Efficient View-Dependent Out-of-Core Visualization

Michael Guthe, Pavel Borodin
and Reinhard Klein

University of Bonn
Computer Graphics Group

Introduction

- Why LOD? Why out-of-core?

372M Triangles (12.5GB) 13M Triangles (500MB)
Related Work

- Point Based Rendering

Problem:
- Too many primitives
  ⇒ GPU overload
Related Work

View-dependent LOD


Problem:

- Vertex based LOD selection & culling
  - CPU overload
  - Graphics hardware optimized for static geometry
Related Work

- Hierarchical LOD
  - C. Erikson, D. Manocha, *HLODs for Faster Display of large static and dynamic environments*, 2000
  - G. Varadhan, D. Manocha, *Out-of-Core Rendering of Massive Geometric Environments*, 2002

- Adjustable Balance: CPU ⇔ GPU
- Unnecessary triangles at subdivisions
  - Hierarchical simplification may run out of memory
  - Frame rate dependent on input model size
- Simplification not error guided
  - More drastic simplification possible
Contributions

- Solution for crack problem
  - General applicable
  - Minimal overhead
  - Framerate independent on input model size
- Error guided simplification
  - All possible collapses performed
  - Better simplification at same quality
- Additional optimizations
  - Improved caching and prefetching

Out-of-core HLOD rendering

1. Segmentation → Preprocessing
2. Simplification
3. HLOD Selection & Culling → Runtime
4. Rendering
Segmentation and Simplification

- Subdivide complex objects using an octree
- Quantization of geometry to $3n$ bits
  - Max. size of projected node into image space at most $2^n \times 2^n$ pixel ($\frac{1}{2}$ pixel screen space error)
- Simplify geometry within nodes independently
  - Avoid cracks within node

Subdivision

- High frequencies along subdivisions prevent efficient simplification
  - Cut triangles along octree cell boundaries
Hierarchical Simplification

- Out-of-Core Simplification
  ⇒ Bottom up approach
- Guaranteed screen space error
  ⇒ Control Hausdorff-error between original model and simplified node geometry
- Problem: impossible to compare to original geometry due to model size
  ⇒ Accumulate Hausdorff-errors during hierarchical simplification

Accumulation of errors

- Hausdorff-error is fixed for each level
  - Fixed ratio: node size / Hausdorff-error
  - Doublicates with each level
- Compare current level to model at two levels below (¼ Hausdorff-error)
- Accumulate errors
  - $\varepsilon + \frac{1}{4} \varepsilon + \frac{1}{16} \varepsilon + \frac{1}{64} \varepsilon + \ldots = \frac{4}{3} \varepsilon$
Hierarchical Simplification

- Cracks from independent simplification not closed by vertex-pair contractions
- Either stitching or „better“ simplifier

Hierarchical Simplification

- Generalized Pair Contractions to close cracks during simplification
- Three new operations:
  - vertex-edge contraction
  - vertex-triangle contraction
  - edge-edge contraction
- Sufficient to connect the closest points of two objects

Geometry Compression

- Quantization of vertex positions to \( n \) bit per coordinate
  - Additional error: node size / \( 2^{n+1} \)
  - E.g. node size / Hausdorff-error = 256
    - \( \frac{1}{2} \) pixel screen space error \( \Rightarrow \) node max. 128x128 pixel
    - 8 bit per coordinate \( \Rightarrow \) \( \frac{1}{4} \) pixel additional error
  - Quantization error less than \( \frac{1}{4} \) pixel \( \Rightarrow \) not visible

- Quantized normals

Connectivity Compression

- For fast decompression:
  - Indexed triangle strips
  - Indexed boundary loops
  - Huffman coding

- In principle any other compression scheme can be used (e.g. Cut-Border)
**Rendering**

**Problem:**
- Cracks from independent simplification of nodes are visible

⇒ Extrude geometry along cuts to close cracks

---

**Crack Filling**

\[ \varepsilon_a + \varepsilon_b \leq 1 \text{pixel} \]
\[ \varepsilon_a \leq \frac{1}{2} \text{pixel} \]
\[ \varepsilon_b \leq \frac{1}{2} \text{pixel} \]
Motion Estimation

- Standard technique used for prefetching
  - Priority depends on necessary resolution change

- Modified priority:
  - Likelihood of a node to be rendered in the next frame

\[ p = \begin{cases} 
  p_{\text{det} \text{ail}} : & \text{visible} \\
  p_{\text{det} \text{ail}} \cdot \cos \alpha : & \text{culled}
\end{cases} \]

\[ \alpha = \max (\gamma, \theta) \]
## Results

### Comparison with previous out-of-core HLOD rendering

<table>
<thead>
<tr>
<th>Model</th>
<th>#Triangles</th>
<th>Constr. simpl. avg. (min.)</th>
<th>Fat Borders avg. (min.)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Armadillo</td>
<td>345,944</td>
<td>123 (72) fps</td>
<td>128 (71) fps</td>
</tr>
<tr>
<td>Dragon</td>
<td>871,414</td>
<td>114 (80) fps</td>
<td>122 (86) fps</td>
</tr>
<tr>
<td>Happy Buddha</td>
<td>1,087,716</td>
<td>101 (62) fps</td>
<td>117 (69) fps</td>
</tr>
<tr>
<td>David 2mm</td>
<td>8,254,150</td>
<td>80 (49) fps</td>
<td>155 (73) fps</td>
</tr>
<tr>
<td>Lucy</td>
<td>28,055,742</td>
<td>29 (8) fps</td>
<td>110 (51) fps</td>
</tr>
<tr>
<td>David 1mm</td>
<td>56,230,343</td>
<td>13 (1) fps</td>
<td>114 (61) fps</td>
</tr>
<tr>
<td>St. Matthew</td>
<td>372,422,615</td>
<td>5 (1) fps</td>
<td>93 (43) fps</td>
</tr>
</tbody>
</table>

Rendering performance on an Athlon 2800+ with ATI Radeon 9800Pro

1. Non-reference implementation of Varadhan's HLOD rendering algorithm
Acknowledgements

- The models used in this talk were provided the UNC Walkthrough Group and the Digital Michelangelo Project.
- This work was funded by the BMBF under the project OpenSG PLUS and by the European Union under the project of RealReflect.

http://cg.cs.uni-bonn.de/project-pages/OpenSG-Plus