Topics

1. Introduction: What is Computer Graphics?
2. Raster Images (image input/output devices and representation)
3. Scan conversion (pixels, lines, triangles)
4. Ray Casting (camera, visibility, normals, lighting, Phong illumination)
5. Ray Tracing (shadows, supersampling, global illumination)
6. Spatial Data Structures (AABB trees, OBB, bounding spheres, octree)
7. Meshes (connectivity, smooth interpolation, uv-textures, subdivision, Laplacian smoothing)
8. 2D/3D Transformations (Translate, Rotate, Scale, Affine, Homography, Homogeneous coordinates)
9. Viewing and Projection (matrix composition, perspective, Z-buffer)
10. Shader Pipeline (Graphics Processing Unit)
11. Animation (kinematics, keyframing, Catmull-Romm interpolation, physical simulation)
12. 3D curves and objects (Hermite, Bezier, cubic curves, curve continuity, extrusion/revolve surfaces)
13. Advanced topics overview

Topic 6.

Spatial Data Structures

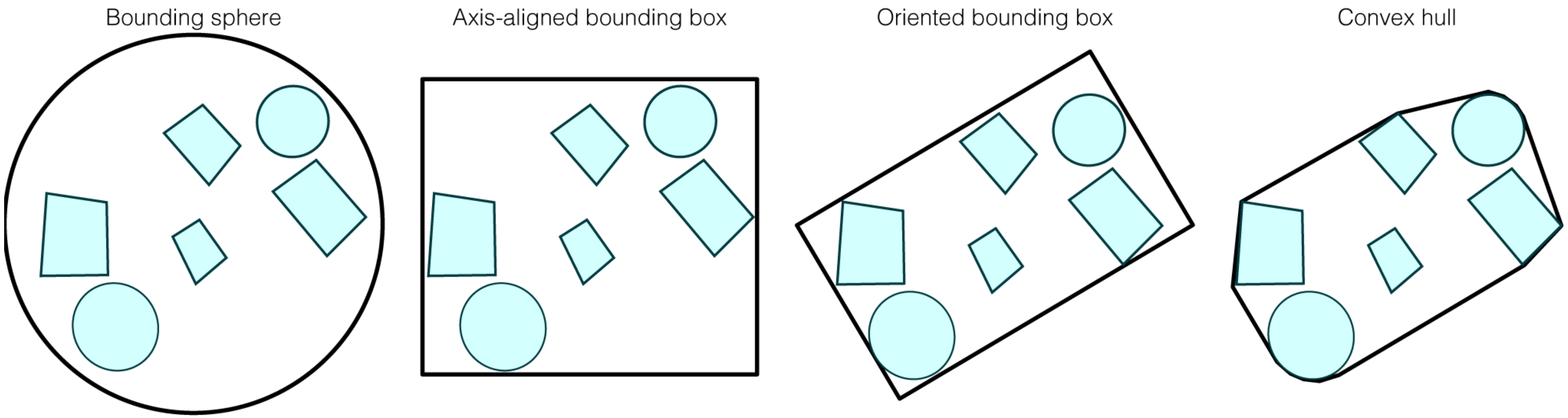
*some slides adapted from Steve Marschner

Bounding volumes

Quick way to avoid intersections and collisions:

bound object with a simple volume

- Object is fully contained in the volume
- If it doesn’t hit the volume, it doesn’t hit the object
- So test **bvol** first, then test object if it hits



