Ray Tracing

Ray Tracing 1
- Basic algorithm
- Overview of pbrt
- Ray-surface intersection (triangles, ...)

Ray Tracing 2
- Problem: brute force = |Image| \times |Objects|
- Acceleration data structures

Ray Tracing Acceleration Techniques

**Approaches**

- Faster Intersection
- Fewer Rays
- Generalized Rays

**Uniform grids**
- Spatial hierarchies
- k-d, oct-tree, bsp
- Hierarchical grids
- Hierarchical bounding volumes (HBV)

**Tighter bounds**
- Faster intersector

**Early ray termination**
- Adaptive sampling

**Beam tracing**
- Cone tracing
- Pencil tracing
Primitives

pbrt primitive base class

- Shape
- Material reflection and emission

Primitives

- Primitive instance
  - Transformation and pointer to basic primitive

- Aggregate (collection)
  - Treat collections just like basic primitives
  - Incorporate acceleration structures into collections
  - May nest accelerators of different types
  - Types: grid.cpp and kdtree.cpp

Uniform Grids

Preprocess scene
1. Find bounding box
Uniform Grids

Preprocess scene
1. Find bounding box
2. Determine resolution

\[ n_x = n_y = n_z \propto n_o \]
\[ \max(n_x, n_y, n_z) = \frac{d}{\sqrt[3]{n_o}} \]

Uniform Grids

Preprocess scene
1. Find bounding box
2. Determine resolution

\[ \max(n_x, n_y, n_z) = \frac{d}{\sqrt[3]{n_o}} \]

2. Place object in cell, if object overlaps cell
Uniform Grids

Preprocess scene
1. Find bounding box
2. Determine resolution
   \[ \max(n_x, n_y, n_z) = d \sqrt[2]{n_o} \]
3. Place object in cell, if object overlaps cell
4. Check that object intersects cell

Uniform Grids

Preprocess scene
Traverse grid
   3D line – 3D-DDA
   6-connected line

Section 4.3
Caveat: Overlap

Problem: Don’t output first intersection found!

Problem: Redundant intersection tests
Solution: Mailboxes
  - Assign each ray an increasing number
  - Primitive intersection cache (mailbox)
    - Store last ray number tested in mailbox
    - Only intersect if ray number is greater

Spatial Hierarchies

Letters correspond to planes (A)
Point Location by recursive search
Spatial Hierarchies

Letters correspond to planes (A, B)
Point Location by recursive search
Variations

- \text{kd-tree}
- \text{oct-tree}
- \text{bsp-tree}

Ray Traversal Algorithms

Recursive inorder traversal

\[ t^* = (S - O[a]) / D[a] \]

\[
\begin{align*}
& t^* < t_{\text{min}} \\
& t_{\text{min}} < t^* < t_{\text{max}} \\
& t^* < t_{\text{min}}
\end{align*}
\]

Intersect (L, tmin, tmax) Intersect (L, tmin, t*) Intersect (R, tmin, tmax) Intersect (R, t*, tmax)
**Build Hierarchy Top-Down**

Methods to choose axis and splitting plane
- Midpoint
- Median cut (balanced)
- Surface area heuristic

**Cost**

What is the cost of tracing a ray through a node?

\[
\text{Cost(node)} = C_{\text{trav}} + \text{Prob(hit L)} \times \text{Cost(L)} + \text{Prob(hit R)} \times \text{Cost(R)}
\]

- \(C_{\text{trav}}\) = cost of traversing a cell
- \(\text{Cost(L)}\) = cost of traversing left child
- \(\text{Cost(R)}\) = cost of traversing right child
Splitting with Cost in Mind

From Gordon Stoll

Split in the Middle = Bad!

From Gordon Stoll

Makes the L & R probabilities equal
Pays no attention to the L & R costs
**Split at the Median = Bad!**

Makes the L & R costs equal
Pays no attention to the L & R probabilities

From Gordon Stoll

---

**Cost-Optimized Split = Good!**

Automatically and rapidly isolates complexity
Produces large chunks of empty space

From Gordon Stoll
Cost

Need the probabilities

- Turns out to be proportional to surface area

Need the child cell costs

- Triangle count is a good approximation

Cost(cell) = C_trav + Prob(hit L) * Cost(L) + Prob(hit R) * Cost(R)

= C_trav + SA(L) * TriCount(L) + SA(R) * TriCount(R)

C_trav is the ratio of the cost to traverse to the cost to intersect

- C_trav = 1:80 in pbrt
- C_trav = 1:1.5 in a highly optimized version

Surface Area and Rays

Number of rays in a given direction that hit an object is proportional to its projected area

The total number of rays hitting an object is $4\pi \overline{A}$

Crofton’s Theorem:

