Advanced Screenspace Antialiasing

by Arne Schober
Yager Development GmbH

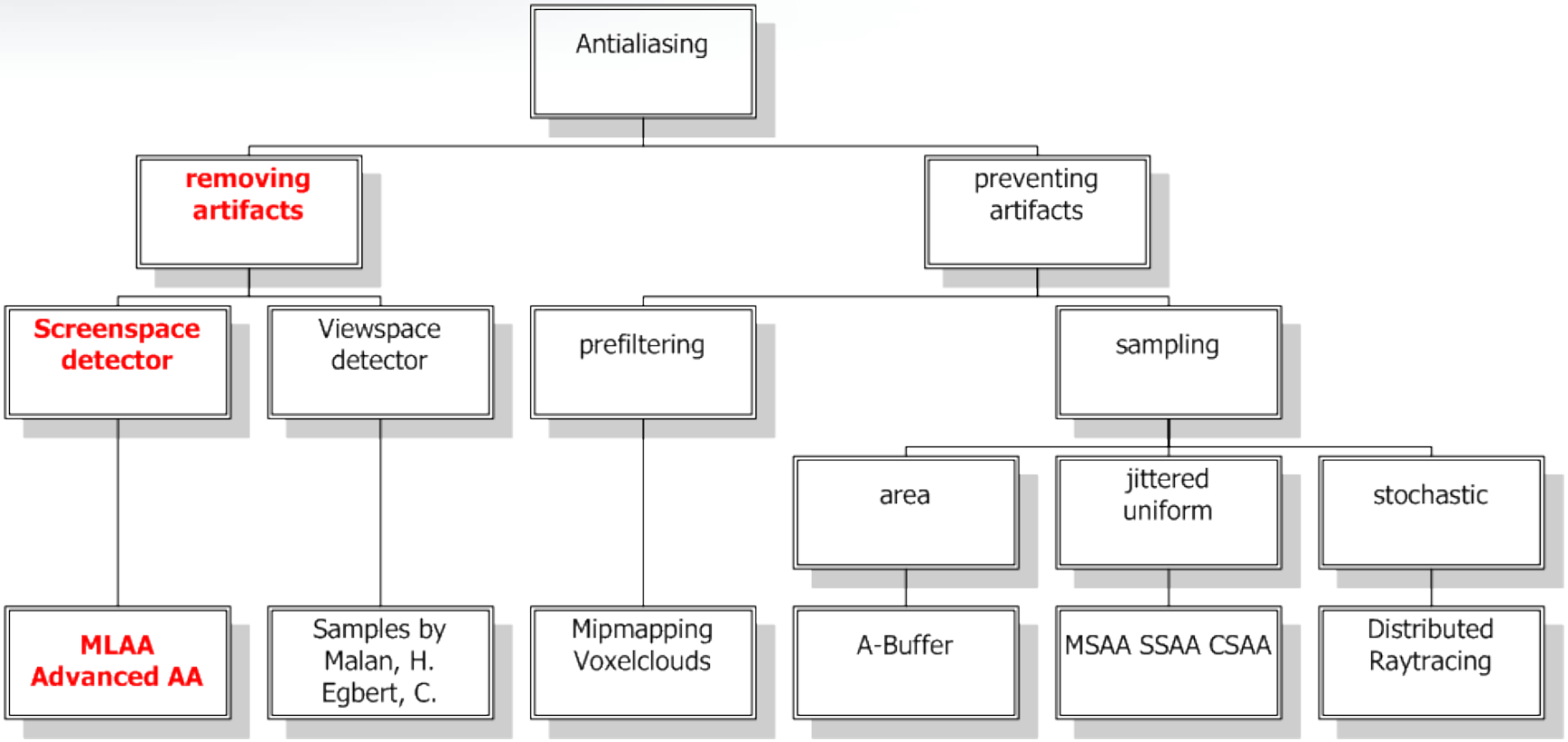
Advanced Screenspace Antialiasing

- Antialiasing Techniques
- What types of artifacts can be removed?
- Morphological Antialiasing
- Advancements in the implementation
  - Discontinuity detector
  - Richards counting optimization
  - Calculating coverage
  - Advanced coverage calculation
- Implementation details
  - Overview
  - Blending optimizations
  - Linear textures on Xbox360
  - Synchronization
  - Counting on the GPU

YAGER

ANTIALIASING TECHNIQUES

Antialiasing Techniques



YAGER

WHAT TYPES OF ARTIFACTS CAN BE REMOVED?

YAGER

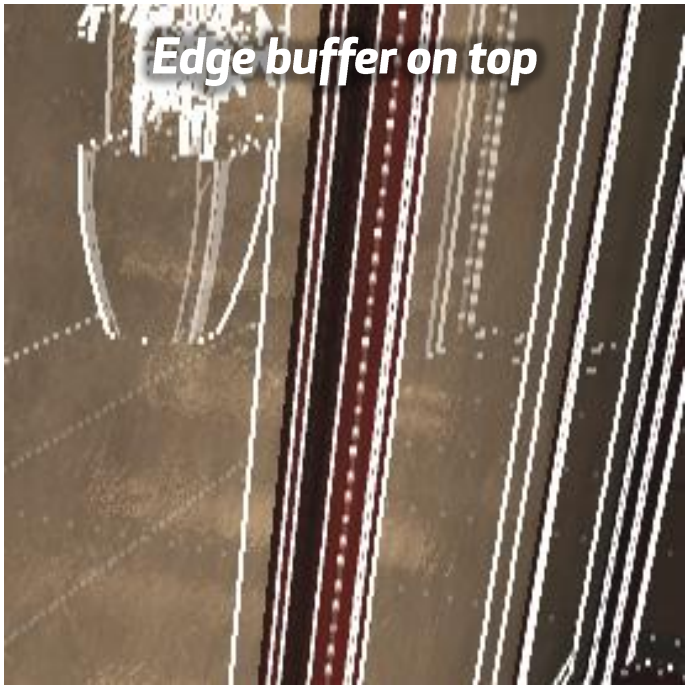
What types of artifacts can be removed?

