Row Tracing with Hierarchical Occlusion Maps

Ravi P. Kammaje, Benjamin Mora
August 9, 2008
Introduction
  Related Work

Row Tracing
  Concept
  Tree Traversal
  Leaf Node Processing
  HOMs

Packet Row Tracing

Results

Conclusion
Rendering methods - Rasterization

➤ Widely used

➤ Brute force approach

➤ High per pixel complexity

➤ Very fast per pixel due to fast scanline intersections

➤ Exceptional performance for relatively small models due to hardware acceleration

➤ Visibility determination
  ➤ Hierarchical Occlusion Maps (HOMs) – Zhang et al. (1997)
Rendering methods - Ray Tracing

- Acceleration structures
  - Spatial Subdivision Structures – kd-trees, BSP trees, octrees, grids, etc
  - Bounding Volume Hierarchies

- Scales well with increase in model size – logarithmic complexity

- Parallelization

Acceleration


Row Tracing

- Combines rasterization and ray tracing concepts
- A form of Packet ray tracing – Packets of rays spanning an entire row
- Row can be
  - A 2D plane
    - Simpler traversal
    - Easy row / triangle intersection – per-pixel cost less than ray / triangle intersections
  - A 1D line – Simplifies clipping, occlusion testing and frustum testing
## Comparison with other methods

<table>
<thead>
<tr>
<th></th>
<th>Rasterization</th>
<th>Ray Tracing</th>
<th>Row Tracing</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Logarithmic Complexity</strong></td>
<td>×</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td><strong>Cheaper per-pixel scanline algorithm</strong></td>
<td>✓</td>
<td>×</td>
<td>✓</td>
</tr>
<tr>
<td><strong>Multi-core Parallelization</strong></td>
<td>×</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td><strong>Shadow Rays</strong></td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td></td>
<td>(with shadow mapping / shadow volumes)</td>
<td>(with shadow mapping / shadow volumes)</td>
<td></td>
</tr>
</tbody>
</table>
Row Tracing

- High level algorithm
  - Traverse row-plane through kd-tree or octree
  - Rasterize leaf node triangles with scanline algorithm

- Very similar to Ray tracing

- Early ray termination not possible

- Use 1D Hierarchical Occlusion Maps to achieve this
Tree Traversal

- Similar to ray tree traversal
- Start traversal from root node
- Traverse nodes in front-to-back order
- Traversal stops if
  - Row-plane does not intersect node
  - Node’s projection is occluded
  - Node’s projection is outside frustum (1D frustum)
- At leaf node – rasterize leaf node triangles
Row-plane / Node intersection

Row node intersection

Row plane

Node vertices to be tested

Row being traced

Image plane

Viewport
Leaf node processing

Intersect Row-plane with leaf node triangles
Leaf node processing

Clip intersection segments
Leaf node processing

Project clipped segments
Hierarchical Occlusion Maps (HOMs)

- Important optimization
  - Indicates already occluded parts of a Row
- 1D version of HOM by Zhang, et al. (1997)

HOM for Dragon model
HOMs

- Lowest level – 1 pixel
- Each upper level – 2 bits of lower level
- For a row with 1024 pixels, lowest level – 128 chars
- Entire HOM – 256 chars

Example HOM for a 16 pixel row
HOMs – Usage

- Initialize prior to traversal
  - Set all bits to zero
  - The entire row is unoccluded
## HOMs – Usage

- Initialize prior to traversal
  - Set all bits to zero
  - The entire row is unoccluded

- Updating the HOM
  - Triangles rasterization
  - Corresponding lowest level bits are set to 1
HOMs – Usage

- Initialize prior to traversal
  - Set all bits to zero
  - The entire row is unoccluded

- Updating the HOM
  - Triangles rasterization
  - Corresponding lowest level bits are set to 1
  - Upper levels updated if necessary
HOMs – Usage

- Initialize prior to traversal
  - Set all bits to zero
  - The entire row is unoccluded

