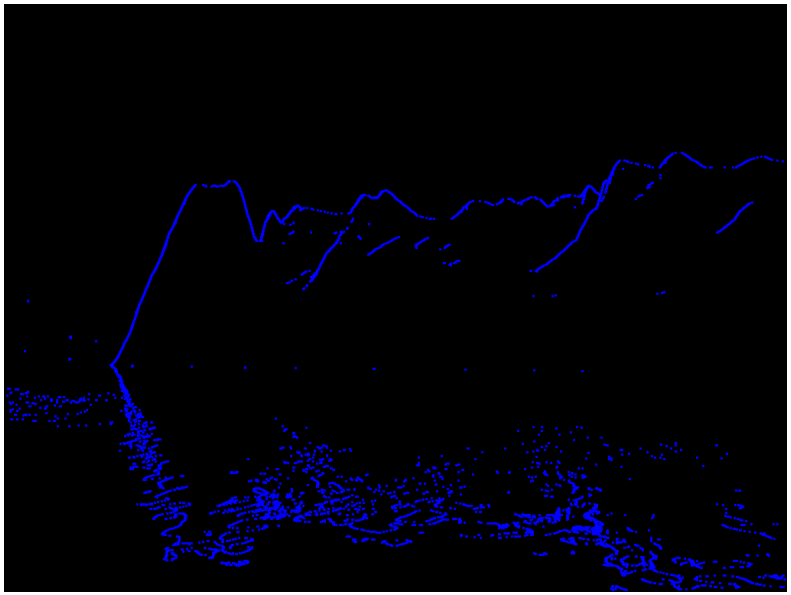


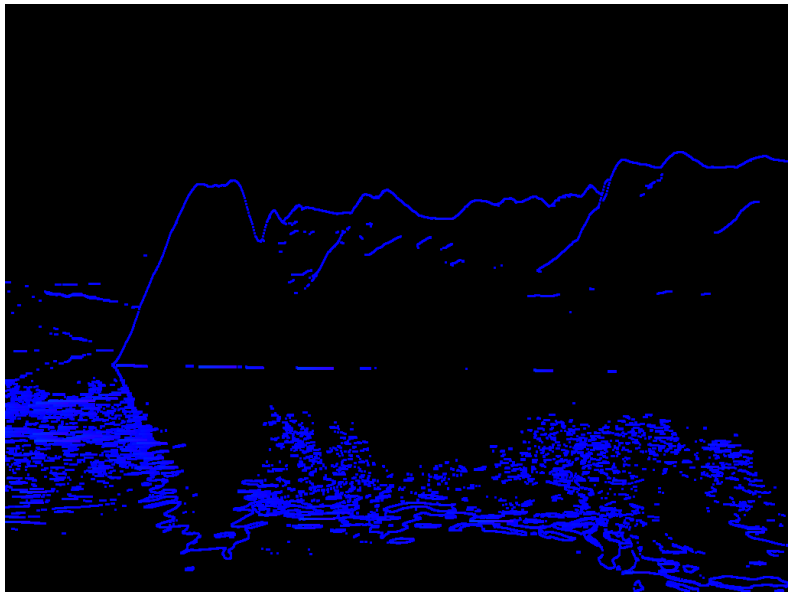
SIGGRAPH2010

Line length computation

2 - Computation of line length and relative position

Number of iteration for row = $log_2(width)$. Number of iteration for columns = log_2(height) Results are stored in textures (one for the rows, one for the columns)