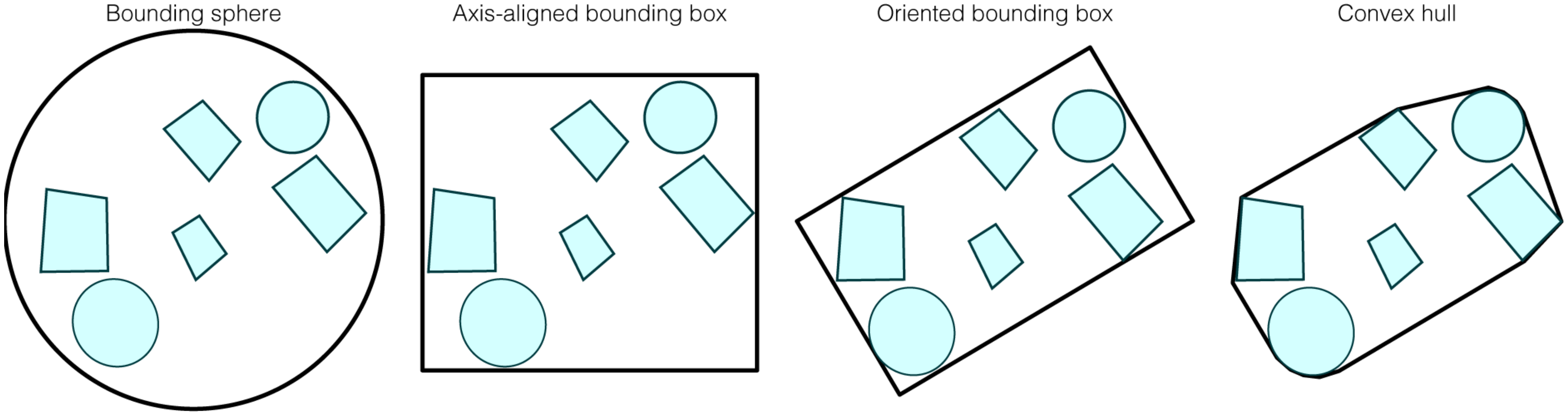
Choice of bounding volumes

Spheres: easy to intersect, not always tight.

Axis-aligned Bounding Boxes (AABBs): easy to intersect, tighter for axis-aligned objects.

Oriented bounding boxes (OBBs): easy to intersect (but cost of transformation), tighter than AABBs.

Convex Hull: not as easy to intersect as the above, tighter than the above.

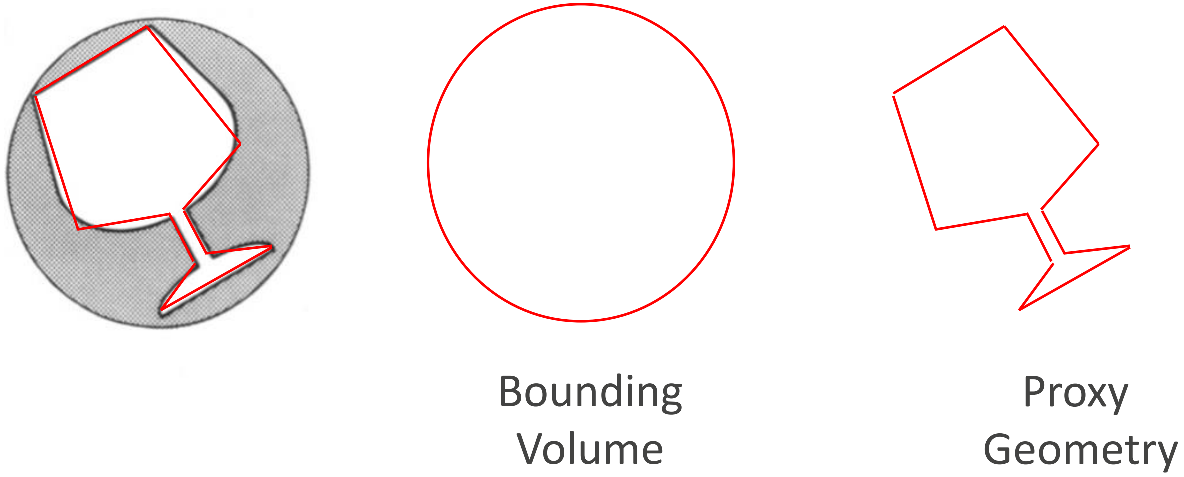


Proxy Geometry vs. Bounding volumes

Another concept often used in CG is proxy geometry or Level-Of-Detail LOD.

This is refers to a simplified representation of the object, that can be used as the object when rendering and processing speed is more important than visual accuracy.

Note, that proxy geometry is a simpler approximation to the object and typically not a bounding volume. Conversely a bounding volume itself is typically not a good visual proxy for an object.



