SIGGRAPH 2012

The **39th** International **Conference** and **Exhibition**
on **Computer Graphics** and **Interactive Techniques**

Graphics Gems for Games

Findings from Avalanche Studios

Emil Persson
Senior Graphics Programmer

@_Humus_

AVALANCHE STUDIOS

SIGGRAPH2012

Graphics Gems for Games

- Particle trimming
- Merge-instancing
- Phone-wire Anti-Aliasing
- Second-Depth Anti-Aliasing

Particle trimming

Particle trimming

- GPUs increasingly more powerful
  - ALU – Through the roof
  - TEX – Pretty decent increase
  - BW – Kind of sluggish
  - ROP – Glacial speed

	9700 Pro (2002)	HD 2900XT (2007)	HD 7970 (2012)	10 year speedup
ALU	33.8 GF/s	475 GF/s	3789 GF/s	112x
TEX	2.6 GT/s	11.9 GT/s	118.4 GT/s	46x
BW	19.84 GB/s	105.6 GB/s	288 GB/s	15x
ROP	2.6 GP/s	11.9 GP/s	29.6 GP/s	11x

Source: Wikipedia [1]

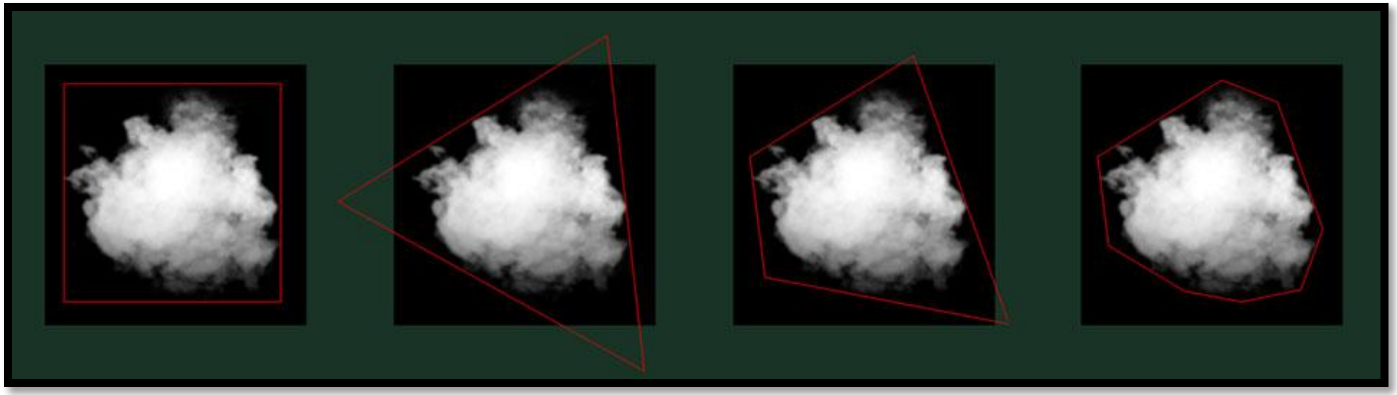
- ROP bound?
  1. Draw fewer pixels
  2. ???
  3. Goto 1

Particle trimming

- Typical ROP bound cases
  - Particles
  - Clouds
  - Billboards
  - GUI elements
- Solutions

