Masked Software Occlusion Culling

Jon Hasselgren
Magnus Andersson
Tomas Akenine-Möller

Intel®
Background

- Potentially Visible Sets
  - Precomputed – very efficient
  - Scene (occluders) must be static
  - Difficult to handle general scenes

Quake 2
Half-Life 2
• Dynamic occlusion culling increasingly popular
  - Modern games have more complex and dynamic worlds
  - Simpler content pipeline, no complex pre-computation

Battlefield 4  Assasin's Creed Unity
Dynamic Occlusion Culling

- **Hardware occlusion queries**
  - GPU is extremely efficient at rasterization
  - Long pipeline delay, takes long to get the result of a query
  - May require sending result back to CPU

- **Software occlusion culling**
  - Short delay, no readback → easier to integrate with scene traversal
  - Software rasterization not as efficient as GPU
Hierarchical Z Buffer (HiZ) [Greene93]

- Rasterize occluders to full resolution z buffer
Hierarchical Z Buffer (HiZ) [Greene93]

- Rasterize occluders to full resolution z buffer
- Create hierarchical z buffer
  - Find the maximum z in each 8x8 tile
Hierarchical Z Buffer (HiZ) [Greene93]

- Rasterize occluders to full resolution z buffer
- Create hierarchical z buffer
  - Find the maximum z in each 8x8 tile
- Perform occlusion queries with hierarchical z buffer
Masked Depth Culling [AHAM15]
Masked Depth Culling [AHAM15]

Hierarchical Z buffer
[Greene93]

Depth buffer

HiZ buffer

Depth buffer

Masked HiZ buffer

Not needed

$Z_{max}^0$  $Z_{max}^1$
Masked Depth Culling [AHAM15]

Hierarchical Z buffer [Greene93]

Depth buffer

HiZ buffer

Masked depth buffer [AHAM15]

Depth buffer

Masked HiZ buffer

Not needed

$Z_{max}^0$ $Z_{max}^1$
Masked Depth Culling [AHAM15]
Masked Software Occlusion Culling

- **Masked Depth Culling [AHAM15]**
  - Was originally intended for graphics hardware
  - Directly update hierarchical z buffer without computing full res z buffer
  - Decouples coverage sampling (rasterization) and depth computation

- **Could it be really fast for software?**
  - Much less memory to read/write than full resolution z buffer
  - Updates use bitmasks, can process 256 pixels in parallel using AVX
Algorithm
Compute Bounding Box

- Padded to 32x8 pixel supertiles
Traverse Supertiles
Traverse Supertiles

AVX register
AVX Register Layout

- One Scanline per SIMD-lane
Edge Slopes

• Compute slopes ($\Delta y/\Delta x$) during triangle setup
  - Similar to regular scanline rasterizers
  - Some precision caveats due to tile size
Compute Break Points

- Compute break point for each scanline
  - Eight scanlines in parallel using AVX
Compute Coverage Mask

- Start with full coverage mask
Compute Coverage Mask

- Start with full coverage mask
  - Shift each lane (scan line) to break point
  - AVX2 and later support per-lane shift
Repeat for Next Edge

- Repeat the same process for next edge
Repeat for Next Edge

- Repeat the same process for next edge
  - Edge is facing right → invert mask

Coverage mask

Breakpoints
Combine Masks

- Combine mask of all overlapping edges

Coverage mask
Combine Masks

- Combine mask of all overlapping edges
  - Using bitwise AND
Resulting Coverage Mask

- Combine mask of all overlapping edges
  - Using bitwise AND
Shuffle mask

- Shuffle mask to form better shaped tiles
  - Before: each SIMD-lane is a scanline
Shuffle mask

- Shuffle mask to form better shaped tiles
  - Before: each SIMD-lane is a scanline
  - After: each SIMD-lane is a 8x4 tile
Masked Depth Buffer Update

- Masked z update similar to previous work [AHAM15]
  - Optimized for AVX and software implementation
  - Less accurate than original, more dependent on render order
  - Easier to control render order than for HW culling