For a convex body $\overline{A} = \frac{S}{4}$

For a sphere $S = 4\pi r^2$ and $\overline{A} = A = \pi r^2$
Surface Area and Rays

The probability of a ray hitting a convex shape enclosed by another convex shape is

\[
\Pr[r \cap S_o \mid r \cap S_c] = \frac{S_o}{S_c}
\]

Basic Build Algorithm (Triangles)

1. Pick an axis, or optimize across all three
2. Build a set of “candidate” split locations
   Note: Cost extrema must be at bbox vertices
   - Vertices of triangle
   - Vertices of triangle clipped to node bbox
3. Sort or bin the triangles
4. Sweep to incrementally track L/R counts, cost
5. Output position of minimum cost split

Running time: \[ T(N) = N \log N + 2T(N/2) \]
\[ T(N) = N \log^2 N \]
Sweep Build Algorithm

2n splits

\[ P_a = \frac{S_a}{S} \quad P_b = \frac{S_b}{S} \]

Termination Criteria

When should we stop splitting?

- Bad: depth limit, number of triangles
- Good: When split does not lower the cost

Threshold of cost improvement

- Stretch over multiple levels
- For example, if cost doesn’t go down after three splits in a row, terminate

Threshold of cell size

- Absolute probability \( \frac{SA(\text{node})}{SA(\text{scene})} \) small
Best Reported Timings

<table>
<thead>
<tr>
<th>Scene</th>
<th>Framerate (FPS) @ 1024x1024 resolution</th>
<th>OpenRT @ 2.5 GHz P4 1 thread</th>
<th>MLRTA @ 2.4 GHz P4 1 thread</th>
<th>MLRTA @ 3.2 GHz P4 with HT 2 threads</th>
</tr>
</thead>
<tbody>
<tr>
<td>Erwē 804</td>
<td>– shader</td>
<td>7.1</td>
<td>70.2</td>
<td>109.8</td>
</tr>
<tr>
<td></td>
<td>+ shader</td>
<td>2.3</td>
<td>37.8</td>
<td>50.7</td>
</tr>
<tr>
<td>Conference 274K</td>
<td>– shader</td>
<td>4.55</td>
<td>11.2</td>
<td>19.5</td>
</tr>
<tr>
<td></td>
<td>+ shader</td>
<td>1.93</td>
<td>9.5</td>
<td>15.6</td>
</tr>
<tr>
<td>Soda Hall 2195K</td>
<td>– shader</td>
<td>4.12</td>
<td>21.1</td>
<td>35.5</td>
</tr>
<tr>
<td></td>
<td>+ shader</td>
<td>1.8</td>
<td>15.3</td>
<td>24.1</td>
</tr>
</tbody>
</table>

Reshetov, Soupikov, Hurley, SIGGRAPH 2005

Superoptimizations

Lots of optimizations

- Carefully written inner loop (no recursion)
- Use vector instructions SSE2
- 64 bits per kd-tree node
  - 32 bit position
  - 32 bit pointer to pair of child nodes
  - 2 bits for split plane direction (x, y, or z)
- Trace packet of rays
  - 4 or more rays at a time
- Intersect beam at top of tree
- Encourage empty nodes
- Special case axis-aligned triangles
- ...

CS348B Lecture 3
Pat Hanrahan, Spring 2006
Ray Tracing Hardware

Custom designed chips
- AR250/350 ray tracing processor
  www.art-render.com
- SaarCOR
- RPU

Ray tracing on programmable GPUs

Comparison

<table>
<thead>
<tr>
<th></th>
<th>Spheres</th>
<th>Rings</th>
<th>Tree</th>
</tr>
</thead>
<tbody>
<tr>
<td>Uniform Grid d=1</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>244</td>
<td>129</td>
<td>1517</td>
</tr>
<tr>
<td>Uniform Grid d=20</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>38</td>
<td>83</td>
<td>781</td>
</tr>
<tr>
<td>Hierarchical Grid</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>34</td>
<td>116</td>
<td>34</td>
</tr>
</tbody>
</table>

V. Havran, Best Efficiency Scheme Project
http://sgi.felk.cvut.cz/BES/
Comparison

Theoretical Nugget 1

Computational geometry of ray shooting

1. Triangles (Pellegrini)
   - Time: $O(\log n)$
   - Space: $O(n^{5+\varepsilon})$

2. Sphere (Guibas and Pellegrini)
   - Time: $O(\log^2 n)$
   - Space: $O(n^{5+\varepsilon})$
Theoretical Nugget 2

Optical computer = Turing machine
Reif, Tygar, Yoshida

Determining if a ray starting at y₀ arrives at yn is undecidable

\[ y = y + 1 \]
\[ y = -2 \times y \]
\[ \text{if}( y > 0 ) \]