Particle trimming

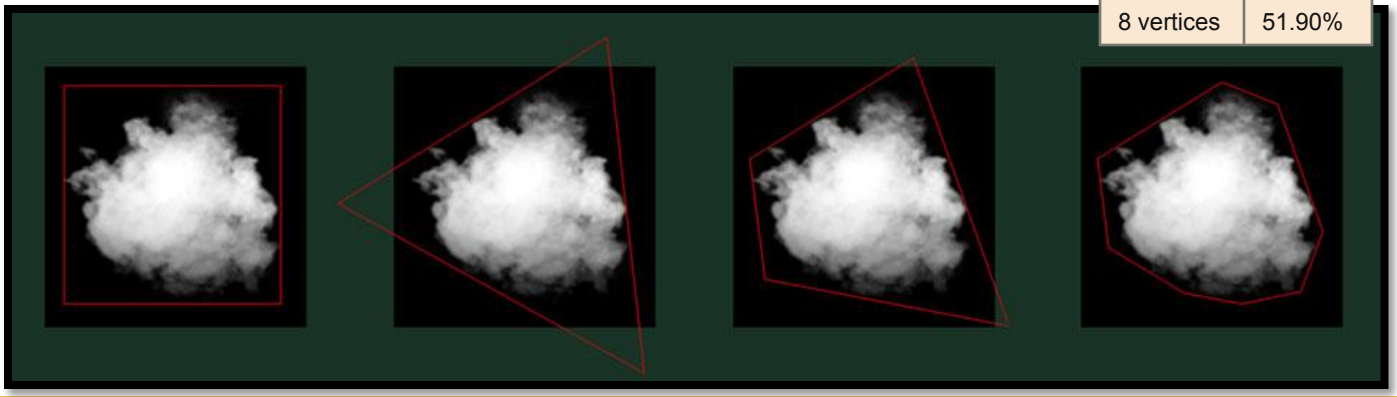
- Common with large alpha=0 areas
  - Wasted fillrate
  - Adjust particle geometry to minimize waste
  - Automated tool [2]



Particle trimming

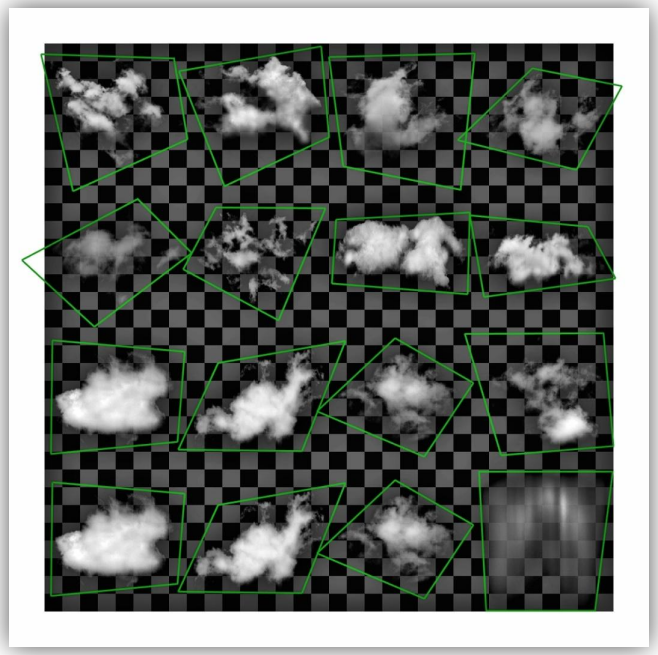
- Huge fillrate savings
  - More vertices ⇒ Bigger saving
  - Diminishing returns
  - Just Cause 2 used 4 for clouds, 8 for particle effects

Original	100%
Tight rect	69.23%
3 vertices	70.66%
4 vertices	60.16%
5 vertices	55.60%
6 vertices	53.94%
7 vertices	52.31%
8 vertices	51.90%



Particle trimming

- First attempt: Manual trimming
  - Tedious, but proved the concept
  - OK for our cloud atlas
    - > 2x performance
  - Dozens of atlased particle textures
- Automatic tool [3]
  - Input:
    - Texture and Alpha threshold
    - Vertex count
  - Output:
    - Optimized enclosing polygon

