Ray Intersection Acceleration

Readings

Chapter 2 – Geometry & Transformations

Chapter 3 – Shapes

Chapter 4 – Primitives & Intersection Acceleration

Covers basic math and PBRT implementation: read on your own

We'll cover this in class

Reading

Chapter 2: Geometry and Transformations

2.1-2.5	Review basic geometry
2.6	3D Bounding boxes in PBRT
2.7-2.8	Transformation & applying them in PBRT
2.5.1 & 2.9	Differential geometry

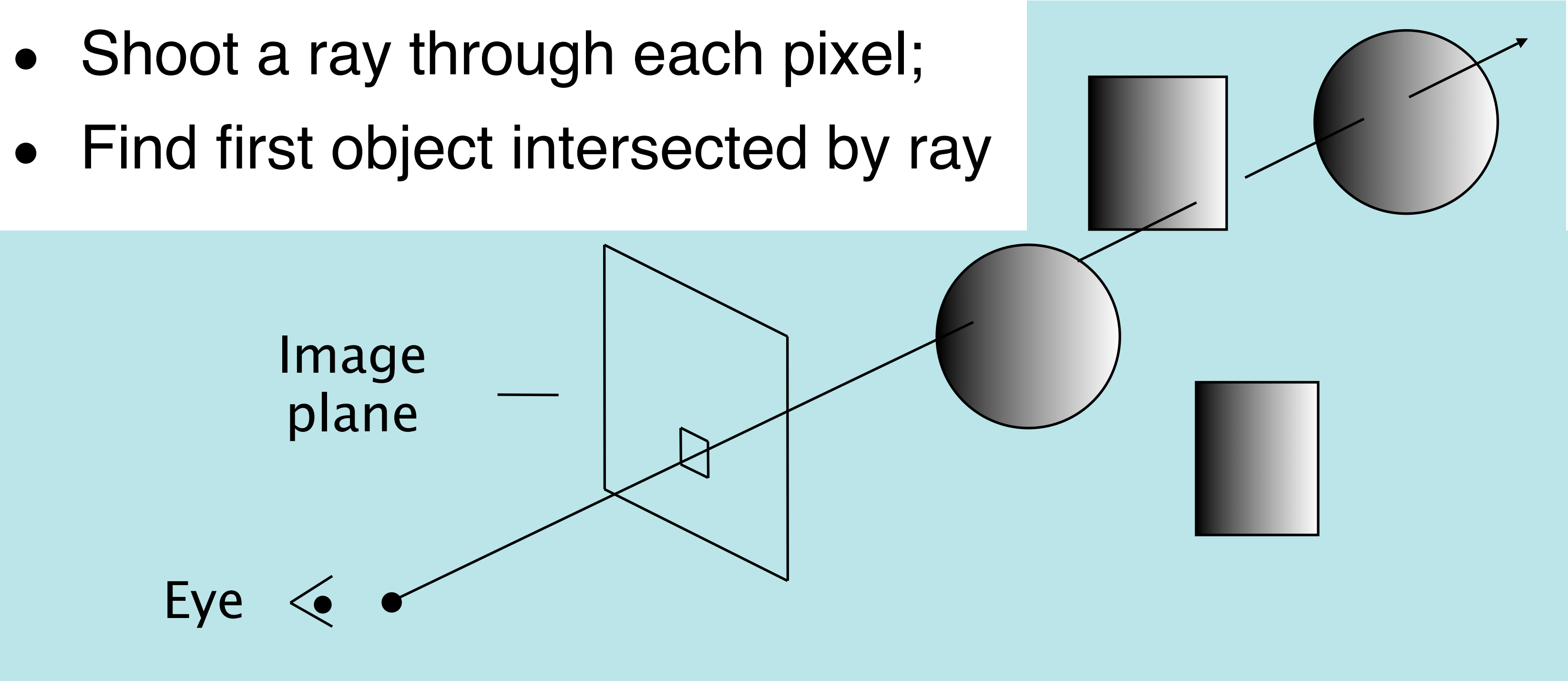
Reading

Chapter 3: Shapes

3.1	Basic PBRT shape interface
3.2-3.5	Specific shapes - quadrics
3.6	Triangles and meshes
3.7	Subdivision surfaces

Ray Tracing

- Shoot a ray through each pixel;
- Find first object intersected by ray

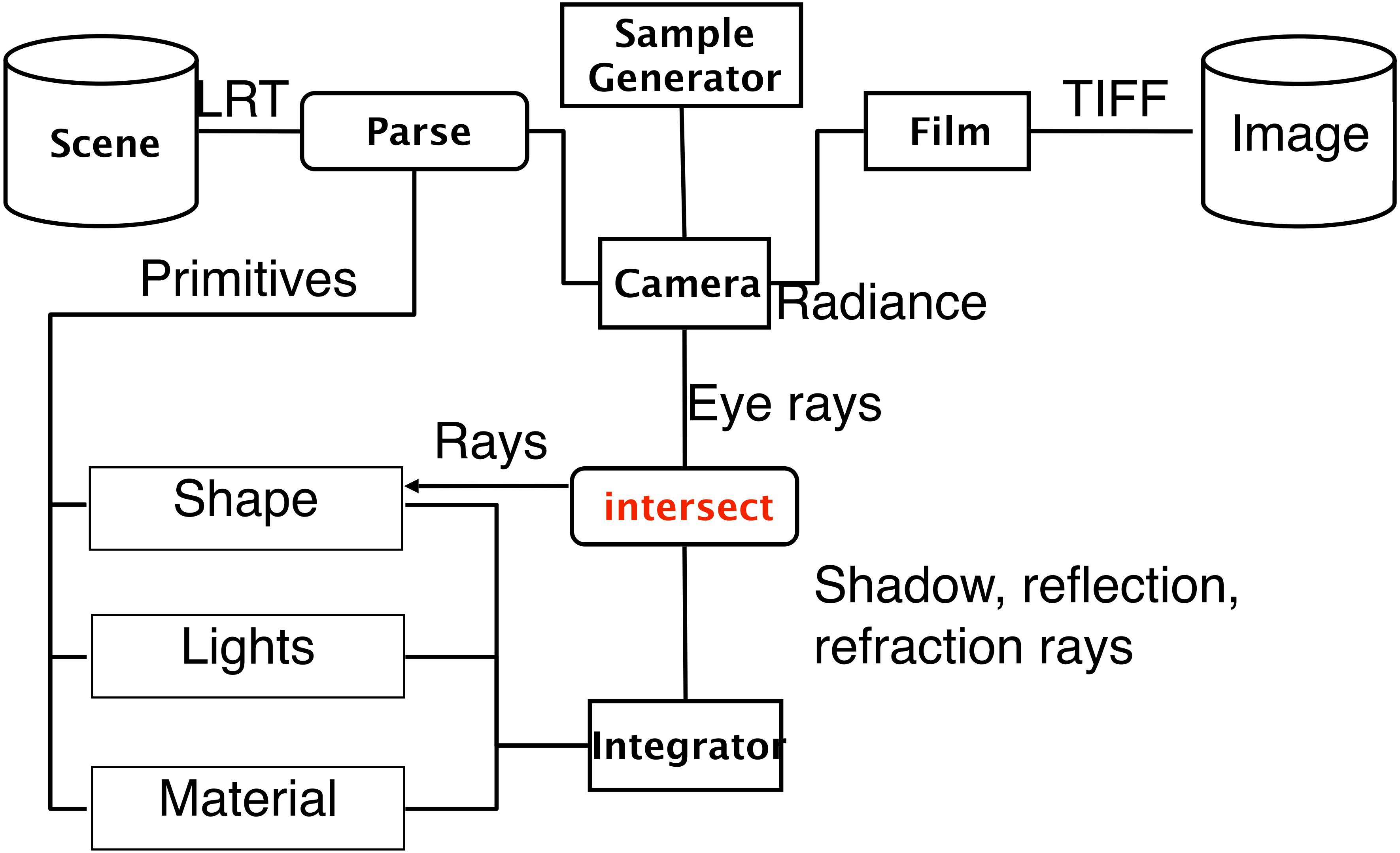


Compute ray. (More linear algebra.)

Compute ray-object intersection.

Spawn more rays for reflection and refraction

Ray Tracing Architecture



Scene

LRT

Parse

Sample Generator

Film

TIFF

Image

Primitives

Camera

Radiance

Eye rays

Rays

Shape

intersect

Shadow, reflection, refraction rays

Lights

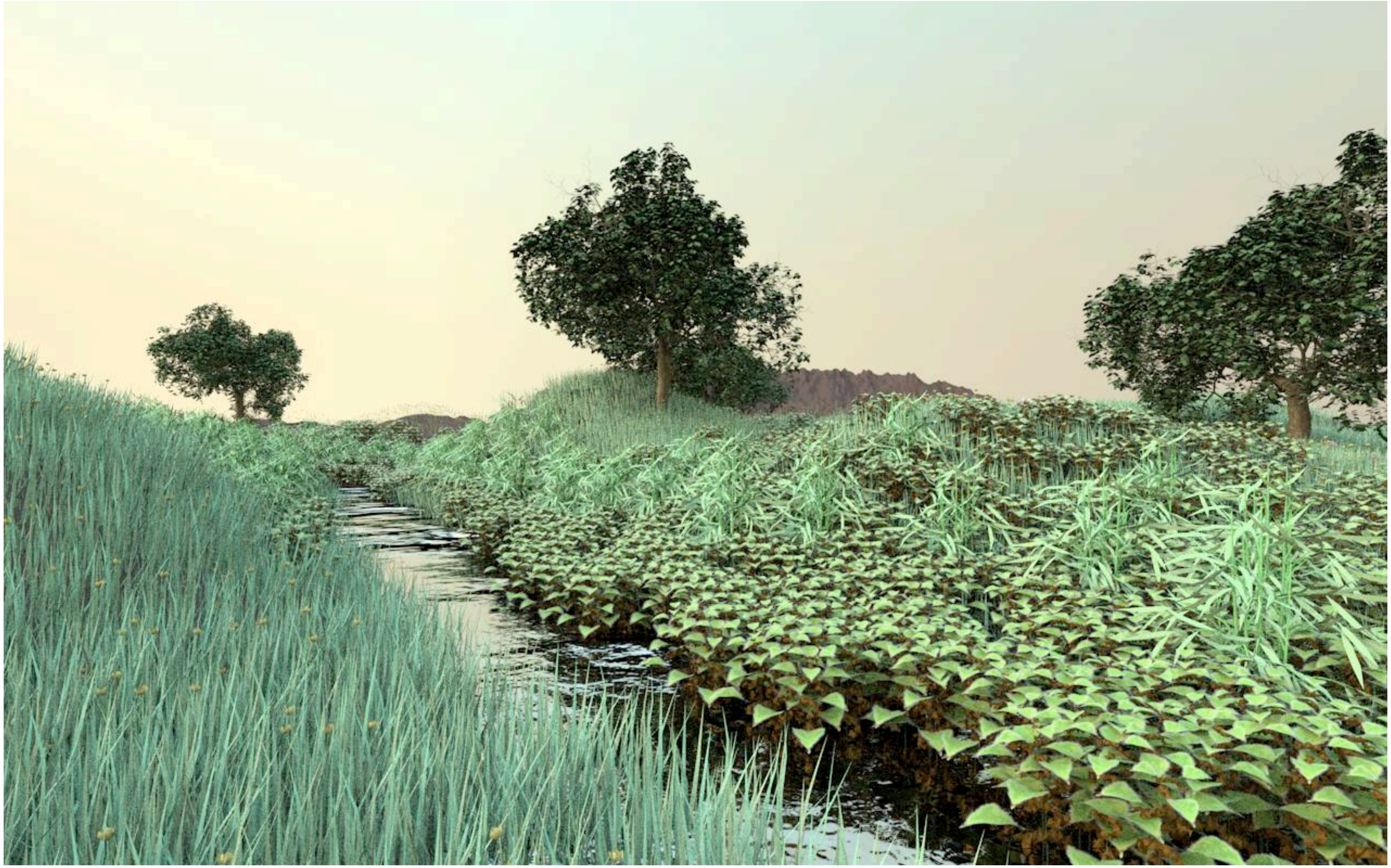
Integrator

Material

Optimizing Ray Tracing

- Main computation load is ray-object intersection
- 50-90% of run time when profiled
- <u>Test</u> for possible intersection before committing to computing intersections