- Information that has been lost cannot be recovered!
  - under sampling artifacts cannot be removed

YAGER

What types of artifacts can be removed?

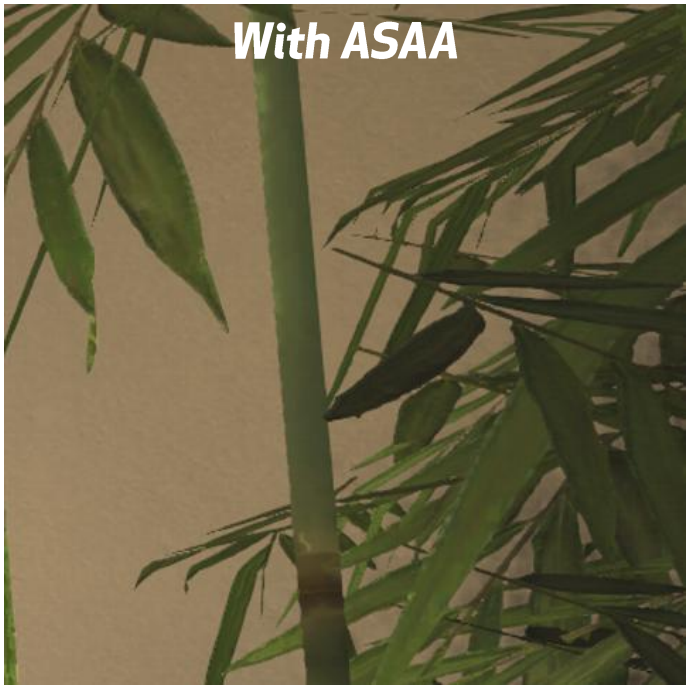
- Transparencies are not handled in our implementation
  - depth peeling or a stencil routed K-buffer might fix this
  - avoid intersection of transparent geometry (window & frame)
  - add an opaque edge where transparencies occur



YAGER

What types of artifacts can be removed?

- Alpha Test geometry as well as geometry edges are handled



YAGER

MORPHOLOGICAL ANTIALIASING

YAGER

Morphological Antialiasing

- Original idea from Reshetov, A. (HPG 2009)
- GPU implementation available from Biri, V. (Siggraph 2010)
  - http://igm.univ-mlv.fr/~biri/mlaa-gpu/
- Uses a color discontinuity detector
  - ☹ Might miss important edges (where the difference in color is small)
  - ☹ Might be overly sensitive (detects texture details)
  - ☺ Will detect transparencies
  - ☺ no need of extra buffers (e.g. normal, texcoord & depth)
- Edge detection convolution kernels

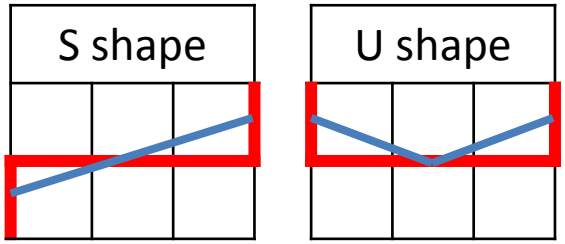
vertical		
0	0	0
0	1	0
0	-1	0

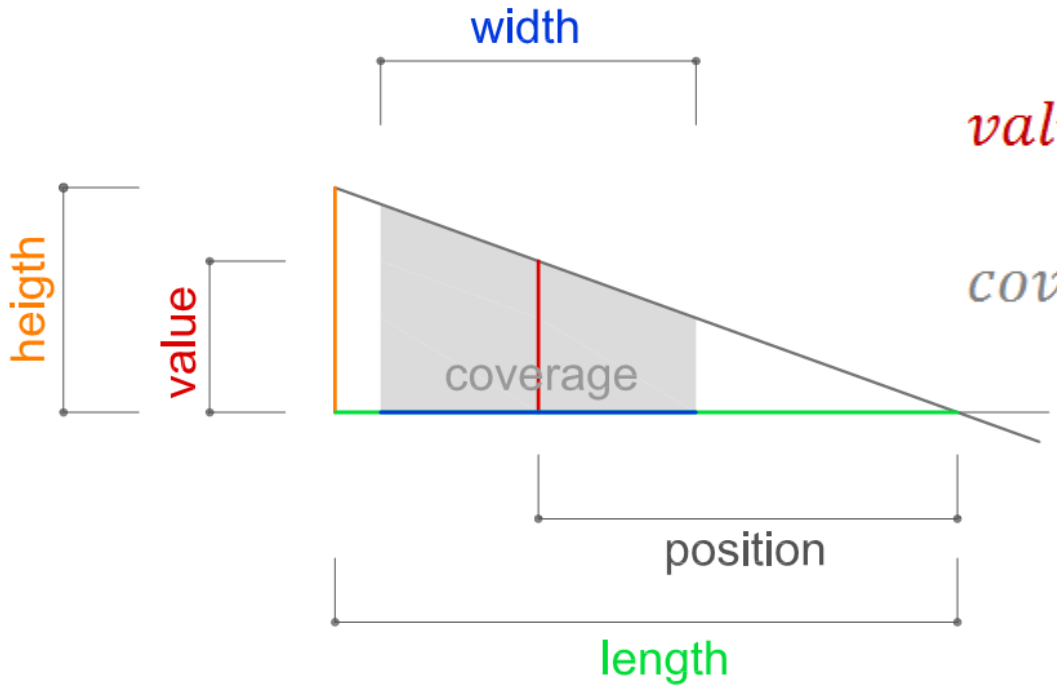
horizontal		
0	0	0
0	1	-1
0	0	0

YAGER

Morphological Antialiasing

- Detect various types of line segments
  - S shapes & U shapes composed of 2 L shapes
- Actual length and pixel position needed
  - Coverage can be determined using the Intercept theorem