Bounding volume Intersections

Cost: more for hits and near misses, less for far misses

Worth doing? It depends:

- Cost of **bvol** intersection test should be small
  Therefore use simple shapes (spheres, boxes, …)
- Cost of object intersect test should be large
  bvol most useful for complex objects
- Tightness of fit should be good
  Loose fit leads to extra object intersections
  Tradeoff between tightness and **bvol** intersection cost

Implementing a bounding volume

Add new Surface subclass, *BoundedSurface*

- Contains a *bvol* and a reference to a *surface*
- Intersection method:

```
if (!bvol.intersect(ray,t))
        return false;
else
        return surface.intersect(ray,t);
```

- This change is transparent to the renderer (only it might run faster).

Implementing a bounding volume hierarchy

A *BoundedSurface* can contain a *surface* list.

Any *surface* in this list might also be a *BoundedSurface*

=> A bounding volume hierarchy

Axis aligned bounding boxes

Probably easiest to implement

Computing for primitives

- Cube: duh!
- Sphere, cylinder, etc.: pretty obvious
- Groups or meshes: min/max of component parts

How to intersect them

- Treat them as an intersection of slabs (see book 12.3.1)

  Box plane equations look like $x = xmin$, $x = xmax$, …and similarly for y and z.

  => Intersection with ray $\boldsymbol{e}+t\boldsymbol{d}$ is $t=(xmin-e_x)/d_x$

AABB Tree Construction

Make AABB for whole scene/object, then split into two parts

- Recurse on parts.
- Stop when there is one (or a few) object/triangle in your box.

How to split into parts?

Space based: partition objects based on value relative to the center of longest dimension.

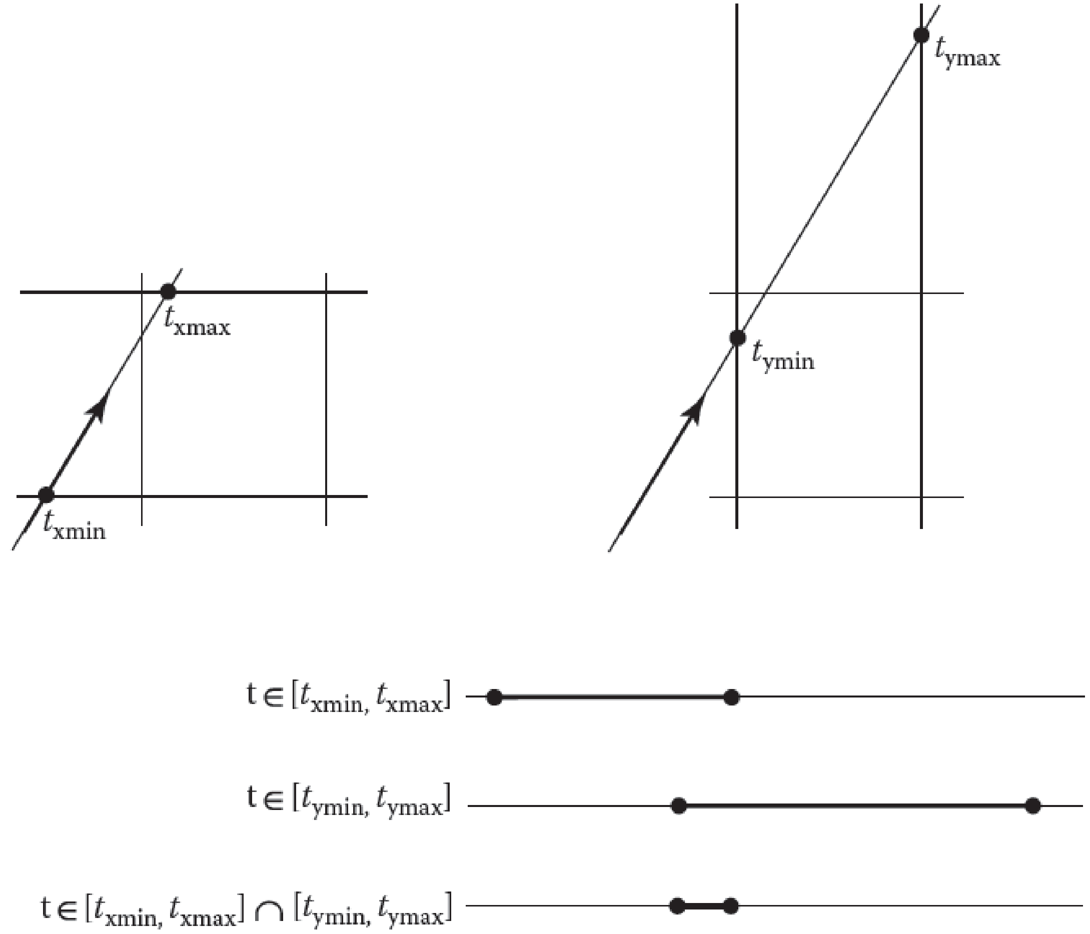
Object based: sort the objects along the longest dimension and divide them equally.

