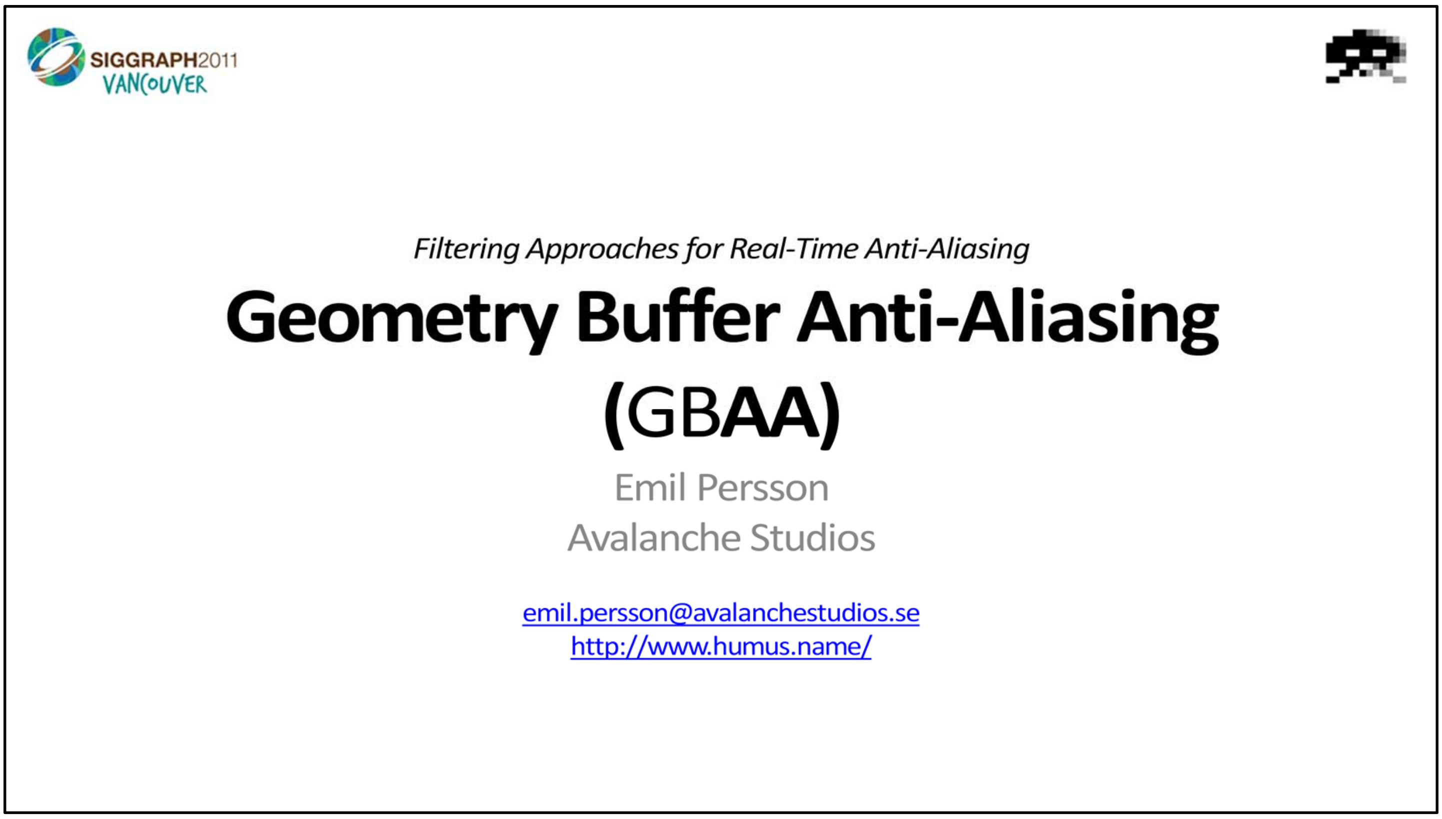
Filtering Approaches for Real-Time Anti-Aliasing

http://www.iryoku.com/aacourse/



Filtering Approaches for Real-Time Anti-Aliasing

Geometry Buffer Anti-Aliasing (GB**AA**)

Emil Persson
Avalanche Studios

emil.persson@avalanchestudios.se
http://www.humus.name/

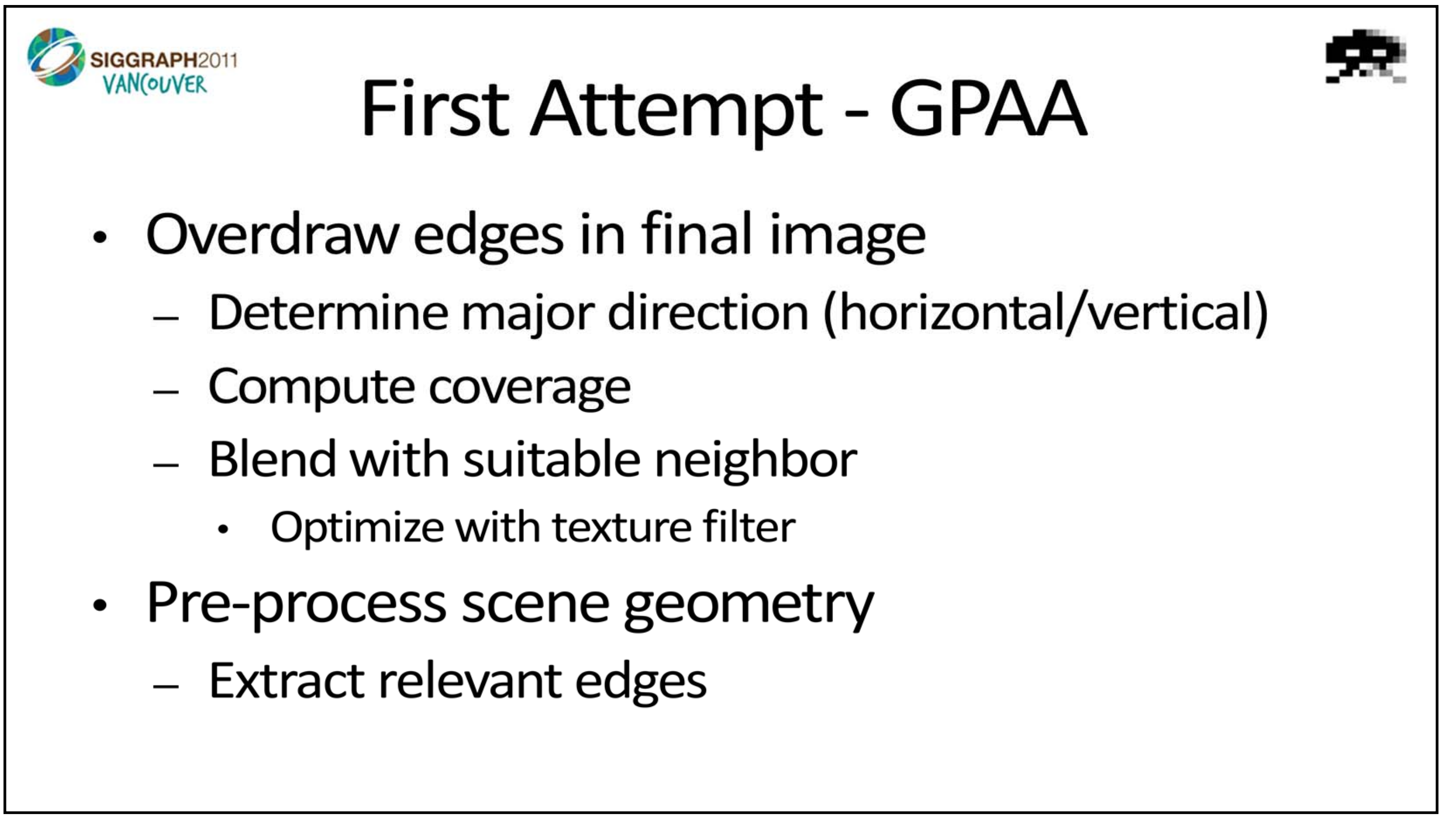
Basic idea

MLAA and friends recover edges from backbuffer (and optionally depth-buffer)

Idea:

Game engine knows where the edges are.

Let's use that!



First Attempt - GPAA

- Overdraw edges in final image
  - Determine major direction (horizontal/vertical)
  - Compute coverage
  - Blend with suitable neighbor
    - Optimize with texture filter
- Pre-process scene geometry
  - Extract relevant edges

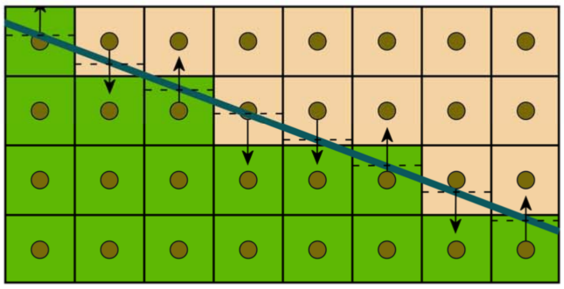
Pre-process eliminates internal edges in a surface and removes duplicates from shared edges.