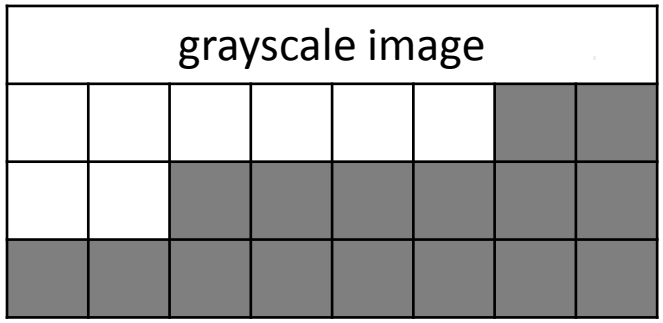
$$value = \frac{heigth * position}{length}$$

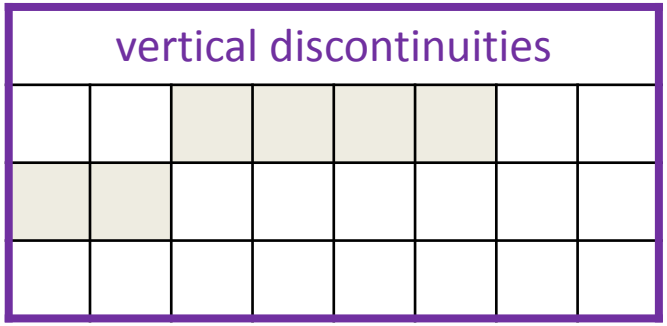
$$coverage = value * width$$

YAGER

Horizontal counting

- 2 additional buffers for the edge detection (only one shown)





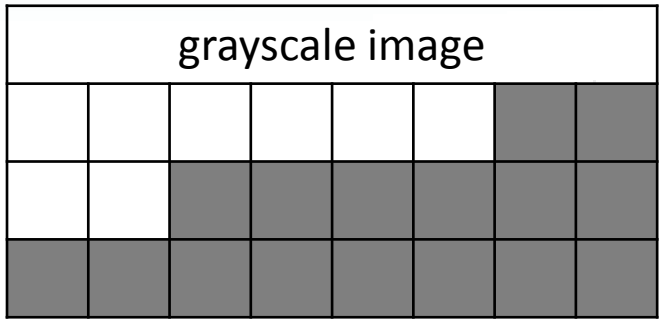
- Counting technique with 4 additional buffers (←, →, ↑, ↓)

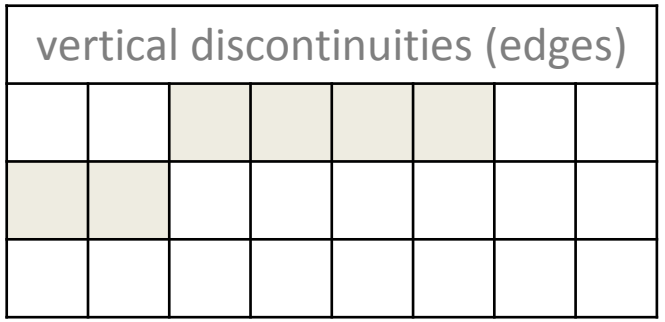
→ counting buffer							
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0

← counting buffer							
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0

YAGER

Horizontal counting





- Always reject all pixels not in the vertical discontinuity buffer

→ counting buffer & edges							
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0

← counting buffer & edges							
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0

YAGER

Horizontal counting

→ counting buffer & edges							
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0

← counting buffer & edges							
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0

- start with level = 1: let x = lookup level pixel to the left or right and only set value to (level + x) when ceil(level /2) pixels to the left or right has not been rejected else reject

→ counting buffer & edges							
0	0	0	1	1	1	0	0
0	1	0	0	0	0	0	0
0	0	0	0	0	0	0	0

← counting buffer & edges							
0	0	1	1	1	0	0	0
1	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0

Horizontal counting

- continue with level *= 2 (level is now 2):

→ counting buffer & edges							
0	0	0	1	2	3	0	0
0	1	0	0	0	0	0	0
0	0	0	0	0	0	0	0

← counting buffer & edges							
0	0	3	2	1	0	0	0
1	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0

- do this repeatedly until no further pixels are drawn or maximum precision is reached
- the maximum number of needed passes will be:

$$maxPasses = \log_2\left(2^{numBits}\right)$$

- which is in fact the number of bits per counting value used

Morphological Antialiasing

→ counting- & horizontal discontinuity buffer							
0	0	0	1	2	3	0	0
0	1	0	0	0	0	0	0
0	0	0	0	0	0	0	0

← counting- & horizontal discontinuity buffer							
0	0	3	2	1	0	0	0
1	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0