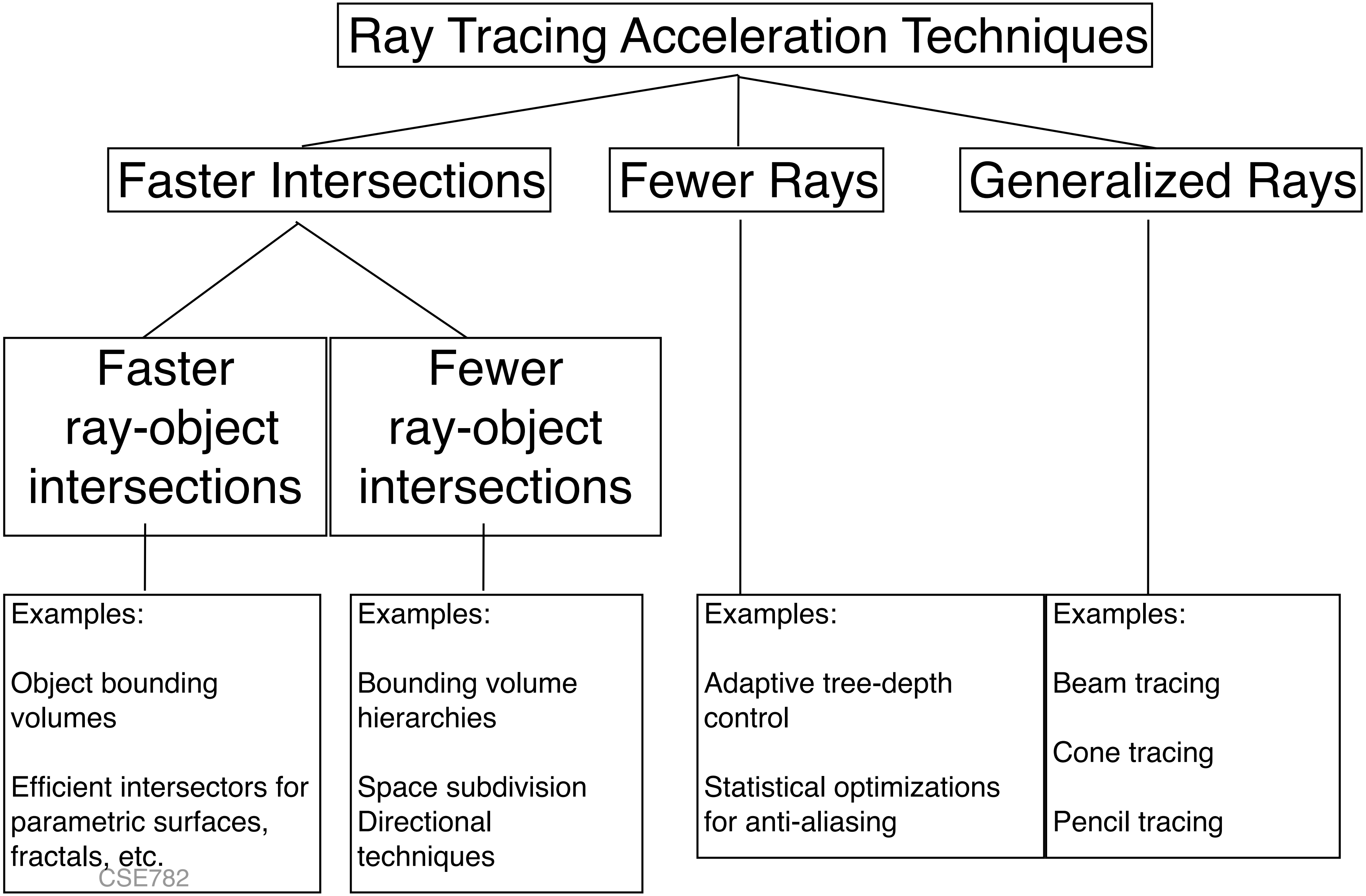
Consider this



Complexity !

- I rays or pixels in image
- N objects
- O(NI)
- Can we do O(I logN) ?

Ray Intersection Acceleration



Ray Tracing Acceleration Techniques

Faster Intersections

Fewer Rays

Generalized Rays

Faster ray-object intersections

Fewer ray-object intersections

Examples:

Object bounding volumes

Efficient intersectors for parametric surfaces, fractals, etc.

Examples:

Bounding volume hierarchies

Space subdivision
Directional techniques

Examples:

Adaptive tree-depth control

Statistical optimizations for anti-aliasing

Examples:

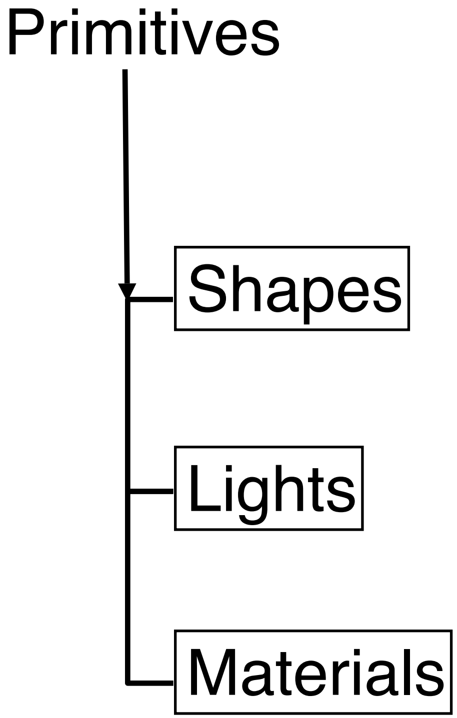
Beam tracing

Cone tracing

Pencil tracing

Pbrt and Intersections

- Primitive base class
- Shapes are subclasses of primitive
- Aggregate class
- Methods
  - WorldBound
  - CanIntersect
  - Intersect
  - IntersectP
  - Refine
- First four return Intersection structures
- Last returns Primitives



Primitives

Shapes

Lights

Materials

Pbrt and Intersections

WorldBound	Returns a bounding box in world space
Intersect	Return ‘true’ if an intersection and an intersection structure
IntersectP	Return ‘true’ if an intersection occurs but does not return an intersection structure
Refine	If non-intersectable, refines shape into (some) intersectable new shapes

Intersection Geometry

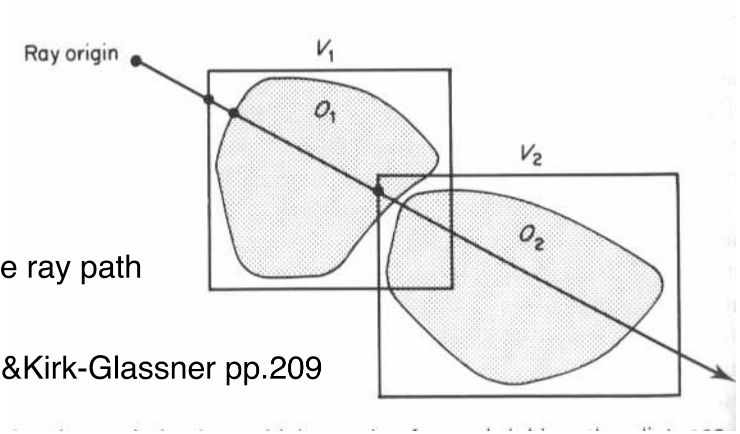
- Shape independent representation for intersections
- DifferentialGeometry Intersection::dg
  - Point P
  - Normal N
  - Parametric (u,v)
  - Partial derivatives
    Tangents: dpdu, dpdv
    change in normal: dndu, dndv

Speeding up Intersection Calculation
Object-based vs. World-based

- Common dichotomy in graphics
  - objects situated in (world) space
  - (world) space in which objects reside
- Bounding volumes are object-based
- Spatial Subdivision is world-based approach
- Sub-linear search – logarithmic ?

Bounding Volumes

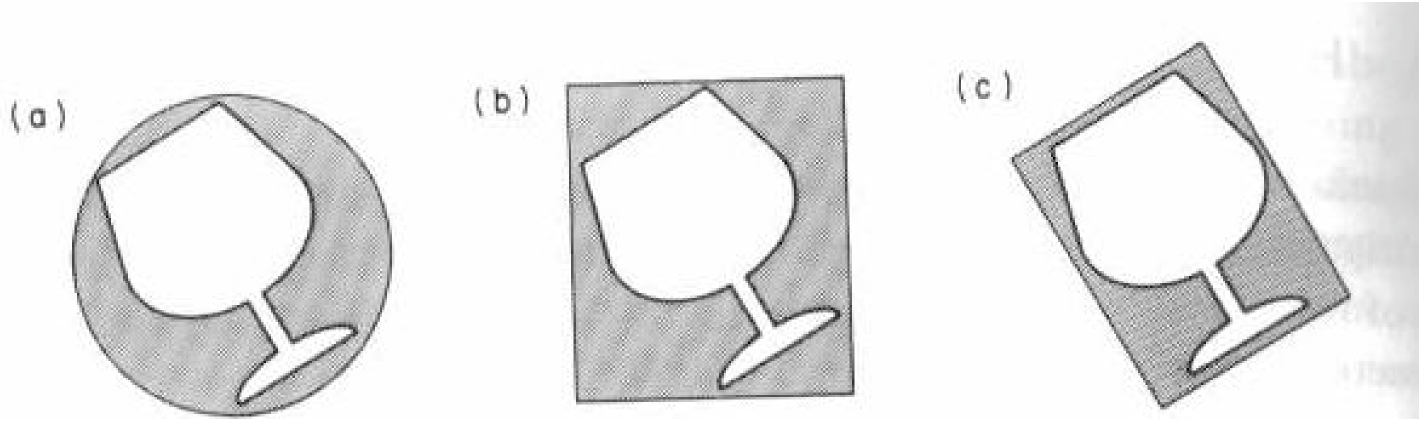
- Surround object with a simple volume
- Test ray against volume first
- Test object-space or world-space bound? (pros and cons)
- Cost model - N*cb + pi*N*co
  - N (number of rays) is given
    pi – fraction of rays intersecting bounding volume
  - Minimize cb (cost of intersecting bounding volume) and co (cost of intersecting object)
  - Reduce ray path
  - Minimize cost/fit ratio