Line equations are computed in screen-space and passed to pixel shader.

SIGGRAPH2011 VANCOUVER

GPAA – Shader

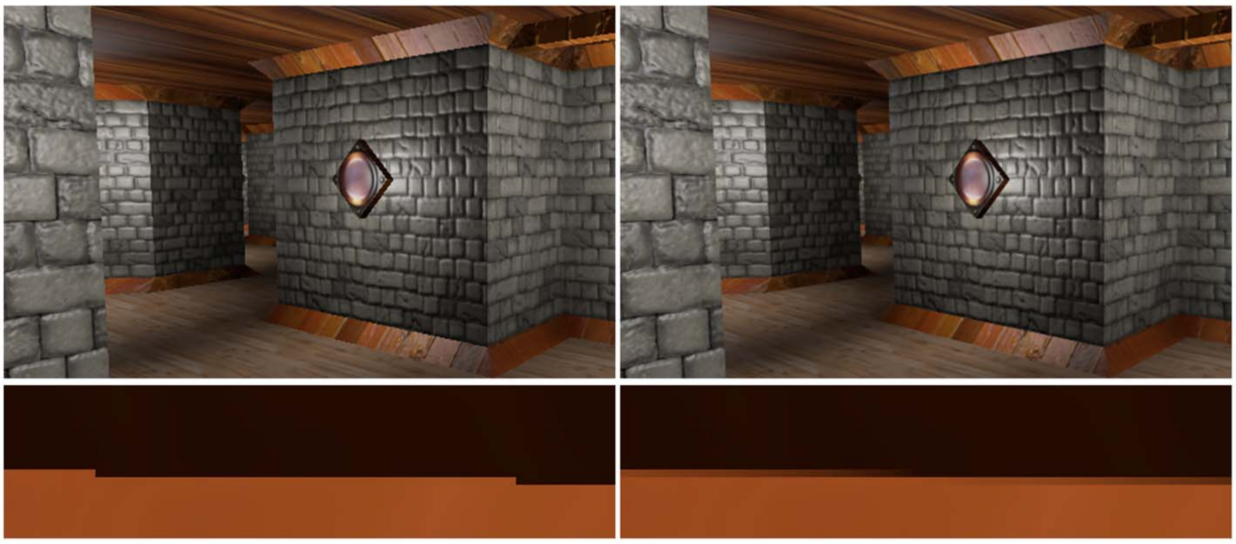
```
// Compute the difference between geometric line and sample position
float diff = dot(In.KMF.xy, In.Position.xy) + In.KMF.z;

// Compute the coverage of the neighboring surface
float coverage = 0.5f - abs(diff);
float2 offset = 0;

if (coverage > 0) {
    // Select direction to sample a neighbor pixel
    float off = (diff >= 0)? 1 : -1;
    if (asuint(In.KMF.w))
        offset.y = off;
    else
        offset.x = off;
}

// Blend pixel with neighbor pixel using texture filtering and shifting the coordinate appropriately.
return BackBuffer.Sample(Filter, (In.Position.xy + coverage * offset.xy) * PixelSize);
```

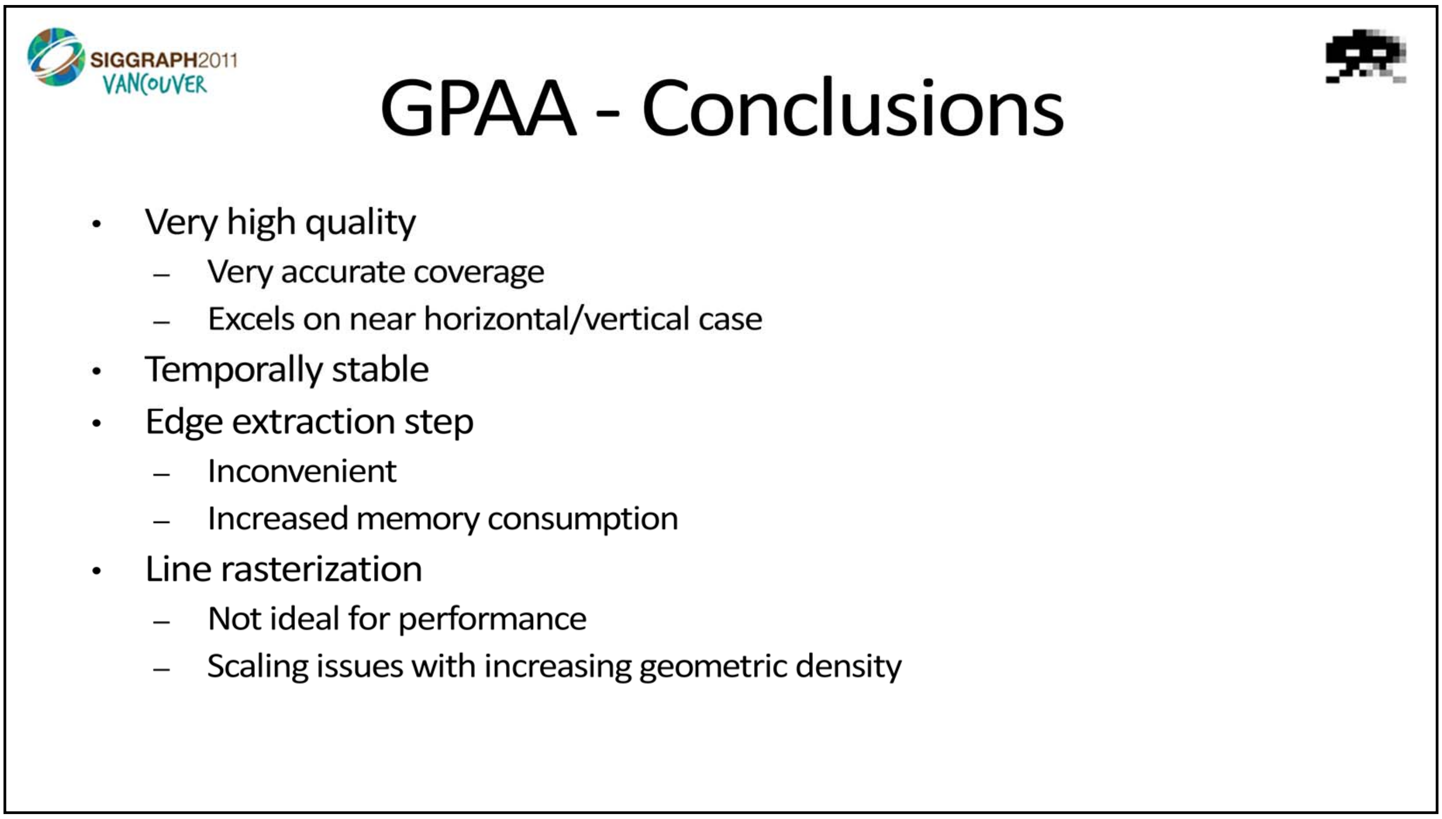


In.KMF contains line equation for either horizontal or vertical direction. Direction flag is stored in w.
Sampling direction is selected based on which side the pixel center the line cuts the pixel.
Coverage is computed from distance.