- Input for an 8x4 tile
  - Tri: Coverage mask (32b) + Zmax value (32b float)
  - Buffer: Selection mask (32b) + 2 Zmax values (2x32b float)
Masked Depth Buffer Update

- \( Z_{\text{max}}^1 \) is the working layer
  - Updated as: \( \max(Z_{\text{max}}^0, Z_{\text{tri}}^1) \)
  - Mask is updated using bitwise or

- \( Z_{\text{max}}^0 \) is the reference layer
  - Whenever working layer mask is full, overwrite reference layer
  - Clear working layer

Rasterized triangle: \( Z_{\text{tri}}^1 = - (0.5) \)
Buffer entry: \( Z_{\text{max}}^0 = - (0.9) \), \( Z_{\text{max}}^1 = - (0.75) \)
Updated Buffer entry: \( Z_{\text{max}}^0 = - (0.9) \), \( Z_{\text{max}}^1 = - (0.75) \)
Update Heuristic Results

- Silhouettes can leak through geometry
  - Reason: partial working layer contaminates foreground layer, which would otherwise completely overwrite the tile
Revised Update

• Discard working layer if drawing a new object
  - Throw away partial data avoid contaminating layers in front
  - How to know if we begin drawing a new object?

• Discard heuristic
  - If $Z_{\text{max}} - Z_{\text{max}}^{\text{tri}} > Z_{\text{max}}^0 - Z_{\text{max}}^1$, discard working layer
  - Avoids fixed threshold value

![Rasterized triangle](image1)

![Buffer entry](image2)

![Updated Buffer entry](image3)
Update Heuristic Results

No discard

Discard heuristic
Results
Results

Intel Occlusion Culling Sample

- Integrated in Intel occlusion culling sample
  - Uses low-poly occluder meshes
  - Two pass occlusion culling (rasterize occluders, perform queries)
  - Contains an AVX2 optimized version of the HiZ algorithm
  - Integrated our algorithm making minimal changes
Results
Intel Occlusion Culling Sample

- Algorithm timing breakdown
  - Clear: Clearing the depth buffer
  - Geom: Transform & project geometry
  - Rast: Triangle setup & occluder rasterization
  - Gen: Compute hierarchical z buffer from full resolution z buffer
  - Test: Perform occlusion queries

<table>
<thead>
<tr>
<th></th>
<th>Clear</th>
<th>Geom</th>
<th>Rast</th>
<th>Gen</th>
<th>Test</th>
<th>Total</th>
</tr>
</thead>
<tbody>
<tr>
<td>HiZ</td>
<td>377</td>
<td>196</td>
<td>2145</td>
<td>509</td>
<td>278</td>
<td>3505</td>
</tr>
<tr>
<td>Mask</td>
<td>23</td>
<td>194</td>
<td>584</td>
<td>0</td>
<td>255</td>
<td>1056</td>
</tr>
</tbody>
</table>

16x 3.7x
Results
Intel Occlusion Culling Sample

- Performance comparison for camera animation

![Image of performance comparison graph]

- Total Frame Time (CPU & GPU)

- Time (ms)

- Frame Number

- Colors: Red - Frustum, Cyan - HiZ, Black - Mask
Results

Standalone framework

- Standalone engine tailored for our algorithm
  - One pass: interleaving occluder rasterization and occlusion queries
  - Early exit: don't perform occlusion culling in occluded regions
  - Modified version of the hierarchical z buffer (HiZ) algorithm

MPI Informatics Building
Mesh: 72 MTris, Occluder: 143 KTris

Rungholt
Mesh: 7 MTris, Occluder: 7 MTris
Results

MPI Informatics Building

- Performance during camera flythrough
Rungholt

• Live demo
Conclusion

- **Efficient algorithm for rasterizing occlusion buffers**
  - More than 3x better performance than previous work
  - Can be integrated tightly with traversal algorithm (low latency)
  - Very accurate culls 98% of all triangles culled by hierarchical z buffer

- **Future work**
  - Efficient multi-threading
  - Better update heuristics for masked z buffer
  - GPU implementation
Thank you

• Source code available
  - www.github.com/GameTechDev/MaskedOcclusionCulling

• Questions