Announcements

• HW 2 is out

• Project discussion will be next week
  – Proposals: Oct 26
  – Final projects due date

• Exam moved to Nov 11 or Nov 18 (Thursday)?
  – Vote

Fewer Ray-Object Intersections

• From $O(N)$ to $O(\log N)$
• How?
  – Apply the idea of bounding boxes hierarchically
  – Cluster objects hierarchically
  – Single intersection might eliminate cluster

• Bounding volume hierarchy
• Space subdivision
  – Octree, Kd-tree, BSP-trees

Bounding Volume Hierarchy

• Hierarchical object bounding volumes
• Spheres, axis-aligned bounding boxes (AABB), oriented bounding boxes (OBB): fast

Intersection Acceleration

• If no intersection, eliminate tests with all children!

BVH: Construction

• Group objects together
  – Top-down: how to split?
  – Bottom-up: minimize surface area?
**Fewer Ray-Object Intersections**

- From $O(N)$ to $O(\log N)$
- Bounding volume hierarchy
- Space subdivision
  - Octree (Quadtree in 2D)
  - Non-uniform (kd-tree)
  - BSP-tree

**Spatial Hierarchy**

- Hierarchical spatial subdivision
  - Divides up space
- Children are distinct and cover parent

**Intersection Acceleration**

1. Intersect ray with root: $p = \text{root.intersect}(\text{ray})$
   - If no intersection, done
2. Find $p$ in tree (node $j = \text{root.find}(p)$)
3. Test ray against elements in node $j$
   - If intersection found, done
   - Else find exit point ($q$) from node $j$, $p = q$, goto 2

**Octree Properties**

- Front to back traversal

**Solutions**

- Split object
  - No repeated intersections and correct
  - But, could create lots of little objects
- Use mailboxes
  - Store intersection in the object: avoids repeated intersection
  - What about correctness?
    - Need to check that intersection is in “current” bounding box

**Octree Problems**

- Distribution of objects
- Chops up objects
**K-dimensional (kd) Tree**
- Spatial subdivision
  - Subdivide only 1 dimension
  - Do not subdivide at the center
- Tracing with kd-tree unchanged

**Construction**
- Which axis to pick?
- What point on the axis to pick?
- One heuristic:
  - Sort objects on each axis
  - Pick point corresponding to “middle” object
  - Pick axis that has “best” distribution of objects
  - \( L = n/2, R = n/2 \) (ideal)
  - Realistically, minimize (L-R) and
    - \( L \) approx. \( n/2 \), \( R \) approx. \( n/2 \)

**BSP Tree**
- Generalization of kd-trees
- Splitting plane is not axis aligned
- Used in games: DOOM

**BSP Construction**
- Use a polygon to define the splitting plane
- Other objects either split or stored high up

**How to construct?**
- Least-crossed criterion (random selection of polygons)
  - Do not split many polygons
  - Why are polygons split? Depends on use
- Try to make it balanced

**BSP Construction**
- Top-down
- Input: set of polygons
  - Select a partition plane
    - \( Ax + By + Cz + D = 0 \)
  - Partition the set of polygons with the plane
  - Recurse on both new sets
BSP Traversal

- Front to back ordering
- BSP traversal similar to kd-tree

Other acceleration structures

- Axis-aligned BSP for coherent ray tracing: same as our kd-tree

Uniform Grid

- Ray marching is trivial (additions)
- But, lots of cells (potentially empty)
- Bad for bi-modal distributions

Bounding Volume vs. Spatial Hierarchy

- Object subdivision
  - Hierarchical object representation
  - Hierarchically cluster objects
  - Siblings could overlap
  - Object in single leaf
  - Ray marches down
  - AABB, OBB, Spheres
- Spatial subdivision
  - Hierarchical spatial representation
  - Hierarchically cluster space
  - Siblings distinct
  - Object in >1 leaf (higher)
  - Ray marches across
  - Octree, kd-tree, Grid

Fewer Ray/Object Intersections

- Issues with hierarchical data structures:
  - Does it take long to initialize?
  - Does it require a lot of memory?
  - Is it as efficient for shadow and secondary rays as for view rays?
  - Can it accommodate time-varying data?

Using Acceleration Structures

- Acceleration structures for:
  - Ray tracing
  - Visibility determination
    - Culling: hardware and software
  - Point finding
  - Collision detection
Photon Maps

- Find n closest photons

Photon Maps: Balanced kd-tree

- Find n closest photons
- Balanced kd-tree for photon maps
  - Points (photons) as nodes
    - Compact
  - Balanced: implicit structure
    - Child of node i is 2i and 2i+1
  - Search: Same as before

Edge-and-point Rendering

- Kd-tree for edge-and-point rendering to find silhouettes and shadows
- How to efficiently find silhouette and shadow discontinuities in complex scenes made of polygon meshes?

Silhouettes

\[ N_1 \cdot V > 0 \text{ (forward facing)} \]
\[ N_2 \cdot V < 0 \text{ (backward facing)} \]

Umbral and Penumbral Conditions

- Event plane tangential to light and blocker
  \[ L \cdot N_{\text{blocker}} = L \cdot N_{\text{light}} = 0 \]
  \[ N_{\text{light}} \cdot N_{\text{blocker}} = 1 \text{ (umbral), -1 (penumbral)} \]

Normal–Position Tree

- Novel data structure similar to bounding-volume hierarchy
- Node represents a set of object polygons: stores boxes for normals and positions
  - Position interval: \([x_0, x_1] \times [y_0, y_1] \times [z_0, z_1]\)
  - Can be computed efficiently
- Equations (e.g., \(L \cdot N_{\text{blocker}} = 0\)) evaluated conservatively using interval arithmetic
Tree Traversal

- Fast traversal with interval evaluation of formulas
- Efficient shadow event computation with non-convex objects and area lights

Culling of Complex Scenes

- Remove geometry that is not visible … cull it away
  - View Frustum Culling
  - Hierarchical z-buffer
  - Cell-portal visibility
  - Many others….

View Frustum Culling

- Construct view frustum
  - 6 plans
- Test objects in scene against frustum
  - Cull them if they do not lie in frustum
- Complexity: $O(n)$
  - So what’s the point?

Hierarchical View Frustum Culling

- Use an octree/BVH
- Start at $o = \text{root of octree/BVH}$
- Test(Node $o$) {
  - Check 6 planes of frustum for intersection with $\text{bbox}(o)$
  - If in or out, terminate testing
  - If it intersects
    - For each child $c = \text{child}[i]$, Test ($c$)

Example

Occlusion Culling

- Occlusion Culling/Visibility Culling
- Don’t send all polygons to hardware
  - Remove polygons that are not visible
  - Conservative: find visible superset
Occlusion Culling

- **On-line**
  - Remove geometry on-the-fly

- **Off-line**
  - Determine potentially visible set (PVS)
  - When rendering only display PVS

Hierarchical Z-buffer

- **On-line**
- Use nearby polygons to remove far polygons
- Construct an octree subdivision of scene
  - Could use other data structures as well

How Hierarchical Z-buffer works

- When rendering:
  - Traverse octree from front to back
    - Enumeration order of octree cells can be determined by ray direction
  - Test z-value in z-buffer against octree cell
  - Consider cell b from octree
  - Let b project to pixels p0, ..., pn
  - If pi.z < b.Minz Eliminate octree cell
  - Else recurse

Hierarchical

- Have to do it for every pixel
  - Too slow

- Instead do it for a quadtree subdivision of z-buffer
  - Check if the whole square of pixels is in front of the box b