SIGGRAPH2011 VANCOUVER

GPAA - Results



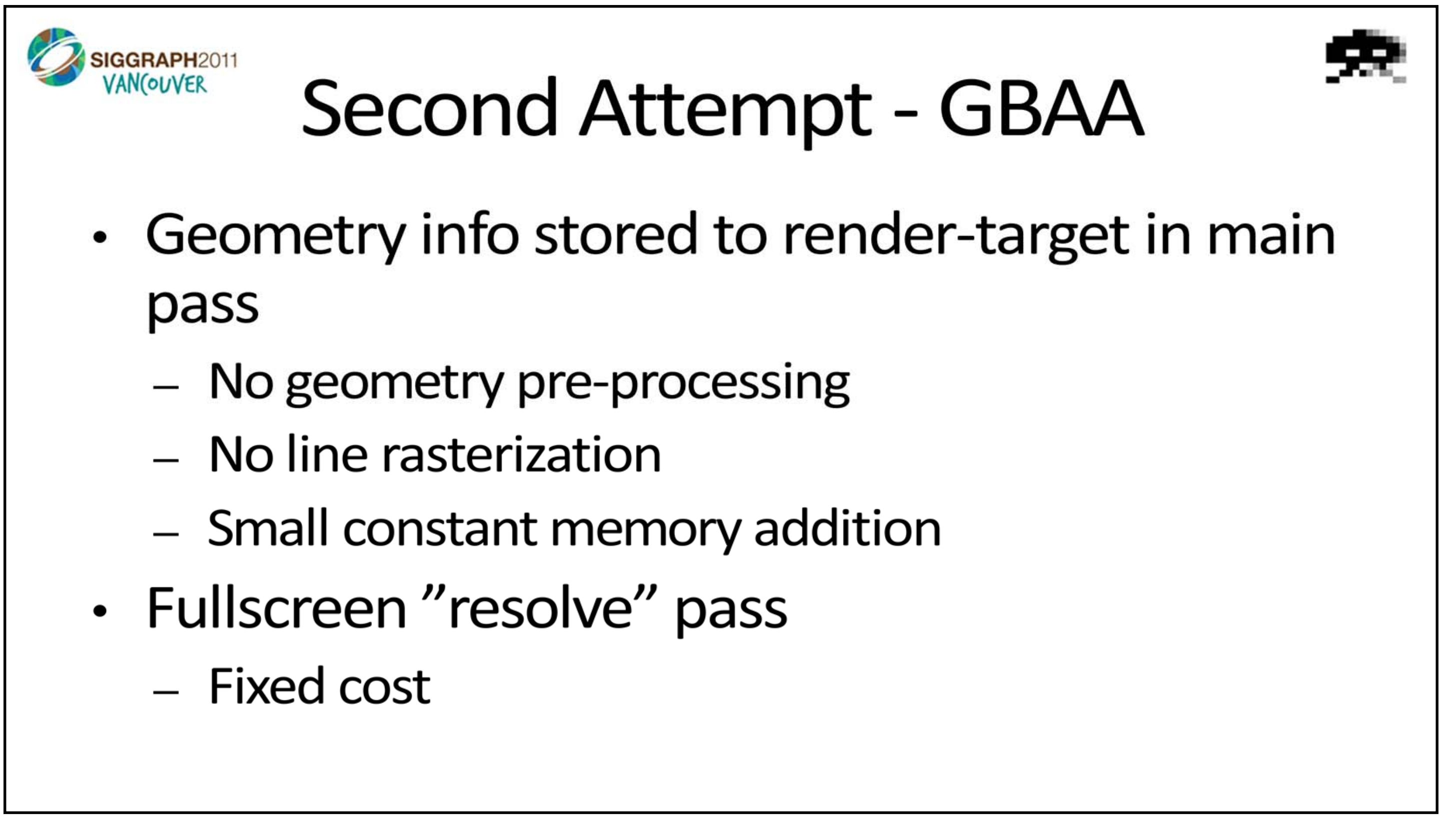


SIGGRAPH2011 VANCOUVER

GPAA - Conclusions

- Very high quality
  - Very accurate coverage
  - Excels on near horizontal/vertical case
- Temporally stable
- Edge extraction step
  - Inconvenient
  - Increased memory consumption
- Line rasterization
  - Not ideal for performance
  - Scaling issues with increasing geometric density

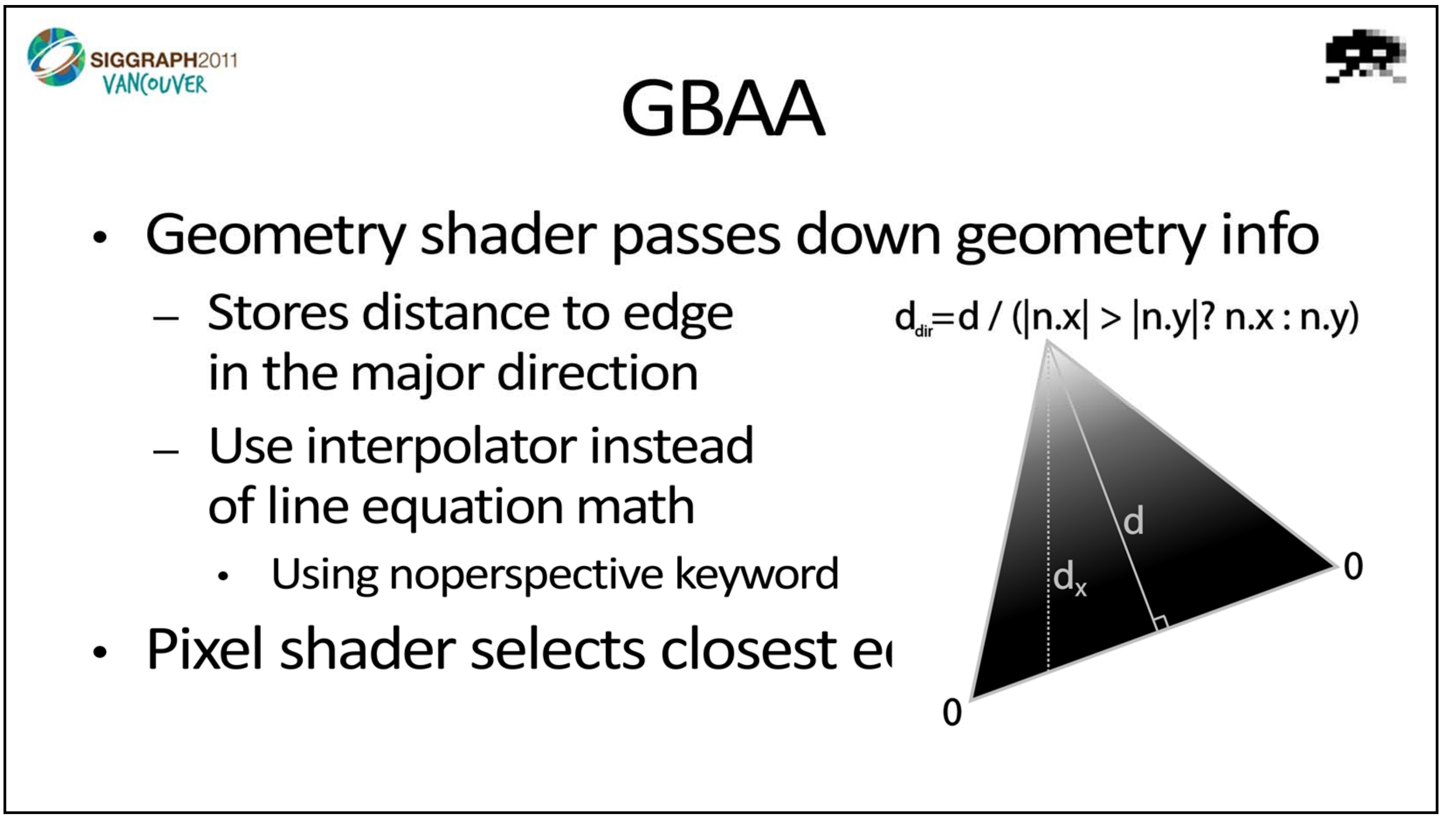
Demo with source available for download at http://www.humus.name



Second Attempt - GBAA

- Geometry info stored to render-target in main pass
  - No geometry pre-processing
  - No line rasterization
  - Small constant memory addition
- Fullscreen ”resolve” pass
  - Fixed cost

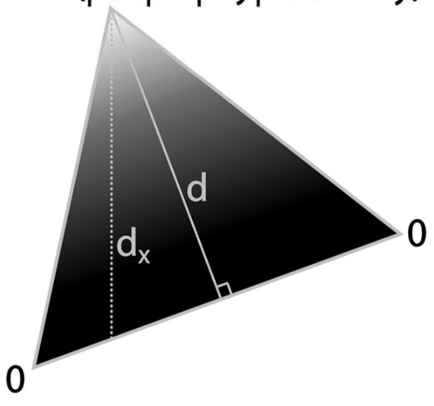
Geometry Buffer is fullscreen, two channels, signed format. Could pack into single channel if necessary (one bit stores flag for horizontal/vertical)



GBAA

- Geometry shader passes down geometry info
  - Stores distance to edge in the major direction
  - Use interpolator instead of line equation math
    - Using noperspective keyword
- Pixel shader selects closest e

