

Ray Tracing Acceleration

- Adaptive depth control
- Bounding Hierarchies
- Spatial Subdivision
 - Octrees
 - 3D Line Rasterization (grids/SEADS)
 - 5D Subdivision
- Hybrid Ray Tracing
- Shadow Feeler acceleration

CS174 Winter 00 Lecture 2

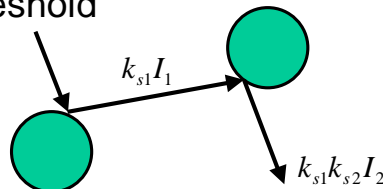
Copyright © Mark Meyer

Adaptive Depth Control

Every time a ray hits a surface it is attenuated by a reflection/transmission coefficient, k_s

The recursive ray is attenuated by the product of the coefficients, $k_{s1}k_{s2}k_{s3} \dots$

Terminate the recursion if this value is below some threshold



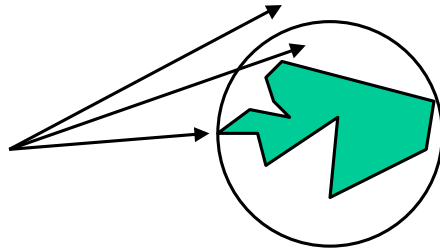
CS174 Winter 00 Lecture 2

Copyright © Mark Meyer

Bounding Volumes

Surround the object with an object that is easier to intersect with

If there is no intersection with the bounding object, there is no intersection with the object



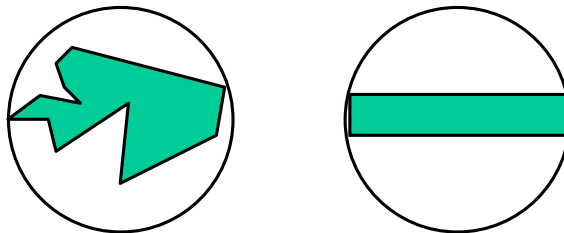
CS174 Winter 00 Lecture 2

Copyright © Mark Meyer

Bounding Spheres

Surround the object with a sphere

- 1) Find the object center
- 2) Find the distance to the furthest point from the center



Easy to intersect with, but may not fit object well

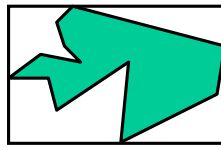
CS174 Winter 00 Lecture 2

Copyright © Mark Meyer

Bounding Boxes

Axis-Aligned Bounding Boxes (AABB)

- 1) Find the max and min in each coordinate direction in world space



Easy to intersect with, but may not fit object that well
Intersections can be done in world space

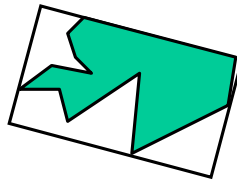
CS174 Winter 00 Lecture 2

Copyright © Mark Meyer

Bounding Boxes

Object-Aligned Bounding Boxes (OBB)

- 1) Find the max and min in each coordinate direction in object space



Easy to intersect with, usually fits object well

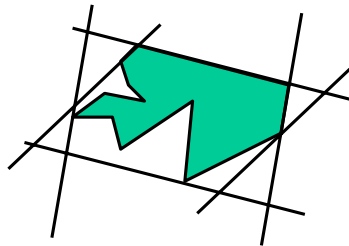
CS174 Winter 00 Lecture 2

Copyright © Mark Meyer

Bounding Slabs

Surround the object with sets of parallel slabs

- 1) Find the max and min extents in a set of specified directions



The directions must span the space
Harder to intersect with, but may fit object very well

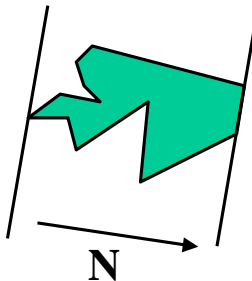
CS174 Winter 00 Lecture 2

Copyright © Mark Meyer

Bounding Slabs

Finding the max and min extents

$$d_i = \mathbf{x}_i \cdot \mathbf{N}$$
$$d_{\max} = \max(d_i)$$
$$d_{\min} = \min(d_i)$$



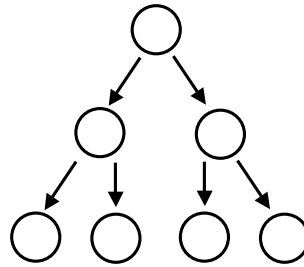
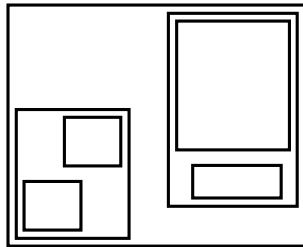
Store a set of (global) directions and (local) extent pairs

