Point-Based Rendering of Large 3D Models

Szymon Rusinkiewicz  
Princeton University

Marc Levoy  
Stanford University
Motivation

• 3D scanning makes it possible to capture large, detailed models of real-world objects
Motivation

- Models may be dense
  - Hundreds of millions of samples
  - Can’t decimate without losing detail
Goals

- An interactive viewer for large models (10^8 – 10^9 samples)
- Fast startup and progressive loading
- Maintains interactive frame rate
- Compact data structure
- Fast preprocessing
Sample Renderings of a 127-million-sample Model

Interactive (8 frames/sec)

High quality (8 sec)
Previous Systems for Rendering Large Models

- Level of detail control in architectural walkthrough, terrain rendering systems [Funkhouser 93, Duchaineau 97]
- Progressive meshes [Hoppe 96, Hoppe 97]
- These systems often have expensive data structures or high preprocessing costs
Outline

• Data structure: bounding sphere hierarchy
• Rendering algorithm: traverse tree and splat
• Point rendering: when is it appropriate?
QSplat Data Structure

- Key observation: a single bounding sphere hierarchy can be used for
  - Hierarchical frustum and backface culling
  - Level of detail control
  - Splat rendering [Westover 89]
Creating the Data Structure

- Start with a triangle mesh produced by aligning and integrating scans [Curless 96]
Creating the Data Structure

- Place a sphere at each node, large enough to touch neighbor spheres
Creating the Data Structure

• Build up hierarchy
## QSplat Node Structure

<table>
<thead>
<tr>
<th></th>
<th>Position and Radius</th>
<th>Tree Structure</th>
<th>Normal</th>
<th>Width of Cone of Normals</th>
<th>Color (Optional)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bits</td>
<td>13</td>
<td>3</td>
<td>14</td>
<td>2</td>
<td>16</td>
</tr>
</tbody>
</table>

Total: 6 bytes
QSplat Node Structure

- Position and radius encoded relative to parent node
  - Hierarchical coding vs. delta coding along a path for vertex positions
QSplat Node Structure

Position and Radius  Tree Structure  Normal  Width of Cone of Normals  Color (Optional)

13 bits  3 bits  14 bits  2 bits  16 bits

Uncompressed
## QSplat Node Structure

<table>
<thead>
<tr>
<th>Position and Radius</th>
<th>Tree Structure</th>
<th>Normal</th>
<th>Width of Cone of Normals</th>
<th>Color (Optional)</th>
</tr>
</thead>
<tbody>
<tr>
<td>13 bits</td>
<td>3 bits</td>
<td>14 bits</td>
<td>2 bits</td>
<td>16 bits</td>
</tr>
</tbody>
</table>

**Delta Coding**

[Deering 96]
QSplat Node Structure

Position and Radius

Tree Structure

Normal

Width of Cone of Normals

Color (Optional)

13 bits

3 bits

14 bits

2 bits

16 bits

Hierarchical Coding
### QSplat Node Structure

<table>
<thead>
<tr>
<th>Data Type</th>
<th>Bits</th>
</tr>
</thead>
<tbody>
<tr>
<td>Position and Radius</td>
<td>13</td>
</tr>
<tr>
<td>Tree Structure</td>
<td>3</td>
</tr>
<tr>
<td>Normal</td>
<td>14</td>
</tr>
<tr>
<td>Width of Cone of Normals</td>
<td>2</td>
</tr>
<tr>
<td>Color (Optional)</td>
<td>16</td>
</tr>
</tbody>
</table>

- **Number of children** (0, 2, 3, or 4) – 2 bits
- **Presence of grandchildren** – 1 bit
QSplat Node Structure

- Normal quantized to grid on faces of a cube

Position and Radius | Normal | Width of Cone of Normals | Color (Optional)
--- | --- | --- | ---
13 bits | 14 bits | 2 bits | 16 bits

52×52×6
QSplat Node Structure

- Each node contains bounding cone of children’s normals
- Hierarchical backface culling [Kumar 96]
QSplat Node Structure

- Position and Radius
  - 13 bits

- Tree Structure
  - 3 bits

- Normal
  - 14 bits

- Width of Cone of Normals
  - 2 bits

- Color (Optional)
  - 16 bits

Viewer

Culled

Not Culled
### QSplat Node Structure

<table>
<thead>
<tr>
<th>Field</th>
<th>Bits</th>
</tr>
</thead>
<tbody>
<tr>
<td>Position and Radius</td>
<td>13</td>
</tr>
<tr>
<td>Tree Structure</td>
<td>3</td>
</tr>
<tr>
<td>Normal</td>
<td>14</td>
</tr>
<tr>
<td>Width of Cone of Normals</td>
<td>2</td>
</tr>
<tr>
<td>Color (Optional)</td>
<td>16</td>
</tr>
</tbody>
</table>

- Per-vertex color is quantized 5-6-5 (R-G-B)
QSplat Rendering Algorithm

• Traverse hierarchy recursively

```plaintext
if (node not visible)
    Skip this branch
else if (leaf node)
    Draw a splat
else if (size on screen < threshold)
    Draw a splat
else
    Traverse children
```

Hierarchical frustum / backface culling

Point rendering

Level of detail control

Adjusted to maintain desired frame rate
Frame Rate Control

• Feedback-driven frame rate control
  – During motion: adjust recursion threshold based on time to render previous frame
  – On mouse up: redraw with smaller thresholds
  – Consequence: frame rate may vary

• Alternative:
  – Predictive control of detail [Funkhouser 93]
Loading Model from Disk

• Tree layout:
  – Breadth-first order in memory and on disk

• Working set management:
  – Memory mapping disk file
  – Consequence: lower detail for new geometry
  – Alternative: Active working set management with prefetching [Funkhouser 96, Aliaga 99]
Tradeoffs of Splatting

For rendering large 3D models, what are the tradeoffs of:

<table>
<thead>
<tr>
<th>Polygons</th>
<th>QSplat</th>
</tr>
</thead>
<tbody>
<tr>
<td>Good for large, flat or subtly curved regions</td>
<td>Good for models with detail everywhere</td>
</tr>
<tr>
<td>Highly-efficient rasterization with 3D graphics hardware</td>
<td>Higher per-pixel cost, but less slowdown in absence of 3D hardware</td>
</tr>
<tr>
<td>Decimation or creating LOD data structures is often expensive</td>
<td>Fast preprocessing</td>
</tr>
</tbody>
</table>
Demo – St. Matthew

- 3D scan of 2.7 meter statue at 0.25 mm
- 102,868,637 points
- File size: 644 MB
- Preprocessing time: 1 hour
Conclusion

- Non-polygonal rendering
  - Works well when \# samples \gg \# pixels
  - Lack of connectivity may = simpler algorithms
  - Bad for flat regions

- Time, space efficiency important for big data sets