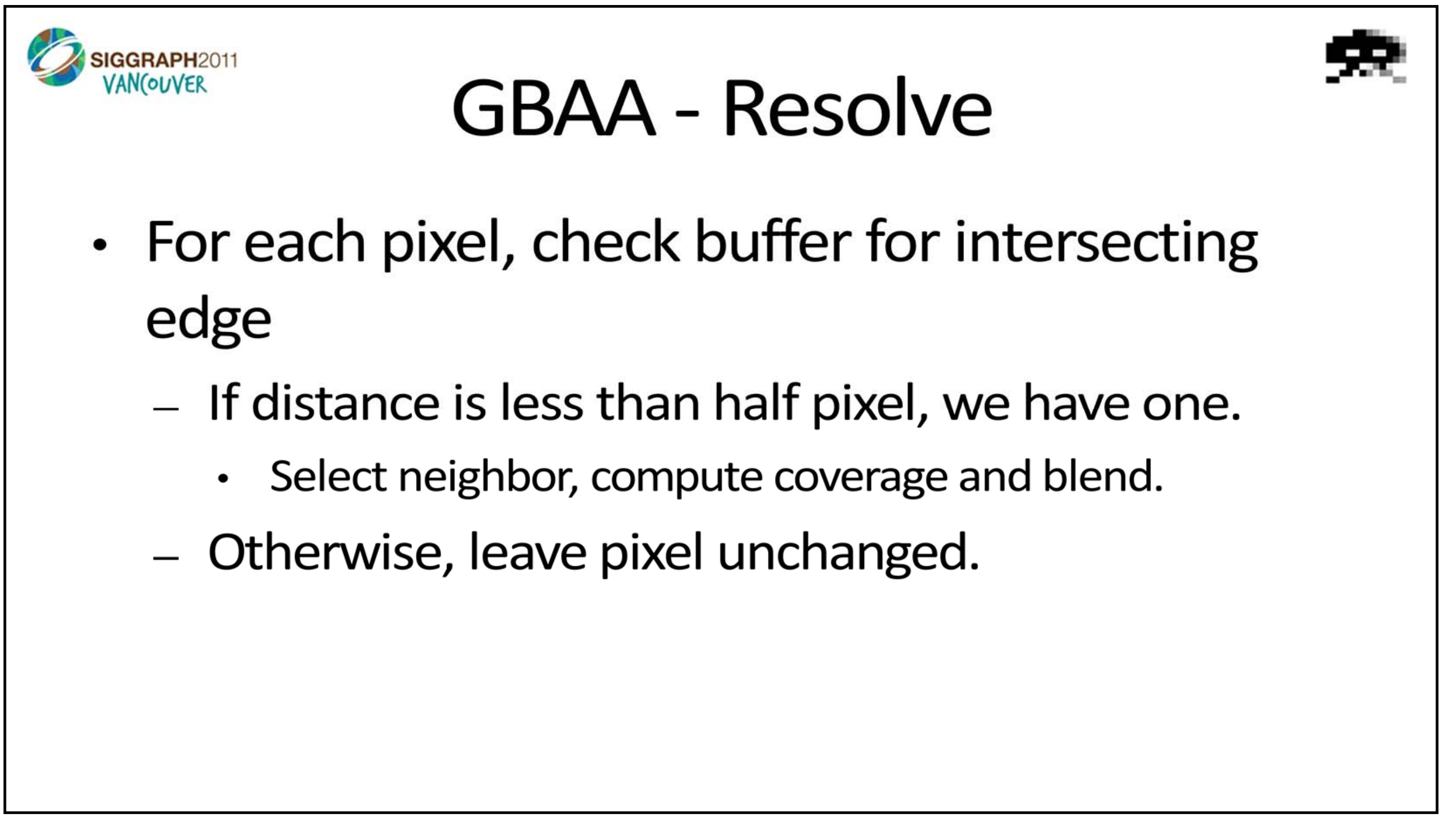
$d_{dir} = d / (|n.x| > |n.y| ?\ n.x : n.y)$



The two vertices on the edge are naturally of zero distance to that edge.

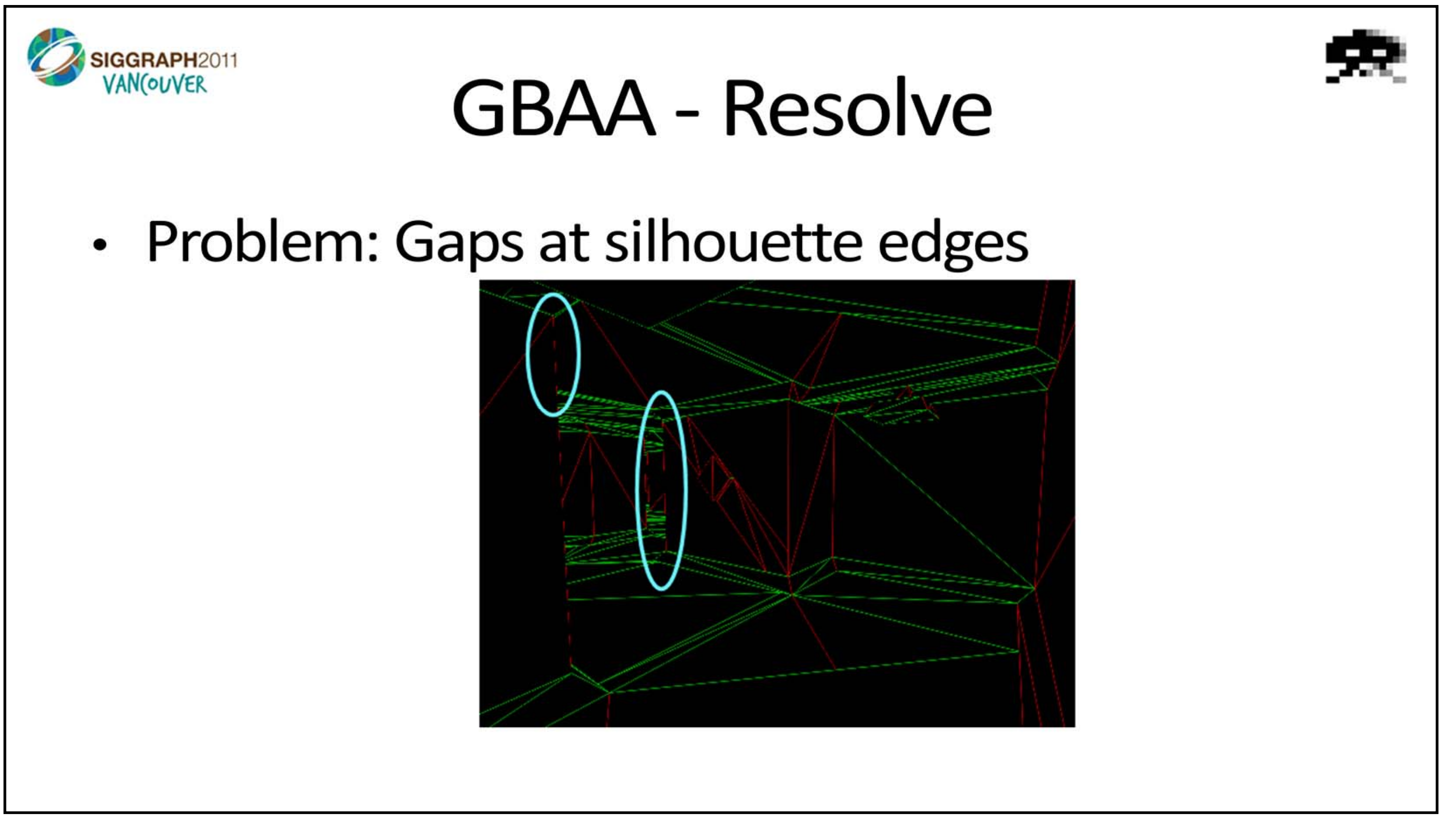
The top vertex is assigned the vertical distance because the edge is horizontal.

The perpendicular distance (d) is computed first, then scaled by a factor to become the vertical distance (dx).



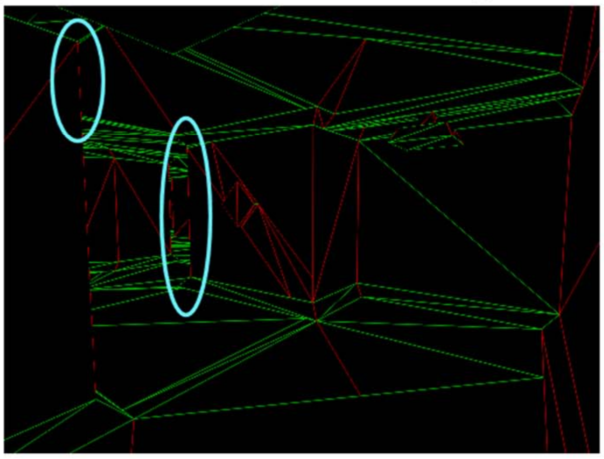
GBAA - Resolve

- For each pixel, check buffer for intersecting edge
  - If distance is less than half pixel, we have one.
    - Select neighbor, compute coverage and blend.
  - Otherwise, leave pixel unchanged.