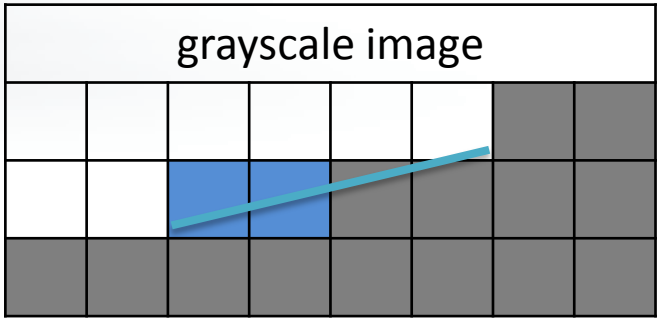
- The length of a L-shape and position on that shape can be determined:

$$position = \frac{|left - right|}{2} \qquad length = \frac{left + right + 1}{2}$$

- The blending value is rejected when in the left case left + 1 pixel to the left or in the right case right pixel to the right is not marked in the **horizontal** discontinuity buffer

YAGER

Special Case: downside of S-Shapes



- for blending of the blue pixels there are no immediate values therefore we have to look one line above

→ counting- & horizontal discontinuity buffer

0	0	0	1	2	3	0	0
0	1	0	0	0	0	0	0
0	0	0	0	0	0	0	0

← counting- & horizontal discontinuity buffer

0	0	3	2	1	0	0	0
1	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0

- If we have more than one blending value → **use the maximum**

YAGER

Morphological Antialiasing

- ☺ The Algorithm handles U shapes always in an unambiguous way
- ☹ The implementation of Biri, V. uses a lot of memory
  - 2 ARGB counting buffers
  - 1 ARGB blending buffer
  - 1 RG discontinuity buffer
  - 1 512x512px float Lookup Table for coverage calculations
- ☹ With clever packaging and buffer sharing the overall memory consumption could be reduced significantly (at additional computational costs)
- ☹ The algorithm uses a lot of conditional branching in the pixelshader
- ☹ The algorithm uses a lot of passes with no stencil early outs
- ☹ Therefore it is unsuitable for the current console generation

YAGER

ADVANCEMENTS IN THE IMPLEMENTATION

Discontinuity detector

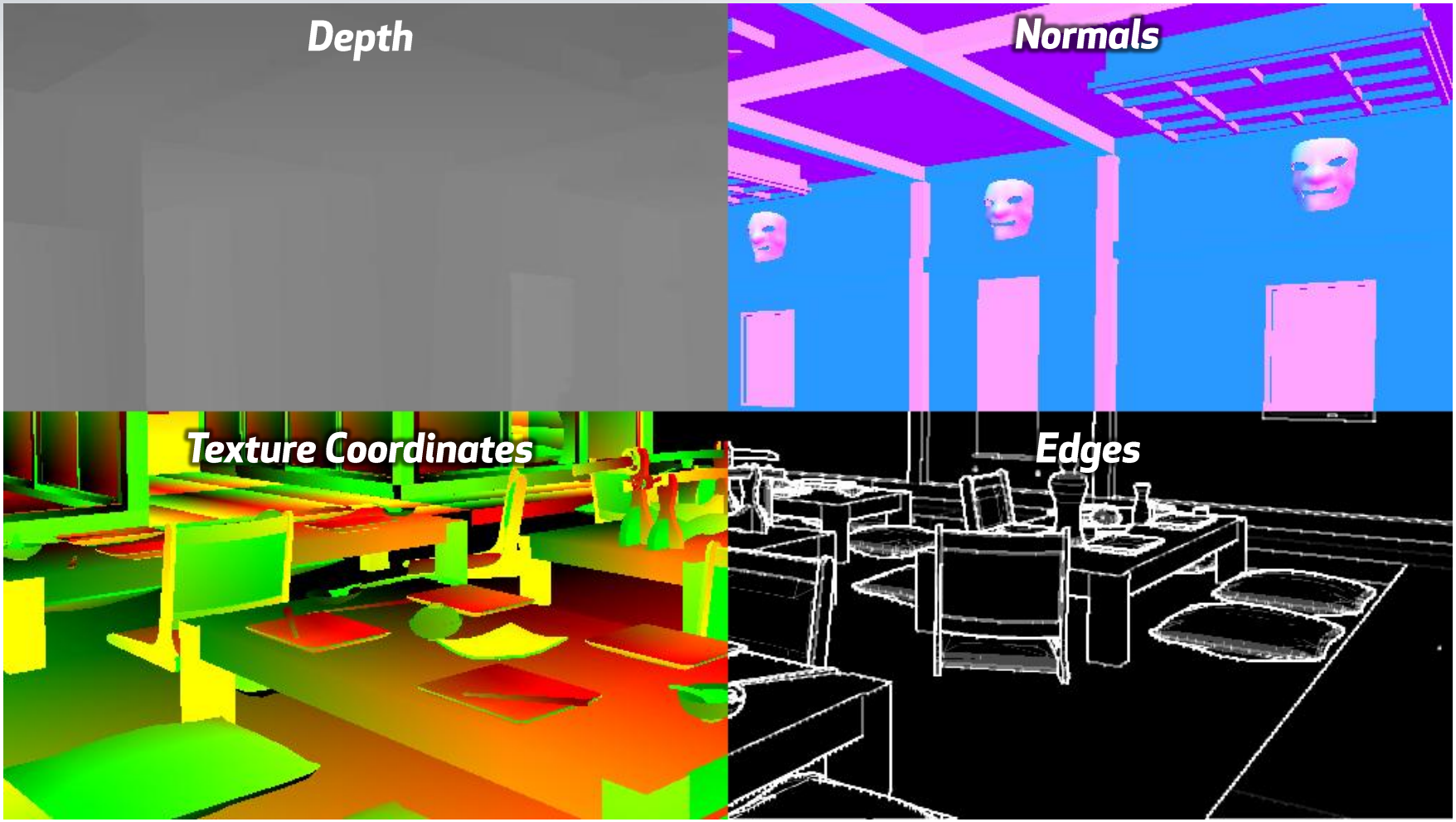
- Our implementation did not use color for discontinuity detection because we thought depth would be more precise
- Depth revealed not all edges and fine grained detail was lost
- Added normal and texture coordinates as additional edge detection hints
  - These two additional buffers could be safely shared with the scene color for the time they are being used
- A laplacian convolution kernel was used for discontinuity detection → detects discontinuities on both sides of an edge

