Lecture 7:
Ray Tracing Intersection Acceleration

Image Synthesis
Stanford CS348b, Spring 2014
Efficient Ray Tracing

- **Today:**
  - Problem: brute force = $|\text{Image}| \times |\text{Objects}|$
  - Solution: culling $\approx |\text{Image}| \times \log |\text{Objects}|$

- **Next lecture:**
  - Low-level issues in efficient intersection calculations

- **Later**
  - How to minimize the number of rays traced?
  - How to choose rays (sample) efficiently?
Uniform Grids

- Preprocess scene
  - Find bounding box
Uniform Grids

- Preprocess scene
  - Find bounding box
  - Determine resolution
Uniform Grids

- Preprocess scene
  - Find bounding box
  - Determine resolution
  - Place objects in cell if bbox overlaps cell
Uniform Grids

- Preprocess scene
  - Find bounding box
  - Determine resolution
  - Place objects in cell if bbox overlaps cell, if surface intersects cell
Uniform Grids

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

\[ n_v = n_x n_y n_z \propto n_o \]
\[ \max(n_x, n_y, n_z) = d^{\frac{3}{2}}n_o \]
\[ d \approx 3 \]
Grid Traversal State

- Current cell \((x, y, z)\)
- Distance along ray to next cell \((x, y, z)\)
- Distance to closest found cell

- Visit cells in ray order until hit found is inside the current cell
Caveat: Overlap

Mistake: output first intersection found!
Caveat: Overlap

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: primitive intersection cache (mailbox)
  - Assign each ray an increasing number
  - Store last checked ray id with each primitive
  - Don’t re-test if ids match

- Problem: more complex with multiple threads...
When Grids Work Well

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

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

“The 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).


Problem: non-uniform size distribution, varying density
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

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)
Boxes at leaves correspond to regions
Variations

kd-tree

oct-tree

bsp-tree
Building The Hierarchy

Midpoint?  Median?
Which Hierarchy Is Fastest?

What is the cost of tracing a ray?

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

\(C_{\text{visit}} = \text{cost of visiting a note}\)

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

\(\text{Cost}(\text{R}) = \text{cost of traversing right child}\)
Splitting With Cost In Mind
Split In The Middle: Bad!

Midpoint: makes left and right probabilities equal
Cost of R greater than cost of L
Split At The Median: Bad!

Median: makes left and right costs equal
Probability of hitting L greater than R
Cost-Optimized Split

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

- Need the probabilities
  - Turn out to be proportional to the surface area
- Need the child cell costs
  - Triangle count is a good approximation

\[
\text{Cost(cell)} = C_{\text{visit}} + \text{SurfArea}(L) \times \text{TriCount}(L) + \text{SurfArea}(R) \times \text{TriCount}(R)
\]

\(C_{\text{trav}}\) 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

The number of rays in a given direction that hit an object with surface area $S$ is proportional to its projected area $A$

Average projected area: $\bar{A} = \frac{1}{4\pi} \int A \, d\omega$

Crofton’s theorem: $\bar{A} = \frac{S}{4}$  \hspace{1cm} (Convex shapes only)
Surface Area and Ray Intersection Probability

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

\[ Pr[r \cap S_o | r \cap S_c] = \frac{S_o}{S_c} \]
kd-Tree Sweep Build Algorithm

2n candidate splits
kd-Tree Sweep Build Algorithm

\[ p_a = \frac{S_a}{S} \quad p_b = \frac{S_b}{S} \]

\[ N_a \quad N_b \]
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, costs
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 cost improvement
  - Stretch over multiple levels—e.g., terminate if cost doesn’t go down after three splits in a row

- Threshold of cell size
  - Absolute probability SA(node)/SA(scene) low
kd-Tree Representation

- 8 bytes per node
- Compact encoding of topology:
  - Left child next node in array
  - Right child at aboveChild
- Flags encode split axis

```c
struct KdAccelNode {
    union {
        float split;            // Interior
        uint32_t onePrimitive;  // Leaf
        uint32_t *primitives;   // Leaf
    };
    union {
        uint32_t flags;         // Both
        uint32_t nPrims;        // Leaf
        uint32_t aboveChild;    // Interior
    };
    uint32_t nPrimitives() { return nPrims >> 2; } 
    uint32_t SplitAxis() { return flags & 3; } 
    bool IsLeaf() { return (flags & 3) == 3; } 
    uint32_t AboveChild() { return aboveChild >> 2; } 
};
```

```c
vector<KdAccelNode> nodes;
vector<Primitive> primitives;
```
Recursive Inorder Traversal

$t* = (S - O[a]) / D[a]$
Primitive 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
BVH Node Representation

- Same encoding of topology as kd-tree
- 32 bytes (24 of which for the bounding box...)

```cpp
struct LinearBVHNode {
    BBox bounds;
    union {
        uint32_t primitivesOffset;    // leaf
        uint32_t secondChildOffset;   // interior
    };

    uint8_t nPrimitives;  // 0 -> interior node
    uint8_t axis;         // interior node: xyz
    uint8_t pad[2];       // ensure 32 byte total size
};
vector<LinearBVHNode> tree;
vector<Primitive> primitives;
```
Building Primitive Hierarchies

- Can also apply sweep build

Candidate split cost = $N_A \times S_A + N_B \times S_B$
Building Primitive Hierarchies

- Can also apply sweep build

Candidate split cost = $N_A * S_A + N_B * S_B$
Building Primitive Hierarchies

- Can also apply sweep build
  - May consider all three axes for primitive partitioning

\[
\text{Candidate split cost} = N_A \times S_A + N_B \times S_B
\]

(This is probably the optimal one)
Theory

- Computational Geometry of Ray Shooting

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

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