Ray Intersection Acceleration

Readings
Chapter 2 – Geometry & Transformations
Chapter 3 – Shapes
Chapter 4 – Primitives & Intersection Acceleration
We’ll cover this in class

Covers basic math and PBRT implementation: read on your own

Reading
Chapter 2: Geometry and Transformations
2.1-2.5 Review basic geometry
2.6 3D Bounding boxes in PBRT
2.7-2.8 Transformation & applying them in PBRT
2.5.1 & 2.9 Differential geometry

Reading
Chapter 3: Shapes
3.1 Basic PBRT shape interface
3.2-3.5 Specific shapes - quadrics
3.6 Triangles and meshes
3.7 Subdivision surfaces

Ray Tracing
• Shoot a ray through each pixel;
• Find first object intersected by ray
Compute ray. (More linear algebra.)
Compute ray-object intersection.
Spawn more rays for reflection and refraction

Ray Tracing Architecture

Optimizing Ray Tracing
• Main computation load is ray-object intersection
• 50-90% of run time when profiled
• Test for possible intersection before committing to computing intersections
Consider this

**Complexity!**
- I rays or pixels in image
- N objects
- $O(NI)$
- Can we do $O(I \log N)$?

**Ray Intersection Acceleration**

Ray Tracing Acceleration Techniques
- Faster Intersections
- Fewer Rays
- Generalized Rays

- Faster ray-object intersections
- Fewer ray-object intersections

Examples:
- Object bounding volumes
- Efficient intersectors for parametric surfaces, fractals, etc.

Examples:
- Bounding volume hierarchies
- Space subdivision
- Directional techniques

Examples:
- Adaptive tree-depth control
- Statistical optimizations for anti-aliasing
- Beam tracing
- Cone tracing
- Pencil tracing

**Pbrt and Intersections**

- Primitive base class
- Shapes are subclasses of primitive
- Aggregate class
- Methods
  - WorldBound
  - CanIntersect
  - Intersect
  - IntersectP
  - Refine
- First four return Intersection structures
- Last returns Primitives

**Intersection Geometry**

- Shape independent representation for intersections
- DifferentialGeometry Intersection::dg
  - Point $P$
  - Normal $N$
  - Parametric $(u,v)$
  - Partial derivatives
  - Tangents: $dpdu, dpdv$
  - Change in normal: $dndu, dndv$
Speeding up Intersection Calculation
Object-based vs. World-based

- Common dichotomy in graphics
  - objects situated in (world) space
  - (world) space in which objects reside
- Bounding volumes are object-based
- Spatial Subdivision is world-based approach
- Sub-linear search – logarithmic ?

Bounding Volumes

- Surround object with a simple volume
- Test ray against volume first
- Test object-space or world-space bound? (pros and cons)
- Cost model: \( N^*c_b + p_i^*N^*c_o \)
  - \( N \) (number of rays) is given
  - \( p_i \) – fraction of rays intersecting bounding volume
  - Minimize \( c_b \) (cost of intersecting bounding volume)
  - \( c_o \) (cost of intersecting object)
  - Reduce ray path
  - Minimize cost/fit ratio

Bounding Volumes

- Bounding sphere
  - Difficult to compute good one
  - Easy to test for intersection
- Bounding box
  - Easy to compute for given object
  - Relatively difficult to intersect (maybe ?)

Pbrt's Bounding Boxes

- Virtual BBox ObjectBound() const=0;
- Virtual BBox WorldBound() const {
  return ObjectToWorld(ObjectBound());
}
- Bool BBox::IntersectP(Const Ray &ray, Float *hit0, Float *hit1) const {

Bounding Box

- Compute min/max for x,y,z
- 3 options
  - Compute in world space
    - Chance of ill fitting b-box
  - Compute in object space and transform w/object
    - Object space b-box probably better fit than world space
    - Need to intersect ray with arbitrary hexahedral in world sp.
  - Compute in object space and test in object space
    - Inverse transform ray into object space
**Ray & Cube**

\[ P(t) = s + tc \]

\[ t_{x1} = \frac{(x1 - s)}{c} \]

\[ t_{x2} = \frac{(x2 - s)}{c} \]

\[ t_{y1} = \frac{(y1 - s)}{c} \]

\[ \ldots \]

**Square/Cube**

Note entering and leaving intersections separately

Ray is inside after last entering and before first leaving

**Algorithm**

set \( T_{near} = -\infty \), \( T_{far} = \infty \)

\[ \text{Ray } (t) = O + t \times \text{Ray} \]

For each pair of planes \( P \) associated with \( X, Y, \) and \( Z \) do:

1. if direction \( \text{Ray} \times x = 0 \) then the ray is parallel to the \( X \) planes
2. if origin \( Ox \) is not between the slabs \( (Ox < X_l \text{ or } Ox > X_h) \) then return false
3. if the ray is not parallel to the plane then:
   - compute the intersection distance of the planes
   - if \( T1 > T2 \) swap \( (T1, T2) \) - since \( T1 \) intersection with near plane
   - if \( T1 > T_{near} \) \( T_{near} = T1 \) - want largest \( T_{near} \)
   - if \( T2 < T_{far} \) \( T_{far} = T2 \) - want smallest \( T_{far} \)
4. If Box survived all above tests, return true with intersection point \( T_{near} \) and exit point \( T_{far} \)

**Bounding Sphere**

- Find min/max points in \( x,y,z \) -> 3 pairs
- Use maximally separated pair to define initial sphere
- For each point:
  - If point is outside of current sphere, increase old sphere to just include new point

**Bounding Slabs**

- More complex to compute
- Better fit of object
- Use multiple pairs of parallel planes to bound object
- Can add more slabs to get tighter fit

© 2004 Pharr, Humphreys
**Bounding Slabs**
- Use algorithm for axis aligned bounding box
- Intersect ray with arbitrary plane \( P \cdot N = D \)

\[
P(t) \cdot N = D_1
\]
\[
(P + tR) \cdot N = D_2
\]
\[
t = \frac{D_2 - P \cdot N}{R \cdot N}
\]

**Approximate Convex Hull**
- Find highest vertex
- Find plane through vertex parallel to ground plane
- Find second vertex that makes minimum angle with first vertex and up vector
- Find third vertex that makes plane whose normal makes minimum angle with up vector

For any unmatched edge, find unused vertex such that the plane of the vertex and edge makes a minimum angle with the plane of edge's face

**Hierarchical Bounding Volumes**
- Compute bounding volume for groups of objects
- Compute bounding volume for groups of groups of objects

**Problem**
- Subtrees overlap
- Does not contain all objects it overlaps
- Balance

**Spatial Enumeration**
- Divide space into ‘voxels’
- Bucket sort objects in voxels they intersect
  - Object goes into each voxel it touches
  - Reuse results from one voxel calculation
- Determine voxels that a ray intersects
  - Only deal with the objects in those voxels
Spatial Enumeration

- Identifying voxels hit is like a line drawing algorithm

Uniform Grids

- Preprocess scene
- Find Big bounding box

- Determine grid resolution (how?)

- Place object in cell if its bounding box overlaps the cell

- Check that object overlaps cell (expensive!)

Add Sorting

- If objects/voxels/cells are processed in front-to-back sorted order, stop processing when first intersection is detected

- e.g., process cells in bottom to top, left to right order and stop at first intersection
**Uniform Grids**

- Preprocess scene
- Traverse grid
  - 3D line = 3D-DDA
  - 6-connected line
- pbrt algorithm (grid accelerator)

---

**Amanatides & Woo Algorithm**

```
Amanatides & Woo Algorithm
```


- `Step[X,Y] +/- 1`
- `tMax[X,Y]` – first intersection
- `tDelta[X,Y]` - voxel distance in `[X,Y]`

---

**A&W Algorithm Results**

- Rendering time for different levels of subdivision

---

**Objects Across Multiple Voxels**

- Mailboxes eliminate redundant intersection tests
- Objects have mailboxes
- Assign rays numbers
- check against objects last tested ray number
- Intersection must be within current voxel

---

**Hierarchical Spatial Subdivision**

- Recursive subdivision of space
- 1-1 Relationship between scene points and leaf nodes
- Example: point location by recursive search (log time)
- Solves the lack-of-adaptivity problem
- DDA works
- Effective in practice

---
**Creating Spatial Hierarchies**

```c
Insert(node, prim) {
    if (overlap(node->bound, prim)) {
        if (leaf(node)) {
            if (node->nprims > MAXPRIMS && node->depth < MAXDEPTH) {
                foreach child in node
                    insert(child, prim);
            } else list_insert(node->prims, prim);
            foreach child in node
                insert(child, prim);
        } else list_insert(node->prims, prim);
    }
    // Typically MAXDEPTH=16, MAX PRIMS = 2-8
```

**Comparison**

<table>
<thead>
<tr>
<th>Scheme</th>
<th>Spheres</th>
<th>Rings</th>
<th>Tree</th>
</tr>
</thead>
<tbody>
<tr>
<td>Uniform grid</td>
<td>D=1</td>
<td>244</td>
<td>129</td>
</tr>
<tr>
<td></td>
<td>D=20</td>
<td>38</td>
<td>83</td>
</tr>
<tr>
<td>Hierarchical grid</td>
<td></td>
<td>34</td>
<td>116</td>
</tr>
</tbody>
</table>


**Questions?**

- “Teapot in a stadium” versus uniform distribution
- Multiplicative constants important
- Adaptivity allows robustness
- Cache effects are important