- Updating the HOM
  - Triangles rasterization
  - Corresponding lowest level bits are set to 1
  - Upper levels updated if necessary

- Testing for Occlusion
  - Skip occluded nodes
  - Optimize rasterization
Node occlusion test

- Use HOM to find if node is occluded
Node occlusion test

- Use HOM to find if node is occluded

- Main challenge – to find the pixels of the node projection
  - Accurate projection
    - Needs 8 vertices’ projection
    - Too expensive – 16 dot products and 8 divides
  - Calculate an overestimate
  - Much cheaper – 4 dot products and 2 divides
Occlusion testing leaf node triangle projections
Packet Row Tracing

- Row-Packet / Node intersection
  - Case 1 – All rows in packet hit the node
  - Case 2 – Row packet misses node
  - Case 3 – Divergence nodes - Trace individual rows from these nodes

![Diagram showing cases of row-packet/node intersections](image-url)
Packet Row Tracing

- **Row-Packet / Node intersection**
  - Case 1 – All rows in packet hit the node
  - Case 2 – Row packet misses node
  - Case 3 – Divergence nodes - Trace individual rows from these nodes

- **Occlusion testing** – Test each row individually

- **Leaf node** – All rows are rasterized with leaf node’s triangles
SSE and Muti-threading

- SSE used to optimize key parts of algorithm
  - Triangle intersections
  - Projection calculation

- Multi-threading
  - Good load balancing – essential to obtain best speed-ups
    - Round robin allocation of work

Load balancing for a Quad core processor.
## Results

<table>
<thead>
<tr>
<th></th>
<th>ERW6</th>
<th>Sponza</th>
<th>Armadillo</th>
<th>Soda Hall</th>
<th>PowerPlant</th>
</tr>
</thead>
<tbody>
<tr>
<td>Shapes</td>
<td><img src="image1.png" alt="Shape" /></td>
<td><img src="image2.png" alt="Shape" /></td>
<td><img src="image3.png" alt="Shape" /></td>
<td><img src="image4.png" alt="Shape" /></td>
<td><img src="image5.png" alt="Shape" /></td>
</tr>
<tr>
<td>Triangles</td>
<td>804</td>
<td>66454</td>
<td>345944</td>
<td>2.1 Million</td>
<td>12.7 Million</td>
</tr>
</tbody>
</table>

- Intel Core 2 Quad 2.4 GHz – 4 threads
- GeForce 8800 GTX with 768 MB Video RAM
- Performance comparison
  - Packet ray tracing \((2 \times 2, 4 \times 4, 8 \times 8, 16 \times 16 \text{ rays})\)
  - OpenGL with display lists
**Results**

<table>
<thead>
<tr>
<th>Model</th>
<th>Triangles</th>
</tr>
</thead>
<tbody>
<tr>
<td>ERW6</td>
<td>804</td>
</tr>
<tr>
<td>Sponza</td>
<td>66454</td>
</tr>
<tr>
<td>Armadillo</td>
<td>345944</td>
</tr>
<tr>
<td>Soda Hall</td>
<td>2.1 Million</td>
</tr>
<tr>
<td>PowerPlant</td>
<td>12.7 Million</td>
</tr>
</tbody>
</table>

Frames per second - Multi-Threaded version (4 threads).
Results – Fastest version of Row tracing vs OpenGL

<table>
<thead>
<tr>
<th>ERW6</th>
<th>Sponza</th>
<th>Armadillo</th>
<th>Soda Hall</th>
<th>PowerPlant</th>
</tr>
</thead>
<tbody>
<tr>
<td>804 Triangles</td>
<td>66454 Triangles</td>
<td>345944 Triangles</td>
<td>2.1 Million Triangles</td>
<td>12.7 Million Triangles</td>
</tr>
</tbody>
</table>

Frames per second - Multi-Threaded version (4 threads).
Results – Fastest versions of Row tracing vs Ray tracing