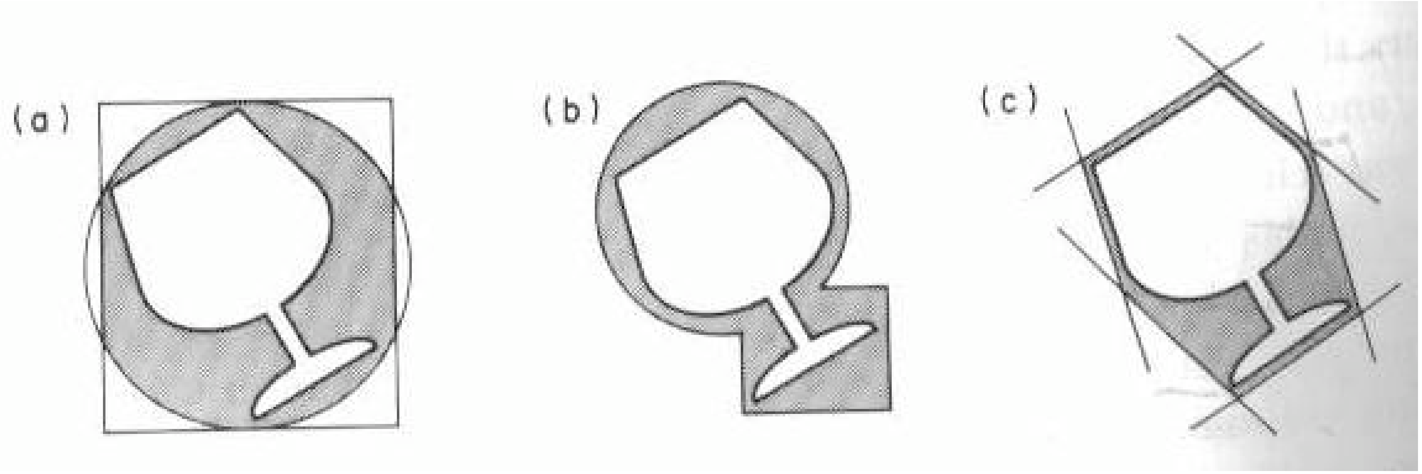


Reduce ray path

Arvo&Kirk-Glassner pp.209

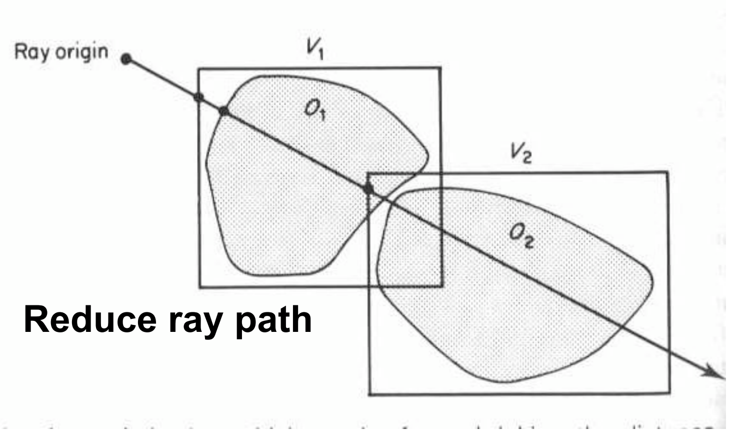
Bounding Volumes





Bounding Volumes

- Bounding sphere
  - Difficult to compute good one
  - Easy to test for intersection
- Bounding box
  - Easy to compute for given object
  - Relatively difficult to intersect (maybe ?)



Reduce ray path

Arvo&Kirk-Glassner pp.209

Pbrt's Bounding Boxes

- Virtual BBox ObjectBound() const=0;
- Virtual BBox WorldBound() const {
  return ObjectToWorld(ObjectBound());
  }
- Bool BBox::IntersectP(Const Ray &ray, Float *hit0, Float *hitt1) const { }

Bounding Box

- Compute min/max for x,y,z
- 3 options
  - Compute in world space
    - Chance of ill fitting b-box
  - Compute in object space and transform w/object
    - Object space b-box probably better fit than world space
    - Need to intersect ray with arbitrary hexahedral in world sp.
  - Compute in object space and test in object space
    - Inverse transform ray into object space

Ray & Cube

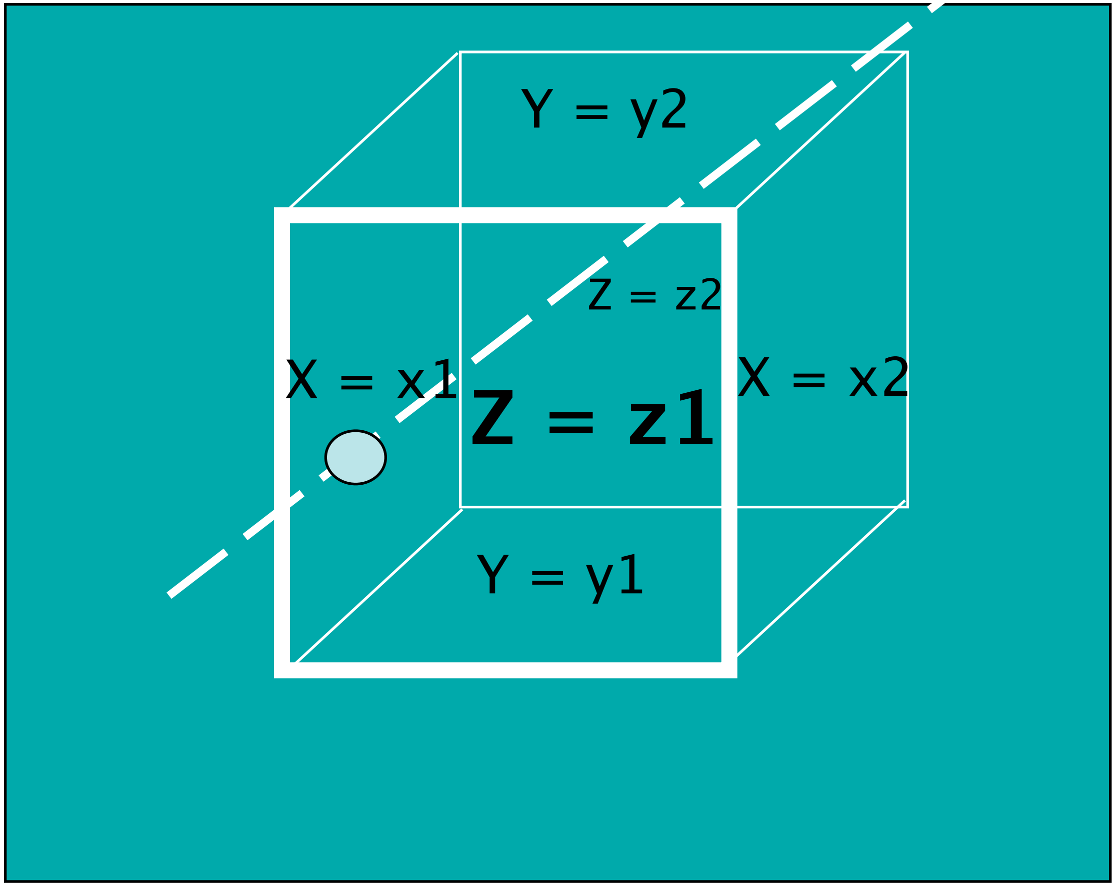
$P(t) = s + tc$

$t_{x1} = (x1 - s_x)/c_x$

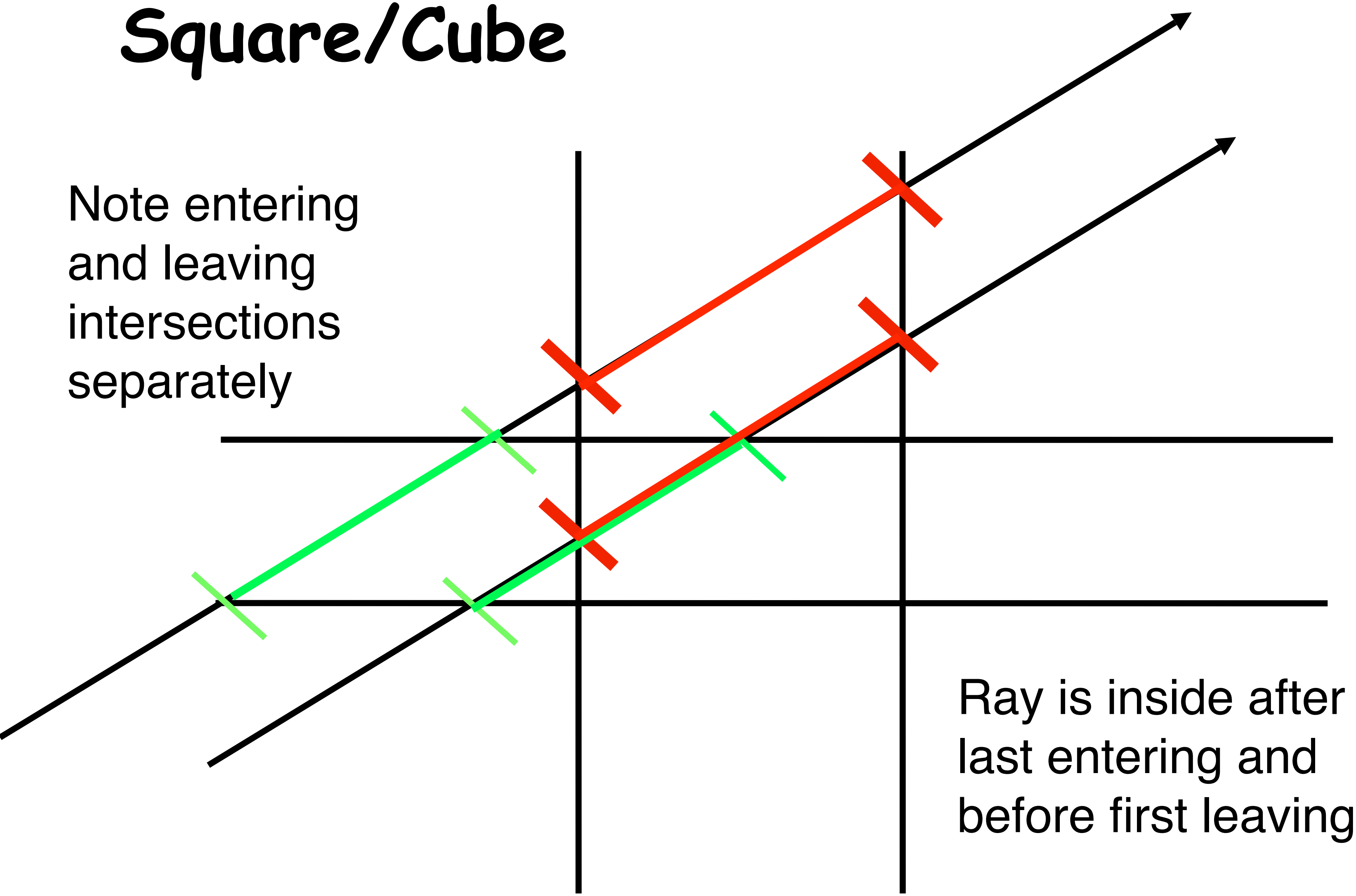
$t_{x2} = (x2 - s_x)/c_x$

$t_{y1} = (y1 - s_x)/c_x$

...



Square/Cube



Note entering and leaving intersections separately

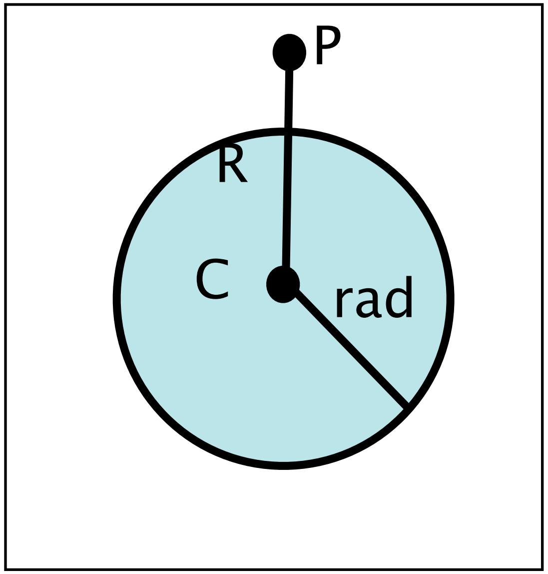
Ray is inside after last entering and before first leaving

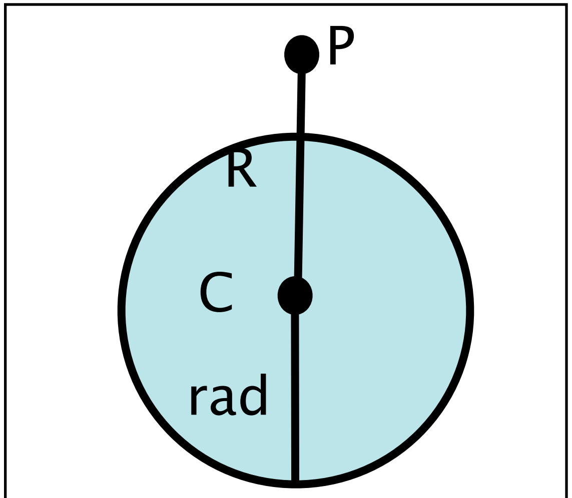
Algorithm

```
set Tnear = - infinity, Tfar = infinity

Ray (t) = O + t * Ray

For each pair of planes P associated with X, Y, and Z do:
                                        (example using X planes)
    if direction $Ray_x = 0$ then the ray is parallel to the X planes

    if origin Ox is not between the slabs ( Ox < Xl or Ox > Xh) then

    return false
    else

    if the ray is not parallel to the plane then
    begin
      compute the intersection distance of the planes
      T1 = (Xl - Ox) / Xd
      T2 = (Xh - Ox) / Xd
      If T1 > T2 swap (T1, T2) - since T1 intersection with near plane

      If T1 > Tnear      Tnear =T1  - want largest Tnear
      If T2 < Tfar       Tfar="T2"  - want smallest Tfar
      If Tnear > Tfar                         - box is missed so return false
      If Tfar < 0                             - box is behind ray return false

    end
If Box survived all above tests, return true with intersection point Tnear and exit point Tfar.
```