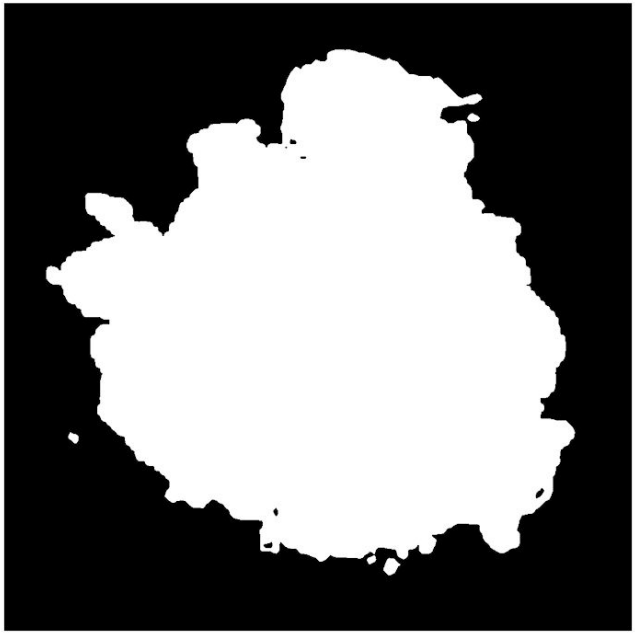


Particle trimming

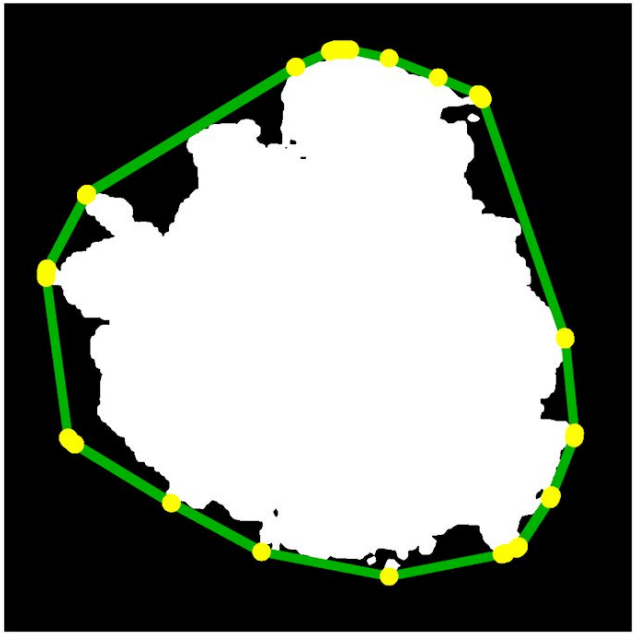
- Algorithm
  - Threshold alpha
  - Add each solid pixel to convex hull
    - Optimize with potential-corner test
  - Reduce hull vertex count
    - Replace least important edge
    - Repeat until max hull vertex count
  - Brute-force through all valid edge permutations
    - Select smallest area polygon

