Progressive Buffers: View-dependent Geometry and Texture LOD Rendering
Pedro V. Sander, ATI Research
Jason L. Mitchell, Valve Software
International Conference on Computer Graphics and Interactive Techniques, 2005

Sushrut Pavanaskar
21st April 2009

Motivation

Outline
- Background
  - Level of Detail
- Research Concepts
  - Data structure
  - Algorithms
  - Rendering
  - Pre-processing
- Summary & Remarks

Background
- Definitions / Acronyms
  - Level of Detail (LOD)
  - Geomorphing
  - Out-of-core (OOC)
  - Mip-mapping
- More about LOD

Level of Detail (LOD)
- Changing complexity of an object/model based on distance from camera or other metrics

Level of Detail (LOD)
- Depends on
  - Distance from the camera (eye)
  - Importance, speed of camera etc.
- Why?
  - Speedy, efficient, visually-pleasing rendering
  - To avoid tiny, sub-pixel sized polygons
Changing LOD

Polygons: 4500 3500 3500 2500 2500

LODs and New Problems

- Smooth transition of LODs
- Not so smooth
- Sudden change in LODs

“Our eyes are very good at seeing sharp changes but not very good at seeing smooth changes. If there is a way to smooth a hard transition you can get away with a lot more than if you didn’t.”

Brian Karis
Lead Graphics Programmer
Human Head Studios

LOD Pop

LODs with Morphing

Outline

- Background
  - Level of Detail
- Research Concepts
  - Data structure
  - Algorithms & related
  - Rendering
  - Pre-processing
- Summary & Remarks

Progressive Buffer Data Struct.

- Progressive Buffer: Series of static buffers
  - One static buffer ≡ one LOD
- Two vertex buffers
- One index buffer
**Outline**
- Background
  - Level of Detail
- Research Concepts
  - Data structure
  - Algorithms
  - Rendering
  - Pre-processing
- Summary & Remarks

**Algorithms**
- Continuous & View dependent LOD
- Geomorphing
- Cluster formation
- Coarse Buffer Hierarchy
- Out of core loading

**Continuous LOD**
- LERP(static buffers, weight)
- Weights depending on distance
- \( \text{Coarse}_1 = \text{Fine}_{i-1} \)
- With distance increasing
  - Weight of coarse buffer increases
  - \( \text{Fine}_3 \rightarrow \text{Coarse}_3 \rightarrow \text{Fine}_2 \rightarrow \text{Coarse}_2 \)
  - \( (\text{Coarse}_3 = \text{Fine}_2) \)

**Need of View-dependent LOD**
- Continuous LOD good for static scene
  - Entire mesh rendered with
    - Same geomorphic weight
    - Same fine and coarse (static) buffers
- For dynamism, view dependent LOD
  - Different regions of mesh with different LOD

**Different LODs in single mesh**

**View Dependent LOD**
- Divide mesh into clusters
  - Clusters can have different LODs
- To avoid cracks at boundary vertices
  - Ensure consistent positions at all LOD
  - Same geomorphing weights across clusters
  - **Per vertex computation (GPU)**
- Assign view-dependent LODs to clusters
  - Depending on distance of cluster center to eye
Cracks

Algorithms
- Continuous & View dependent LOD
- Geomorphing
- Cluster formation
- Coarse Buffer Hierarchy
- Out of core loading

Geomorph Zone
- Geomorph zone is away from boundary by $r$
- Ensures all vertices have geomorphed

Vertex LOD Distribution
- Geomorphing zone away from boundary
- Consistent $P_{B_i}$ (static buffer) at boundary

Clusters
- Scene segmented into clusters
- Progressive buffer for each cluster
- Same static buffer inside a cluster
  - Depends on bounding sphere center distance

Algorithms
- Continuous & View dependent LOD
- Geomorphing
- Cluster formation
- Coarse Buffer Hierarchy
- Out of core loading
Segmentation
- Voxelization of space for clustering
  - Voxels further divided to **charts**
    - Planar regions, homeomorphic to discs
    - Parameterized (L² metric)
  - "Packed" into one ATLAS per cluster

Chartification (Re)Building

Algorithms
- Continuous & View dependent LOD
- Geomorphing
- Cluster formation
- Coarse Buffer Hierarchy
- Out of core loading

Coarse Buffer Hierarchy (CBH)
- CBH built with clusters as leaves
- Bottom up greedy merging based on radius
  - Indicates size & distance based LOD as criteria
Coarse Buffer Hierarchy (CBH)

- CBH built with clusters as leaves
- Bottom up greedy merging based on radius
  - Indicates size & distance based LOD as criteria
- Rendering engine uses CBH
- Minimized draw() calls

Out of Core (Pre-fetching)

- Many textures/buffers are too large
- Systematic loading-unloading required
  - 4 priority levels, based on Least Recently Used
- By tracking LOD of cluster sphere centers
- Auto-deduced thresholds for memory

<table>
<thead>
<tr>
<th>Priority</th>
<th>Sparse rendering</th>
<th>Video memory</th>
<th>Sample overhead</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Yes</td>
<td>Yes</td>
<td>Low</td>
</tr>
<tr>
<td>1</td>
<td>Yes</td>
<td>Yes</td>
<td>Medium</td>
</tr>
<tr>
<td>2</td>
<td>No</td>
<td>Yes</td>
<td>High</td>
</tr>
<tr>
<td>3</td>
<td>No</td>
<td>Yes</td>
<td>Medium</td>
</tr>
<tr>
<td>4</td>
<td>No</td>
<td>No</td>
<td>Low</td>
</tr>
</tbody>
</table>