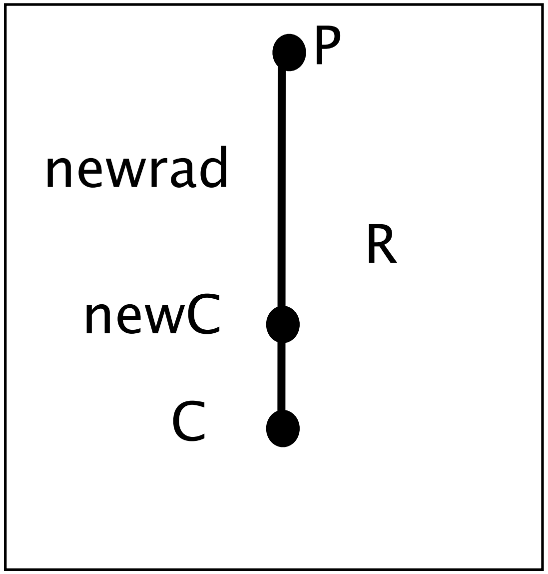
Bounding Sphere

- Find min/max points in x,y,z -> 3 pairs
- Use maximally separated pair to define initial sphere
- For each point
  - If point is outside of current sphere, increase old sphere to just include new point

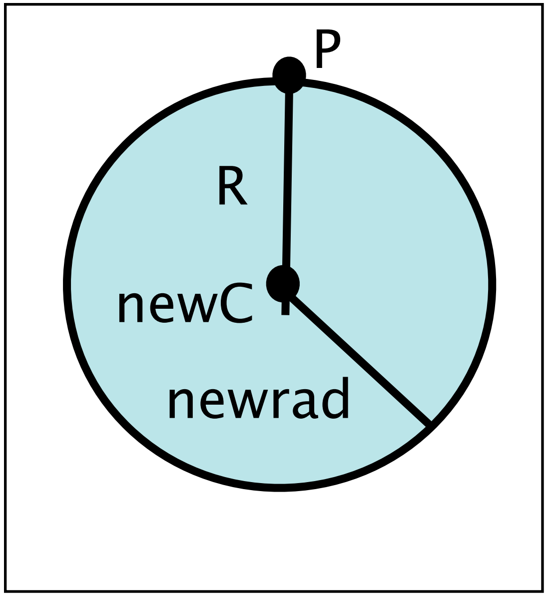




newrad = (R+rad)/2

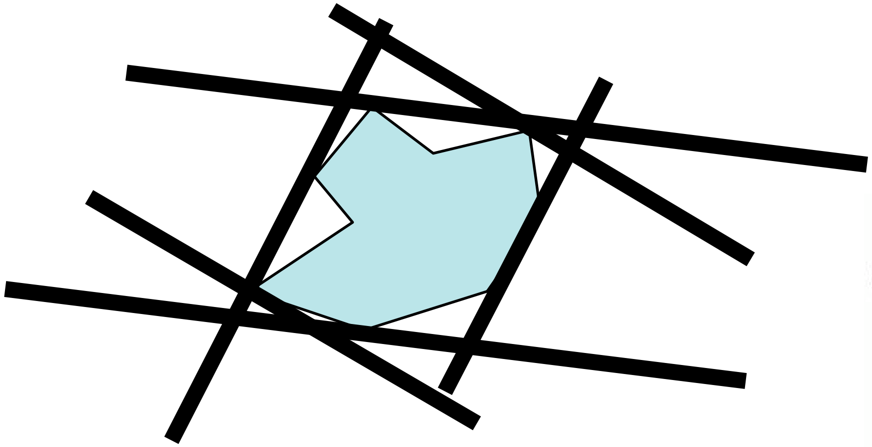


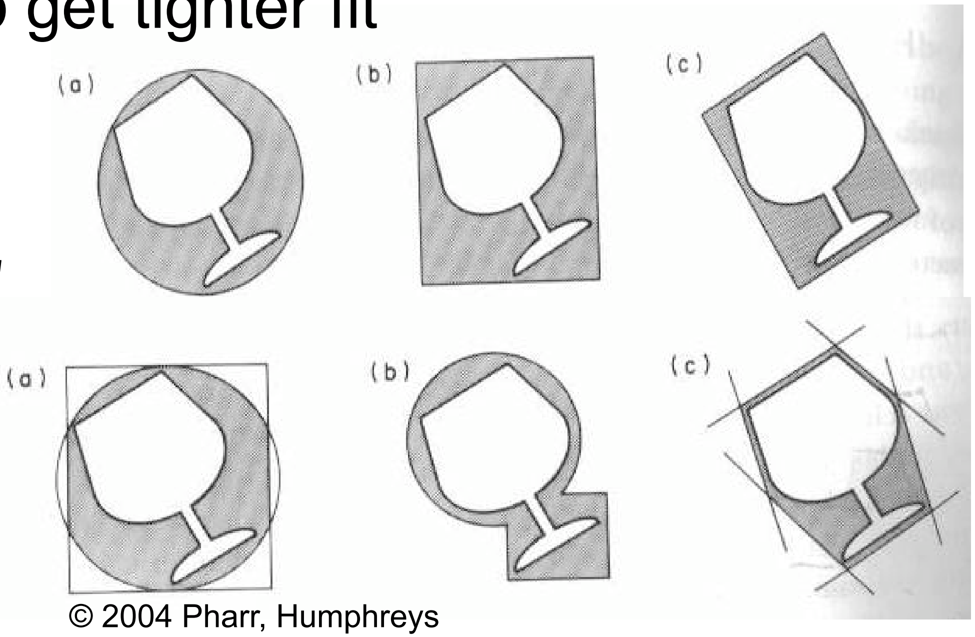
newC = P+(newrad/R)(C-P)



Bounding Slabs

- More complex to compute
- Better fit of object
- Use multiple pairs of parallel planes to bound object
- Can add more slabs to get tighter fit

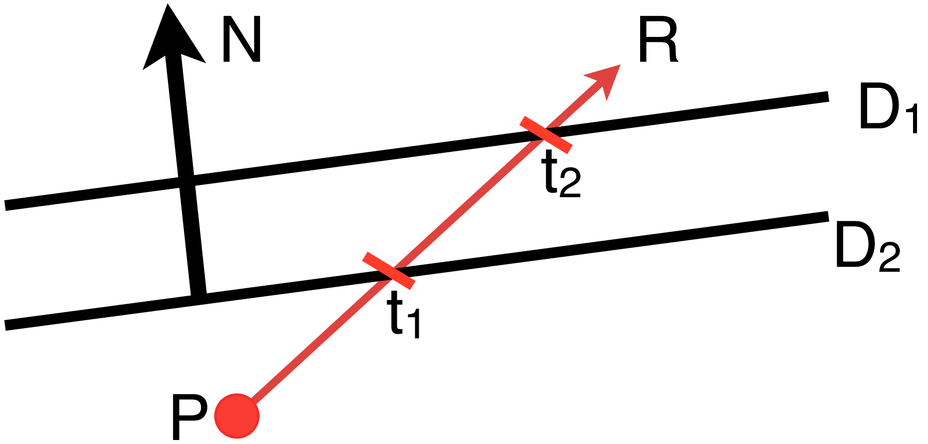




© 2004 Pharr, Humphreys

Bounding Slabs

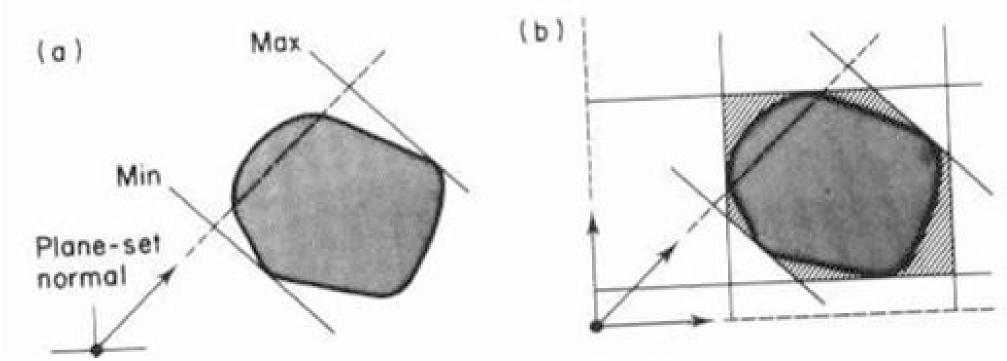
- Use algorithm for axis aligned bounding box
- intersect ray with arbitrary plane $P \cdot N = D$



$$P(t) \cdot N = D_i$$

$$(P + tR) \cdot N = D_i$$

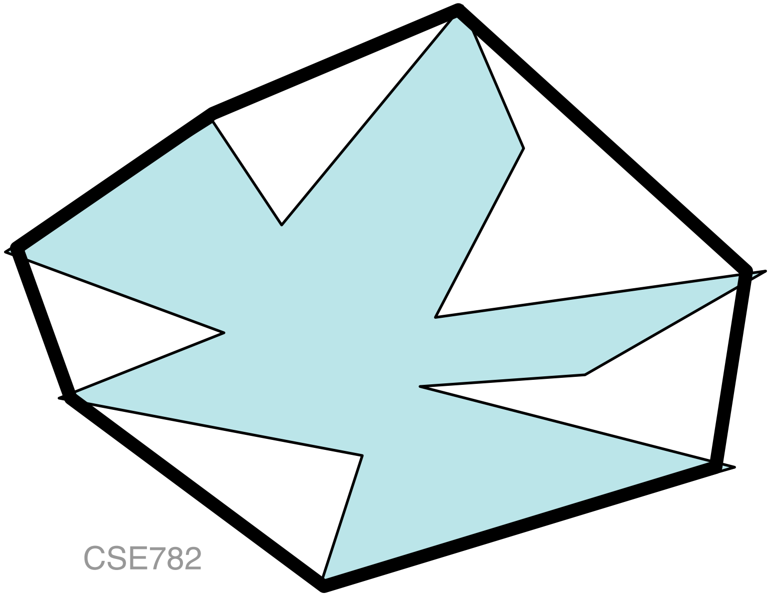
$$t = \frac{D_i - P \cdot N}{R \cdot N}$$



Slabs - More effort to compute, better fit

Approximate 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
- Find third vertex that makes plane whose normal makes minimum angle with up vector

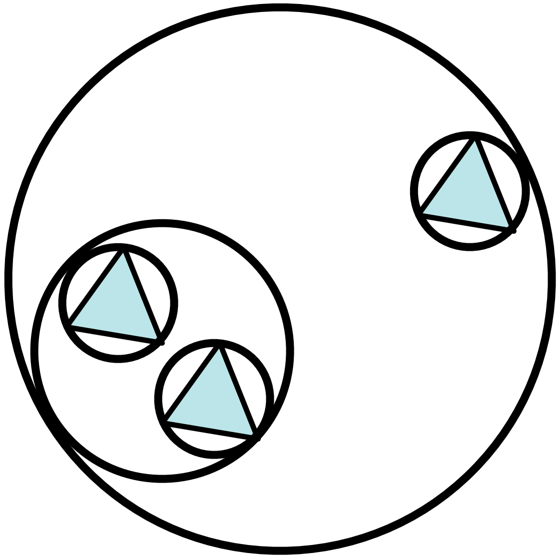


For any unmatched edge, find unused vertex such that the plane of the vertex and edge makes a minimum angle with the plane of edge's face

