Bounding Volumes

Use simple volume
enclose object(s)
if ray doesn’t intersect volume
it doesn’t intersect what’s inside
tradeoff for rays where there is
extra intersection test for
object intersections
volume intersections, but not object intersections
v.
quick test for no intersection for no volume intersection

Bounding Volumes

3 approaches:
Bound object
Bound screen area that object projects to
Bound area of world space
Can use hierarchical organization of bounding volumes

Bound Object

Easy-to-compute approximation to object
Easy to test for ray-bounding-object intersection
Trade-off complexity of computation v. tightness of fit
Can bound object in object space or world space
Axis Aligned Bounding Box

- Easiest bounding volume to compute
- Compute min/max for x, y, z of vertices
- Some computational expense to test for intersection
- Not tight fitting for some objects

Limit\[3\][0] = 10000000000; limit\[3\][1] = -1000000000;

for each point, for each dimensions

if p[i][j] < limit[j][0] then limitIndex[j][0] = i;
if p[i][j] > limit[j][1] then limitIndex[j][1] = i;

Bounding Spheres

- Takes some effort to compute optimal bounding sphere
- Easy to compute approximation (non-optimal fit)
- Easy to test for intersection (ray-sphere)
- Not tight fitting for some objects

Bounding Spheres

Loop through points and record min/max in x,y,z

Use maximally separated pair of points and their midpoint as initial approximation to sphere

For each point in original set, adjust the bounding sphere to include the point
Bounding Spheres

limit[3][0] = -10000000000; limit[3][1] = 1000000000;
for each point, for each dimensions
if p[i][j] < limit[j][0] then limitIndex[j][0] = i;
if p[i][j] > limit[j][1] then limitIndex[j][1] = i;

k=0;
if (|limit[1][1]-limit[1][0]| > |limit[0][1]-limit[0][0]|) k=1;
if (|limit[2][1]-limit[2][0]| > |limit[k][1]-limit[k][0]|) k=2;
midpoint = (p[limit[k]]+p[limit[k]])/2;
radius = (p[limit[k]]-p[limit[k]])/2;

Bounding Spheres

• For each point
  – if p[i] is outside of radius of midpoint
  – radius = (radius + dist(midpoint,p[i]))/2
  – center = p[i]+radius*(center-p[i])/|center-p[i]|;

Bounding Slabs

For each slab, user defines normal to use for slab pair
For each object, compute 2 d’s for each N
Takes some effort to compute d’s - how?
Takes some effort to test for intersection - how?
Can add more slabs to get tighter fit
Bounding Slabs

SLAB: pair of parallel planes
for all points, compute $P \cdot N = d$
find min, max $d$ for all points

Slab defined by: $N$, $d_1$, $d_2$

- ray defined by $P(t) = P + tD$
- intersection with plane: $N \cdot P + t(N \cdot D) = d$

for each slab
- retrieve $d_1$, $d_2$, $N$
- compute
  - $t_1 = (d_1-N \cdot P)(N \cdot D) / (N \cdot D)$
  - $t_2 = (d_2-N \cdot P)(N \cdot D) / (N \cdot D)$
  - keep track of entering max, exiting min
  - how to determine entering, exiting status?

Bounding Slabs

Convex Hull

Smallest convex polyhedron containing object (point set)

Takes some effort to test for intersection - how?
Convex Hull

Find highest vertex
Find plane through vertex parallel to ground plane
Find second vertex that makes minimum angle with first vertex and up vector

In the final convex hull, each edge will be shared by two and only two triangles
For each unmatched edge (until there are no more), find vertex that, when a triangle is formed with the edge, will minimize angle between its normal and normal of shared face

Hierarchical Bounding Volumes

Compute bounding volume for groups of objects

Compute bounding volume for groups of groups of objects
Test higher-level bound volumes first
Bound Area of Projection

Project object to picture plane
Bound columns and rows that object projects to
Only intersect first-level rays with pixels in projected area
i.e., only good for ray-casting part of ray-tracing

Bound Area of Projection

Project vertices onto picture plane
Find 2D bounding box on picture plane

Bound Area of Projection

Areas of projection can be grouped hierarchically

Bound Area of Projection

Project rays through pixels inside bound
Only test object that was bounded
Bound Area, not Object
In 2D - it looks like this:
Bucket sort objects into cells

Bound Area of 3D World
Divide world space into cells
Dump objects into cells: an object is dumped into each cell it touches

Bound Area of World
Trace ray through cells from closest to farthest
Intersect ray with each object in cell
Stop when it hits closest object in cell

Bound Area of World
Traverse cells in order – test all objects in cell
If no intersections, step to next cell
If one or more intersections, get closest intersection
Bound Area of World

Use octree (quadtree in 2D)

Hierarchical approach: cells, then subcells.
Takes significant coding to keep track of level
Overhead in popping up and down in hierarchy

Binary Spatial Partitioning