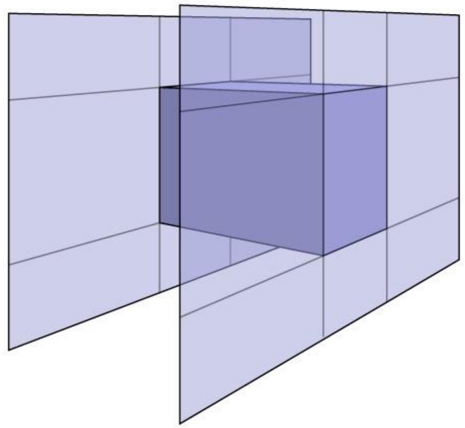
Ray and AABB intersection

Treat AABB as an intersection of slabs (see book 12.3.1)

Box plane equations look like $x = xmin$, $x = xmax$, …and similarly for y and z.

=> Intersection with ray $\boldsymbol{e}+t\boldsymbol{d}$ and $x = xmin$ has $t=(xmin-e_x)/d_x$

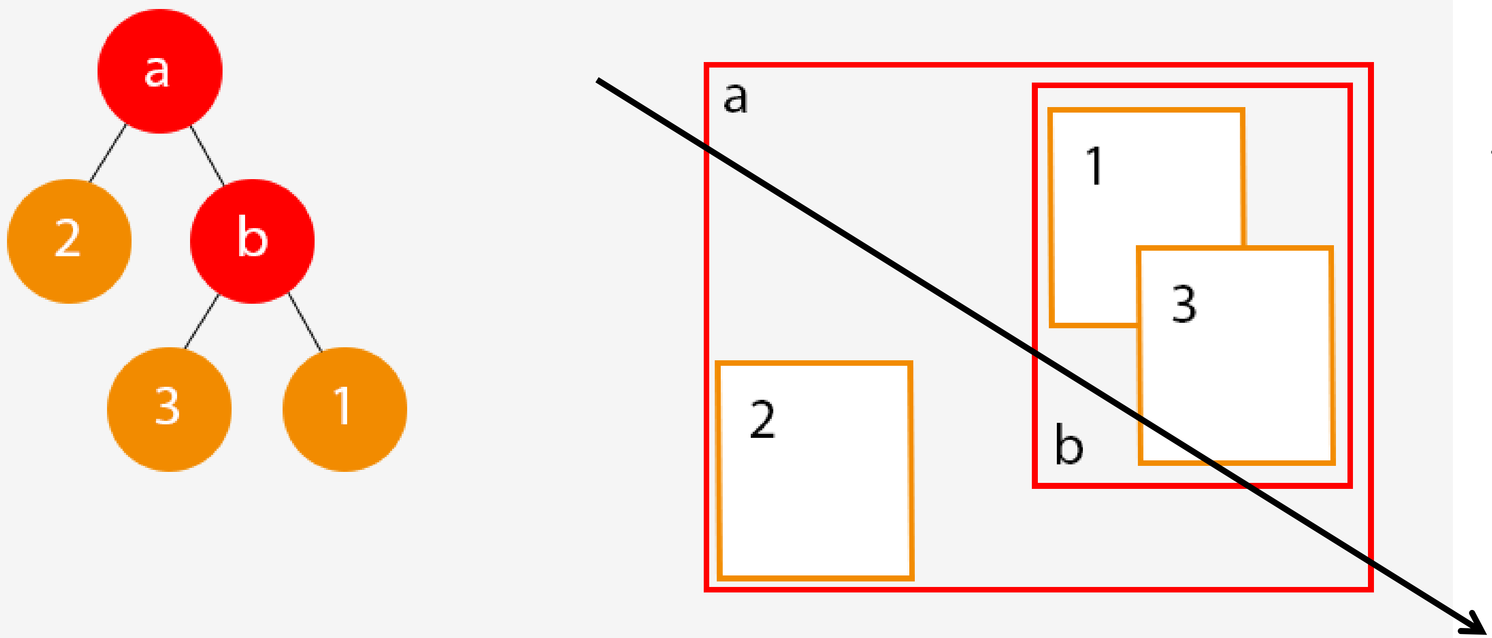




Intersection of the ray with a *xyz*-slab is an interval in *t.*
Intersection of the 3 intervals gives the intersection of the ray and AABB.

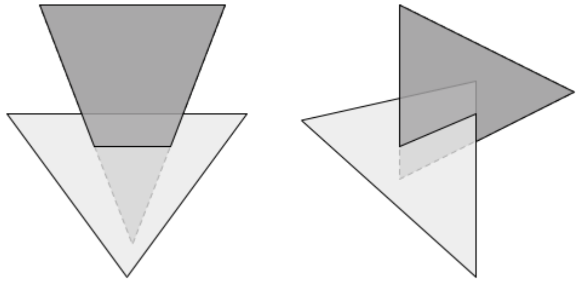
Note: This is shown on the left in 2D.