Hierarchical Bounding Volumes

- Compute bounding volume for groups of objects
- Compute bounding volume for groups of groups of objects

Hierarchical Bounding Volumes



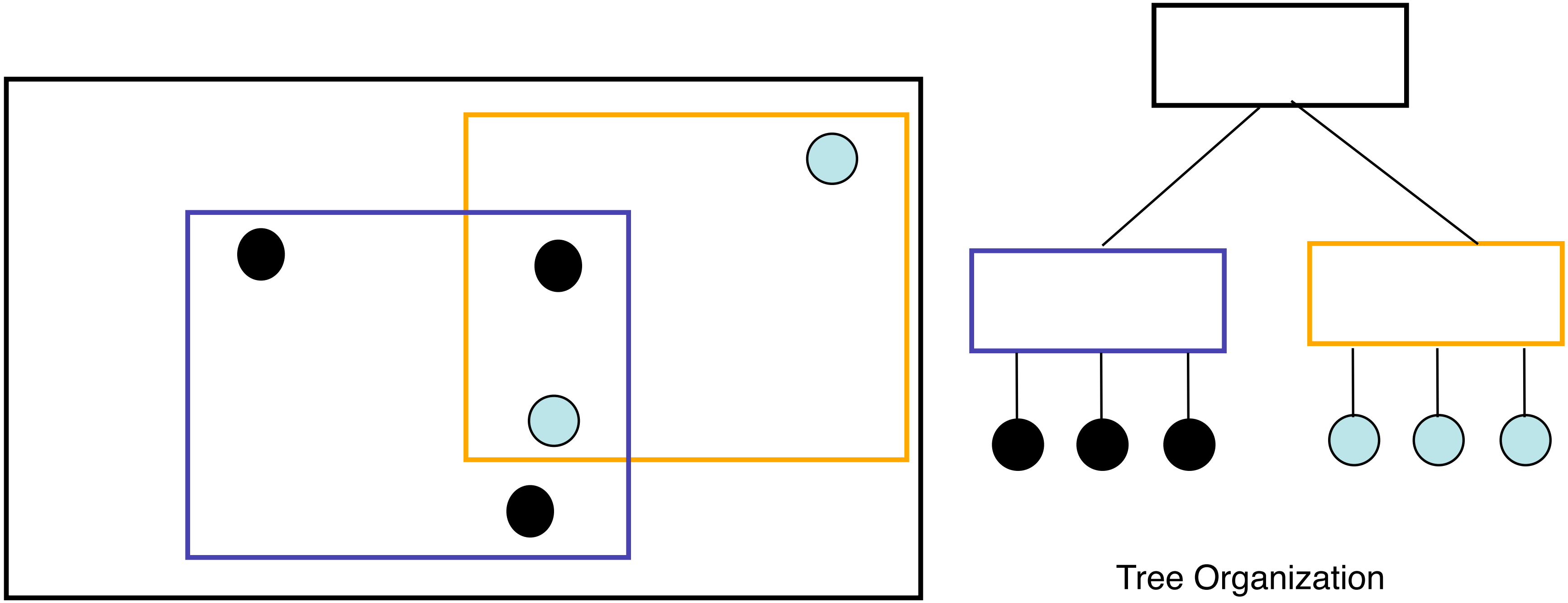
- Create tree of bounding volumes
- Children are contained within parent
- Creation preprocess
  - From model hierarchy
  - Automatic clustering
- Search

```
intersect(node,ray,hits) {
 if( intersectp(node->bound,ray)
        if( leaf(node) )
                intersect(nodeprims,ray,hits)
        else
                for each child
                        intersect(child,ray,hits)
}
```

Return the closest of all hits !

Problem

- Subtrees overlap
- Does not contain all objects it overlaps
- Balance

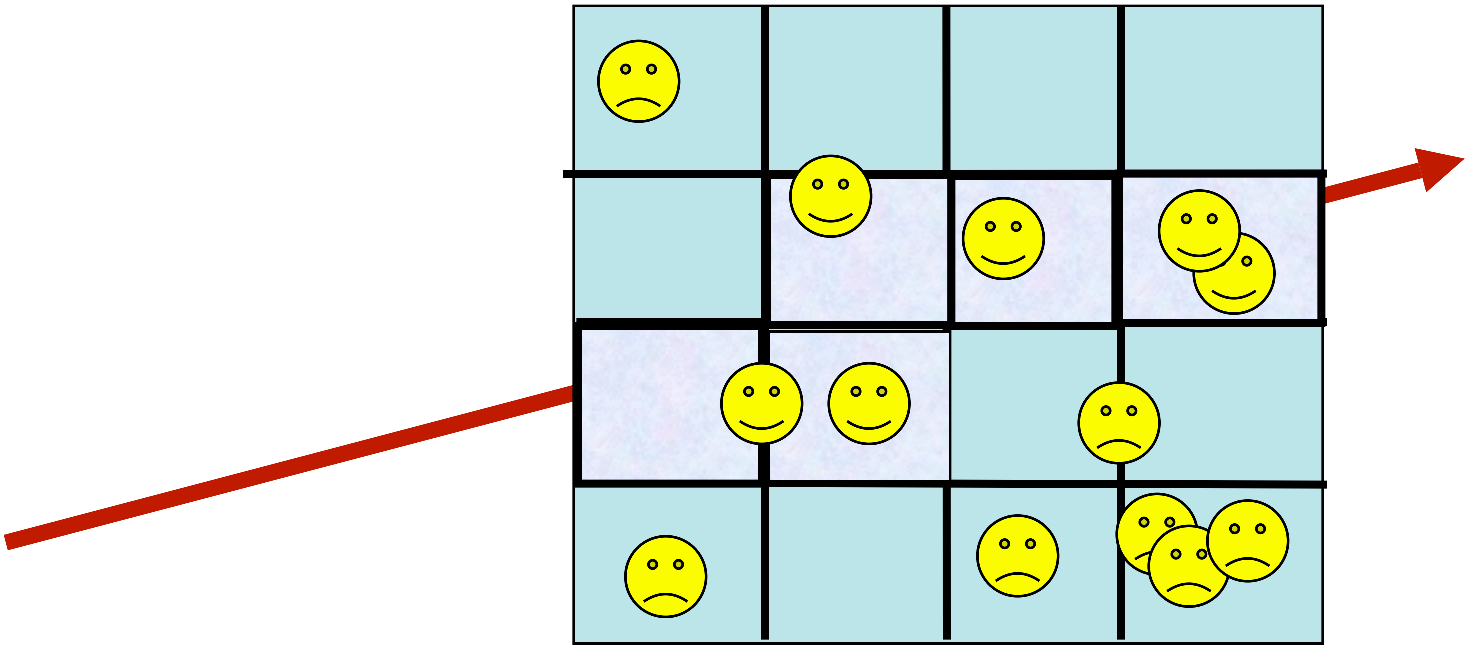


Spatial Enumeration

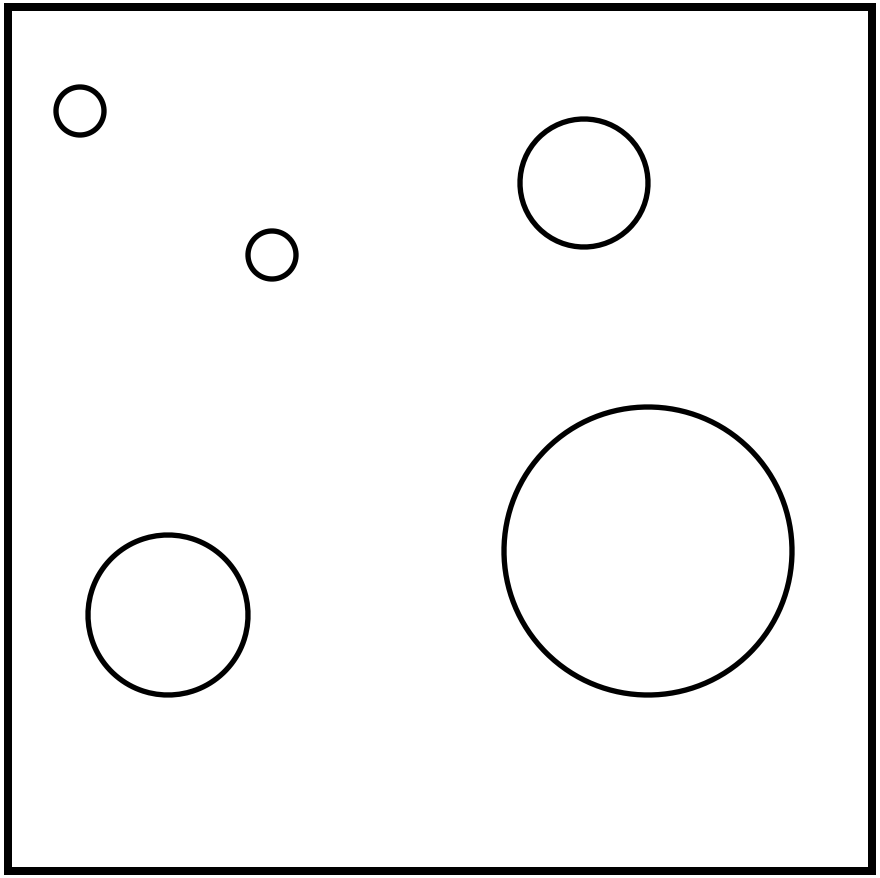
- Divide space into ‘voxels’
- Bucket sort objects in voxels they intersect
  - Object goes into each voxel it touches
  - Reuse results from one voxel calculation
- Determine voxels that a ray intersects
  - Only deal with the objects in those voxels

