Accelerating Ray-Scene Intersection Calculations

Connelly Barnes

CS 4810: Graphics

Acknowledgment: slides by Jason Lawrence, Misha Kazhdan, Allison Klein, Tom Funkhouser, Adam Finkelstein and David Dobkin
Overview

• Acceleration techniques
  o Bounding volume hierarchies
  o Spatial partitions
    » Uniform grids
    » Octrees
    » BSP trees
Goal

- Find intersection with front-most primitive in group

Intersection FindIntersection(Ray ray, Scene scene)
{
    min_t = ∞
    min_shape = NULL
    For each primitive in scene
    {
        t = Intersect(ray, primitive);
        if (t > 0 and t < min_t) then
            min_shape = primitive
            min_t = t
    }
    return Intersection(min_t, min_shape)
}
Acceleration Techniques

• A direct approach tests for an intersection of every ray with every primitive in the scene.

• Acceleration techniques:
  o Grouping:
    Group primitives together and test if the ray intersects the group. If it doesn’t, don’t test individual primitives.
  o Ordering:
    Test primitives/groups based on their distance along the ray. If you find a close hit, don’t test distant primitives/groups.
Bounding Volumes

• Check for intersection with the bounding volume:
  o Bounding cubes
  o Bounding boxes
  o Bounding spheres
  o 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 hierarchy of bounding volumes
  o Bounding volume of interior node contains all children
Bounding Volume Hierarchies

• Grouping acceleration

```c
FindIntersection(Ray ray, Node node) {
    min_t = \infty
    min_shape = NULL
    
    // Test if you intersect the bounding volume
    if( !intersect ( node.boundingVolume ) ) {
        return (min_t,min_shape);
    }
    
    // Test the children
    for each child {
        (t, shape) = FindIntersection(ray, child)
        if (t < min_t) {min_shape=shape}
    }
    return (min_t, min_shape);
}
```
Bounding Volume Hierarchies

• Use hierarchy to accelerate ray intersections
  o Intersect node contents only if hit bounding volume
Bounding Volume Hierarchies

• Use hierarchy to accelerate ray intersections
  ○ Intersect node contents only if hit bounding volume

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

• Grouping + Ordering acceleration

```c
FindIntersection(Ray ray, Node node) {
    // Find intersections with child node bounding volumes
    ...
    // Sort intersections front to back
    ...
    // Process intersections (checking for early termination)
    min_t = ∞
    min_shape = NULL
    for each intersected child {
        if (min_t < bv_t[child]) break;
        (t, shape) = FindIntersection(ray, child);
        if (t < min_t) {
            min_t = t
            min_shape = shape
        }
    }
    return (min_t, min_shape);
}
```
Bounding Volume Hierarchies

- Use hierarchy to accelerate ray intersections
  - Intersect nodes only if you haven’t hit anything closer
Bounding Volume Hierarchies

• Use hierarchy to accelerate ray intersections
  • Intersect nodes only if you haven’t hit anything closer

• Don’t need to test shapes A, B, D, E, or F
• Need to test groups 1, 2, and 3
• Need to test shape C
Overview

• Acceleration techniques
  • Bounding volume hierarchies
  • Spatial partitions
    » Uniform grids
    » Octrees
    » BSP trees
Uniform (Voxel) Grid

- 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

• Trace rays through grid cells
  o Fast
  o Incremental

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

- Potential problem:
  - How choose suitable grid resolution?

Too much cost if grid is too fine

Too little benefit if grid is too coarse
“Teapot in a Stadium” Problem

Could have much complicated geometry (e.g. a teapot) inside a single cell of the voxel grid. Why is this problematic?
Ray-Scene Intersection

» Acceleration techniques
  o Bounding volume hierarchies
  o Spatial partitions
    » Uniform grids
    » Octrees
    » BSP trees
Octrees

- We can think of a voxel grid as a tree.
  - The root node is the entire region
  - Each node has eight children obtained by subdividing the parent into eight equal regions
Octrees

- We can think of a voxel grid as a tree.
  - The root node is the entire region
  - Each node has eight children obtained by subdividing the parent into eight equal regions
Octrees

- We can think of a voxel grid as a tree.
  - The root node is the entire region
  - Each node has eight children obtained by subdividing the parent into eight equal regions
Octrees

- We can think of a voxel grid as a tree.
  - The root node is the entire region
  - Each node has eight children obtained by subdividing the parent into eight equal regions
Octrees

- In an octree, we only subdivide regions that contain more than one shape.
Octrees

- In an octree, we only subdivide regions that contain more than one shape.
Octrees

- In an octree, we only subdivide regions that contain more than one shape.
Octrees

• In an octree, we only subdivide regions that contain more than one shape.
Octrees

- In an octree, we only subdivide regions that contain more than one shape.
Octrees

- In an octree, we only subdivide regions that contain more than one shape.
- Adaptively determines grid resolution.
Ray-Scene Intersection

- Intersections with geometric primitives
  - Sphere
  - Triangle

  > Acceleration techniques
  - Bounding volume hierarchies
  - Spatial partitions
    - Uniform (Voxel) grids
    - Octrees
    - BSP trees
Binary Space Partition (BSP) Tree

- Recursively partition space by planes
Binary Space Partition (BSP) Tree

- Recursively partition space by planes
  - Generate a tree structure where the leaves store the shapes.
Binary Space Partition (BSP) Tree

