Accelerating Shadow Rays Using Volumetric Occluders and Modified kd-Tree Traversal

Peter Djeu*, Sean Keely*, and Warren Hunt‡

* University of Texas at Austin  ‡ Intel Labs
Shadow Rays

• Shadow rays are often time consuming
• Intersection (isect) required with lots of triangles

<table>
<thead>
<tr>
<th>Baseline</th>
<th>Our approach</th>
</tr>
</thead>
<tbody>
<tr>
<td>Triangles</td>
<td>Volumetric Occluders and Triangles</td>
</tr>
</tbody>
</table>

vs

Ray Tracing and Shadows

• Goal: faster shadows without modifying result
• 50% reduction in time spent on shadow rays
• Identical images
What is a Volumetric Occluder?

- An axis aligned bounding box within the mesh interior
- Kd-tree nodes used to represent volumetric occluders
Outline

1. Create volumetric occluders

2. Modify kd-tree traversal to use volumetric occluders
   • Two novel algorithms

3. Reuse volumetric occluders

4. Results
   • Monte Carlo soft shadows
One prerequisite: Manifold mesh

- Input mesh must be manifold
  - Watertight
  - Consistent face orientation
  - No self intersections
Example: Oval mesh

- Normal build – SAH
Classification

- For each leaf node in kd-tree, classify as:
  - Boundary node – non-empty
  - Opaque node – empty and inside
  - Clear node – empty and outside
Classifying the oval mesh

- Classification is easy on manifold mesh
- Opaque nodes are volumetric occluders
But are they useful?

• Intersection with a volumetric occluder implies intersection with mesh geometry provided:
  • At least one ray endpoint is outside of the mesh

• Volumetric occluders accelerate shadow rays
  • Cheap to isect
  • Often larger – better occlusion
One major problem

- Volumetric occluders are inaccessible under normal kd-tree traversal order!
Modifying kd-tree traversal to use volumetric occluders

• We present two ways to modify kd-tree traversal order
  • Goal: encounter volumetric occluders during traversal

• Both solutions perform the same task at different cost

• Both solutions enable encountering volumetric occluders during traversal
  • Intersection becomes a bit-mask and compare
Traversals Mod 1: Extended Ray Order

- Extend the ray past boundary nodes
- Defer geometry intersect using Deferment List

Node A
Deferment List
Extended Ray Order
Extended Ray Order Summary

• Speculatively take extra traversal steps

• If speculation pays off, we encounter a volumetric occluder
Extended Ray Order Expected Behavior

+ High chance geometry isects decrease
– Traversal steps guaranteed to increase
Intro to Traversal Mod 2: BFS

- Breadth-first search (BFS)
  - Shallow nodes are visited before deep ones
  - Code change: Stack → Queue
Intro to Traversal Mod 2: BFS

- Breadth-first search (BFS)
  - Shallow nodes are visited before deep ones
  - Code change: Stack → Queue

BFS Order

Queue
0

BFS Order
Intro to Traversal Mod 2: BFS

- Breadth-first search (BFS)
  - Shallow nodes are visited before deep ones
  - Code change: Stack → Queue

BFS Order
Intro to Traversal Mod 2: BFS

- Breadth-first search (BFS)
  - Shallow nodes are visited before deep ones
  - Code change: Stack → Queue

BFS Order
Intro to Traversal Mod 2: BFS

• Breadth-first search (BFS)
  • Shallow nodes are visited before deep ones
  • Code change: Stack → Queue
Intro to Traversal Mod 2: BFS

- Breadth-first search (BFS)
  - Shallow nodes are visited before deep ones
  - Code change: Stack → Queue
Intro to Traversal Mod 2: BFS

- Breadth-first search (BFS)
  - Shallow nodes are visited before deep ones
  - Code change: Stack $\rightarrow$ Queue

BFS Order

BFS Order

Queue

5, 6, 7
Intro to Traversal Mod 2: BFS

- Breadth-first search (BFS)
  - Shallow nodes are visited before deep ones
  - Code change: Stack → Queue

BFS Order

Queue
6, 7, 8
Traversals Mod 2: QDBFS

- Quick descent bread-first search (QDBFS)
  - Immediately descend in one child case

BFS Order
Nodes visited: 9
Pushes: 10
Pops: 8

QDBFS Order
Nodes visited: 9
Pushes: 8
Pops: 4
Quick descent

- Can discover volumetric occluders sooner

BFS Order

- Nodes visited: 8
- Pushes: 10
- Pops: 7

QDBFS Order

- Nodes visited: 7
- Pushes: 6
- Pops: 3
BFS and QDBFS Summary

• BFS
  • Finds large volumetric occluders

• QDBFS
  • Less queue traffic
  • 15% - 20% faster than BFS

• QDBFS will be used for rest of talk
QDBFS Expected Behavior

+ High chance geometry intersects decrease
+ Traversal steps may decrease
  – Traversal steps may increase
+ Preference for large occluders
Volumetric occluder intersection

- Very, very cheap

- Volumetric occ. tag stored in flag bits in the kd-node

- Intersection leverages all work from kd-tree traversal
Reusing volumetric occluders

- The volumetric occluder cache (VC)
  - Software managed
  - Stores most recently used occluders