Ray and AABB tree Intersection




```
bvh::intersect(ray,t)
{
   if (aabb== null || !aabb.intersect(ray,t))
            return false;
   else
   {
     i1=left.intersect(ray,t1);
     i2=right.intersect(ray,t2);
     if (i1 && i2) {t=min(t1,t2); return true;}
     if (i1) {t=t1; return true;}
     if (i2) {t=t2; return true;}
     return false;
   }
}
```

Triangle-Triangle intersection



T_1 intersects T_2 <=> at least one tri edge intersects the other tri.

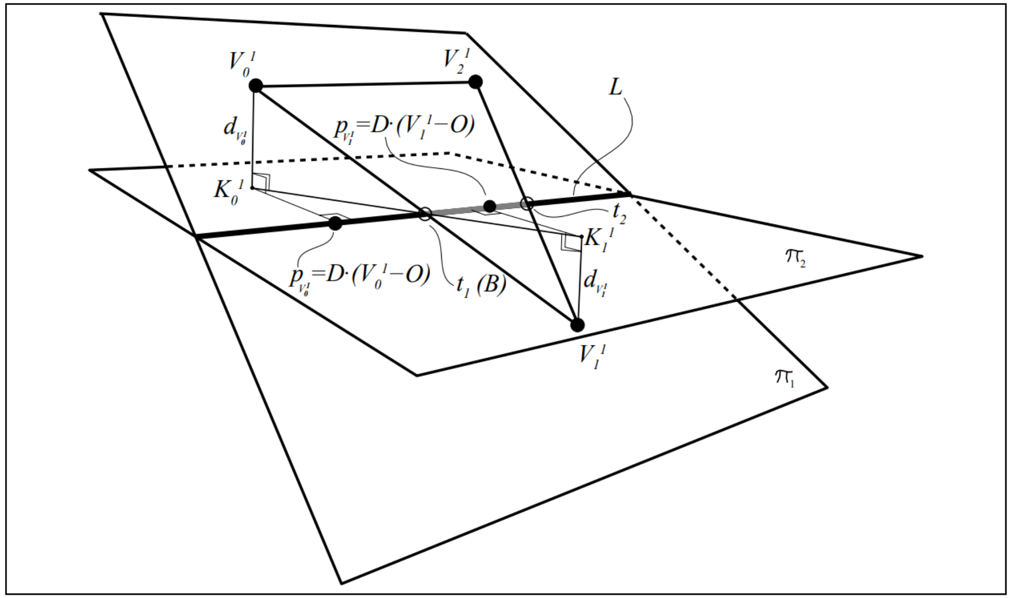
Algorithm 1: Test edge-tri intersection for all 6 edges.

