About PBRT_SAMPLED_SPECTRUM...

CIE D65 Illuminant

Wavelength (nm)
About PBRT_SAMPLED_SPECTRUM...

“rgb L” [1 1 1] “rgb scale” [1 1 1]

CIE D65 Illuminant

Wavelength (nm)
The Hinged View, Olafur Eliasson (via Eric Haines)
The Hinged View, Olafur Eliasson (via Eric Haines)
The Hinged View, Olafur Eliasson (via Eric Haines)
The Hinged View, Olafur Eliasson (via Eric Haines)
Ray Tracing

Ray Tracing 1

■ Basic algorithm
■ Ray-surface intersection (triangles, ...)

Ray Tracing 2

■ Problem: Brute force = \(|\text{Image}| \times |\text{Objects}|\)
■ Solution: Acceleration \(\approx |\text{Image}| \times \log |\text{Objects}|\)

Later

■ How to choose the best rays to trace?
■ How to minimize the number of rays traced?
Primitives

pbrt Primitive base class

- Shape
- Material (emission, reflection and transmission)

```cpp
class Primitive {
public:
    virtual Bounds3f WorldBound() const = 0;
    virtual bool Intersect(const Ray &r,
                           SurfaceInteraction *) const = 0;
    virtual bool IntersectP(const Ray &r) const = 0;
    virtual const AreaLight *GetAreaLight() const = 0;
    virtual const Material *GetMaterial() const = 0;
    virtual void ComputeScatteringFunctions(...) const = 0;
};
```
Primitives

Collections

- TransformedPrimitive: Transformation + primitive
- Aggregate
  - Treat acceleration data structures as primitives
  - Two types of accelerators: `kdtree.cpp`, and `bvh.cpp`
  - May nest accelerators of different types

```cpp
class Scene {
    // ...
    bool Intersect(const Ray &ray,
                   SurfaceInteraction *isect) const {
        return aggregate->Intersect(ray, isect);
    }
    std::shared_ptr<Primitive> aggregate;
};
```
300k tris
Time: Accel 27.4%, Ray-Triangle 10.2%
Avg. 5.8 tri intersect tests / ray
Memory: Accel 19 MB, Tris 21 MB
5.3M tris
Time: Accel 42.7%, Ray-Triangle 13.3%
Avg. 4.5 tri intersect tests / ray
Memory: Accel 339 MB, Tris 901 MB
3.1B tris
Time: Accel 74.7%, Ray-Triangle 13.3%
Avg. 32.9 tri isect tests / ray
Memory: Accel 1.47 GB, Tris 4.58 GB
Uniform Grids
Preprocess Scene

1. Find bounding box
Preprocess Scene

1. Find bounding box
2. Determine resolution
Preprocess Scene

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

1. Find bounding box
2. Determine resolution
3. Place objects in cells
   if bbox overlaps
   if surface intersects
Ray-Intersection

For each ray,
step through grid
3D line – 3D-DDA
Output all grid cells
the ray pierces
In 3D, 6-connected line
Resolution?

1 cell
No speedup
Resolution?

Large number of cells
Too many empty cells
Extra cost!
Resolution?

Heuristic

\[ n_v = n_x n_y n_z \propto n_o \]

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

\[ d \approx 3 \]
Objects Overlapping Multiple Cells

Mistake: Output first intersection found
Objects Overlapping Multiple Cells

Solution: Check whether intersection is inside cell
Objects Overlapping Multiple Cells

Solution: Check whether intersection is inside cell
Mailboxes

Solution: Check whether intersection is inside cell

Problem: Objects tested for intersection multiple times
Mailboxes

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 \( n \) w/ each primitive
  - Only re-intersect if ray \( n \) is greater than last ray \( n \)
Mailboxes and Parallelism...

Storing ray id with primitive doesn’t work well with many concurrent threads

- Per-thread id: uses lots of storage
- Single id: collisions (ok), cache overhead (bad)

Alternative: store primitive ids with ray

- e.g. a small hash table or an array of n primitive pointers
Uniform Grids: When They Work Well

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

http://www.kevinboulanger.net/grass.html
Uniform Grids: When They Work Poorly

Sphereflake

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
Nested Grids
Nested Grids

Refine grid cells with many primitives to have second-level grids in those cells.
Nested Grids

Refine grid cells with many primitives to have second-level grids in those cells.
Spatial Hierarchies
Spatial Hierarchies

Letters correspond to planes (A)
Spatial Hierarchies

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

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]

Intersect(L, t_{min}, t_{max})
Intersect(L, t_{min}, t^*)
Intersect(R, t_{min}, t_{max})
Intersect(R, t^*, t_{max})
Building the Hierarchy

Midpoint?  Median?
Which Hierarchy is Fastest?

What is the cost of tracing a ray?

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

\[
C\_\text{trav} = \text{cost of traversing a cell}
\]

\[
\text{Cost}(L) = \text{cost of traversing left child}
\]

\[
\text{Cost}(R) = \text{cost of traversing right child}
\]
Splitting with Cost in Mind
Split in the Middle = Bad!

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!

\[
\text{Cost(cell)} = C_{\text{trav}} + \text{Prob(hit L)} \times \text{Cost(L)} + \text{Prob(hit R)} \times \text{Cost(R)}
\]
The probability of ray in a given direction hitting an object with surface area $S$ is proportional to its projected area $A$ in the direction of the ray.
The probability of a ray in any direction hitting an object with surface area $S$ is proportional to its average projected area.