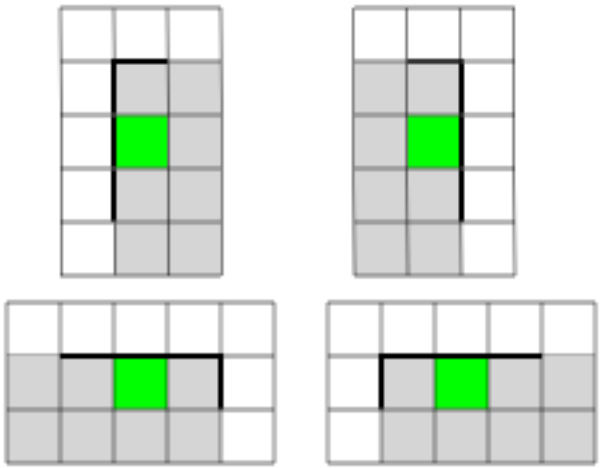
SIGGRAPH2010

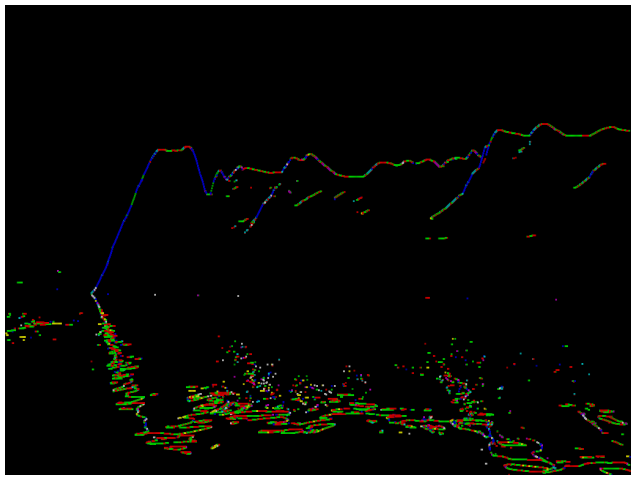
Blending map

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

8 cases



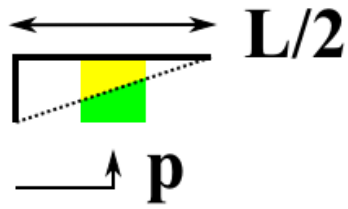


SIGGRAPH2010

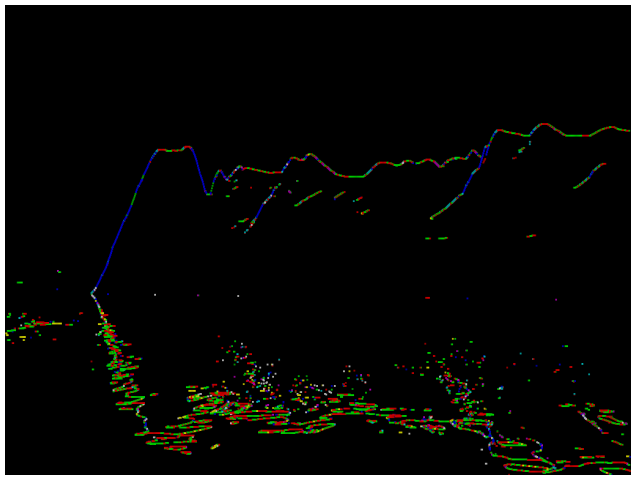
Blending map

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