Particle trimming

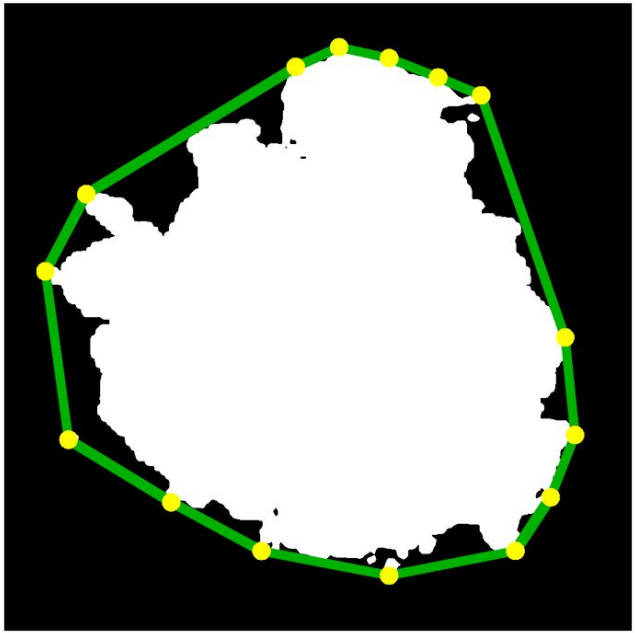
Particle trimming



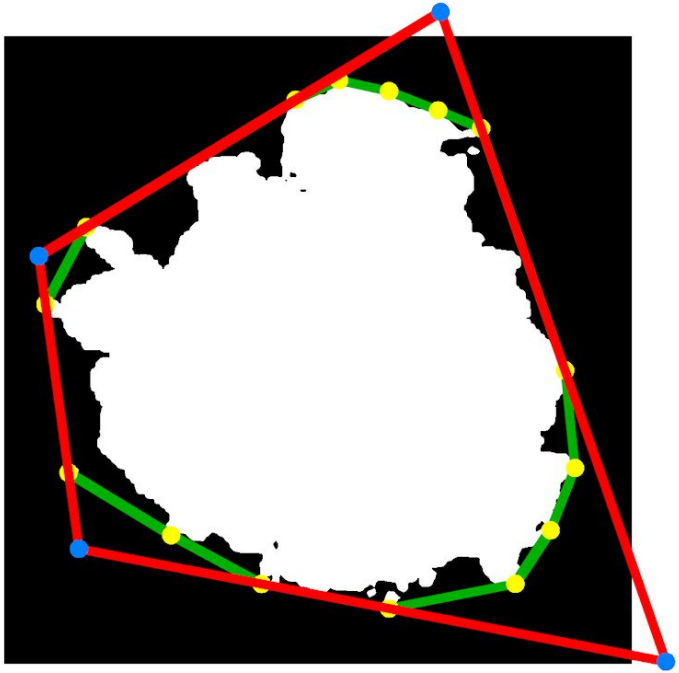
Particle trimming



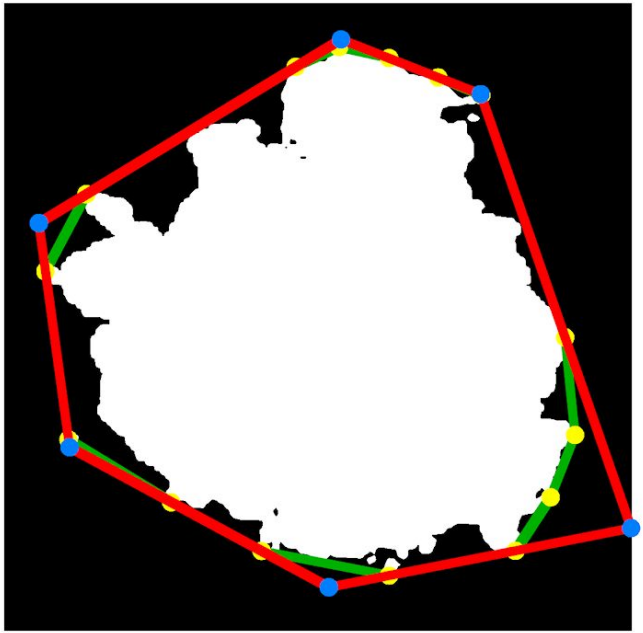
Particle trimming



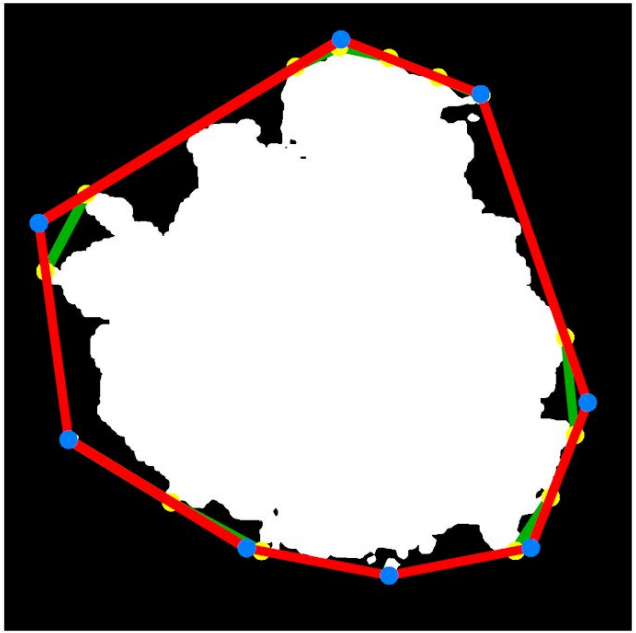
Particle trimming



Particle trimming



Particle trimming



Particle trimming

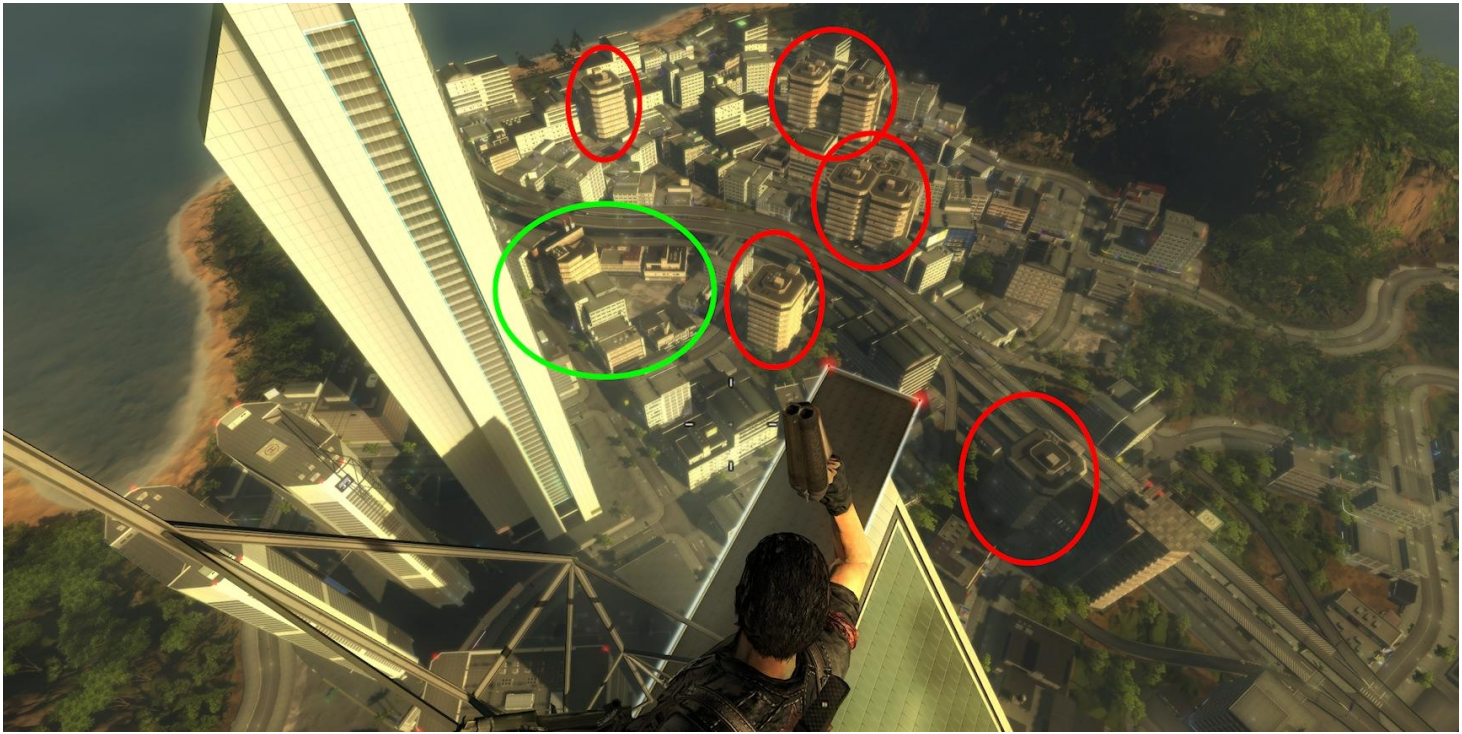
- Issues
  - Polygon extending outside original quad
    - No problem for regular textures. Use CLAMP.
    - May cut into neighboring atlas tiles …
    - Compute all hulls first, reject solutions that intersect another hull.
    - Revert to aligned rect if no valid solution remains
  - Performance
    - Brute-force
    - Keep convex hull vertex count reasonably low
  - Filtering
    - Add all four corners of a pixel (faster), or interpolate subpixel alpha values (accurate)
  - Handling ”weird” textures