Texture LOD

- Each mip level of texture ≡ static buffer
- Single texture used
- OOC Loading/unloading from/to disk

Texture LOD table:
- RAM: 1GB VRAM: 256MB

Texture LOD details:
- Each mip level of texture ≡ static buffer
- Single texture used
- OOC Loading/unloading from/to disk

Outline

- Background
  - Level of Detail
- Research Concepts
  - Data structure
  - Algorithms & related
  - Rendering
  - Pre-processing
- Summary & Remarks

Rendering

- Compute LOD, based on clusters
- Basic rendering
- CBH rendering for efficiency
- Controlling / (Auto) tuning LOD

LOD Computation

- Calculated cluster-wise at runtime
- Based on cluster center distance to camera
- Maintains constant triangle size on screen
  - Thus maintains constant screen space area
- Distance 2x ≡ Area 1/4th ≡ Next LOD
- Successive LODs have 1/4th vertices
- Same factor convenient for mipmapping
- Thus, vertex and texture LOD in unison
LOD
- $s$ is bias
- $k$ is scaling
- $d$ is distance from camera

Basic Rendering
- Traverses through all clusters
- Tests bounding sphere against frustum
  - If passes visibility test, computes LOD
  - Computation aided by preloading buffers
- Vertex shader (thematic):
  - $d = \text{length} (p1 - \text{eye})$
  - $w = \text{smoothstep} (\text{geomorphing zone limits}, d)$
  - $\text{Pos} = \text{LERP} (p1, p2, w)$

Basic Rendering (Continued)
- Normals, textures coordinates geomorphed
  - Same parameterization helps
  - Normals to be re-normalized
  - Geomorphing done on shader (GPU)
    - Two texture fetches
    - Multiplied by geomorphic weight

Controlling LOD
- For portability, stability on all GPUs
- Varying $k$ to adjust LOD (until $k > r + \epsilon$)
- Then, $s$ to be reduced, even $< 0$
  - Support for older/cheaper cards

Parameter Tuning
- Whether LOD tuning required for a state
  - Either decided by frames per second (fps)
    - If $> 65$, increase
    - If $< 55$, decrease
      - But, fps not always a good upper bound
  - Or upper bounds on the memories
    - Exceeding memory levels causes fps crash
    - Video memory, system memory, triangle count
  - All bounds set during initialization

Outline
- Background
- Level of Detail
- Research Concepts
  - Data structure
  - Algorithms & related
  - Rendering
  - Pre-processing
- Summary & Remarks
Pre-processing
- Triangle mesh → Progressive buffer(s)
  - Segmentation
  - Hierarchy construction
  - LOD compliant parameterization
  - Progressive mesh creation
  - Vertex and index buffer creation
  - Texture sampling

Progressive Mesh (Hoppe 96)
- Edge collapsing / Vertex merging
- Successive collapsing with some metrics

Progressive Mesh Results
- 40000 pixel “baboon” to 400 node mesh

Progressive Meshes
- 13546 faces to 150 faces in 3 steps
Progressive Mesh Creation

- Simplification of mesh
  - Half-edge collapse
  - Memory-less appearance preserving metric
- Clusters simplified in unison
- Boundaries simplified equally, so no cracks

Geomorphing

- Weighted interpolation
- From Stage 2 to Stage 3
- 1000 to 500 faces

<table>
<thead>
<tr>
<th>w</th>
<th>0</th>
<th>0.25</th>
<th>0.50</th>
<th>0.75</th>
<th>1</th>
</tr>
</thead>
</table>

Vertex and Index Buffers

- By extracting meshes at various LODs
- Each LOD has 1/4th vertices of previous
- Vertex buffer contains info about parents
  - i.e. index buffer

Vertex Buffer Compression

- Since we require 2X space (fine/coarse)
  - Optimized/compressed buffer
  - Precision reduced for normals, texture coords
  - Reduced precision NOT OK for positions
  - So stored as \( p/r \)
    - \( p \) = position, \( r \) = cluster sphere radius
  (How does it help: try storing 1.0000000000006)

Outline

- Background
- Level of Detail
- Research Concepts
  - Data structure
  - Algorithms & related
  - Rendering
  - Pre-processing
- Summary & Remarks
Results

- Instanced geometry
- 45 million
- 240 million to <1 million

Results

- Fixed LOD band size against flexible
- When using flexible (automatic control)
  - Lesser number of triangles, consistent fps
  - Lesser memory footprint
  - Thus optimal performance

Results

- Continuous LOD control
- Special attention to avoiding cracks, pops
- Portable to most graphic cards (NV/ATI)
- Offers optimal performance, auto tuned
- First system to provide
  - Out-of-core view-dependent rendering
  - Arbitrary meshes and textures
  - Real time graphics, 3D games

Contributions

- Use in deformable models
- Fly-path look-ahead
- Reducing memory usage
- Use in streaming systems (video games)
- Use with a template “tile” etc.

Future Work

- Use in deformable models
- Fly-path look-ahead
- Reducing memory usage
- Use in streaming systems (video games)
- Use with a template “tile” etc.
Credits and Sources

- http://www.cs.unc.edu/~penry/HYLOD/
- http://files.vrijhefte.org/Members/wiebe/jargon.pdf