$$\alpha = \left(1 + \frac{\frac{L}{2} - p - 1}{\frac{L}{2}}\right) / 4.0$$

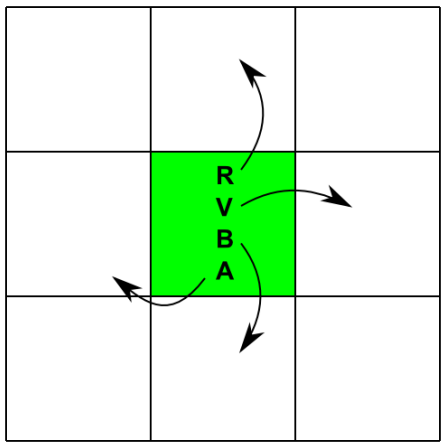


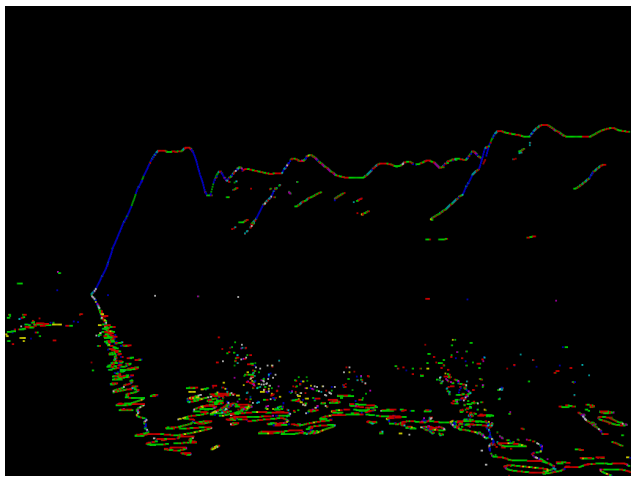
SIGGRAPH2010

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



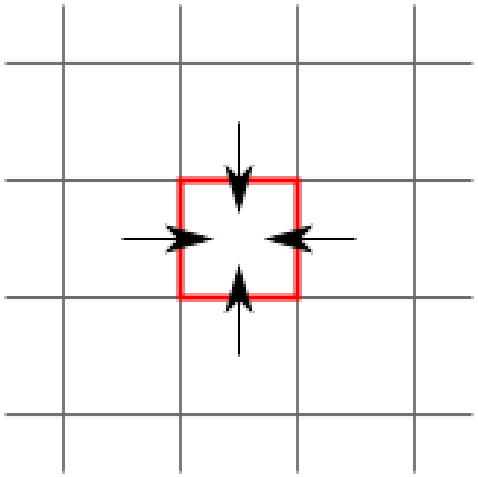


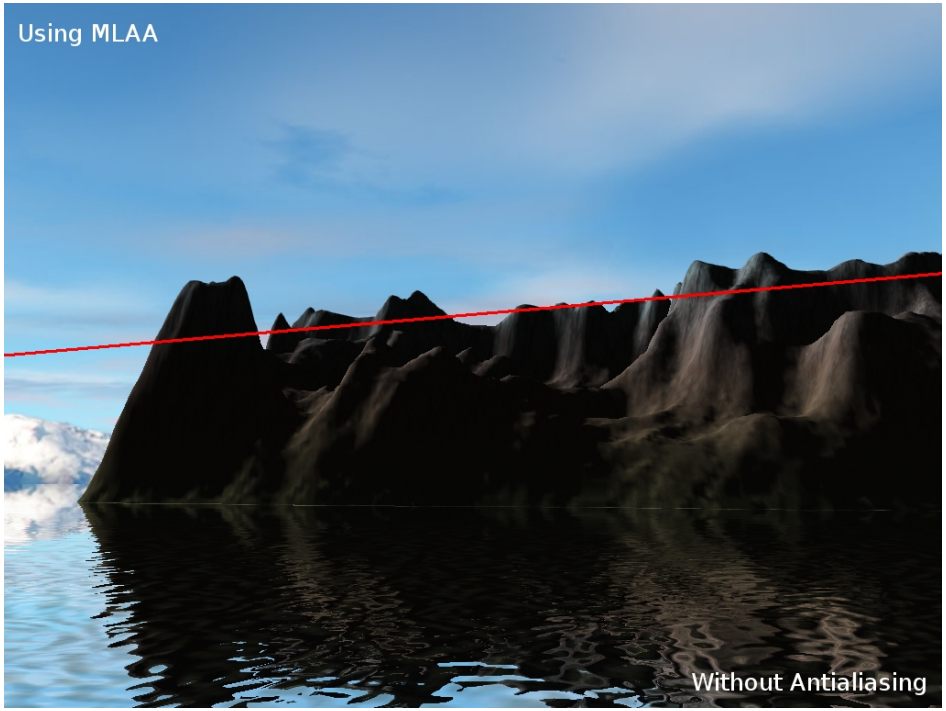
SIGGRAPH2010

Final rendering

4 - Final blending