<table>
<thead>
<tr>
<th>Scene</th>
<th>Triangles</th>
<th>Frames per second (Multi-Threaded 4 threads)</th>
</tr>
</thead>
<tbody>
<tr>
<td>ERW6</td>
<td>804 Triangles</td>
<td>120</td>
</tr>
<tr>
<td>Sponza</td>
<td>66454 Triangles</td>
<td>80</td>
</tr>
<tr>
<td>Armadillo</td>
<td>345944 Triangles</td>
<td>50</td>
</tr>
<tr>
<td>Soda Hall</td>
<td>2.1 Million Triangles</td>
<td>30</td>
</tr>
<tr>
<td>PowerPlant</td>
<td>12.7 Million Triangles</td>
<td>8</td>
</tr>
</tbody>
</table>

Results - Performance on Octrees

<table>
<thead>
<tr>
<th></th>
<th>ERW6</th>
<th>Sponza</th>
<th>Armadillo</th>
<th>Soda Hall</th>
<th>PowerPlant</th>
</tr>
</thead>
<tbody>
<tr>
<td>Triangles</td>
<td>804</td>
<td>66454</td>
<td>345944</td>
<td>2.1 Million</td>
<td>12.7 Million</td>
</tr>
</tbody>
</table>

Frames per second - Multi-Threaded version (4 threads).
## Results - Scalability over number of Cores

### ERW6
- 804 Triangles

### Sponza
- 66454 Triangles

### Armadillo
- 345944 Triangles

### Soda Hall
- 2.1 Million Triangles

### PowerPlant
- 12.7 Million Triangles

### Speed-up provided by Multi-Threading

<table>
<thead>
<tr>
<th></th>
<th>Octree Tracing</th>
<th>Octree Packet Row Tracing</th>
<th>KdTree Tracing</th>
<th>KdTree Packet Row Tracing</th>
<th>Packet Ray Tracing 2x2</th>
<th>Packet Ray Tracing 4x4</th>
<th>Packet Ray Tracing 8x8</th>
<th>Packet Ray Tracing 16x16</th>
<th>OpenGL Display Lists</th>
</tr>
</thead>
<tbody>
<tr>
<td>ERW6</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Sponza</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Armadillo</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Soda Hall</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>PowerPlant</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Results - Performance vs Tree Size

- **Sponza**
- **Sodahall**

<table>
<thead>
<tr>
<th>KBs</th>
<th>Rendering time in ms.</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1400</td>
</tr>
<tr>
<td>1000</td>
<td>1000</td>
</tr>
<tr>
<td>2000</td>
<td>600</td>
</tr>
<tr>
<td>3000</td>
<td>200</td>
</tr>
<tr>
<td>4000</td>
<td>100</td>
</tr>
<tr>
<td>5000</td>
<td>50</td>
</tr>
<tr>
<td>6000</td>
<td>20</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>KBs</th>
<th>Rendering time in ms.</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>600</td>
</tr>
<tr>
<td>20000</td>
<td>500</td>
</tr>
<tr>
<td>40000</td>
<td>300</td>
</tr>
<tr>
<td>60000</td>
<td>200</td>
</tr>
<tr>
<td>80000</td>
<td>100</td>
</tr>
</tbody>
</table>
Conclusion and Future work

Conclusion

- Row tracing combines concepts from ray tracing and rasterization
- Low per pixel cost
- Excellent performance
- Works well on smaller trees
- Easy and effective parallelization – near perfect speed-up achieved

Future work

- Further optimization – faster row packet occlusion testing
- Dynamic scenes – combine row tracing with fast construction methods
- Shadow rays – shadow mapping technique
Acknowledgements

▶ I. Wald
  ▶ Models and information about importing models.

▶ Stanford Computer Graphics Laboratory
  ▶ Models.
  ▶ PLY import libraries.

▶ Eric Haines et al.
  ▶ Standard Procedural Databases set of models.
Questions?