Spatial Enumeration

- Identifying voxels hit is like a line drawing algorithm

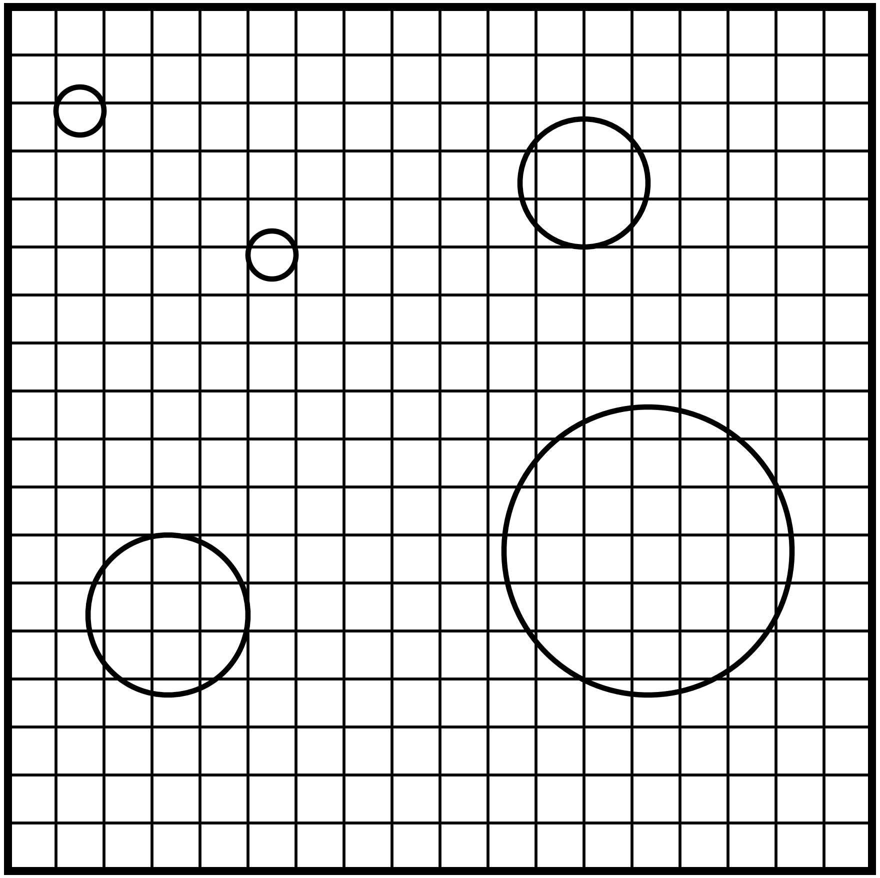


Uniform Grids



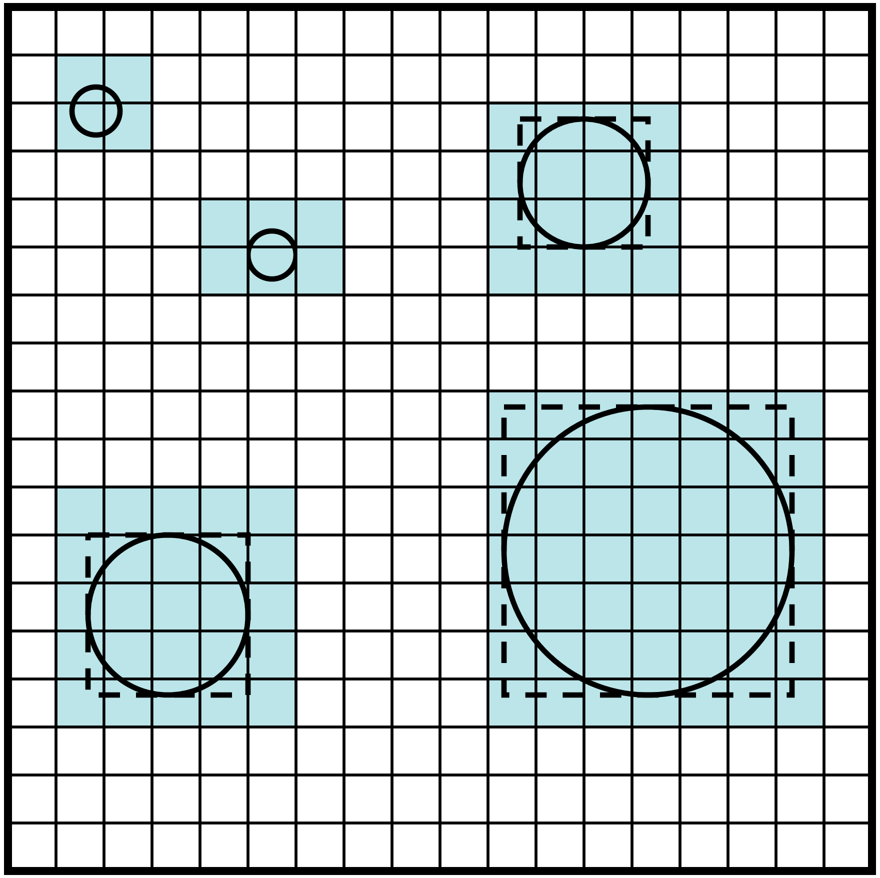
- Preprocess scene
- Find Big bounding box

Uniform Grids



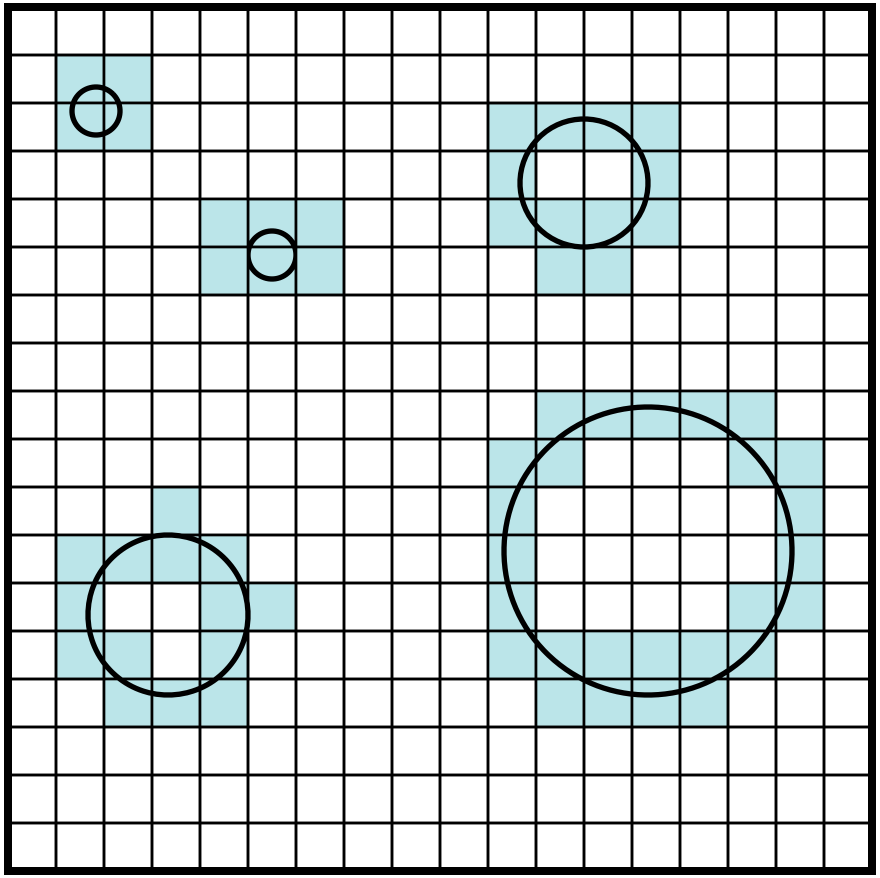
- Preprocess scene
- Find Big bounding box
- Determine grid resolution (how ?)

Uniform Grids



- Preprocess scene
- Find bounding box
- Determine grid resolution
- Place object in cell if its bounding box overlaps the cell

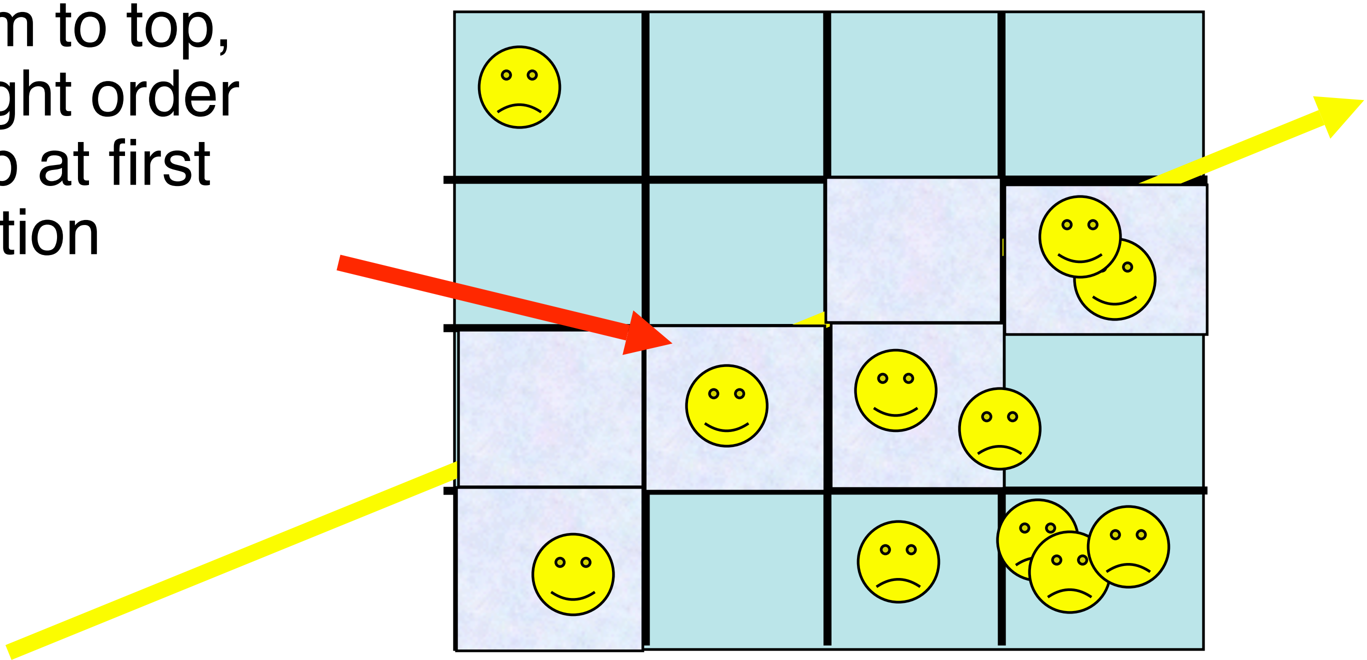
Uniform Grids



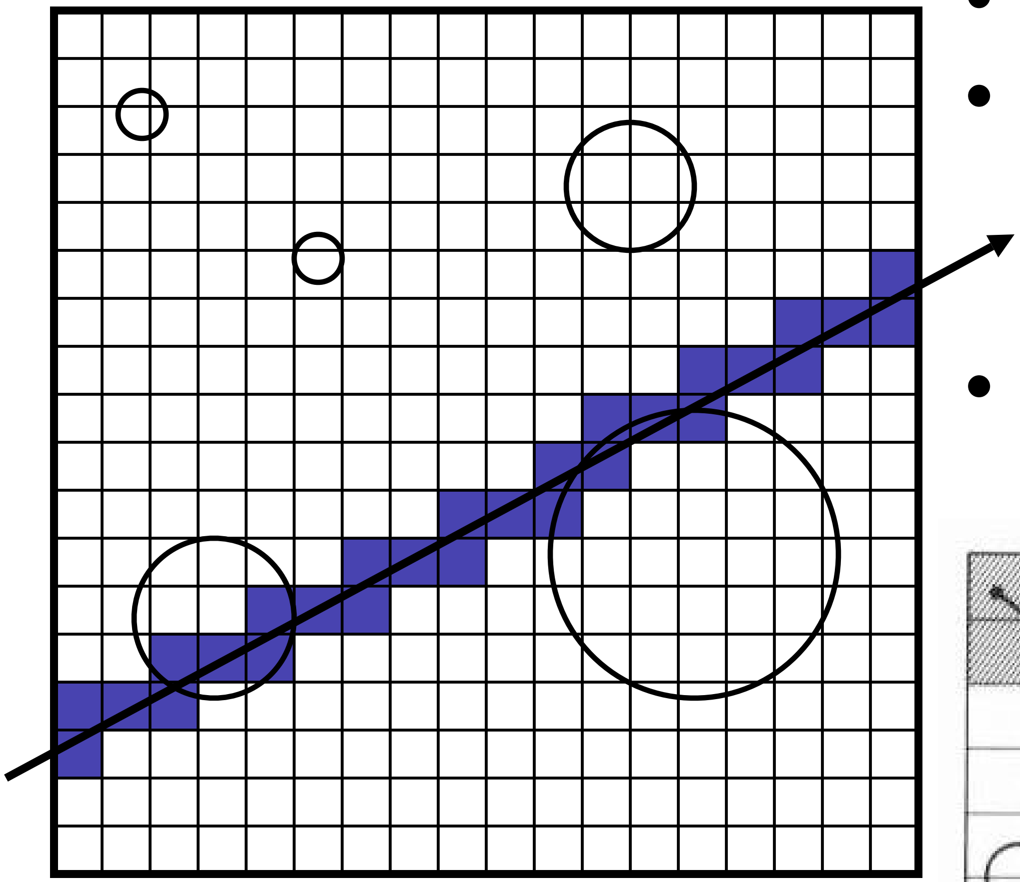
- Preprocess scene
- Find Big bounding box
- Determine grid resolution
- Place object in cell if its bounding box overlaps the cell
- Check that object overlaps cell (expensive!)