vertical		
0	-1	0
0	2	0
0	-1	0

horizontal		
0	0	0
-1	2	-1
0	0	0

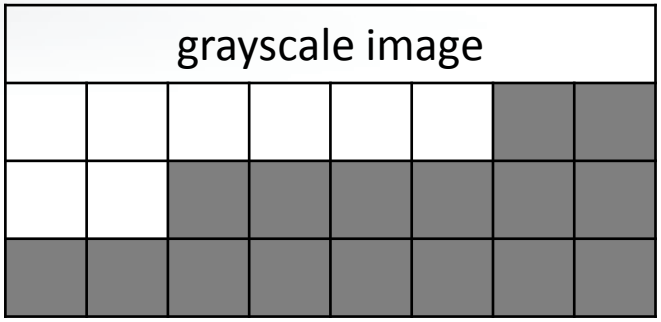
YAGER

Edge detection buffers

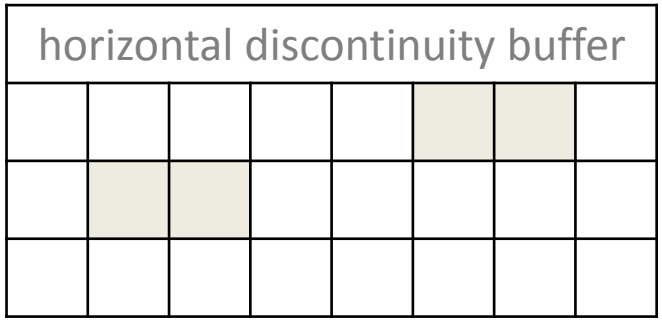


YAGER

Discontinuity detector



- The resulting discontinuity buffers using a laplace convolution



YAGER

Richards counting optimization

- Original 4 counting buffers (←, →, ↑, ↓)

↑ counting- & vertical discontinuity buffer							
1	1	0	0	0	0	3	3
0	0	0	0	0	0	2	2
0	0	1	1	1	1	1	1

→ counting- & horizontal discontinuity buffer							
1	2	3	4	5	0	0	1
1	0	0	1	2	3	4	5
1	2	3	4	5	6	7	8

↓ counting- & vertical discontinuity buffer							
1	1	0	0	0	0	1	1
0	0	0	0	0	0	2	2
0	0	1	1	1	1	3	3

← counting- & horizontal discontinuity buffer							
5	4	3	2	1	0	0	1
1	0	0	5	4	3	2	1
8	7	6	5	4	3	2	1

Richards counting optimization

- Combined results of (↓↑) and (→←) counting using the minimum operator

↓↑ counting- & vertical discontinuity buffer							
1	1	0	0	0	0	1	1
0	0	0	0	0	0	2	2
0	0	1	1	1	1	1	1

→← counting- & horizontal discontinuity buffer							
1	2	3	2	1	0	0	1
1	0	0	1	2	3	2	1
1	2	3	4	4	3	2	1

YAGER

Calculating coverage

horizontal edges		
0	nxt	0
0	this	0
0	nxt	0

vertical edges		
0	0	0
nxt	this	nxt
0	0	0

- Intercept theorem for calculating the coverage between 2 pixel:

$$length = \frac{this + nxt + 1}{2} \qquad position = \frac{nxt - this}{2}$$

$$coverage = saturate\left(\frac{position}{2length}\right)$$

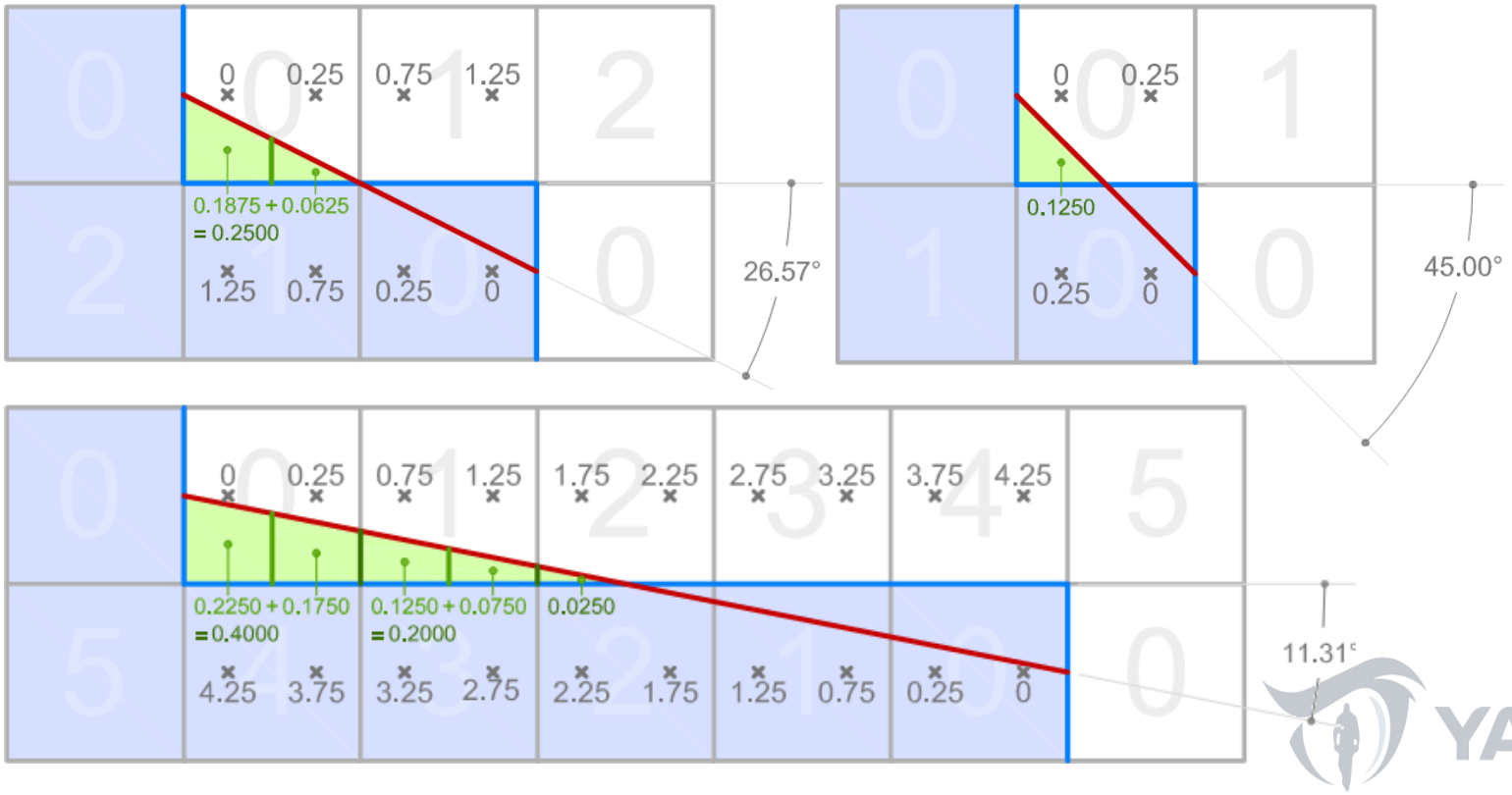
- Calculating coverage this way on 45° angles will result in no coverage at all ☹

YAGER

Advanced coverage calculation

- Use linear interpolation and calculate coverage for half-pixel

$$coverage = saturate\left(\frac{2position + frac(2length)}{4round(2length)}\right)$$



YAGER

Advanced Screenspace Antialiasing

😐 Ambiguous handling of U shapes

$$\lim_{nxt \to \infty} saturate\left(\frac{nxt - this}{2(this + nxt + 1)}\right) = 0.5$$

→← counting & shape					
∞	∞	∞	∞	∞	∞
0	0	1	1	0	0
1	2	3	3	2	1

☺ The memory usage is moderate

- 1 RGB buffer for counting and discontinuities (lower than 2xMSAA)
- 2 G16R16F buffer for normal and texture coordinates
  - Could be shared in our case
  - I recommend compressing these values because the edge detection shader is texture bound

☺ The algorithm uses no conditional branching

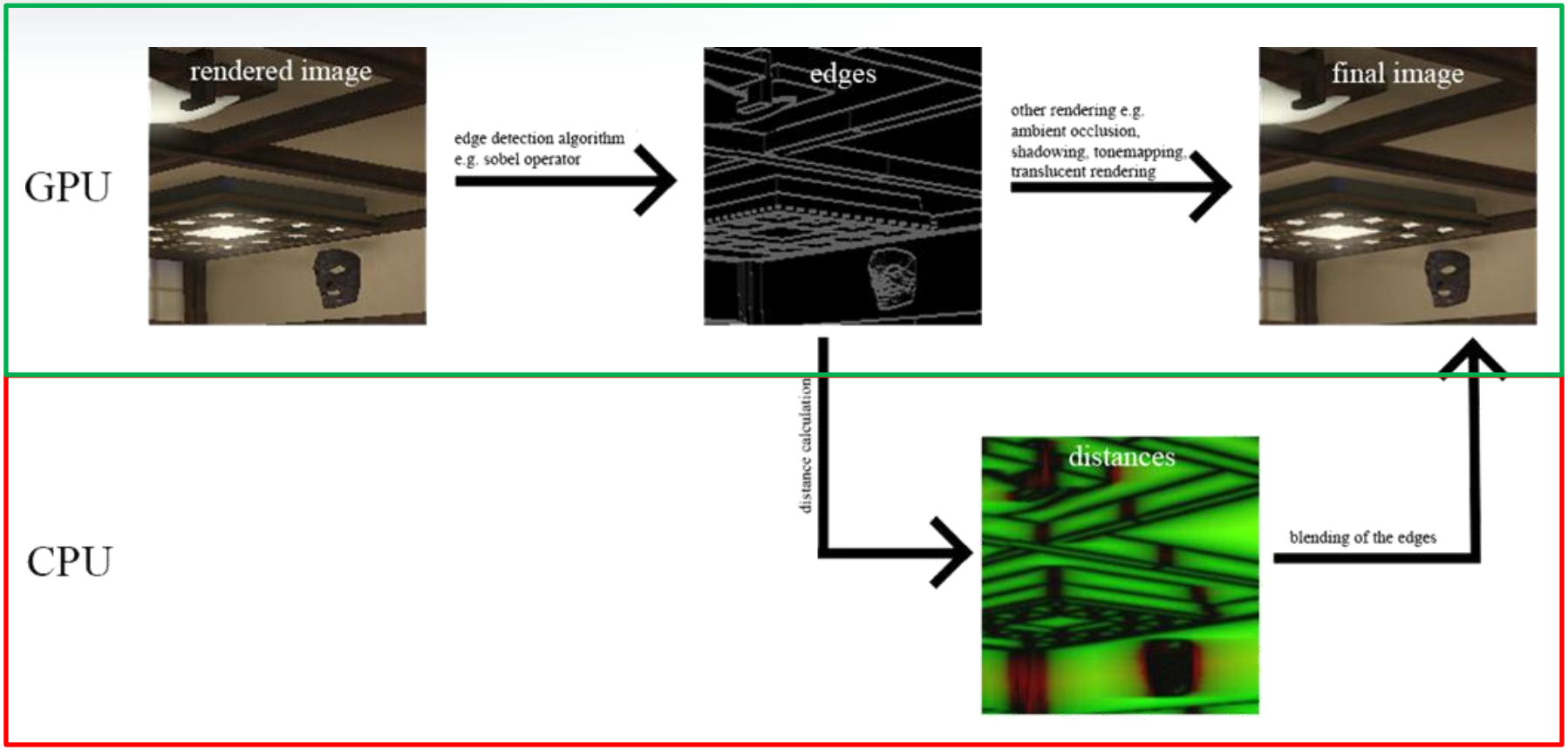
😐 On the Xbox360 and PS3 the counting could be offloaded to the CPU