- Populate

- Lookup
Volumetric occluder cache tradeoffs

+ Traversal steps very likely to decrease
  - On a cache hit, traversal steps drops to 0

– May perform unnecessary bounding box isects
  - If cache is size n, each cache miss costs n isects

• In our experience, the VC always improved run-times
Results

- 4-wide packet tracer
- 1024 x 1024, 5 area lights, ~100 mil shadow rays

<table>
<thead>
<tr>
<th>Model</th>
<th>Size</th>
</tr>
</thead>
<tbody>
<tr>
<td>Armadillo</td>
<td>346 k</td>
</tr>
<tr>
<td>Bunny</td>
<td>70 k</td>
</tr>
<tr>
<td>Dragon</td>
<td>871 k</td>
</tr>
<tr>
<td>Rose</td>
<td>326 k</td>
</tr>
<tr>
<td>Yeah Right</td>
<td>119 k</td>
</tr>
<tr>
<td>Cowboy</td>
<td>7 k</td>
</tr>
</tbody>
</table>
Simple shadows should be cheap

• Our approach: compute cheaper shadows for models that have simple (coherent) shadows
Yeah Right failure case

- Shadow has little coherency
- Shadow plagued by silhouette
New bottleneck: Unoccluded shadow rays

Two types of shadow rays
1. occluded - hits some object
2. unoccluded - hits no objects
Extensions

• This work: controlled environment

• Future work
  • Realistic object configurations
  • Realistic lighting configurations
Summary

• Volumetric occluders provide an opportunity to accel. shadow rays
  • Two new kd-tree traversal algorithms
• Up to 50% reduction in time spent on shadow rays
  • In failure cases, performance degradation is graceful
• Produces identical images
• Applies to any type of query on binary visibility
Acknowledgements

• Bill Mark
• Graphics and Parallel Systems Lab (UT Austin)
• Modelers, especially Techland
• Intel Corporation
Finally... free stuff!

SpiderMind System Library available

http://triangle.csres.utexas.edu/gps/downloads/

Thank You!
Previous Work

Create volumetric occluders using kd-tree nodes

- Inexpensive proxy in an accel. structure
  - Wald et al. 2004 – massive models
  - Yoon et al. 2006 – R-LOD

- Exact volumetric data (closely related)
  - Woo and Amanatides 1990 – voxel occlusion testing
  - Schaufler et al. 2000 – occluder fusion
  - Reshetov et al. 2005 – MLRTA
Classifying kd-nodes

For a given leaf node in the kd-tree:

• If the node has geometry, we are done, it is a BOUNDARY node

• If the node is empty:
  • Cast a test ray, origin at node center, any direction
  • If the test ray hits a back-facing polygon
    • Node is inside mesh, node is OPAQUE
  • If the test ray hits a front-facing polygon or nothing
    • Node is outside mesh, node is CLEAR
Illustration of classification

- Manifold mesh guarantees results are well-defined
Size of the deferment list

- Increase in size usually led to increased performance
- Best performance at size 512 (!)

- Why is this the best policy?
  - Theory: it maximizes # of volumetric occlusions

- VC no longer affects # of triangle isects
Another look: shadow ray rate

• In units of megarays / sec
• Up to a 2x improvement
• Worst case still OK
• Single-ray within 2%

<table>
<thead>
<tr>
<th>Scene</th>
<th>base</th>
<th>vol occ</th>
</tr>
</thead>
<tbody>
<tr>
<td>Armadillo</td>
<td>7.2</td>
<td>13.1</td>
</tr>
<tr>
<td>Bunny</td>
<td>6.4</td>
<td>10.7</td>
</tr>
<tr>
<td>Dragon</td>
<td>6.0</td>
<td>12.2</td>
</tr>
<tr>
<td>Rose</td>
<td>11.6</td>
<td>15.0</td>
</tr>
<tr>
<td>Yeah Right</td>
<td>6.8</td>
<td>6.8</td>
</tr>
<tr>
<td>Cowboy</td>
<td>16.2</td>
<td>17.9</td>
</tr>
</tbody>
</table>
kd-Node to bounding box lookup table

• Volumetric occluders are added to the cache when they are hit during traversal

• Problem: the bounds of the volumetric occluder are not known at traversal time

• Solution: precompute the lookup table as a preprocess, use it to lookup the bounds at run-time
Cowboy failure case

- Volumetric occluders perform well when:
  - Internal volume is large and expansive
  - Mesh tessellation rate is high
- For Cowboy, the ratio is poor
  - Internal volume is too small compared to tessellation rate

<table>
<thead>
<tr>
<th>Scene</th>
<th>v.o. SA / geom SA</th>
</tr>
</thead>
<tbody>
<tr>
<td>Armadillo</td>
<td>45</td>
</tr>
<tr>
<td>Bunny</td>
<td>18</td>
</tr>
<tr>
<td>Dragon</td>
<td>36</td>
</tr>
<tr>
<td>Rose</td>
<td>11</td>
</tr>
<tr>
<td>Yeah Right</td>
<td>9.6</td>
</tr>
<tr>
<td>Cowboy</td>
<td>9.6</td>
</tr>
</tbody>
</table>