CS174 Winter 00 Lecture 2

Copyright © Mark Meyer

Bounding Hierarchies

Cluster the bounding volumes into a hierarchy



CS174 Winter 00 Lecture 2

Copyright © Mark Meyer

Bounding Hierarchies

Criteria for a good hierarchy

- 1) Subtrees should contain objects near one another
- 2) The volume of each node should be minimal
- 3) The sum of the bounding volumes should be minimal
- 4) Nodes near the root are more important
- 5) The time spent constructing the hierarchy should be offset by the savings

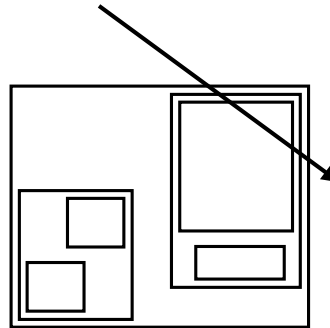
CS174 Winter 00 Lecture 2

Copyright © Mark Meyer

Bounding Hierarchies

Cluster the bounding volumes into a hierarchy

- 1) Start with root node
- 2) Intersect with current node
- 3) If intersected, test children



Significantly reduces the number of intersection computations for complex scenes

CS174 Winter 00 Lecture 2

Copyright © Mark Meyer

Bounding Hierarchy Construction

Calculating the cost of adding a node

Cost is the probability of intersecting the node times the number of children (nc), plus the cost of the children

$$Cost(n) = nc * area(n) + \sum_{i=1}^{nc} Cost(n.child[i])$$

Probability of intersecting a node is proportional to the surface area of the node

CS174 Winter 00 Lecture 2

Copyright © Mark Meyer

Bounding Hierarchy Construction

Incremental cost of adding node m at node n

```
incrementalCost(node n, node m)
  if(leaf(n))
    incr = 2 * area(bound(n,m));
  else
    incr = -area(n)*n.numChildren +
           area(bound(n,m))*(n.numChildren+1)
```

CS174 Winter 00 Lecture 2

Copyright © Mark Meyer

Bounding Hierarchy Construction

Adding a new node m to an existing subtree n

```
insert(n,best,m) //best is the best node to add m to so far
  oldArea = area(n);
  if(!leaf(n))
    for all children calculate incrementalCost(n.child[i],m)
    nc = child with minimum increment
    best = insert(nc, (nc.incr < best.incr) ? nc : best, m)
  if(n == best) addChild(n,m)
  else if (oldArea < area(n))
    n.cost += (area(n) - oldArea)*n.numChildren
  return best
```

CS174 Winter 00 Lecture 2

Copyright © Mark Meyer

Bounding Hierarchy Construction

Add a node m to an existing node n

```
addChild(node n, node m)
```

```
if(leaf(n)) //put both nodes under a single node
```

```
    n.child[0] = new node(n);
```

```
    n.child[1] = m; m.cost = 0;
```

```
else //add the child at the end of the list
```

```
    n.child[n.numChildren] = m;
```

```
    n.numChildren++;
```

```
n.cost += n.incr;
```

CS174 Winter 00 Lecture 2

Copyright © Mark Meyer

Bounding Hierarchy Traversal

Intersecting a ray with the bounding hierarchy

```
while(heap not empty)
```

```
    extract nearest node (n) from the heap
```

```
    if(currentIntersectionPt is closer than n)
```

```
        break
```

```
    if(n is a primitive node)
```

```
        if(ray intersects n && isCloser)
```

```
            update currentIntersectionPt and object
```

```
    else for(each child (i) of n)
```

```
        if(ray intersects i && isCloser)
```

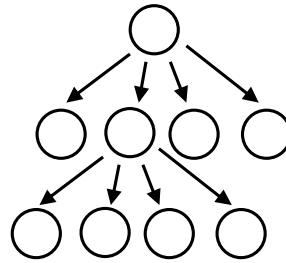
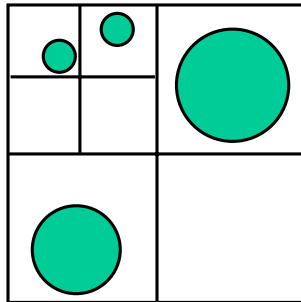
```
            insert i into the active heap
```