Merge-Instancing

Merge-Instancing

- Instancing
  - One mesh, multiple instances
- Merging
  - Multiple meshes, one instance of each
- What about: Multiple meshes, multiple instances of each?
  - Instancing: Multiple draw-calls
  - Merging: Duplication of vertex data
  - Merge-Instancing: One draw-call, no vertex duplication

Merge-Instancing



Merge-Instancing

Instancing

```
for (int instance = 0; instance < instance_count; instance++)
  for (int index = 0; index < index_count; index++)
    VertexShader( VertexBuffer[IndexBuffer[index]], InstanceBuffer[instance] );
```

Merge-Instancing

```
for (int vertex = 0; vertex < vertex_count; vertex++)
{
    int instance = vertex / freq;
    int instance_subindex = vertex % freq;

    int indexbuffer_offset = InstanceBuffer[instance].IndexOffset;
    int index = IndexBuffer[indexbuffer_offset + instance_subindex];

    VertexShader( VertexBuffer[index], InstanceBuffer[instance] );
}
```

Merge-Instancing

- Implemented in Just Cause 2 on Xbox360
  - Draw-calls less of a problem on PS3 / PC
- Xbox360 HW lends itself to this approach
  - No hardware Input Assembly unit
  - Vertex shader does vertex fetching
  - Accessible through inline assembly

Merge-Instancing

- Merging odd sized meshes
  - Choose common frequency
  - Duplicate instance data as needed
  - Pad with degenerate triangles as needed
- Example
  - Mesh0: 39 vertices, Mesh1: 90 vertices
    - Choose frequency = 45
    - Pad Mesh0 with 2 degenerate triangles (6 vertices)
    - Instances[] = {

      (Mesh0, InstanceData[0]),

      (Mesh1, InstanceData[1]),

      (Mesh1 + 45, InstanceData[1]) }

Phone-wire Anti-Aliasing

Phone-wire AA

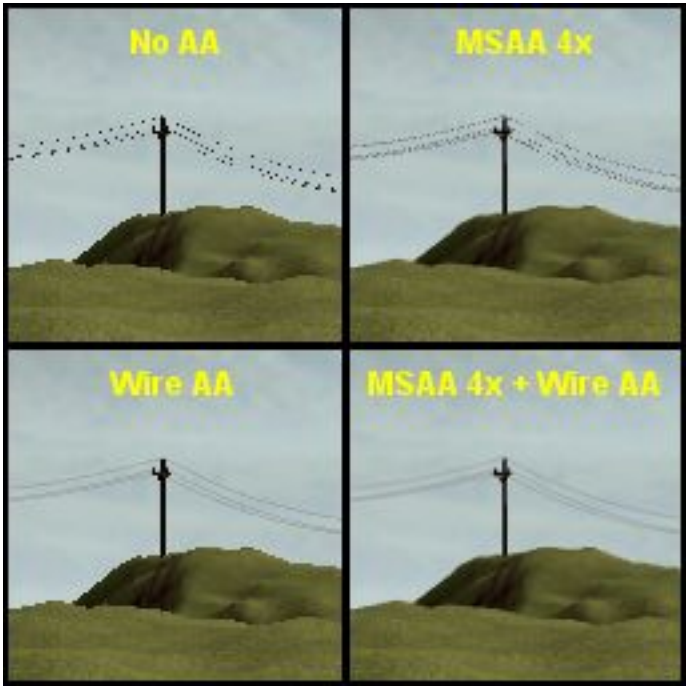
- Sources of aliasing
  - Geometric edges
    - Mostly solved by MSAA
    - Post-AA usually works too
    - Breaks down with thin geometry
  - Shading
    - Sort of solved by mipmapping
    - Poorly researched / understood
    - Few practical techniques for games
    - LEAN mapping [4]