- Recursively partition space by planes
  - Generate a tree structure where the leaves store the shapes.
Binary Space Partition (BSP) Tree

• Recursively partition space by planes
  o Generate a tree structure where the leaves store the shapes.
Binary Space Partition (BSP) Tree

- Recursively partition space by planes
  - Generate a tree structure where the leaves store the shapes.
Binary Space Partition (BSP) Tree

- Recursively partition space by planes
  - Generate a tree structure where the leaves store the shapes.
Binary Space Partition (BSP) Tree

- Recursively partition space by planes
  - Every cell is a convex polyhedron
Binary Space Partition (BSP) Tree

- Example: Point Intersection
Binary Space Partition (BSP) Tree

- Example: Point Intersection
  - Recursively test what side we are on
Binary Space Partition (BSP) Tree

- Example: Point Intersection
  - Recursively test what side we are on
    - Left of 1 (root) → 2
Binary Space Partition (BSP) Tree

- Example: Point Intersection
  - Recursively test what side we are on
    » Left of 2 → 4
Binary Space Partition (BSP) Tree

- Example: Point Intersection
  - Recursively test what side we are on
    - Right of 4 → Test B
Binary Space Partition (BSP) Tree

- Example: Point Intersection
  - Recursively test what side we are on
    - Missed B. No intersection!
Binary Space Partition (BSP) Tree

- Example: Ray Intersection 1
Binary Space Partition (BSP) Tree

- Example: Ray Intersection 1
  - Recursively split the ray and test nearer and farther halves, nearest first. Stop once you hit something:
    » Test half to the left of 1
Binary Space Partition (BSP) Tree

• Example: Ray Intersection 1
  - Recursively split the ray and test nearer and farther halves, nearest first. Stop once you hit something:
  - Test half to the right of 2
Binary Space Partition (BSP) Tree

• Example: Ray Intersection 1
  • Recursively split the ray and test nearer and farther halves, nearest first. Stop once you hit something:
    » Intersection with C. Done!
Binary Space Partition (BSP) Tree

• Example: Ray Intersection 2
  o Recursively split the ray and test nearer and farther halves, nearest first. Stop once you hit something:
    » Test half to the left of 1
Binary Space Partition (BSP) Tree

• Example: Ray Intersection 2
  • Recursively split the ray and test nearer and farther halves, nearest first. Stop once you hit something:
    » Test half to the right of 2
Binary Space Partition (BSP) Tree

- Example: Ray Intersection 2
  - Recursively split the ray and test nearer and farther halves, nearest first. Stop once you hit something:
    » Missed C. Recurse!
Binary Space Partition (BSP) Tree

- Example: Ray Intersection 2
  - Recursively split the ray and test nearer and farther halves, nearest first. Stop once you hit something:
    - Test half to left of 2
Binary Space Partition (BSP) Tree

• Example: Ray Intersection 2
  - Recursively split the ray and test nearer and farther halves, nearest first. Stop once you hit something:
    » Test half to left of 4
Binary Space Partition (BSP) Tree

- Example: Ray Intersection 2
  - Recursively split the ray and test nearer and farther halves, nearest first. Stop once you hit something:
    » Missed A. Recurse!
Binary Space Partition (BSP) Tree

- Example: Ray Intersection 2
  - Recursively split the ray and test nearer and farther halves, nearest first. Stop once you hit something:
    - No half to right of 4.
Binary Space Partition (BSP) Tree

- Example: Ray Intersection 2
  - Recursively split the ray and test nearer and farther halves, nearest first. Stop once you hit something:
    » Test half to right of 1
Binary Space Partition (BSP) Tree

- Example: Ray Intersection 2
  - Recursively split the ray and test nearer and farther halves, nearest first. Stop once you hit something:
    - Test half to left of 3
Binary Space Partition (BSP) Tree

- Example: Ray Intersection 2
  - Recursively split the ray and test nearer and farther halves, nearest first. Stop once you hit something:
    » Intersection with D. Done!
RayTreeIntersect(Ray ray, Node node, double min, double max) {
    if (Node is a leaf)
        return intersection of closest primitive in cell, or NULL if none
    else
        // Find splitting point
        dist = distance along the ray point to split plane of node
        // Find near and far children
        near_child = child of node that contains the origin of Ray
        far_child = other child of node
        // Recurse down near child first
        if the interval to look is on near side {
            isect = RayTreeIntersect(ray, near_child, min, max)
            if( isect ) return isect    // If there’s a hit, we are done
        }
        // If there’s no hit, test the far child
        if the interval to look is on far side
            return RayTreeIntersect(ray, far_child, min, max)
}
Acceleration

• Intersection acceleration techniques are important
  o Bounding volume hierarchies
  o Spatial partitions

• General concepts
  o Sort objects spatially
  o Make trivial rejections quick

Expected time is sub-linear in number of primitives
Summary

• Writing a simple ray casting renderer is easy
  • Generate rays
  • Intersection tests
  • Lighting calculations

Image RayCast(Camera camera, Scene scene, int width, int height) {
    Image image = new Image(width, height);
    for (int i = 0; i < width; i++) {
      for (int j = 0; j < height; j++) {
        Ray ray = ConstructRayThroughPixel(camera, i, j);
        Intersection hit = FindIntersection(ray, scene);
        image[i][j] = GetColor(hit);
      }
    }
    return image;
}
Next Time is Illumination!

Without Illumination

With Illumination