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





SIGGRAPH2010

Computation times

Image size	1248x1024	1600x1200
MLAA CPU	67ms	128ms
MLAA GPU	3.49ms	5.54ms

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

Step	1	2	3	4
Percent	16	64	13	7

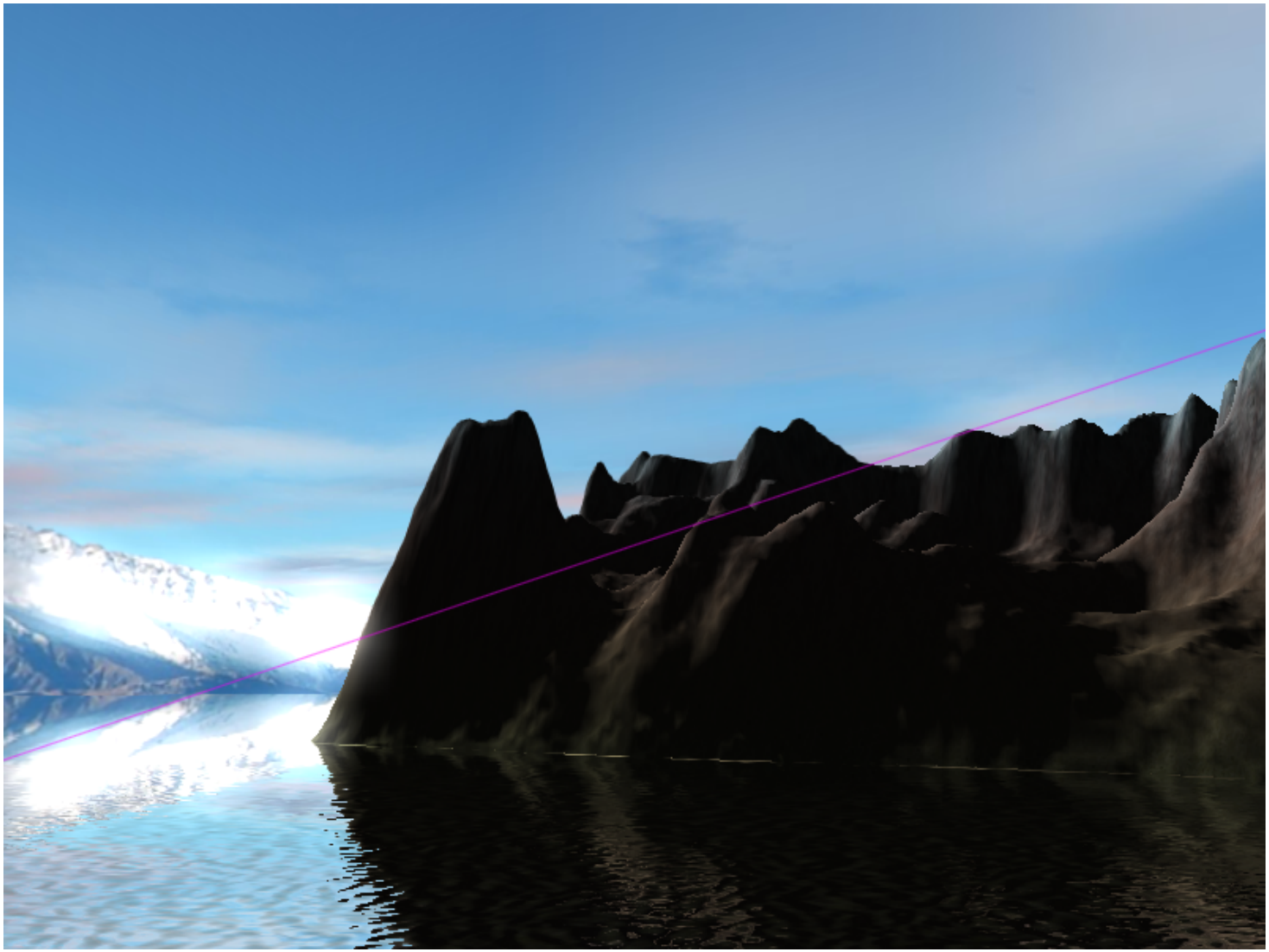
TAB.: Detail of each step cost in the algorithm

Used with ...

