Progressive Volume Rendering of Unstructured Grids on Modern GPUs

Steven P. Callahan¹, Louis Bavoil¹, Valerio Pascucci², and Cláudio T. Silva¹

¹ SCI Institute, University of Utah
² Lawrence Livermore National Laboratory
Progressive Volume Rendering

3% 0.01 sec
33% 7 sec
66% 18 sec
100% 34 sec
Objective

- Interactively explore large data sets through direct volume rendering
  - Show intermediate results
  - Reuse intermediate results
  - Allow user interrupt
  - Only render pertinent data
  - Use both CPU and GPU
Issues

• Tetrahedra are not natively supported
  – Projected Tetrahedra
    • [Shirley and Tuchman ‘90, Wiley et al. ‘02]

• Compositing requires strict order
  – Visibility Sorting
    • [Williams et al. ‘92]
  – Ray Casting
    • [Bunyk et al. ‘97, Weiler et al. ‘03]
  – Hybrids
    • [Farias et al. ‘00, Callahan et al. ‘05]
Issues

- Hierarchical level-of-detail not suitable
  - Regular Sampling
    - [Leven et al. 2002]
  - Geometry Simplification
    - [Cignoni et al. 2005]
  - LOD without hierarchies
    - [Callahan et al. 2005]
Background

- Hardware-Assisted Visibility Sorting
  - Sort in both object-space and image-space

[Callahan et al. 2005]
http://havs.sourceforge.net
## Background

- **The $k$-buffer**
  - Fragment stream sorter
  - Fixed size A-buffer

<table>
<thead>
<tr>
<th>Input</th>
<th>1</th>
<th>3</th>
<th>2</th>
<th>4</th>
<th>6</th>
<th>5</th>
</tr>
</thead>
<tbody>
<tr>
<td>Output</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>6</td>
</tr>
</tbody>
</table>
### Background

- **k-buffer implementation**
  - OpenGL with MRTs and FBOs

<table>
<thead>
<tr>
<th></th>
<th>R</th>
<th>G</th>
<th>B</th>
<th>A</th>
</tr>
</thead>
<tbody>
<tr>
<td>$T_{result}$</td>
<td>$R_{result}$</td>
<td>$G_{result}$</td>
<td>$B_{result}$</td>
<td>$A_{result}$</td>
</tr>
<tr>
<td>$T_{k12}$</td>
<td>$v_1$</td>
<td>$d_1$</td>
<td>$v_2$</td>
<td>$d_2$</td>
</tr>
<tr>
<td>$T_{k34}$</td>
<td>$v_3$</td>
<td>$d_3$</td>
<td>$v_4$</td>
<td>$d_4$</td>
</tr>
<tr>
<td>$T_{k56}$</td>
<td>$v_5$</td>
<td>$d_5$</td>
<td>$v_6$</td>
<td>$d_6$</td>
</tr>
</tbody>
</table>
Background

- Dynamic Level-of-Detail

2.0 fps  5.3 fps  10.0 fps  16.1 fps

[Callahan et al. 2005]
Overview

- Preprocess
- Interactive Mode
- Progressive Mode
- Completed Mode
Preprocess

- **HAVS with LOD**
  - Extract unique triangles
  - Separate boundary triangles
  - Compute triangle centroids
  - Build lookup table
  - Initialize $k$-buffer textures

- **Progressive**
  - Build octree
  - Initialize progressive textures
Preprocess

- **Textures**

<table>
<thead>
<tr>
<th></th>
<th>R</th>
<th>G</th>
<th>B</th>
<th>A</th>
</tr>
</thead>
<tbody>
<tr>
<td>$T_{pro}$</td>
<td>$R_{pro}$</td>
<td>$G_{pro}$</td>
<td>$B_{pro}$</td>
<td>$A_{pro}$</td>
</tr>
<tr>
<td>$T_{k12}$</td>
<td>$v_1$</td>
<td>$d_1$</td>
<td>$v_2$</td>
<td>$d_2$</td>
</tr>
<tr>
<td>$T_{k34}$</td>
<td>$v_3$</td>
<td>$d_3$</td>
<td>$v_4$</td>
<td>$d_4$</td>
</tr>
<tr>
<td>$T_{k56}$</td>
<td>$v_5$</td>
<td>$d_5$</td>
<td>$v_6$</td>
<td>$d_6$</td>
</tr>
<tr>
<td>$T_{approx}$</td>
<td>$R_{approx}$</td>
<td>$G_{approx}$</td>
<td>$B_{approx}$</td>
<td>$A_{approx}$</td>
</tr>
</tbody>
</table>
Overview

- Preprocess
- Interactive Mode
- Progressive Mode
- Completed Mode
Interactive Mode

- Draw the boundary
  - Depth peel front and render into $T_{k12}$
  - Depth peel back and render into $T_{k12}$
  - Composite gap between front and back entries into $T_{\text{approx}}$
- Display $T_{\text{approx}}$
Interactive Mode

• Why depth peeling?
Overview

- Preprocess
- Interactive Mode
- Progressive Mode
- Completed Mode
Progressive Mode

- Render a depth range of the geometry

$Z_{\text{begin}}$  

$Z_{\text{end}}$
Progressive Mode

- CPU
  - Get subset of geometry from octree
    - Calculate min/max ranges for nodes
    - Cull geometry based on min/max and view frustum
  - Sort geometry by centroid
  - Send geometry to GPU for rasterization
Progressive Mode

- GPU
  - Bind $T_{pro}$, $T_{k12}$, $T_{k34}$, $T_{k56}$
  - Render subset into $T_{pro}$ using $k$-buffer

Complete Volume Rendering

Fragments In Buffer

$k$-Buffer ($T_{k12}, T_{k34}, T_{k56}$)

Progressive Buffer ($T_{pro}$)
Progressive Mode

- GPU continued...
  - Bind $T_{approx}$, $T_{k12}$, $T_{k34}$, and $T_{k56}$
  - Approximate empty space by finding front and back fragments in $k$-buffer to store in $T_{approx}$
Progressive Mode

• GPU continued…
  – Bind $T_{pro}$ and $T_{approx}$
  – Composite $T_{pro}$ into $T_{approx}$
  – Display $T_{approx}$

$\text{Approximate} = \text{Complete} + \text{Approximate}$
Overview

- Preprocess
- Interactive Mode
- Progressive Mode
- Completed Mode
Completed Mode

- Flush $k$-buffer
  - Bind $T_{pro}$, $T_{k12}$, $T_{k34}$, and $T_{k56}$
  - Draw $k$-1 screen-aligned planes and composite valid entries into $T_{pro}$
  - Display $T_{pro}$
  - Store $T_{pro}$ for future browsing
Results
Results
Considerations

- **Stream Size**
  - Large:
    - Faster overall
    - Less progressive steps
  - Small:
    - Slower overall
    - More progressive steps

- **Remote rendering**
  - Thin client architecture
Conclusion

- Efficiently uses CPU and GPU
- Progressions increase in quality
- Converges to full-quality renderings
- Allows interactive exploration of large datasets
- Future Work:
  - Transfer functions
  - Cutting planes
  - Time Varying data
Acknowledgments

- Carlos Scheidegger, Huy Vo
- Datasets
  - Neely and Batina (NASA)
  - O’Hallaran and Shewchuck (CMU)
- Funding
  - DOE
  - IBM
  - SNL
  - LLNL
  - ARO
  - University of Utah