SVGPU: Real Time 3D Rendering to Vector Graphics Formats

APOLLO I. ELLIS AND WARREN HUNT AND JOHN C. HART
UIUC OCULUS UIUC
AEILLIS2@ILLINOIS.EDU, WARREN.HUNT@GMAIL.COM, JCH@ILLINOIS.EDU
Intro: What if we could output triangles instead of pixels?

- Benefits of vector graphics representations
  - Vertex correspondences
  - NPR effects
  - Antialiasing
  - Resolution independent compression
- SVGPU (Scalable Vector GPU)
  - Outputs a planar map in real time
  - Currently built in Cuda
SVGPU is simple and fast

- Spatial hashing extracts silhouettes
- Clipping clips triangles to silhouettes
- Occlusion discards hidden surfaces
Previous approaches still short of real time

- Offline systems have been proposed
- Largely CPU programs focusing on quality [Winkenbach and Salesin][Stroila et al.][Karsch and Hart][Eisemann et al.]
- The GPU has been used for fast analytic visibility [Auzinger et al.]
- No one has achieved real time
- We build on Robert’s Algorithm [Roberts, L. 1963]
- Yes, from the 60’s!
First we need to extract silhouettes

- Hash all edges into a spatial hash table
- Minimal Collisions < 5
  - Rarely more than 2
- Second pass uses active keys checking for front-back facing pairs
- If a pair is found we bin the edge
- Reasonably quick, < 2ms
Insight: When triangles are clipped to silhouette edges, the complexity of clipping is reduced.
Clipping: There are basically 3 cases

- The side of the edge that is occluded changes, actually making 6 cases.
Clipping: After round 1 we have three triangles

- Clipping the original triangle creates A, B, and C
Clipping: In round 2 we generate 6 new triangles

- B and C are clipped against next edge
Clipping: Round 3 generates 3 new triangles

- Only one triangle is clipped against the last edge
Clipping: Result, 9 triangles
Insight: Silhouette clipping completely resolves partial visibility cases.

- Triangles are either fully occluded or fully visible
- Suffices to check overlap in X Y and “is closer”
- Compute hyper-plane separation test for overlap
- Then Z test the occludee centroid against the occluder
- Do see why this Z test will holds?
Z Centroid will eventually be occluded
Initial results were sub real time

<table>
<thead>
<tr>
<th>Model</th>
<th>in → out</th>
<th>bins (non-∅)</th>
<th>max</th>
<th>ave. $\frac{\Delta}{\text{bin}}$</th>
<th>t</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bunny</td>
<td>69K → 280K</td>
<td>2,304 (33%)</td>
<td>698</td>
<td>139</td>
<td>41</td>
</tr>
<tr>
<td>Dragon</td>
<td>202K → 950K</td>
<td>2,304 (19%)</td>
<td>1,740</td>
<td>555</td>
<td>222</td>
</tr>
<tr>
<td>Armadillo</td>
<td>212K → 1.2M</td>
<td>1,152 (44%)</td>
<td>2,076</td>
<td>356</td>
<td>256</td>
</tr>
<tr>
<td>Buddha</td>
<td>293K → 1M</td>
<td>2,304 (43%)</td>
<td>712</td>
<td>200</td>
<td>140</td>
</tr>
</tbody>
</table>
Bin Tuning and Re-binning

- Bin sizes help and hurt different phases
- Course binning is good at first (64 bins)
- Occlusion does better with more (2048)
- The cost to change bin resolution (Rebinning) is small.
- ~2ms re-bin for a 14ms improvement.
We are still tuning and improving!

- Currently 75Hz for Bunny
- 30Hz for Armadillo (slowest scene)
- 9X performance over state of the art!
- Optimizing memory usage (e.g. empty bin storage)
- Reducing tessellation
- Applications (Compression, Antialiasing...)
Questions?
References