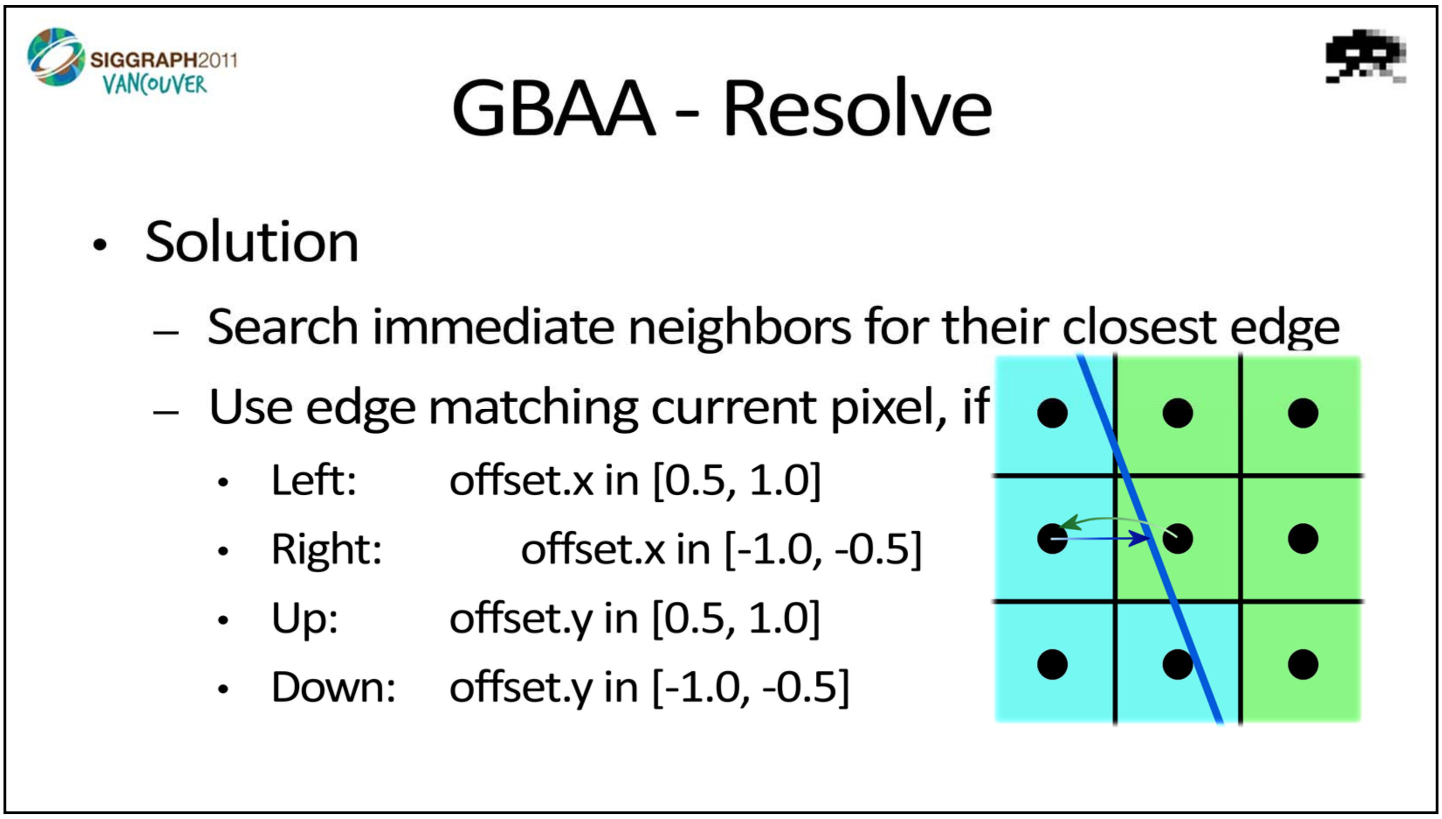


GBAA - Resolve

- Problem: Gaps at silhouette edges

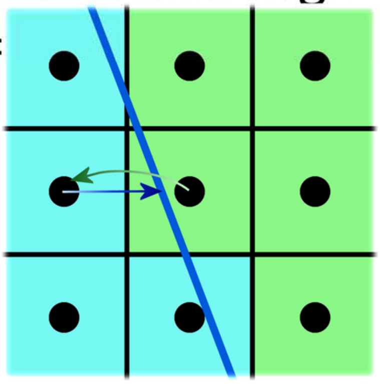


These gaps occur because only one side of the edge has relevant geometry information. The background surface does not know about the foreground geometry. The foreground will blend properly towards the edge, but the background will not blend at all since it has no indication of an edge passing through. This results in visible gaps in the antialiasing along silhouette edges. Edges between adjacent triangles are not affected because both sides are aware of the edge.



GBAA - Resolve

- Solution
  - Search immediate neighbors for their closest edge
  - Use edge matching current pixel, if
    - Left: offset.x in [0.5, 1.0]
    - Right: offset.x in [-1.0, -0.5]
    - Up: offset.y in [0.5, 1.0]
    - Down: offset.y in [-1.0, -0.5]



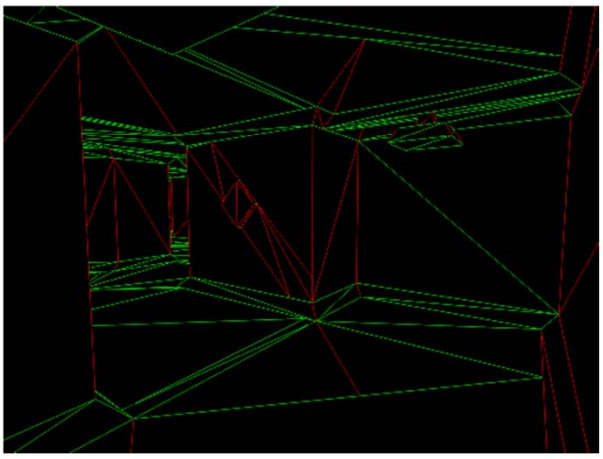
We only need to check the component that can point towards us. For left/right check whether the x-component match this pixel, for up/down check y-component. If the neighbor's distance value is such that it would cut this pixel we use this edge information.

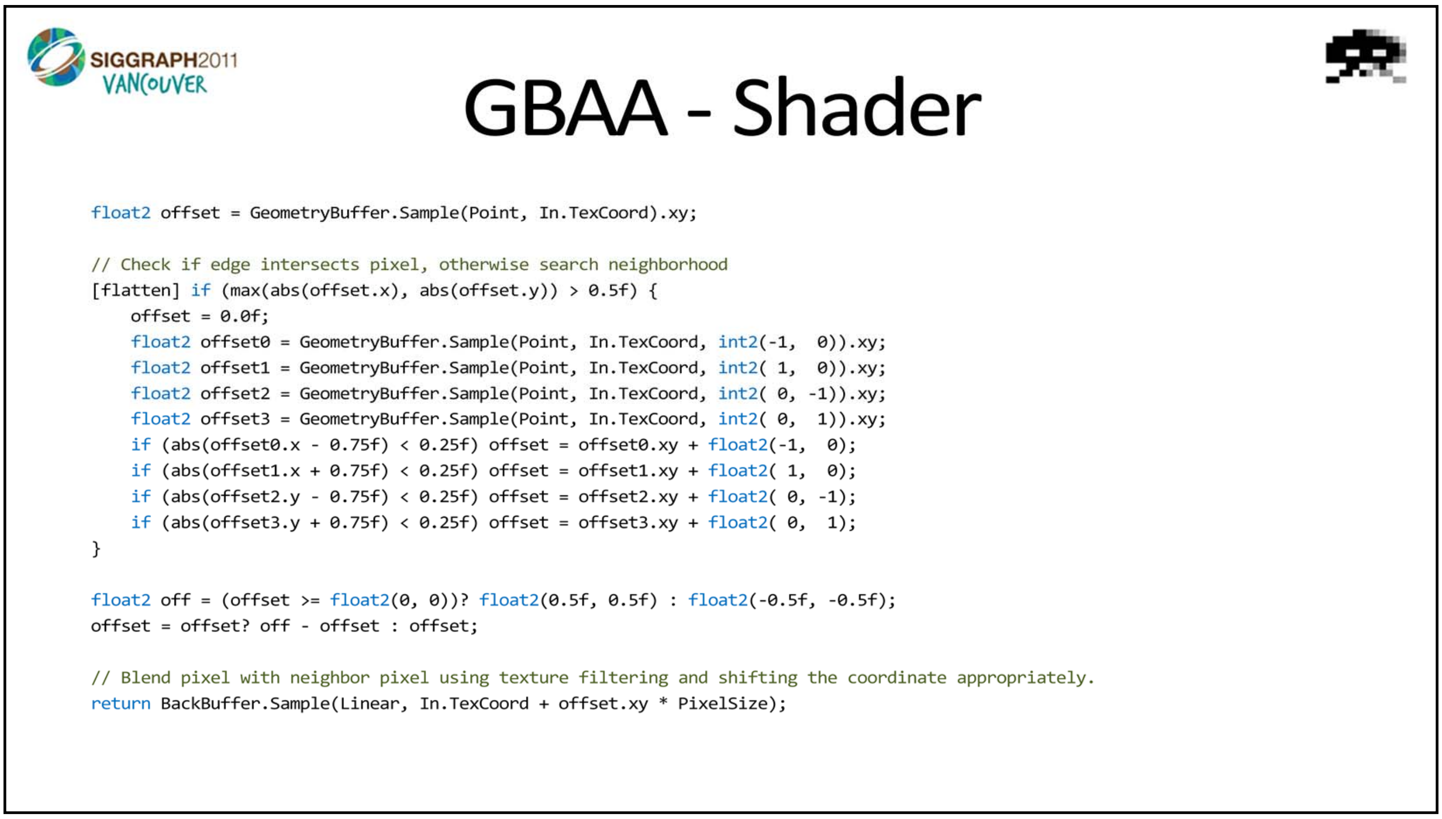
In this illustration the left pixel hold relevant edge info. The value is perhaps 0.8, which is more than 0.5 (pixel's left edge) and less than 1.0 (pixel center). This gives us a distance of 0.8 – 1.0 = -0.2, i.e. edge cuts through 0.2 pixel units to the left of the pixel center.