CS174 Winter 00 Lecture 2

Copyright © Mark Meyer

Octrees

Hierarchically subdivide space into 8 regions

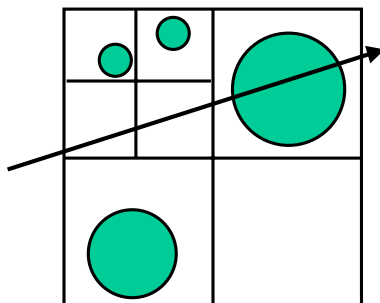


CS174 Winter 00 Lecture 2

Copyright © Mark Meyer

Octrees

Ray Tracing using an octree



- 1) Preprocess the scene by placing objects into the cells which they intersect
- 2) Intersect all objects in the current cell
 - a) If there is an intersection, return the closest point
 - b) Otherwise, find the next cell and repeat (2)

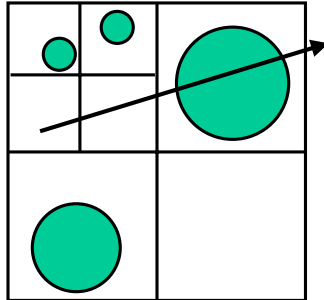
Could also use BSP trees or k-d trees

CS174 Winter 00 Lecture 2

Copyright © Mark Meyer

Octrees

Tracking the ray



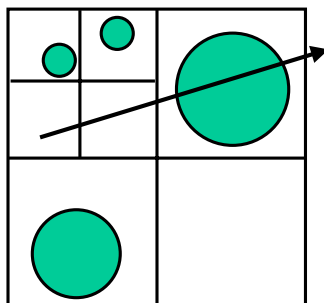
- 1) Find the cell containing the ray start point
- 2) If the next cell along the ray is needed:
 - a) Find the intersection with the cell boundary
 - b) Move further along the ray
 - c) Find the cell containing this new point

CS174 Winter 00 Lecture 2

Copyright © Mark Meyer

Octrees

Finding the cell containing a point



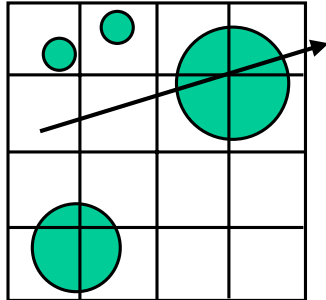
- 1) Start at the root of the tree
- 2) If it is a leaf node, done
- 3) Otherwise, find the child containing the point and repeat (2)

CS174 Winter 00 Lecture 2

Copyright © Mark Meyer

3D Line Rasterization

SEADS



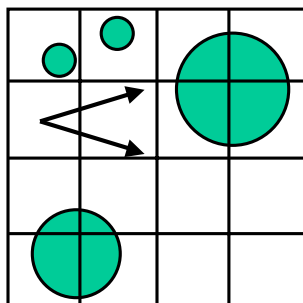
- 1) Subdivide space into equally sized regions
- 2) Preprocess by placing objects into the cells which they overlap
- 3) Intersect all objects in the current cell
 - a) If there is an intersection, return the closest point
 - b) Otherwise, find the next cell by 3D line rasterization and repeat (2)

CS174 Winter 00 Lecture 2

Copyright © Mark Meyer

5D Spatial Subdivision

Subdivide 5D space



Subdivide space by location and direction

Each cell contains a list of candidate objects

Rays index into the 5D table

- Removes the need to track the ray
- Memory requirements are enormous

CS174 Winter 00 Lecture 2

Copyright © Mark Meyer

Hybrid Ray Tracing

Most scenes have an average ray depth < 2

Speed up the first hit calculations

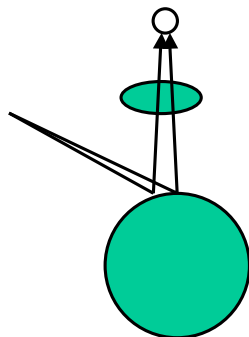
- 1) Calculate the first hit by a modified z-buffer algorithm
- 2) Instead of storing a color buffer, use an object-id buffer containing the id of the first object hit
- 3) Then proceed with normal ray tracing

CS174 Winter 00 Lecture 2

Copyright © Mark Meyer

Shadow Feeler Acceleration

Many rays hitting an object have the same shadowing object



- 1) Store the shadowing object for a given light and object
- 2) When casting a shadow feeler from an object, first test this cached blocker
 - a) If it is intersected, done
 - b) Otherwise proceed as normal

CS174 Winter 00 Lecture 2

Copyright © Mark Meyer