T_1 intersects T_2 => Vertices of T_1 , T_2 straddle plane of T_2 , T_1 respectively.

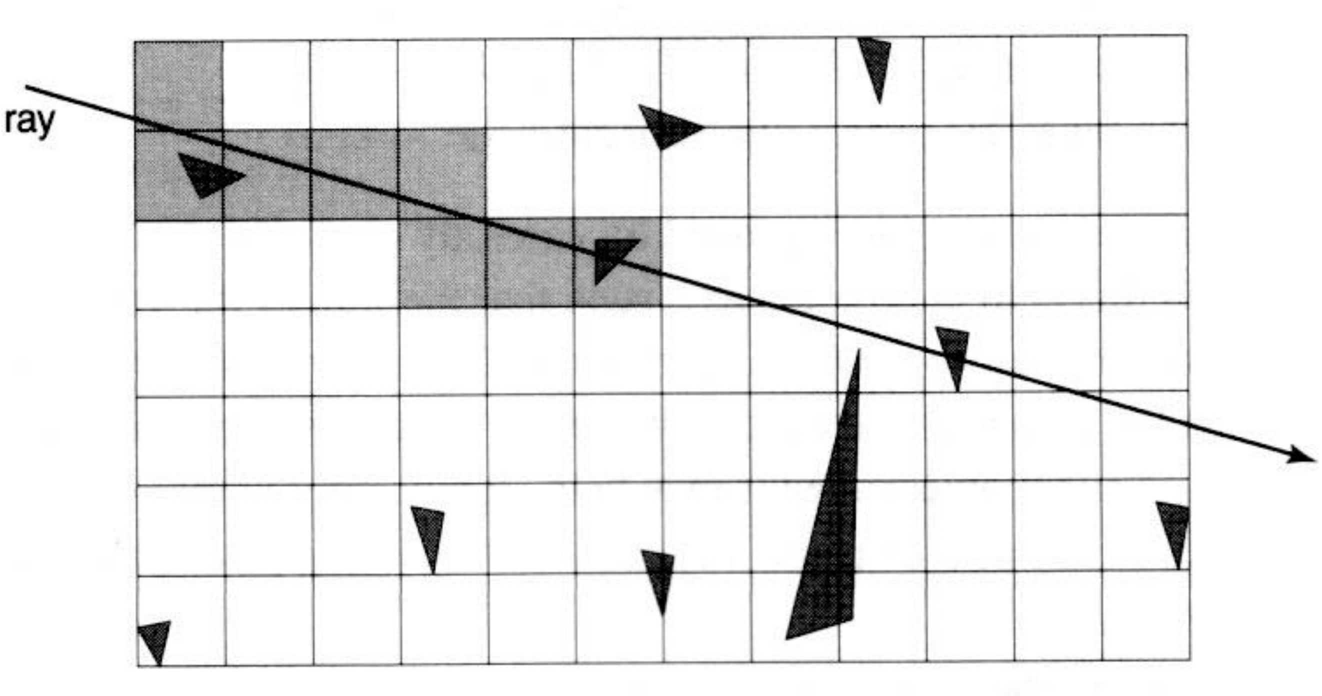
What if T_1 , T_2 are co-planar?