Phone-wire AA

- Sources of aliasing
  - Geometric edges
    - Mostly solved by MSAA
    - Post-AA usually works too
    - **Breaks down with thin geometry**
  - Shading
    - Poorly researched / understood
    - Few practical techniques for games
    - LEAN mapping

Phone-wire AA

- Phone-wires
  - Common game content
  - Often sub-pixel sized
- MSAA helps
  - … but not that much
  - Breaks at sub-sample size
- Idea
  - Let's not be sub-pixel sized!



Phone-wire AA

- Phone-wires
  - Long cylinder shapes
  - Defined by center points, normal and radius
- Avoid going sub-pixel
  - Clamp radius to half-pixel size
  - Fade with radius reduction ratio

Phone-wire AA


```
// Compute view-space w
float w = dot(ViewProj[3], float4(In.Position.xyz, 1.0f));

// Compute what radius a pixel wide wire would have
float pixel_radius = w * PixelScale;

// Clamp radius to pixel size. Fade with reduction in radius vs original.
float radius = max(actual_radius, pixel_radius);
float fade = actual_radius / radius;

// Compute final position
float3 position = In.Position + radius * normalize(In.Normal);
```

- Demo + source available! [5]

Phone-wire AA off, MSAA 4x

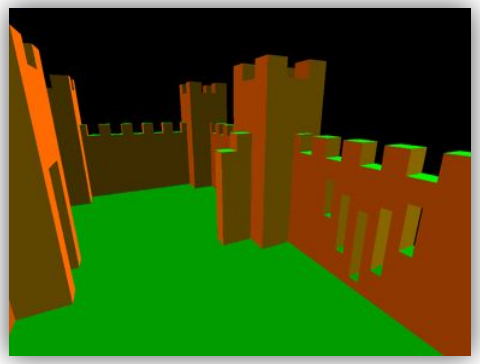
Phone-wire AA on, MSAA 4x

Second-Depth Anti-Aliasing

Second-Depth Anti-Aliasing

- Filtering AA approaches
  - SIGGRAPH 2011 - ”Filtering Approaches for Real-Time Anti-Aliasing” [6]
  - Post-AA
    - MLAA
    - SMAA
    - FXAA
    - DLAA
  - Analytical approaches
    - GPAA
    - GBAA
    - DEAA
    - **SDAA [7]**

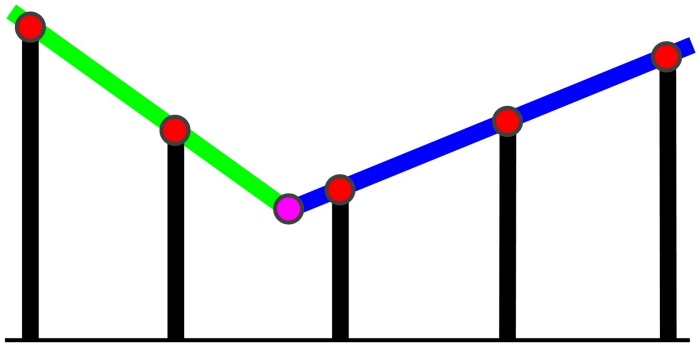
Second-Depth Anti-Aliasing



- Depth buffer and second-depth buffer
- Depth is linear in screen-space
  - Simplifies edge detection
  - Enables prediction of original geometry
- Two types of edges
  - Creases
  - Silhouettes
- Silhouettes require second-depth buffer
  - Do pre-z pass with front-face culling
  - Alternatively, output depth to render target for back-facing geometry