**Average projected area:**

$$
\overline{A} = \frac{1}{4\pi} \int A(\omega) \, d\omega
$$

**Crofton’s theorem:**

$$
\overline{A} = \frac{S}{4} \quad \text{(Convex shapes only)}
$$
Ray Intersection Probability

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

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

Need the probabilities

- Proportional to surface area

Need the child cell costs

- Triangle count is a good approximation

Cost(cell) = C_{trav} + SA(L) \times TriCount(L) + SA(R) \times TriCount(R)

C_{trav}: the ratio of cost to traverse to cost to intersect

C_{trav} = 1:80 in pbrt
C_{trav} = 1:1.5 in a highly optimized version
Build Algorithm

Note: Cost minimum must be at extrema
Build Algorithm

\[ p_a = \frac{S_a}{S} \]

\[ p_b = \frac{S_b}{S} \]

2n candidate split locations
Sweep Build Algorithm (Triangles)

1. Pick an axis, or optimize across $x$, $y$, $z$
2. Build a set of “candidate” split locations
3. Sort the triangles into intervals
4. Sweep to incrementally track L/R counts, cost
5. Output position of minimum cost split

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

$T(N) = N \log^2 N$
Termination Criteria

When should we stop splitting?
- **Bad**: depth limit, number of triangles
- **Good**: When split does not lower the cost

Threshold of cell size
- **Absolute probability** \( \frac{SA(\text{node})}{SA(\text{scene})} \) small
“Split Clipping”

Bounding box is conservative: may incorrectly report primitive overlap
## Results

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td># Triangles</td>
<td>600,127</td>
</tr>
<tr>
<td>Render time</td>
<td>19328s</td>
</tr>
<tr>
<td>kd-tree Build</td>
<td>0.43s (~0%)</td>
</tr>
<tr>
<td>kd-tree Traverse</td>
<td>7775s (40.0%)</td>
</tr>
<tr>
<td>Tri Intersect</td>
<td>6279s (32.5%)</td>
</tr>
<tr>
<td>Ray/Tri tests</td>
<td>107.4B</td>
</tr>
<tr>
<td>Ray/Tri hits</td>
<td>4.4B (4.1%)</td>
</tr>
</tbody>
</table>
Primitive Hierarchies

Bounding Volume Hierarchies (BVH)
Bounding Volume Hierarchies

Spatial subdivision (grids, kd-trees): subdivide 3D space into non-overlapping cells

Primitive subdivision (e.g. bounding volume hierarchies): partition scene primitives into disjoint sets (note bounding volumes may overlap)
Building BVH

Can also apply sweep build

\[ \text{Cost} = N_A \times S_A + N_B \times S_B \]
Building BVH

Can also apply sweep build

Consider all three axes during partitioning, choose lowest cost

Caveat: “greedy” algorithm; not guaranteed to give global optimum
BVH Traversal

stack<Node *> toVisit;
toVisit.push(root);
while (Node *cur = toVisit.pop()) {
    if (cur->bounds->Intersect(ray)) {
        if (cur->isLeaf) {
            /* intersect with primitives */
        } else {
            toVisit.push(cur->near);
            toVisit.push(cur->far);
        }
    }
}
# Results

<table>
<thead>
<tr>
<th></th>
<th>BVH</th>
<th>kd-tree</th>
</tr>
</thead>
<tbody>
<tr>
<td>Render time</td>
<td>12905s</td>
<td>19328s</td>
</tr>
<tr>
<td>Build</td>
<td>0.17s (~0%)</td>
<td>0.43s (~0%)</td>
</tr>
<tr>
<td>Traverse</td>
<td>6155s (47.7%)</td>
<td>7775s (40.0%)</td>
</tr>
<tr>
<td>Tri Intersect</td>
<td>2178s (16.9%)</td>
<td>6279s (32.5%)</td>
</tr>
<tr>
<td>Ray/Tri tests</td>
<td>22.6B</td>
<td>107.4B</td>
</tr>
<tr>
<td>Ray/Tri hits</td>
<td>3.6B (16%)</td>
<td>4.4B (4.1%)</td>
</tr>
</tbody>
</table>
kd-tree # Nodes Visited
BVH # Ray-Tri Tests
kd-tree # Ray-Tri Tests
Original Scene

Rendering time: 27m 38s
Transformed Scene

Rendering time: 1h 55m 45s
Wall and its bounding box

Rotated wall and its bounding box

Axis-alignment and performance
Axis-alignment and performance

Rotated wall and its bounding box

Work-around: refine bounding boxes
BVH # Ray-Tri Tests
Ray-Tracing Dynamic Scenes
Battlefield V
(EA/DICE)
Ray Tracing Dynamic Scenes

Challenge #1: scenes have millions of triangles, many objects are in motion
Challenge #2: relatively few rays traced per frame

For real-time, can allow a few ms / frame
- e.g. @10M tris, 60fps, need 600M tris / second
- pbrt BVH: ~2.5M tris / second

→ Hierarchy construction efficiency really matters
Fast (Parallel) Builds

Trade off BVH quality for build performance

Spatial sort using Morton ordering

Build lower-level BVHs in parallel

Build (small) top-level BVH
Refit rather than rebuild (then fix-up)

[Kopta et al. 2012]
Two-level Acceleration Structures

Top-level acceleration structure

Bottom-level acceleration structures
Computational geometry of ray shooting

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

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