Note that in this illustration the down pixel also points to the right, thus it is of no use, because it has to point up to be useful. Right and up belong to the same surface and hold not useful information in this situation either.

GBAA - Resolve

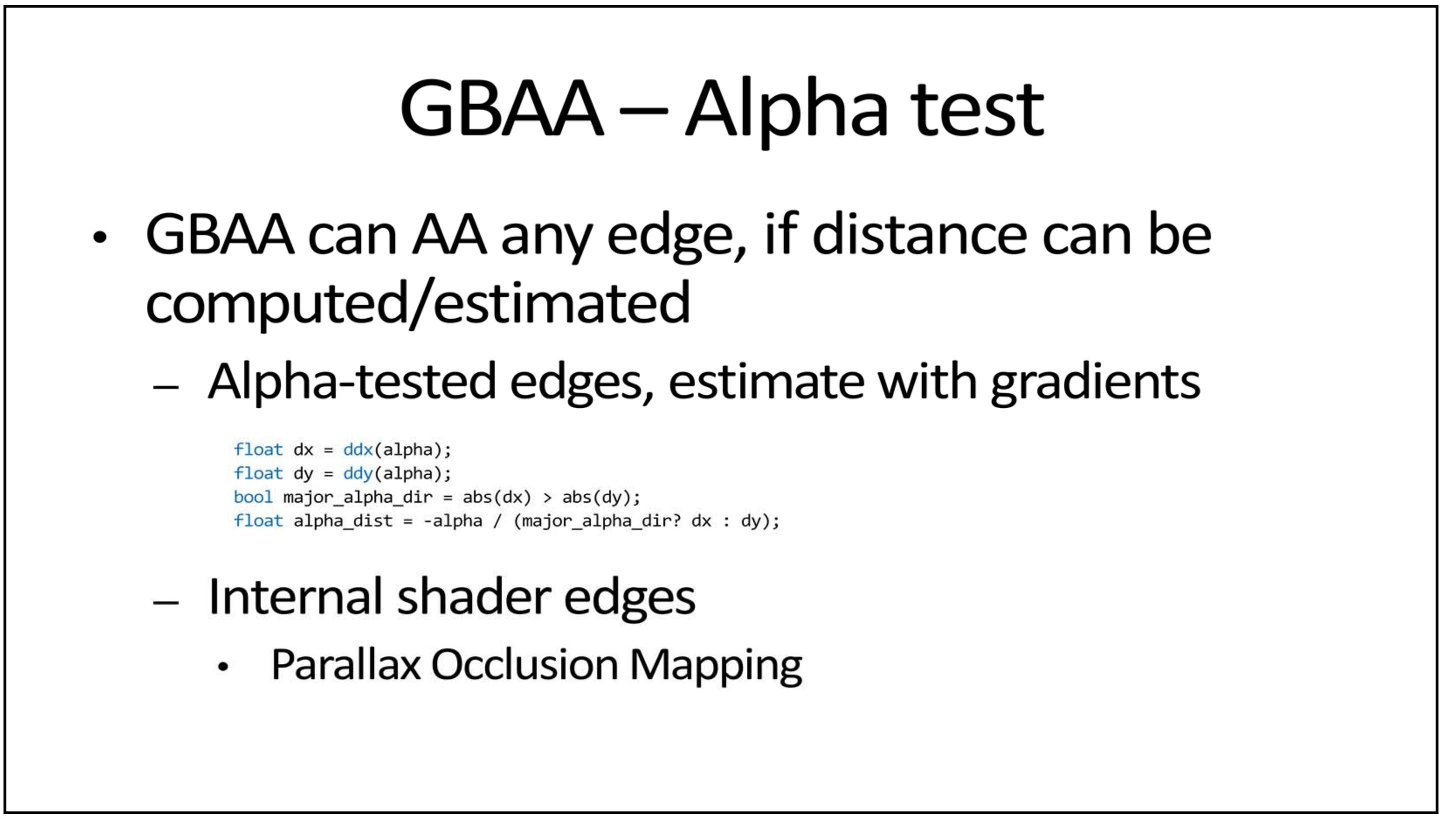
- Edges recovered!





GBAA - Shader

```
float2 offset = GeometryBuffer.Sample(Point, In.TexCoord).xy;

// Check if edge intersects pixel, otherwise search neighborhood
[flatten] if (max(abs(offset.x), abs(offset.y)) > 0.5f) {
    offset = 0.0f;
    float2 offset0 = GeometryBuffer.Sample(Point, In.TexCoord, int2(-1,  0)).xy;
    float2 offset1 = GeometryBuffer.Sample(Point, In.TexCoord, int2( 1,  0)).xy;
    float2 offset2 = GeometryBuffer.Sample(Point, In.TexCoord, int2( 0, -1)).xy;
    float2 offset3 = GeometryBuffer.Sample(Point, In.TexCoord, int2( 0,  1)).xy;
    if (abs(offset0.x - 0.75f) < 0.25f) offset = offset0.xy + float2(-1,  0);
    if (abs(offset1.x + 0.75f) < 0.25f) offset = offset1.xy + float2( 1,  0);
    if (abs(offset2.y - 0.75f) < 0.25f) offset = offset2.xy + float2( 0, -1);
    if (abs(offset3.y + 0.75f) < 0.25f) offset = offset3.xy + float2( 0,  1);
}

float2 off = (offset >= float2(0, 0))? float2(0.5f, 0.5f) : float2(-0.5f, -0.5f);
offset = offset? off - offset : offset;

// Blend pixel with neighbor pixel using texture filtering and shifting the coordinate appropriately.
return BackBuffer.Sample(Linear, In.TexCoord + offset.xy * PixelSize);
```