Add Sorting

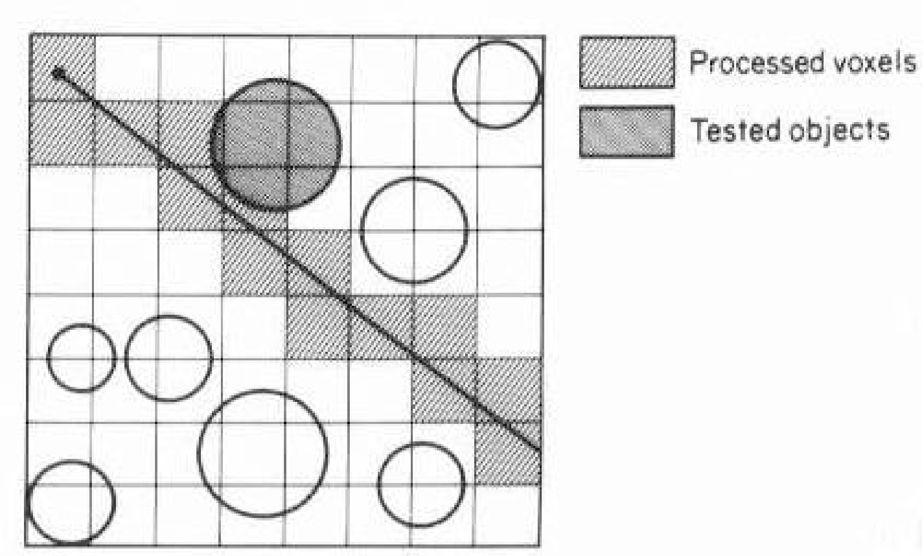
- If objects/voxels/cells are processed in front-to-back sorted order, stop processing when first intersection is detected
- e.g., process cells in bottom to top, left to right order and stop at first intersection



Uniform Grids



- Preprocess scene
- Traverse grid
  - 3D line = 3D-DDA
  - 6-connected line
- pbrt algorithm (grid accelarator)



Amanatides & Woo Algorithm

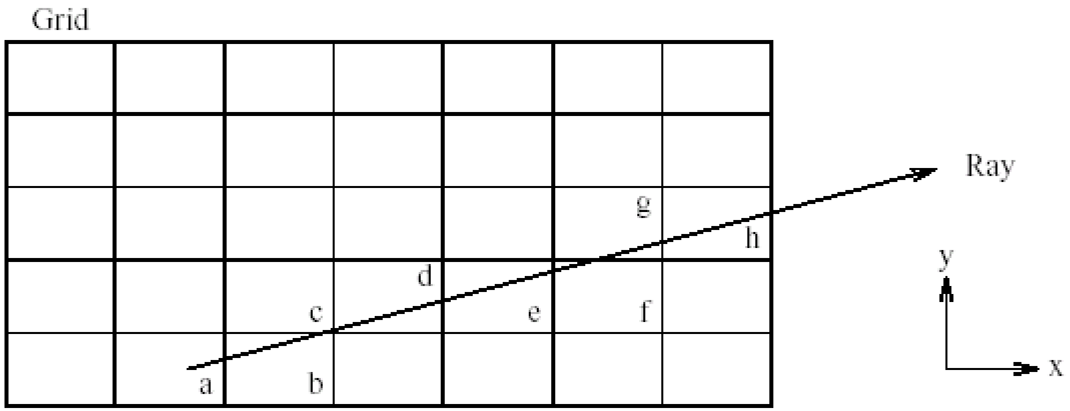


Figure 1

```
loop{
    if(tMaxX < tMaxY) {
        tMaxX= tMaxX + tDeltaX;
        X= X + stepX;
    } else  {
        tMaxY= tMaxY + tDeltaY;
        Y= Y + stepY;
    }
    NextVoxel(X,Y);
}
```

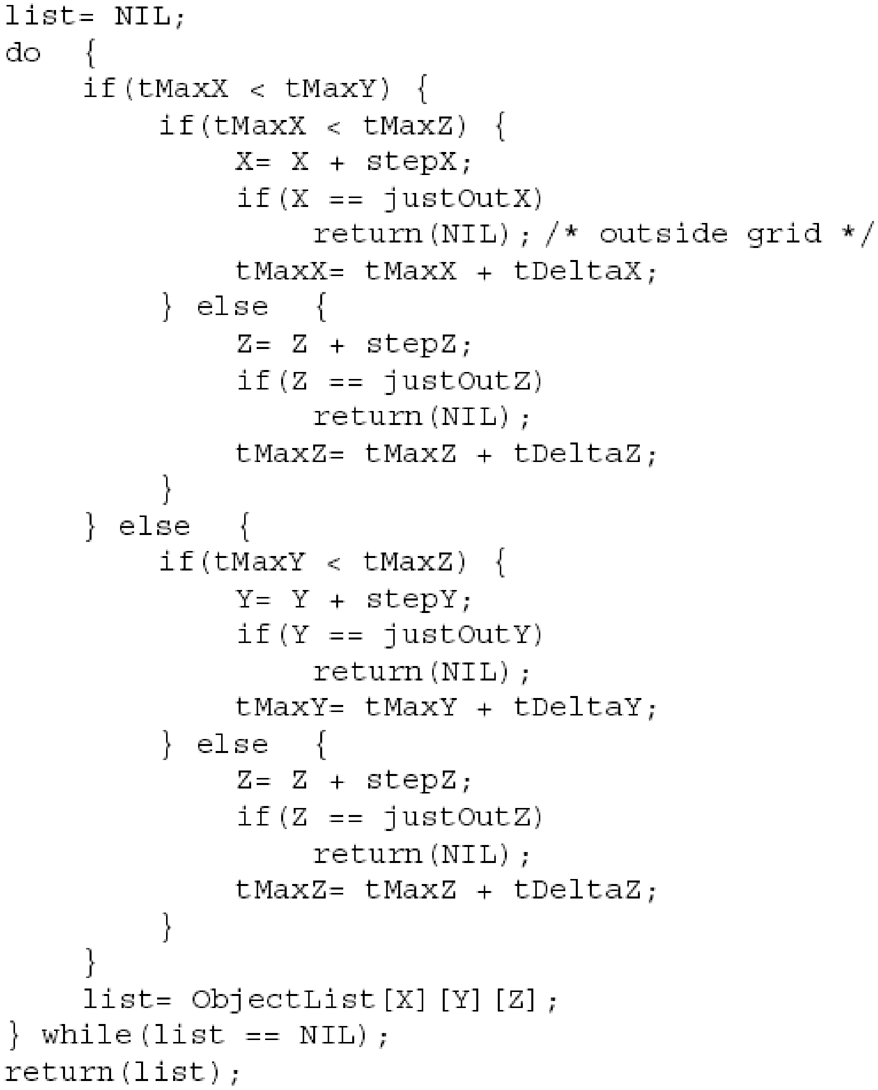
- J. Amanatides and A. Woo, "A Fast Voxel Traversal Algorithm for Ray Tracing", Proc. Eurographics '87, Amsterdam, The Netherlands, August 1987, pp 1-10.

Step[X,Y] +/- 1

tMax[X,Y] – first intersection

tDelta[X,Y] - voxel distance in [X,Y]

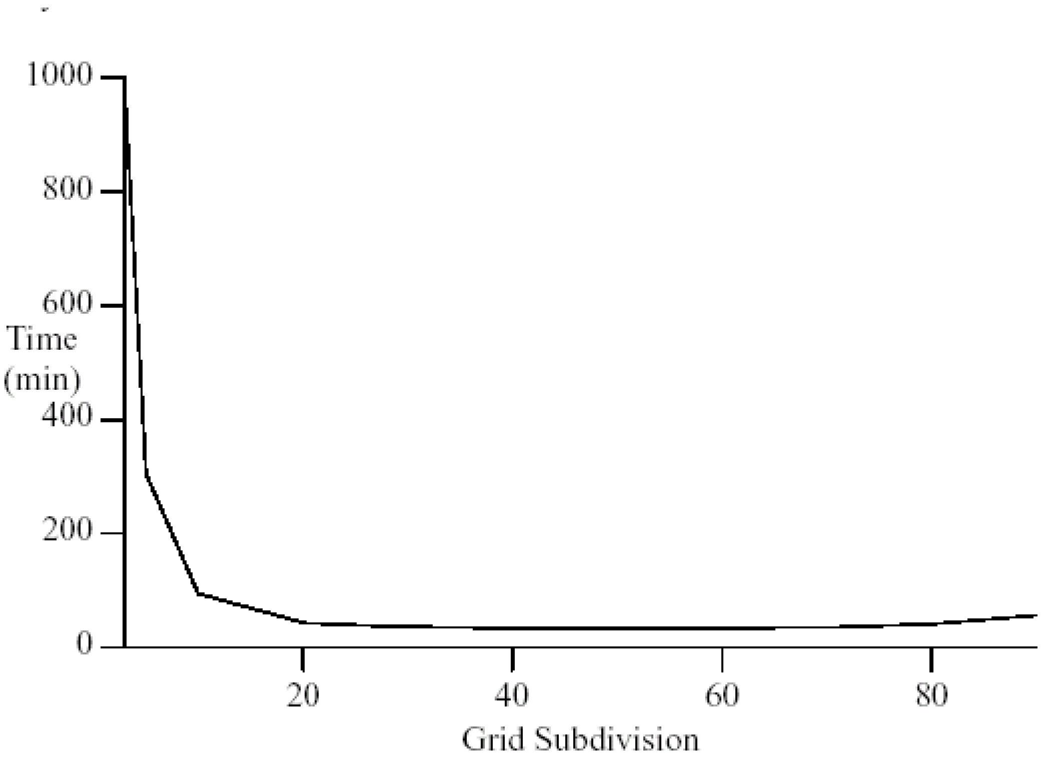
A&W Algorithm



```
list= NIL;
do  {
    if(tMaxX < tMaxY) {
        if(tMaxX < tMaxZ) {
            X= X + stepX;
            if(X == justOutX)
                return(NIL); /* outside grid */
            tMaxX= tMaxX + tDeltaX;
        } else  {
            Z= Z + stepZ;
            if(Z == justOutZ)
                return(NIL);
            tMaxZ= tMaxZ + tDeltaZ;
        }
    } else  {
        if(tMaxY < tMaxZ) {
            Y= Y + stepY;
            if(Y == justOutY)
                return(NIL);
            tMaxY= tMaxY + tDeltaY;
        } else  {
            Z= Z + stepZ;
            if(Z == justOutZ)
                return(NIL);
            tMaxZ= tMaxZ + tDeltaZ;
        }
    }
    list= ObjectList[X][Y][Z];
} while(list == NIL);
return(list);
```

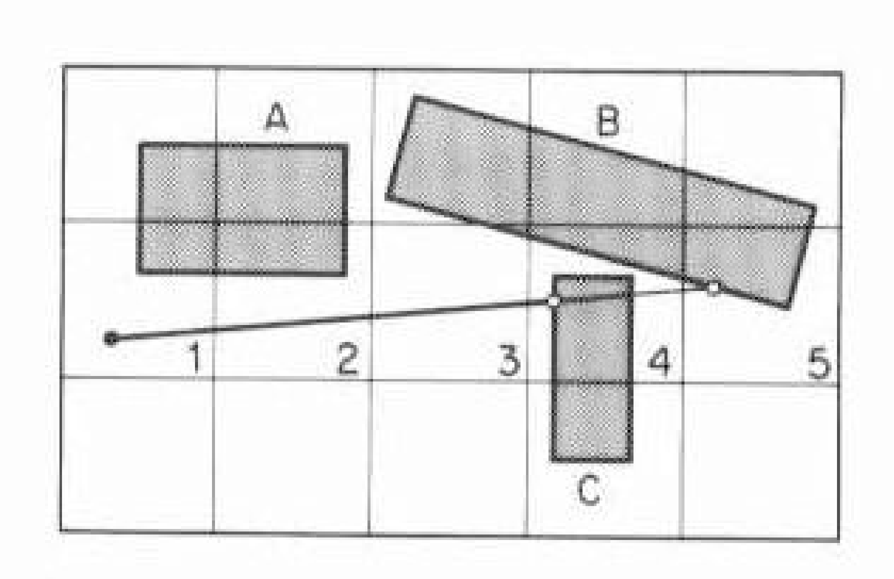
A&W Algorithm Results

- Rendering time for different levels of subdivision



Objects Across Multiple Voxels

- Mailboxes eliminate redundant intersection tests
- Objects have mailboxes
- Assign rays numbers
- check against objects last tested ray number
- Intersection must be within current voxel



