Physically Based Rendering (600.657)

Intersection Acceleration
Intersection Testing

Accelerated techniques try to leverage:

– **Grouping**: To efficiently discard groups of primitives that are guaranteed to be missed by the ray.

– **Ordering**: To test nearer intersections first and allow for early termination if there is a hit.
Intersection Testing

Two Approaches:

1. Space partitions:
   Decompose space into cells and assign primitives to the cells in which they fall.
   ✗ A primitive may appear in multiple cells

2. Object partitions:
   Group objects into clusters
   ✗ Cluster volumes may overlap
Uniform (Voxel) Grid Acceleration

• Construct uniform grid over scene
  – Index primitives according to overlaps with grid cells

• A primitive may belong to multiple cells
• A cell may have multiple primitives
Uniform (Voxel) Grid Acceleration

- Trace rays through grid cells
  - Fast
  - Incremental

Only check primitives in intersected grid cells
Uniform (Voxel) Grid Acceleration

Traversal:

1. Find initial $x$, $y$ intersections and $t$ values
Uniform (Voxel) Grid Acceleration

Traversal:

1. Find initial $x$, $y$ intersections and $t$ values
2. While inside:
Uniform (Voxel) Grid Acceleration

Traversal:

1. Find initial $x$, $y$ intersections and $t$ values
2. While inside:
   1. Find closer hit dir.
Uniform (Voxel) Grid Acceleration

Traversal:

1. Find initial $x$, $y$ intersections and $t$ values

2. While inside:
   1. Find closer hit dir.
   2. Test prims. in voxel
Uniform (Voxel) Grid Acceleration

Traversal:

1. Find initial x, y intersections and t values

2. While inside:
   1. Find closer hit dir.
   2. Test prims. in voxel

3. If hit inside voxel:
   – Done!
Uniform (Voxel) Grid Acceleration

Traversal:

1. Find initial $x$, $y$ intersections and $t$ values

2. While inside:
   1. Find closer hit dir.
   2. Test prims. in voxel
   3. If hit inside voxel:
      – Done!
   4. Update current $t$
Uniform (Voxel) Grid Acceleration

Traversal:

1. Find initial x, y intersections and t values

2. While inside:
   1. Find closer hit dir.
   2. Test prims. in voxel
   3. If hit inside voxel:
      – Done!
   4. Update current t
Uniform (Voxel) Grid Acceleration

Traversal:

1. Find initial $x$, $y$ intersections and $t$ values
2. While inside:
   1. Find closer hit dir.
   2. Test prims. in voxel
   3. If hit inside voxel:
      - Done!
   4. Update current $t$
Uniform (Voxel) Grid Acceleration

Traversal:

1. Find initial $x$, $y$ intersections and $t$ values

2. While inside:
   1. Find closer hit dir.
   2. Test prims. in voxel
   3. If hit inside voxel:
      - Done!
   4. Update current $t$
Uniform (Voxel) Grid Acceleration

Traversal:

1. Find initial $x$, $y$ intersections and $t$ values

2. While inside:
   
   1. Find closer hit dir.
   2. Test prims. in voxel
   
2. While inside:
   
   1. Find closer hit dir.
   2. Test prims. in voxel
   3. If hit inside voxel:
      
      - Done!
   4. Update current $t$

Could’t we just stop the first time we hit C?
Uniform (Voxel) Grid Acceleration

Traversal:

1. Find initial $x, y$ intersections and $t$ values

2. While inside:
   1. Find closer hit dir.
   2. Test prims. in voxel
   3. If hit inside voxel:
      – Done!
   4. Update current $t$
Uniform (Voxel) Grid Acceleration

Traversal:

1. Find initial $x$, $y$ intersections and $t$ values

2. While inside:
   1. Find closer hit dir.
   2. Test prims. in voxel
   3. If hit inside voxel:
      – Done!
   4. Update current $t$
Uniform (Voxel) Grid Acceleration

Traversal:

1. Find initial $x, y$ intersections and $t$ values
2. While inside:
   1. Find closer hit dir.
   2. Test prims. in voxel
   3. If hit inside voxel:
      - Done!
   4. Update current $t$
Uniform (Voxel) Grid Acceleration

Traversal:

1. Find initial $x$, $y$ intersections and $t$ values

2. While inside:
   1. Find closer hit dir.
   2. Test prims. in voxel
   3. If hit inside voxel:
      - Done!
   4. Update current $t$
Uniform (Voxel) Grid Acceleration

Traversal:

1. Find initial $x, y$ intersections and $t$ values
2. While inside:
   1. Find closer hit dir.
   2. Test prims. in voxel
   3. If hit inside voxel:
      - Done!
   4. Update current $t$
Uniform (Voxel) Grid Acceleration

• Potential problem:
  – How choose suitable grid resolution?