☺ Therefore it is suitable for current consoles

YAGER

IMPLEMENTATION DETAILS

Overview



YAGER

Blending optimization

- Blending of horizontal and vertical edges is done separately
- A stencil prepass guaranteed that the expensive blending shader would only be executed on the edges
- The blending itself can be calculated this way:

$$minus = 1 - c0 - c1$$

$$upper' = upper * c0 \quad lower' = lower * c0$$

$$final = center * minus + upper' + lower'$$

- To save one resolve operation between horizontal and vertical blending use alpha blend:

$$DestAlpha = minus \quad SrcAlpha = ONE \quad AlphaOp = ADD$$

$$SrcColor = upper' + lower'$$

YAGER

Linear textures on Xbox360

- Use MEMEXPORT in the pixelshader to write linear textures
  - Saves the resolve operation
- To save the expensive transpose operation on the CPU we pre-transposed the horizontal discontinuity buffer on the GPU
  - Exporting 4bytes per pixel was very expensive → ?cache line stalls?
  - Therefore we exported 48bytes per pixel 64bytes were not effective due to register spilling

```
int2 IntOffsets =  round(InUV.xy * TexelOffset.xy);
int Stride = round(TexStrideWithPadding);
int offset = IntOffsets.y * Stride + IntOffsets.x;
asm
{
          alloc export=2
          mad eA, offset, const01, streamConstant
          mov eM0, OutValue.x
          mov eM1, OutValue.y
          mov eM2, OutValue.z
          mov eM3, OutValue.w
};
```

YAGER

Counting on the GPU (for XNA)

- Initialize the counting buffer with the maximum value

↓↑ counting- & vertical discontinuity buffer							
F	F	F	F	F	F	F	F
F	F	F	F	F	F	F	F
F	F	F	F	F	F	F	F

→← counting- & horizontal discontinuity buffer							
F	F	F	F	F	F	F	F
F	F	F	F	F	F	F	F
F	F	F	F	F	F	F	F

- Zero where there is a discontinuity and mark the pixel for stencil rejection in subsequent passes
- Where there is no edge in the other discontinuity buffer mark the pixel for stencil rejection in subsequent passes

↓↑ counting- & stencil buffer							
F	F	0	0	0	0	F	F
0	0	0	0	0	0	F	F
0	0	F	F	F	F	F	F

→← counting- & stencil buffer							
F	F	F	F	F	0	0	F
F	0	0	F	F	F	F	F
F	F	F	F	F	F	F	F

Counting on the GPU

↓↑ counting- & stencil buffer							
F	F	0	0	0	0	F	F
0	0	0	0	0	0	F	F
0	0	F	F	F	F	F	F

→← counting- & stencil buffer							
F	F	F	F	F	0	0	F
F	0	0	F	F	F	F	F
F	F	F	F	F	F	F	F

- Start with level = 1 and lookup level pixels in both directions (texture border will count as 0) and take the minimum of these values if the resulting value is not equal the maximumValue = F the current pixel is updated with result = value + level and mark the pixel for stencil rejection in subsequent passes else reject the pixel

↓↑ counting- & stencil buffer							
F	F	0	0	0	0	1	F
0	0	0	0	0	0	F	F
0	0	F	F	F	F	F	F

→← counting- & stencil buffer							
F	F	F	F	1	0	0	F
1	0	0	1	F	F	F	F
1	F	F	F	F	F	F	F

Counting on the GPU

↓↑ counting- & stencil buffer							
F	F	0	0	0	0	1	F
0	0	0	0	0	0	F	F
0	0	F	F	F	F	F	F