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

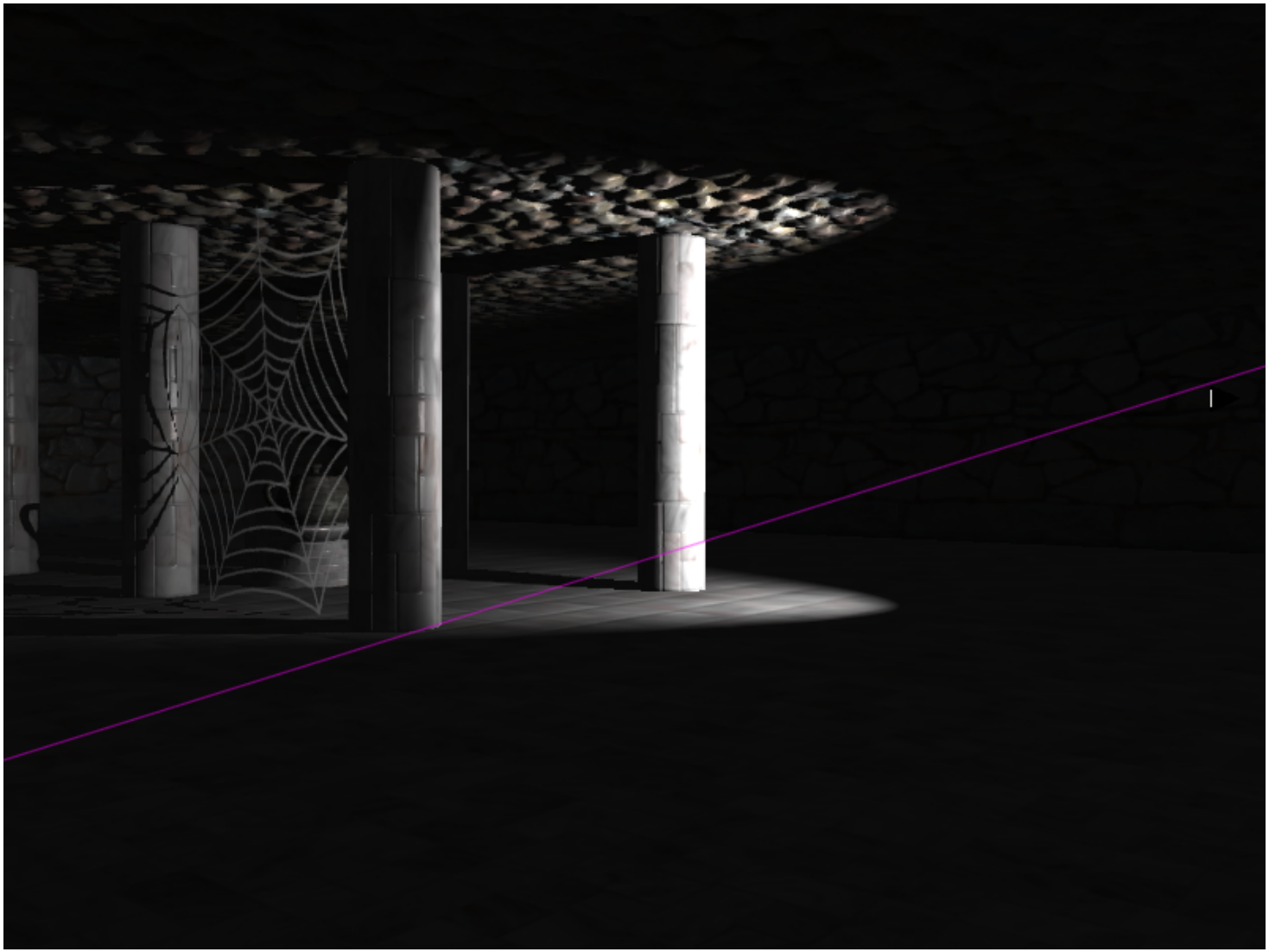
SIGGRAPH2010

Result on bloom



Result on bloom

Result on shadow mapping



Result on shadow mapping

SIGGRAPH2010

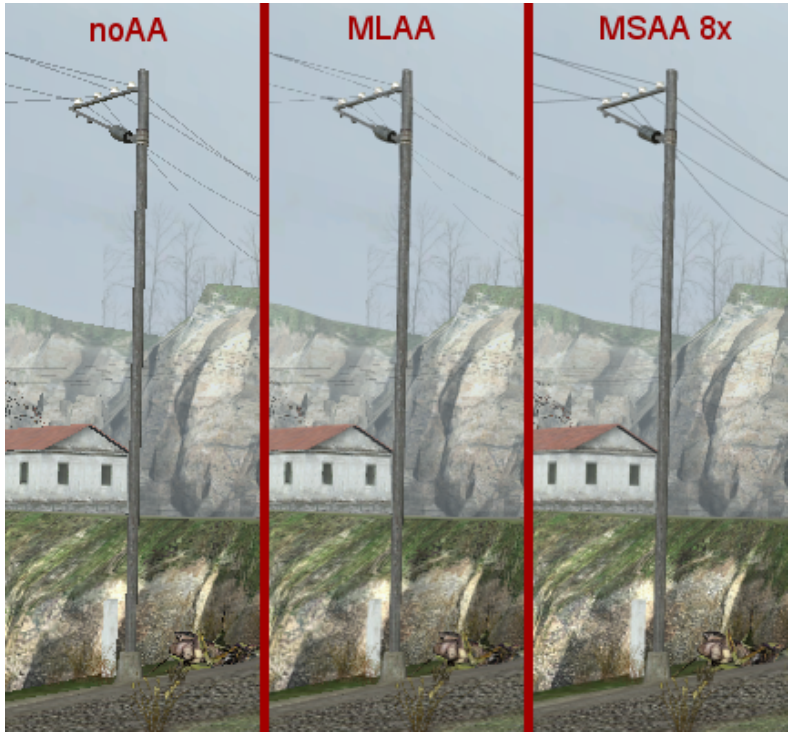
Result on per pixel toonshading

SIGGRAPH2010

Result on per pixel toonshading

SIGGRAPH2010

Comparison



SIGGRAPH2010

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