Too little benefit if grid is too coarse

Too much cost if grid is too fine
Bounding Volumes

• Check for intersection with the bounding volume:
  – Bounding cubes
  – Bounding boxes
  – Bounding spheres
  – Etc.

{ Stuff that’s easy to intersect }
Bounding Volumes

• Check for intersection with the bounding volume
Bounding Volumes

• Check for intersection with the bounding volume
  – If ray doesn’t intersect bounding volume, then it doesn’t intersect its contents
Bounding Volumes

• Check for intersection with the bounding volume
  – If ray doesn’t intersect bounding volume, then it doesn’t intersect its contents

Still need to check for intersections with shape.
Bounding Volume Hierarchies

• Build (binary) hierarchy of bounding volumes
  – Bounding volume of interior node contains all children
Bounding Volume Hierarchies

- Grouping acceleration

```plaintext
Float FindIntersection(Ray ray, Node node) {
    min_t = -1

    if( !intersect ( node.boundingVolume ) ) // Test Bounding box
        return min_t

    foreach shape { // Test node’s shape
        t = Intersect( shape )
        if( t>0 && (t<min_t || min_t<0) ) min_t = t
    }

    for each child { // Test node’s children
        t = FindIntersection(ray, child)
        if (t>0 && (t < min_t || min_t<0) ) min_t = t
    }
    return min_t;
}
```
Bounding Volume Hierarchies

• Build (binary) hierarchy of bounding volumes
  – Bounding volume of interior node contains all children
Bounding Volume Hierarchies

• Build (binary) hierarchy of bounding volumes
  – Bounding volume of interior node contains all children

• Don’t need to test shapes A or B
• Need to test groups 1, 2, 3, 4, and 5
• Need to test shapes C, D, E, and F
Bounding Volume Hierarchies

- Grouping + Ordering acceleration

```c
Float FindIntersection(Ray ray, Node node) {
    // Find intersections with the shapes of the node
    ...
    // Find intersections with child node bounding volumes
    ...
    // Sort child intersections front to back
    ...
    // Process intersections (checking for early termination)
    for each intersected child  {
        if (min_t < bv_t[child]) break;
        t = FindIntersection(ray, child);
        if (t>0 && (t < min_t || min_t<0)) min_t = t
    }
    return min_t
}
```
Bounding Volume Hierarchies

• Build (binary) hierarchy of bounding volumes
  – Bounding volume of interior node contains all children
    • Don’t need to test shapes A, B, D, E, or F
    • Need to test groups 1, 2, 3, 4, and 5
    • Need to test shape C
Bounding Volume Hierarchies

• How do we build up the bounding volume hierarchy?
Bounding Volume Hierarchies

• How do we build up the bounding volume hierarchy?
  Choose an axis:

  1. Evenly split primitives based on centroids.
Bounding Volume Hierarchies

• How do we build up the bounding volume hierarchy?
  Choose an axis:

1. Evenly split primitives based on centroids.
Bounding Volume Hierarchies

• How do we build up the bounding volume hierarchy?
  Choose an axis:
  1. Evenly split primitives based on centroids.
  2. Find the center and split left/right.
Bounding Volume Hierarchies

• How do we build up the bounding volume hierarchy?
  Choose an axis:

1. Evenly split primitives based on centroids.
2. Find the center and split left/right.
Bounding Volume Hierarchies

• How do we build up the bounding volume hierarchy?

3. **Surface area heuristic:**

\[
\min\left( c_{\text{isect}} N, c_{\text{trav}} + c_{\text{isect}} \left( p_A N_A + p_B N_B \right) \right)
\]

- \( c_{\text{isect}} \): Cost of testing for intersection
- \( c_{\text{trav}} \): Cost of traversing a bounding volume
- \( N \): number of primitives in bounding volume
- \( N_{A/B} \): number of primitives in child \( A/B \)
- \( p_{A/B} \): probability that a ray intersects the bounding volume will intersect the child volume \( A/B \)
Minimum Area Heuristic

$$\min(c_{\text{isect}}N_c + c_{\text{isect}}(p_A N_A + p_B N_B))$$

**Challenge 1:**
How do you assign probabilities?

**Challenge 2:**
How do you choose the one out of $2^{N-1}-1$ different (non-trivial) ways to partition?
Minimum Area Heuristic

$$\min\left(c_{\text{isect}} N, c_{\text{trav}} + c_{\text{isect}} \left(p_A N_A + p_B N_B\right)\right)$$

Challenge 1:

How do you assign probabilities?

For two convex shapes $A \subseteq B$, the probability that a ray entering $B$ will also enter $A$ is:

$$\frac{\text{Area}(\partial A)}{\text{Area}(\partial B)}$$
Minimum Area Heuristic

$$\min(c_{\text{isect}} N, c_{\text{trav}} + c_{\text{isect}} (p_A N_A + p_B N_B))$$

**Challenge 2:**

How do you choose the one out of $2^{N-1}-1$ different (non-trivial) ways to partition?