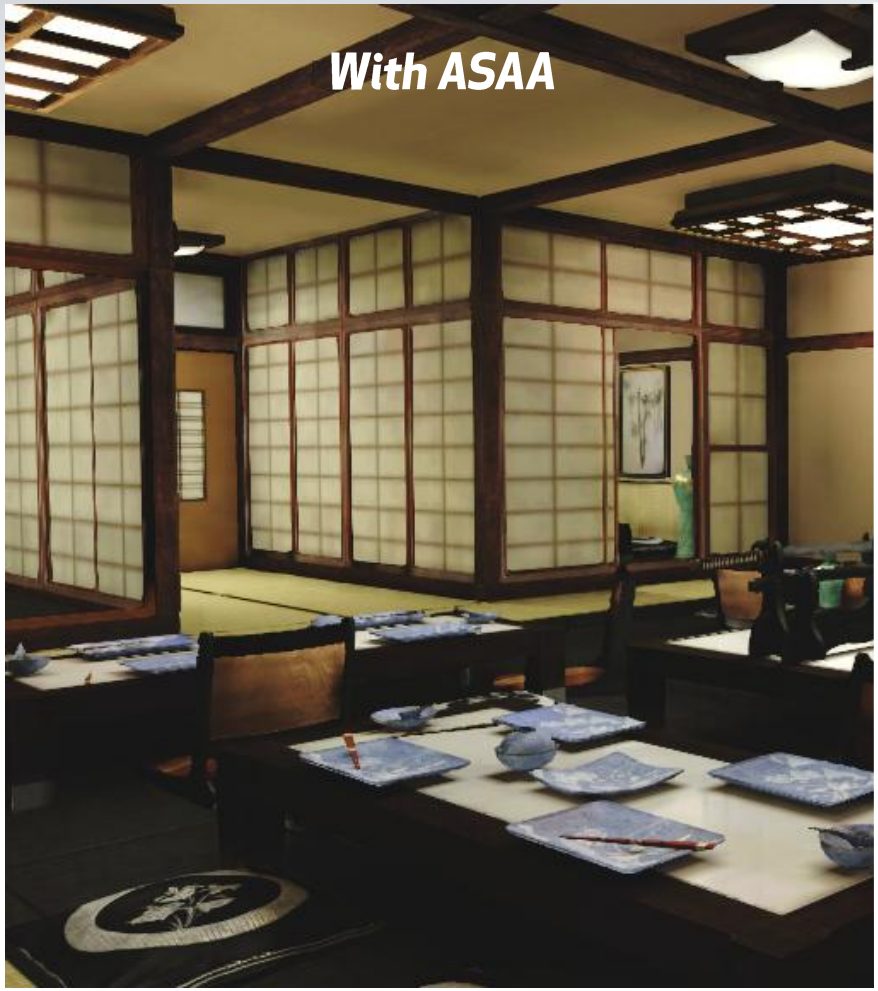
→← counting- & stencil buffer							
F	F	F	F	1	0	0	F
1	0	0	1	F	F	F	F
1	F	F	F	F	F	F	F

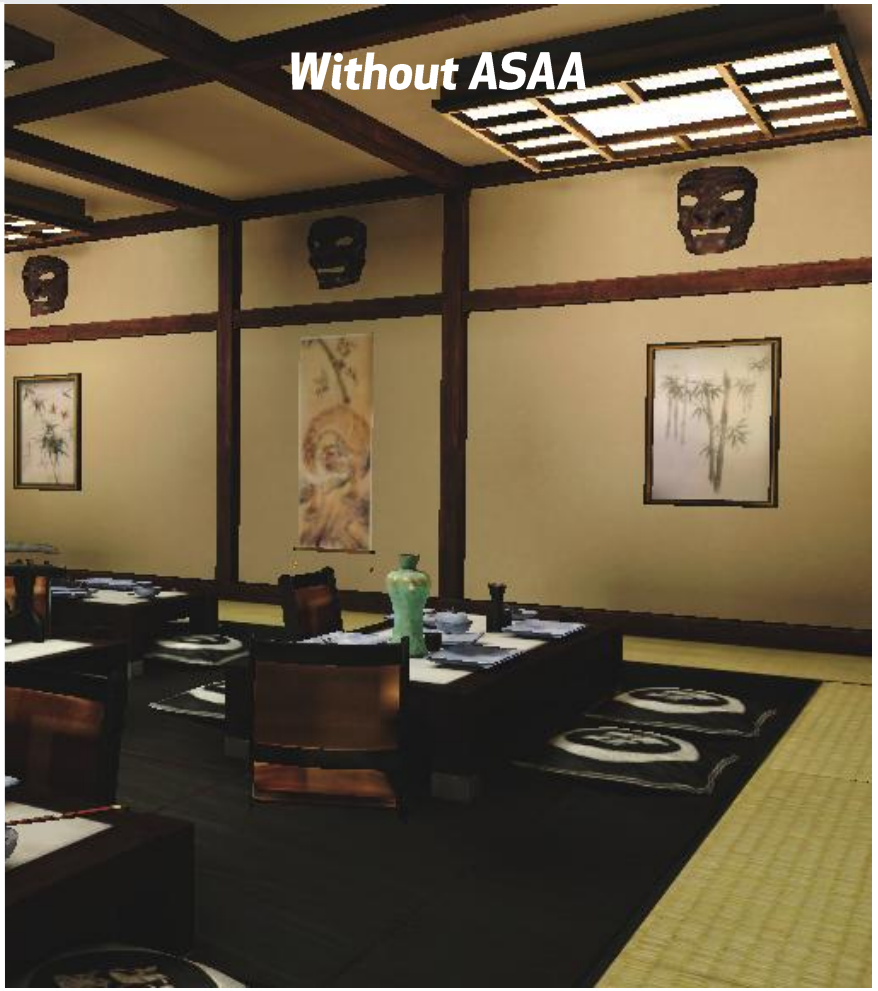
- continue with level *= 2 (level is now 2):
- do this repeatedly until no further pixels are drawn or maximum precision is reached
- Remember the maximum number of passes is equal to the number of bits used per counting value

↓↑ counting- & stencil buffer							
F	F	0	0	0	0	1	F
0	0	0	0	0	0	F	F
0	0	F	F	F	F	F	F

→← counting- & stencil buffer							
F	F	3	2	1	0	0	F
1	0	0	1	2	3	F	F
1	2	F	F	F	F	F	F

Demo





YAGER

“If you can't make it good, at least make it look good.”
Bill Gates

YAGER IS HIRING!

jobs@yager.de
www.yager.de/career.html