Ray Tracing

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

Ray Tracing 2
- Problem: Brute force = |Image| x |Objects|
- Solution: Acceleration ≈ |Image| x log |Objects|

How to minimize the number of rays traced?
How to choose rays (sample) efficiently?

Primitives

pbrt primitive base class
- Shape
- Material (emission, reflection and transmission)

Subclasses
- Primitive instance
  - Transformation and pointer to a primitive
- Aggregate (collection)
  - Build acceleration data structures as primitives
  - Different types of accelerators: grid.cpp and kdtree.cpp
  - May nest accelerators of different types
Uniform Grids

Preprocess scene
1. Find bounding box

Uniform Grids

Preprocess scene
1. Find bounding box
2. Determine resolution
Uniform Grids

Preprocess scene
1. Find bounding box
2. Determine resolution
3. Place objects in cells, if bbox overlaps cell

4. If surface intersects cell
Uniform Grids

For each ray, traverse grid
3D line – 3D-DDA
6-connected line
See Section 4.3

Uniform Grids: Resolution?

• 1 cell
  • No speedup
Uniform Grids: Resolution?

- 1 cell
- No speedup
- Large number of cells
- Extra work walking through empty cells

Uniform Grids: Resolution?

- 1 cell
- No speedup
- Large number of cells
- Extra work walking through empty cells

Heuristic

\[ n_v = n_x n_y n_z \ll n_o \]
\[ \text{max}(n_x, n_y, n_z) = d \sqrt[d]{n_o} \]
\[ d \approx 3 \]
Caveat: Overlap

Mistake: Output first intersection found!

Solution: Check whether intersection is inside cell

Problem: Objects tested for intersection multiple times
Caveat: Overlap

Solution: Check whether intersection is inside cell

Problem: Objects tested for intersection multiple times
Solution: Mailboxes

- Assign each ray an increasing number
- Primitive intersection cache (mailbox)
  - Give each ray a number n
  - Store intersection point and ray n with each primitive
  - Only re-intersect if ray n is greater than last ray n

Grids: When It Works Well

Uniform grids work well for large collections of objects that are approximately uniform in size and distribution

http://www.kevinboulanger.net/grass.html
Grids: When It Works Poorly

“Teapot in the Stadium Problem”: For example, imagine you have a football stadium made of, say, 5K primitives. Sitting on a goal line is a shiny polygonalized teapot of 5K quadrilaterals (note that the teapot is teapot sized compared to the stadium).

Eric Haines
http://tog.acm.org/resources/RTNews/html/rtnews1b.html#art4

Problem: non-uniform size distribution; varying density

Spatial Hierarchies

Letters correspond to planes (A)
Spatial Hierarchies

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

Letters correspond to planes (A, B, C, D)
Boxes at leaves correspond to regions
Variations

kd-tree  oct-tree  bsp-tree

Recursive Inorder Traversal

[Kaplan, Arvo, Jansen]

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

Intersect(L, tmin, tmax)  Intersect(L, tmin, t*)  Intersect(R, t*, tmax)  Intersect(R, tmin, tmax)

Pat Hanrahan, Spring 2012
Building the Hierarchy

Which Hierarchy is Fastest?

What is the cost of tracing a ray?

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

C_trav = cost of traversing a cell
Cost(L) = cost of traversing left child
Cost(R) = cost of traversing right child
Splitting with Cost in Mind

From Gordon Stoll

Split in the Middle = Bad!

From Gordon Stoll

Midpoint: Makes the L & R probabilities equal
Cost of R greater than cost of L
Split at the Median = Bad!

Median: Makes the L & R costs equal
Probability of hitting L greater than R

Cost-Optimized Split = Best!

Cost(cell) = C_trav + Prob(hit L) * Cost(L) + Prob(hit R) * Cost(R)
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\text{\_trav} + SA(L) * TriCount(L) + SA(R) * TriCount(R)

C\text{\_trav} is the ratio of the cost to traverse to the cost to intersect

C\text{\_trav} = 1:80 in pbrt

C\text{\_trav} = 1:1.5 in a highly optimized version

Projected Area and Surface Area

Number of rays in a given direction that hit an object with surface area $S$

is proportional to its projected area $\bar{A}$

Average projected area

$$\bar{A} = \frac{1}{4\pi} \int A d\omega$$

Crofton’s theorem:

$$\bar{A} = \frac{S}{4}$$
**Surface Area and Ray Intersection Prob.**

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

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

**Sweep Build Algorithm**

\[
P_a = \frac{S_a}{N_a} \quad P_b = \frac{S_b}{N_b}
\]
Basic Build Algorithm (Triangles)

1. Pick an axis, or optimize across x, y, z
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 the triangles into intervals
4. Sweep to incrementally track L/R counts, cost
5. Output position of minimum cost split

Running time: $T(N) = N \log N + 2T(N/2)$

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 $SA(node)/SA(scene)$ small
Best Reported Timings

Millions of Rays per Second

<table>
<thead>
<tr>
<th>Scene # of triangles and shader (+/-)</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>Erw6 884 - shader</td>
<td>2.3</td>
<td>37.8</td>
<td>50.7</td>
<td></td>
</tr>
<tr>
<td>+ shader</td>
<td>7.1</td>
<td>70.2</td>
<td>109.8</td>
<td></td>
</tr>
<tr>
<td>Conference 274K - shader</td>
<td>1.93</td>
<td>9.5</td>
<td>15.6</td>
<td></td>
</tr>
<tr>
<td>+ shader</td>
<td>4.55</td>
<td>11.2</td>
<td>19.5</td>
<td></td>
</tr>
<tr>
<td>Soda Hall 2159K - shader</td>
<td>4.12</td>
<td>21.1</td>
<td>35.5</td>
<td></td>
</tr>
<tr>
<td>+ shader</td>
<td>1.8</td>
<td>15.3</td>
<td>24.1</td>
<td></td>
</tr>
</tbody>
</table>

Reshetov, Soupirov, Hurley, SIGGRAPH 2005

Superoptimizations

Lots of optimizations

- Carefully written inner loop (no recursion)
- Use vector instructions SSE/AVX/LNI
- 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
- …
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: $O(n^{5+\varepsilon})$