Dynamic Spatial Partitioning for Real-Time Visibility Determination

Joshua Shagam
Computer Science
Problem

- Complex 3D environments have large numbers of objects
- Computer hardware can only render a finite number at any given speed
- Need to determine which ones will be visible
- Simple/intuitive approaches are extremely slow
Outline

- Prior work
- *Dynamic AABB Tree Structure* - definition, maintenance, usage
- Real-world performance example
- Heuristic comparison
- Conclusions
Visibility Determination

- Heavily-studied [Cohen-Or et al., 2000]
- Hundreds of algorithms
- Dozens of approaches
- Only a few in practical use
- Those in use are very limited
Spatial Partitioning

- Most visibility approaches use spatial partitioning.
- Divide (partition) environment into cells (regions) in organized manner.
- Many partitioning algorithms (most are static or limited in update ability).
Partitioning Approaches

- Binary Space Partition (BSP)
- K-D Tree/Quadtree/Octree
- Axis-Aligned Bounding Box (AABB) Tree
Dynamic AABB Tree

- Group objects based on relative position
  - Grouping determined by heuristic
  - Several heuristics available
- Recursively divide groups
- Group-level visibility determination
Data Structure Definition

- Axis-Aligned Bounding Box (AABB)
  - Defined by two corner points
  - Sides are parallel to coordinate axes
- 0 or more child nodes
- 0 or more objects
Constraints

- Node’s AABB must encompass
  - objects
  - child nodes
- No other constraints
Splitting Nodes
Visibility
Occlusion
(Spatial coherence)
Occlusion

(Temporal + spatial coherence)
Fragmentation Avoidance

- Moving/adding objects may cause tree imbalance
- When AABBs are recomputed, recompute split points w/ approximation
- New split point applies “pressure” to heuristic; stochastic rebalancing
Video clip

Realtime render, 1.1GHz Athlon, Radeon 9700 using temporal coherence occluders only
## Comparisons

**Scene: tunnel2**

<table>
<thead>
<tr>
<th></th>
<th>Objects Considered</th>
<th>Objects Rendered</th>
<th>Frames per Second</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Brute-Force</strong></td>
<td>1736</td>
<td>302</td>
<td>21</td>
</tr>
<tr>
<td><strong>AABB Visibility</strong></td>
<td>554</td>
<td>302</td>
<td>30</td>
</tr>
<tr>
<td><strong>AABB Occlusion</strong></td>
<td>554</td>
<td>302(95)</td>
<td>26</td>
</tr>
</tbody>
</table>
## Comparisons

**Scene: stress**

<table>
<thead>
<tr>
<th></th>
<th>Objects Considered</th>
<th>Objects Rendered</th>
<th>Frames per Second</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Brute-Force</strong></td>
<td>1346</td>
<td>489</td>
<td>17</td>
</tr>
<tr>
<td><strong>AABB Visibility</strong></td>
<td>772</td>
<td>489</td>
<td>23</td>
</tr>
<tr>
<td><strong>AABB Occlusion</strong></td>
<td>568</td>
<td>101(100)</td>
<td>35</td>
</tr>
</tbody>
</table>
## Comparisons

**Scene:** stress5

<table>
<thead>
<tr>
<th></th>
<th>Objects Considered</th>
<th>Objects Rendered</th>
<th>Frames per Second</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Brute-Force</strong></td>
<td>1761</td>
<td>528</td>
<td>23</td>
</tr>
<tr>
<td><strong>AABB Visibility</strong></td>
<td>928</td>
<td>528</td>
<td>24</td>
</tr>
<tr>
<td><strong>AABB Occlusion</strong></td>
<td>862</td>
<td>182(182)</td>
<td>33</td>
</tr>
</tbody>
</table>
Nesting Heuristics

- Determine how objects are distributed
- **Goals:**
  - Maximize tree balance
  - Minimize tree depth
  - Minimize number of visibility tests
Nesting Heuristics

- Two variants of each
  - Leafy - all objects stored in leaf nodes
  - Non-leafy - larger objects stored in internal modes
- Each named after a tree concept (not a strict implementation of the tree type)
Ternary
Octree
Icoseptree
Timings

- Generate large number of random objects
- Distribute throughout region
  - Variety of distributions
  - Consistent overall density
- Record avg. time for region query, object movement
Optimum Split Threshold

- Split threshold affects objects/node
  - Node test slower than object test
  - Need to balance node test vs. wasted object tests
- Find fastest average query per threshold with a dense uniform packing
Optimum Split: K-D

![Graph showing culling time vs. split threshold for different datasets with and without leafy features.](image-url)
Optimum Split: Ternary
Optimum Split: Octree

![Graph showing culling time vs. split threshold]
Optimum Split: Icoseptree
Immediate Observations

- Split threshold can make noticeable difference (nearly double for icoseptree)
- Wide ranges for performance plateau
- Non-leafy variant always performs at least as good as leafy
  - No point in considering leafy further
Best Overall Performance

- Compare performance of heuristics
  - Vary number of objects up to 1 million
  - Four different random distributions (uniform, sphere, cluster, Lissajous)
Clustered
Sphere

![Graph showing culling time vs. total objects for different data structures: K-D, Ternary, Octree, Icosahedron](image)

- **K-D, s=30**
- **Ternary, s=20**
- **Octree, s=30**
- **Icosahedron, s=50**

**Y-axis:** Culling time (ms)

**X-axis:** Total objects (log)
Lissajous
Conclusions

- *D-AABB trees* provide fast queries and updates on fully-dynamic environments
- Works w/ simple occlusion culling; accurate visibility w/o precomputation
- Icoseptree heuristic scales best of those tested
Further Work

- Determine efficient occlusion algorithm for spatial-coherence culling
- Better heuristics than icoseptree might be possible
Acknowledgments

- Dr. Joseph Pfeiffer, Jr.
- NMSU Computer Science Department
- GAANN fellowship program
- Anonymous ACM SIGGRAPH review panel
Questions