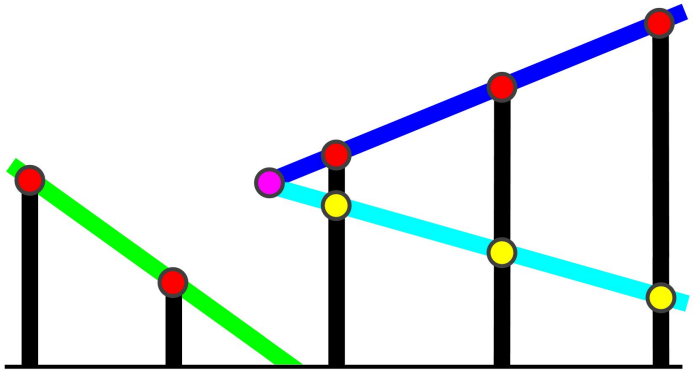
Second-Depth Anti-Aliasing

- Attempt crease case first
- Look at depth slopes
- Compute intersection point
  - Valid if distance < one pixel
  - Used if distance < half pixel
- If invalid, try silhouette



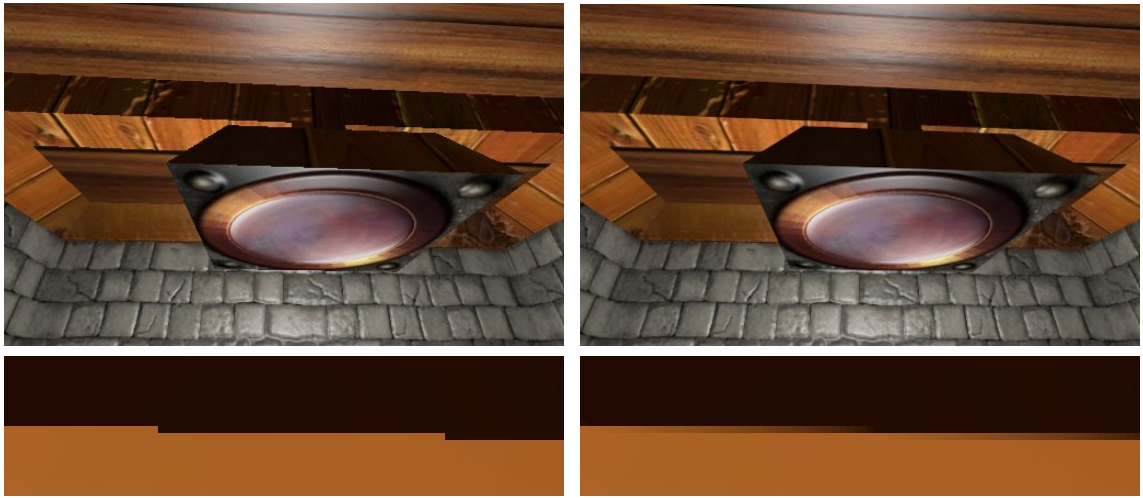
Second-Depth Anti-Aliasing

- Try as silhouette
- Neighbor depths useless
  - Look at second-depths
- Compute intersection point
  - Used if distance < half pixel



Second-Depth Anti-Aliasing

- Results



- Demo + source available! [7]

References

[1] http://en.wikipedia.org/wiki/Comparison_of_AMD_graphics_processing_units
[2] http://www.humus.name/index.php?page=News&ID=266
[3] http://www.humus.name/index.php?page=Cool&ID=8
[4] http://www.csee.umbc.edu/~olano/papers/lean/
[5] http://www.humus.name/index.php?page=3D&ID=89
[6] http://iryoku.com/aacourse/
[7] http://www.humus.name/index.php?page=3D&ID=88

Thank you!

Emil Persson
Avalanche Studios

 @_Humus_