Splitting by looking at centroids relative to a split position, there are only N-1 split positions to consider.
Minimum Area Heuristic

$$\min\left(c_{\text{isect}} N, c_{\text{trav}} + c_{\text{isect}} (p_A N_A + p_B N_B)\right)$$

**Challenge 2:**

How do you choose the one out of $2^{N-1}-1$ different (non-trivial) ways to partition?

Splitting by looking at centroids relative to a split position, there are only N-1 split positions to consider.

Choose the cost minimizer among these possibilities.
Minimum Area Heuristic

$$\min\left(c_{\text{isect}} N, c_{\text{trav}} + c_{\text{isect}} \left(p_A N_A + p_B N_B \right)\right)$$

Challenge 2:
How do you choose the one out of $2^{N-1}-1$ different (non-trivial) ways to partition?

Splitting by looking at centroids relative to a split position, there are only $N-1$ split positions to consider.

Choose the cost minimizer.

All splits between successive centers give the same partition.
Kd Trees

• Split along the coordinate axes
Kd Trees

• Split along the coordinate axes
Kd Trees

- Split along the coordinate axes
Kd Trees

- Split along the coordinate axes
Kd Trees

• Split along the coordinate axes
Kd Trees

- Split along the coordinate axes
Kd Trees

• Split along the coordinate axes

Note:
• Primitives may belong to multiple nodes.
Minimum Area Heuristic

\[
\min \left( c_{\text{isect}} N, c_{\text{trav}} + c_{\text{isect}} \left( p_A N_A + p_B N_B \right) \right)
\]

Challenge 2:
How do you choose a partition location?
Minimum Area Heuristic

\[
\min\left( c_{\text{isect}} N, c_{\text{trav}} + c_{\text{isect}} \left( p_A N_A + p_B N_B \right) \right)
\]

**Challenge 2:**
How do you choose a partition location?

Look at the end-points of the projection of the bounding boxes on the axes.
Minimum Area Heuristic

\[
\min \left( c_{\text{isect}} N, c_{\text{trav}} + c_{\text{isect}} \left( p_A N_A + p_B N_B \right) \right)
\]
Minimum Area Heuristic

\[ \min \left( c_{\text{isect}} N, c_{\text{trav}} + c_{\text{isect}} (p_A N_A + p_B N_B) \right) \]

**Challenge 2:**

How do you choose a partition location?

Look at the end-points of the projection of the bounding boxes on the axes.

Choose the partition that minimizes the cost.

All splits between successive end-points give the same partition.
Minimum Area Heuristic

\[
\min \left( c_{isect} N, c_{trav} + c_{isect} \left( p_A N_A + p_B N_B \right) \right)
\]

Challenge 2:
How do you choose a partition location?

For efficiency, generate \( k \) buckets and only consider partitioning on bucket boundaries.

Choose the partition that minimizes the cost.
Minimum Area Heuristic

$$\min\left( c_{\text{isect}}N, c_{\text{trav}} + c_{\text{isect}}(p_A N_A + p_B N_B) \right)$$

Challenge 2:
How do you choose a partition location?

For efficiency, generate $k$ buckets and only consider partitioning on bucket boundaries.

Choose the partition that minimizes the cost.
Minimum Area Heuristic

$$\min\left( c_{\text{isect}} N, c_{\text{trav}} + c_{\text{isect}} \left( p_A N_A + p_B N_B \right) \right)$$

Challenge 2:
How do you choose a partition location?

For efficiency, generate $k$ buckets and only consider partitioning on bucket boundaries.

Choose the partition that minimizes the cost.
Minimum Area Heuristic

\[
\min \left( c_{\text{isect}} N, c_{\text{trav}} + c_{\text{isect}} \left( p_A N_A + p_B N_B \right) \right)
\]

Challenge 2:
How do you choose a partition location?
For efficiency, generate \( k \) buckets and only consider partitioning on bucket boundaries.
Choose the partition that minimizes the cost.
Minimum Area Heuristic

**Note:**
In contrast to bounding volumes, it may make sense to create kd-tree nodes with no primitives (to cull out large empty space):

\[
\min\left(c_{\text{isect}}N, c_{\text{trav}} + c_{\text{isect}} \left( p_A N_A + p_B N_B \right) \right)
\]

\[
\min\left(c_{\text{isect}}N, c_{\text{trav}} + c_{\text{isect}} \left( 1 - \beta \right) \left( p_A N_A + p_B N_B \right) \right)
\]

with \(0 \leq \beta \leq 1\) and \(\beta = 0\) if \(N_A, N_B \neq N\)