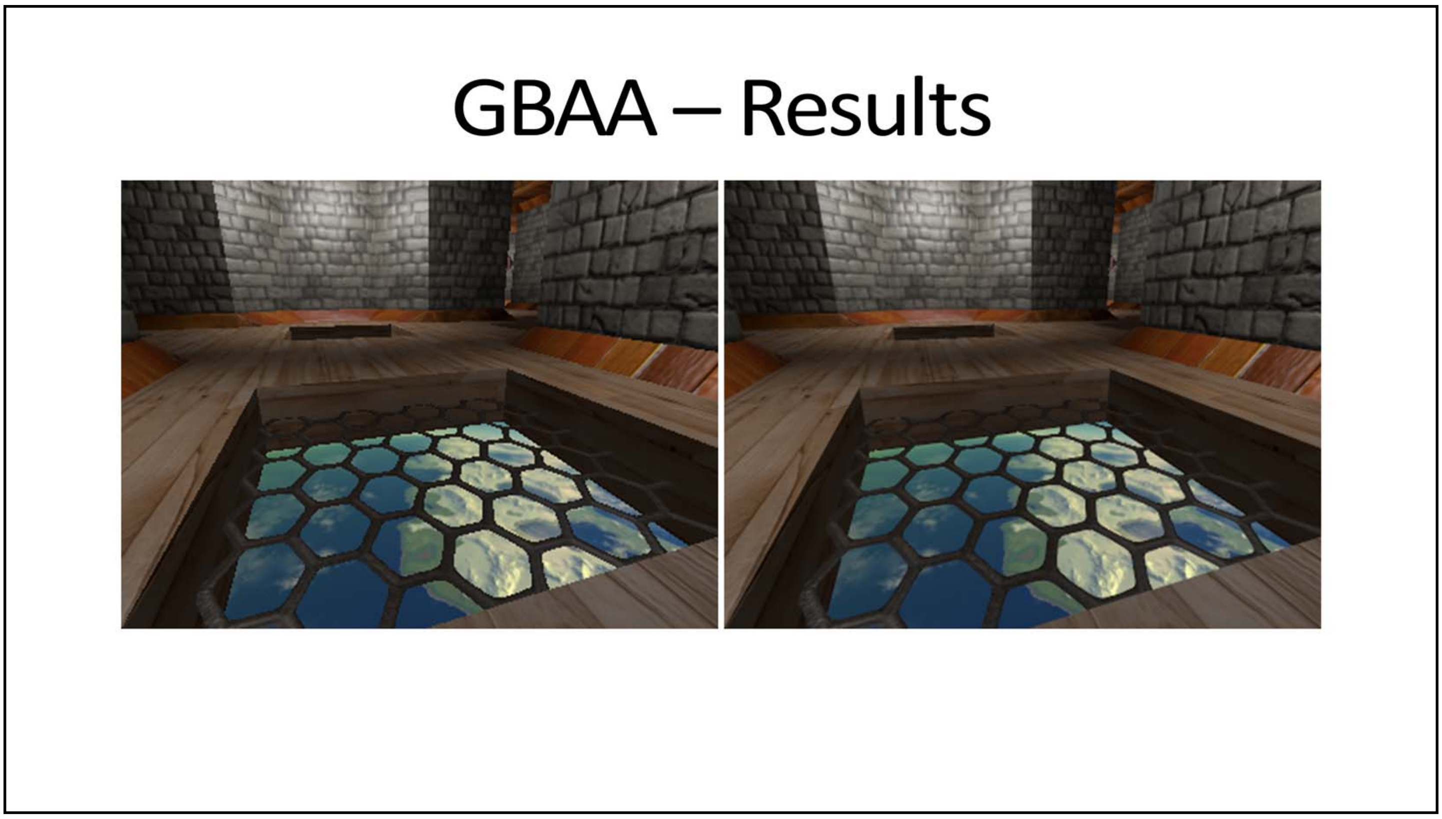
GBAA – Alpha test

- GBAA can AA any edge, if distance can be computed/estimated
  - Alpha-tested edges, estimate with gradients

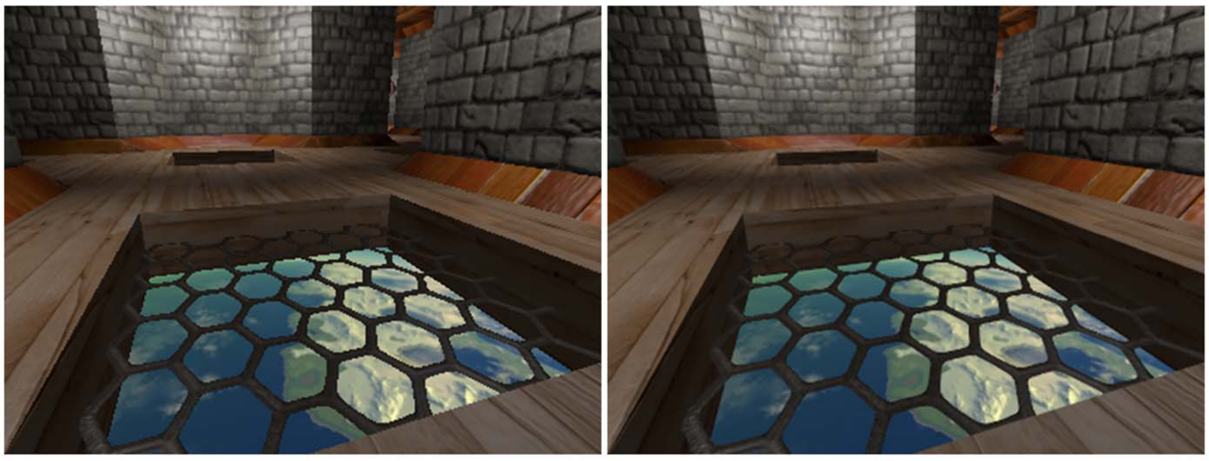
```
float dx = ddx(alpha);
float dy = ddy(alpha);
bool major_alpha_dir = abs(dx) > abs(dy);
float alpha_dist = -alpha / (major_alpha_dir? dx : dy);
```

  - Internal shader edges
    - Parallax Occlusion Mapping

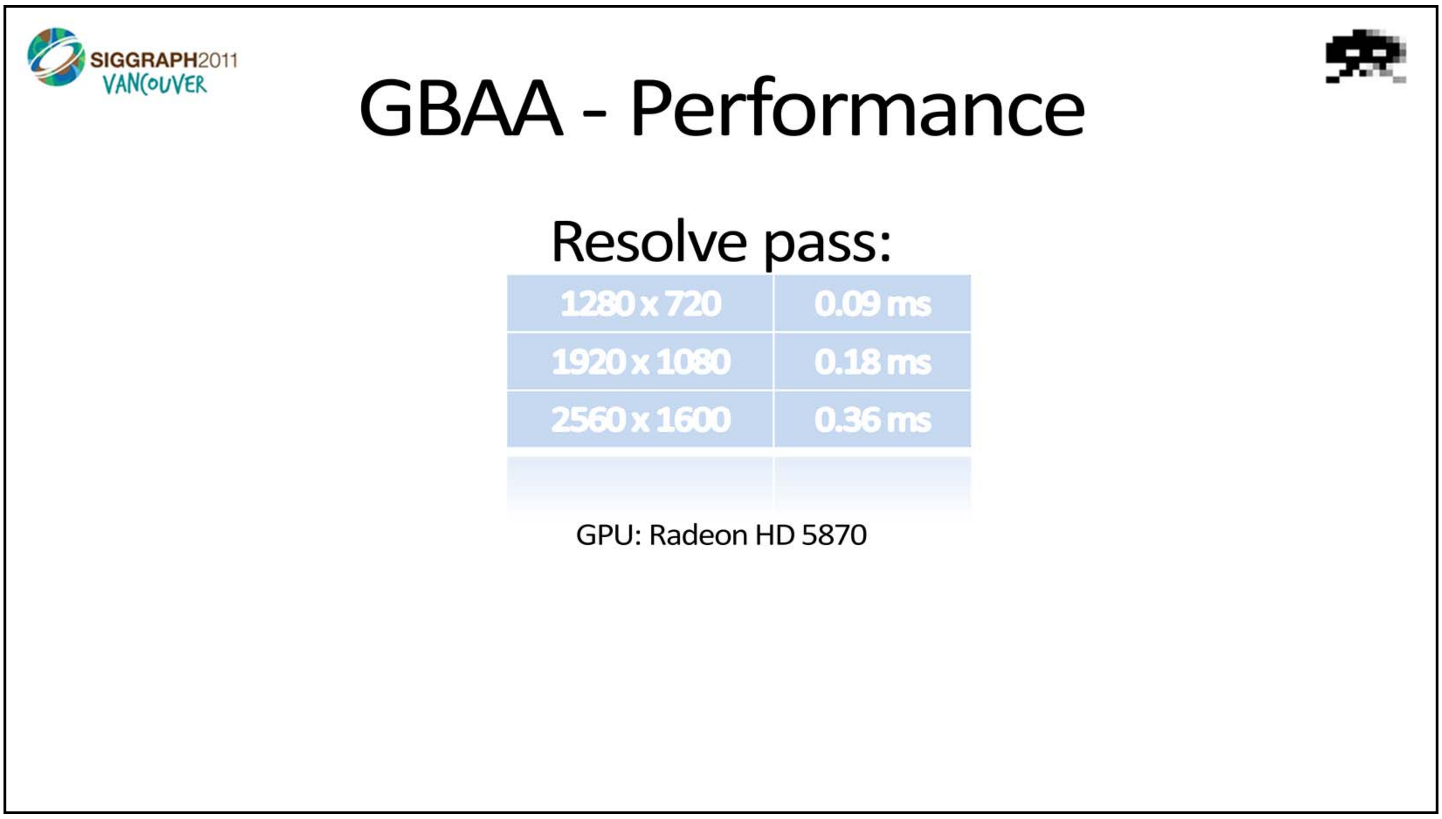
Alpha-test can be better estimated by taking gradients of texture coordinate and sample texture left/right/up/down. This is more costly though, and the above works good enough in practice.



GBAA – Results



Results as good as GPAA, but also AA on alpha-tested surfaces.