Hierarchical Spatial Subdivision

- Recursive subdivision of space
- 1-1 Relationship between scene points and leaf nodes
- Example: point location by recursive search(log time)
- Solves the lack-of-adaptivity problem
- DDA works
- Effective in practice

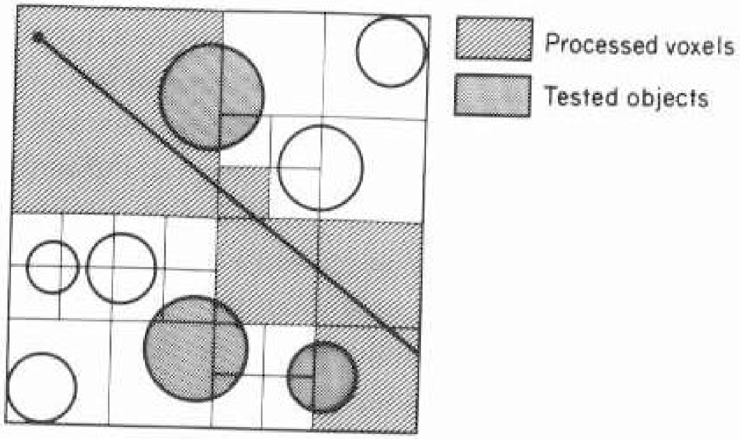
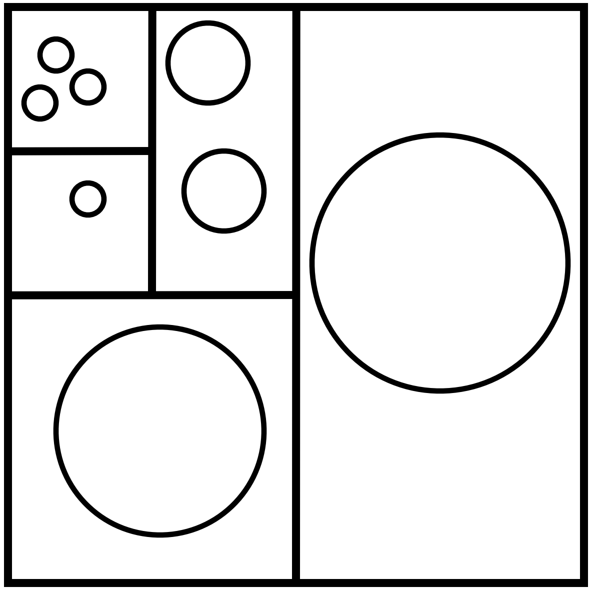
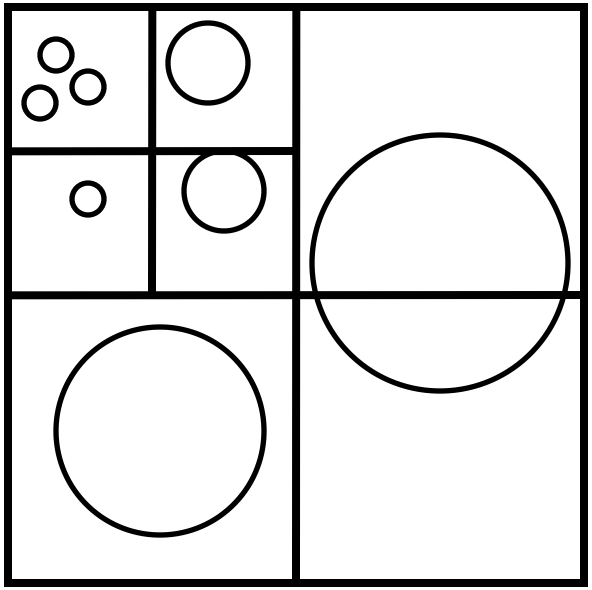


Fig. 13. Non-uniform spatial subdivision via an octree. The ray shown here causes five of the voxels to be examined and three of the eight objects to be tested for intersection. Finer subdivision can decrease the number of ray-object tests at the expense of additional voxel processing overhead.

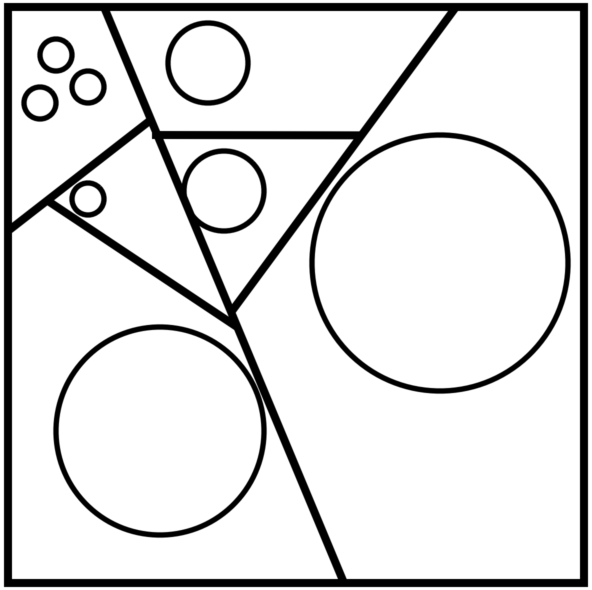
Variations



KD tree

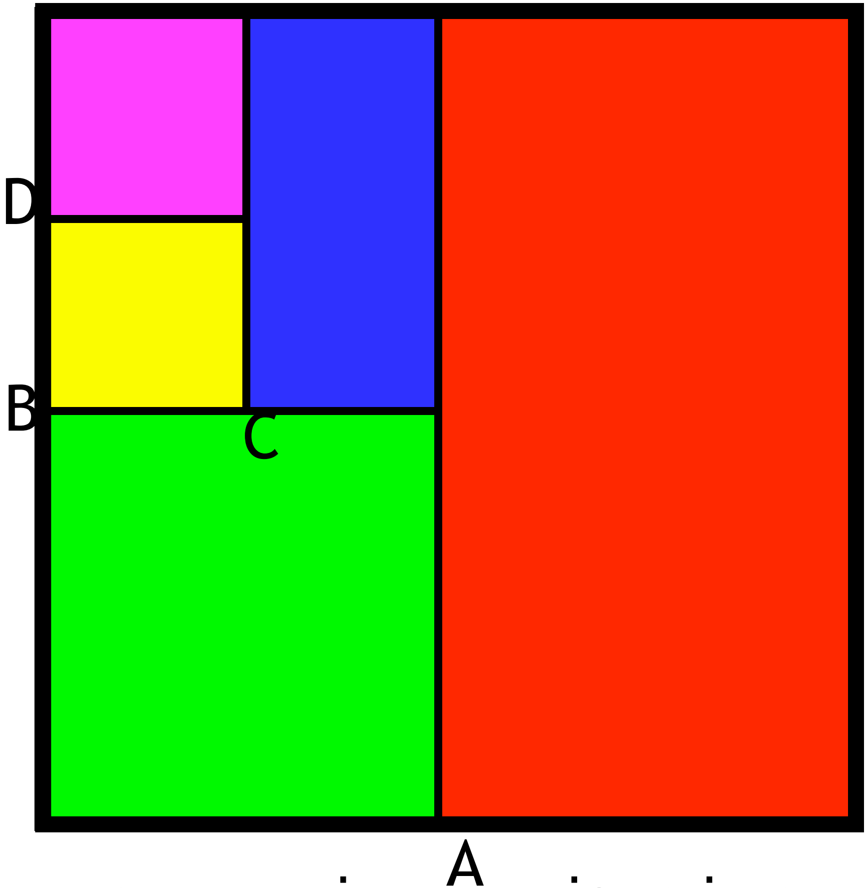


octtree



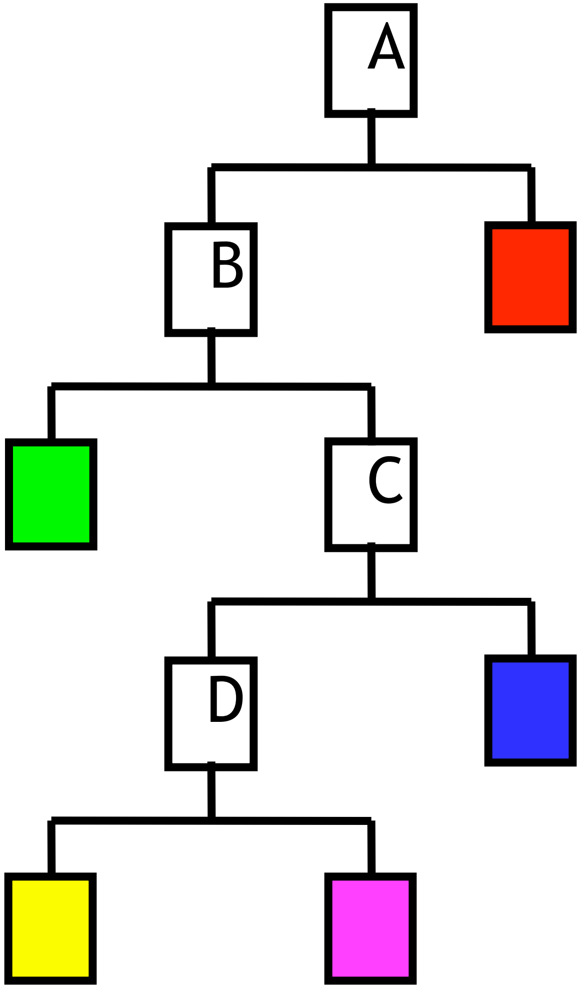
BSP tree

Example



Leaves are unique regions in space

Recursive search

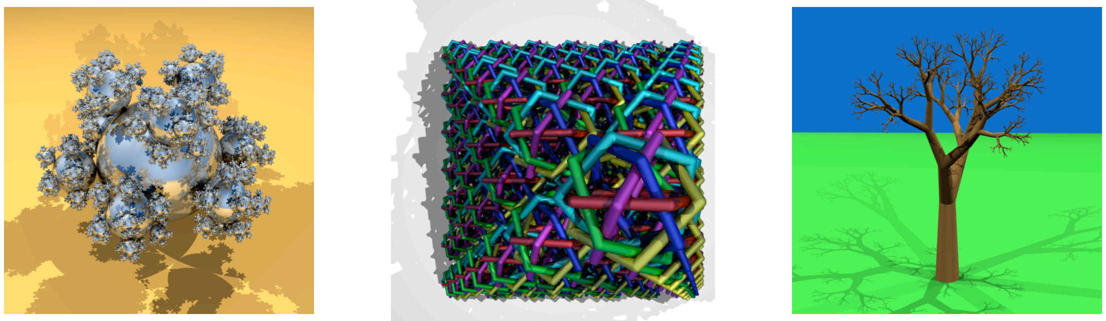


KdTreeAccel - pbrt

Creating Spatial Hierarchies

```
Insert(node,prim) {

    If (overlap(node->bound,prim)) {
        If (leaf(node)) {
            If (node->nprims > MAXPRIMS && node->depth < MAXDEPTH) {
                subdivide(node);
                foreach child in node
                    insert(child,prim)
            }
            else list_insert(node->prims,prim);
        }
        foreach child in node
            insert(child,prim)
    }
}

// Typically MAXDEPTH=16, MAX PRIMS = 2-8
```

Comparison



Scheme		Spheres	Rings	Tree
Uniform grid	D=1	244	129	1517
	D=20	38	83	781
Hierarchical grid		34	116	34

- See “A Proposal for Standard Graphics Environments”, IEEE Computer Graphics and Applications, vol. 7, no. 11, November 1987, pp. 3-5

Questions?

- “Teapot in a stadium” versus uniform distribution
- Multiplicative constants important
- Adaptivity allows robustness
- Cache effects are important