GBAA - Performance

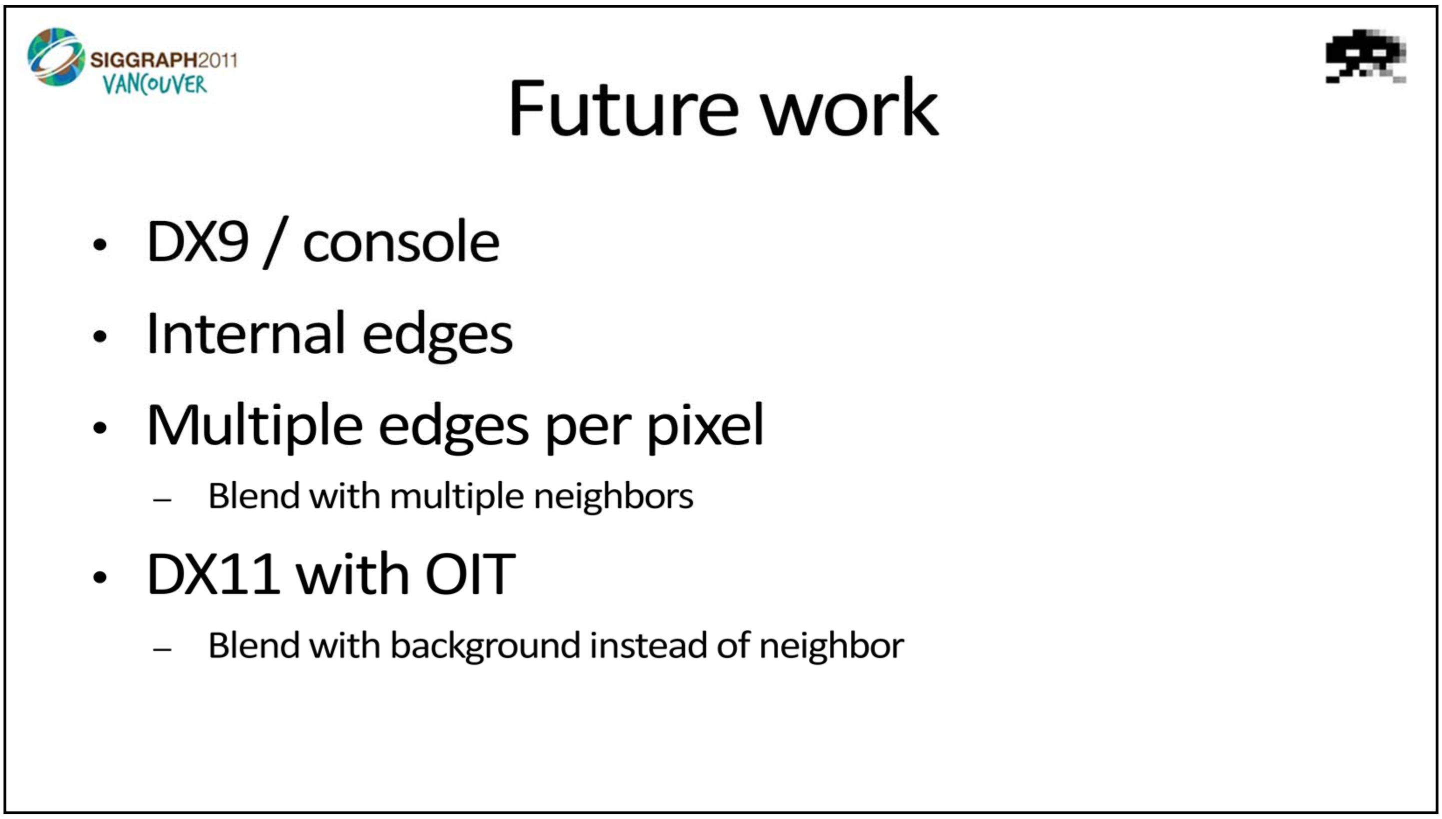
Resolve pass:

1280 x 720	0.09 ms
1920 x 1080	0.18 ms
2560 x 1600	0.36 ms

GPU: Radeon HD 5870

Resolve pass is cheap and has fairly constant cost. I used clip() to reduce writes to backbuffer, which improved performance somewhat. This may not be an optimization for dense geometry though.

Note that these numbers are the resolve pass only. This technique also incur a cost during main scene rendering, which is content dependent and may vary from scene to scene.



SIGGRAPH2011 VANCOUVER

Future work

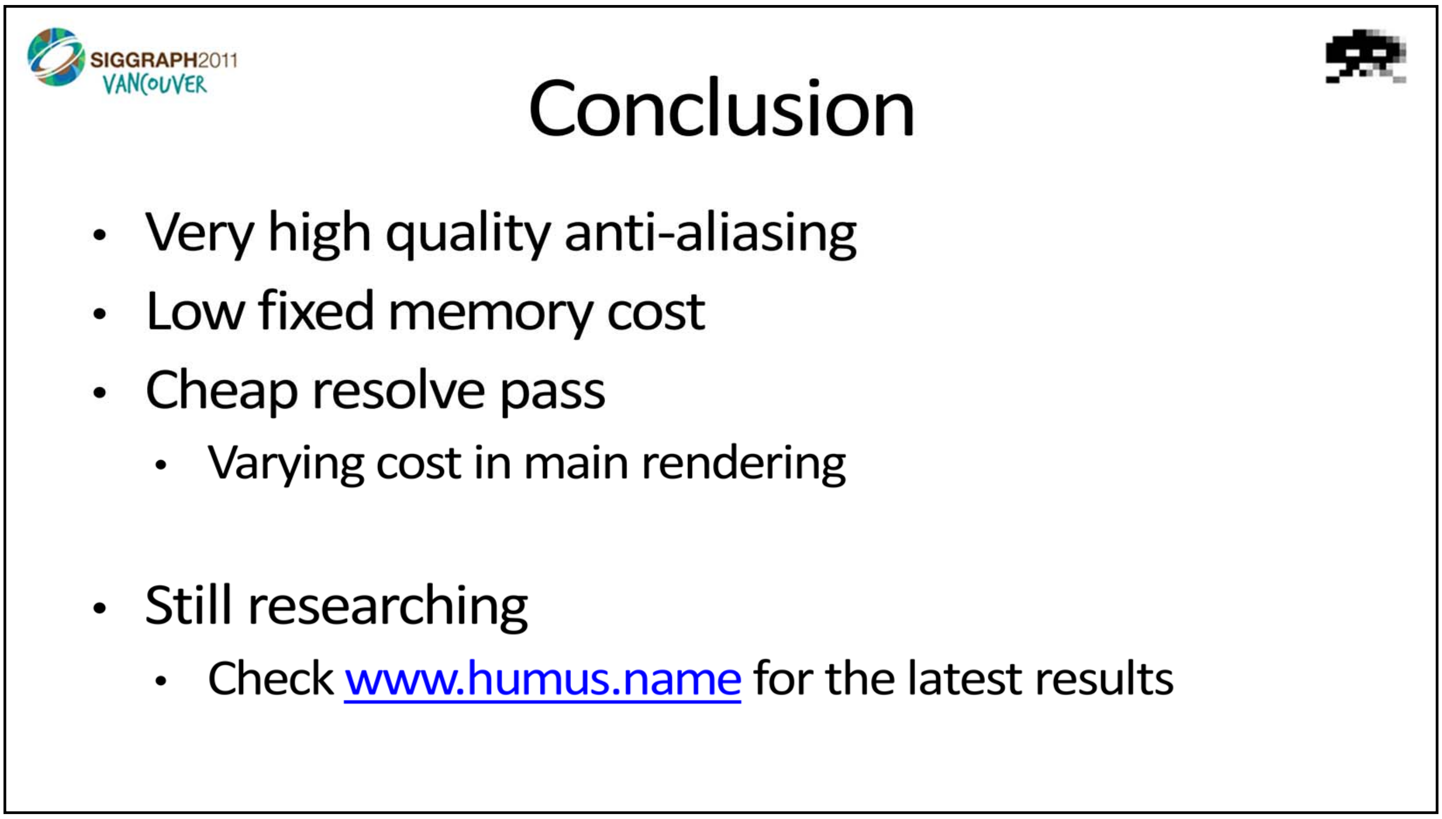
- DX9 / console
- Internal edges
- Multiple edges per pixel
  - Blend with multiple neighbors
- DX11 with OIT
  - Blend with background instead of neighbor

Current implementation depends on geometry shader, DX10. It is possible to implement in DX9 though, but would be more costly in terms of memory. Ways around this would be an interesting research topic.

Internal edges should not be antialiased. Current implementation does. It appears to do no serious harm, but ideally should be avoided.

Corners and T-junctions can sometimes cause minor artifacts. Using all edges and blend more neighbors could probably clean up these cases.

Blending down in the depth is more correct than blending to the side. With order-independent-translucency becoming more practical, this may be the way we do post-AA in the future.



Conclusion

- Very high quality anti-aliasing
- Low fixed memory cost
- Cheap resolve pass
  - Varying cost in main rendering
- Still researching
  - Check www.humus.name for the latest results

GBAA demo available for download at www.humus.name.

Avalanche Studios is hiring for Stockholm and New York.

Next up is Tiago Sousa from Crytek who will talk about “Anti-Aliasing Methods in CryENGINE 3”