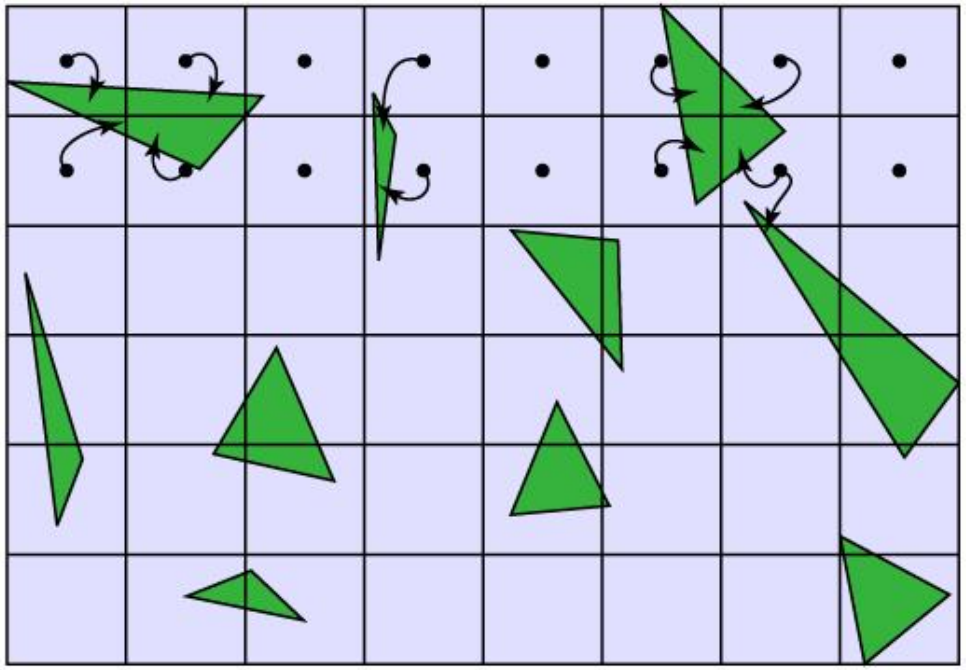
Algorithm 2: if (vertices of T_1 , T_2 on the same side of plane of T_2 , T_1) return false;

The triangles intersect iff the triangle intervals along line of intersection do.



Regular space subdivision

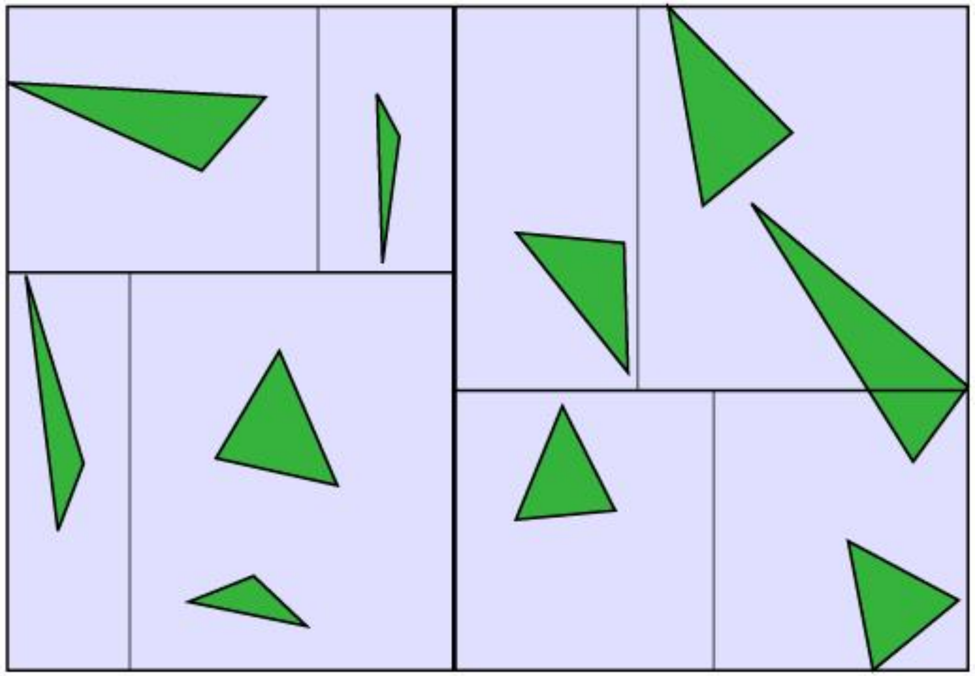




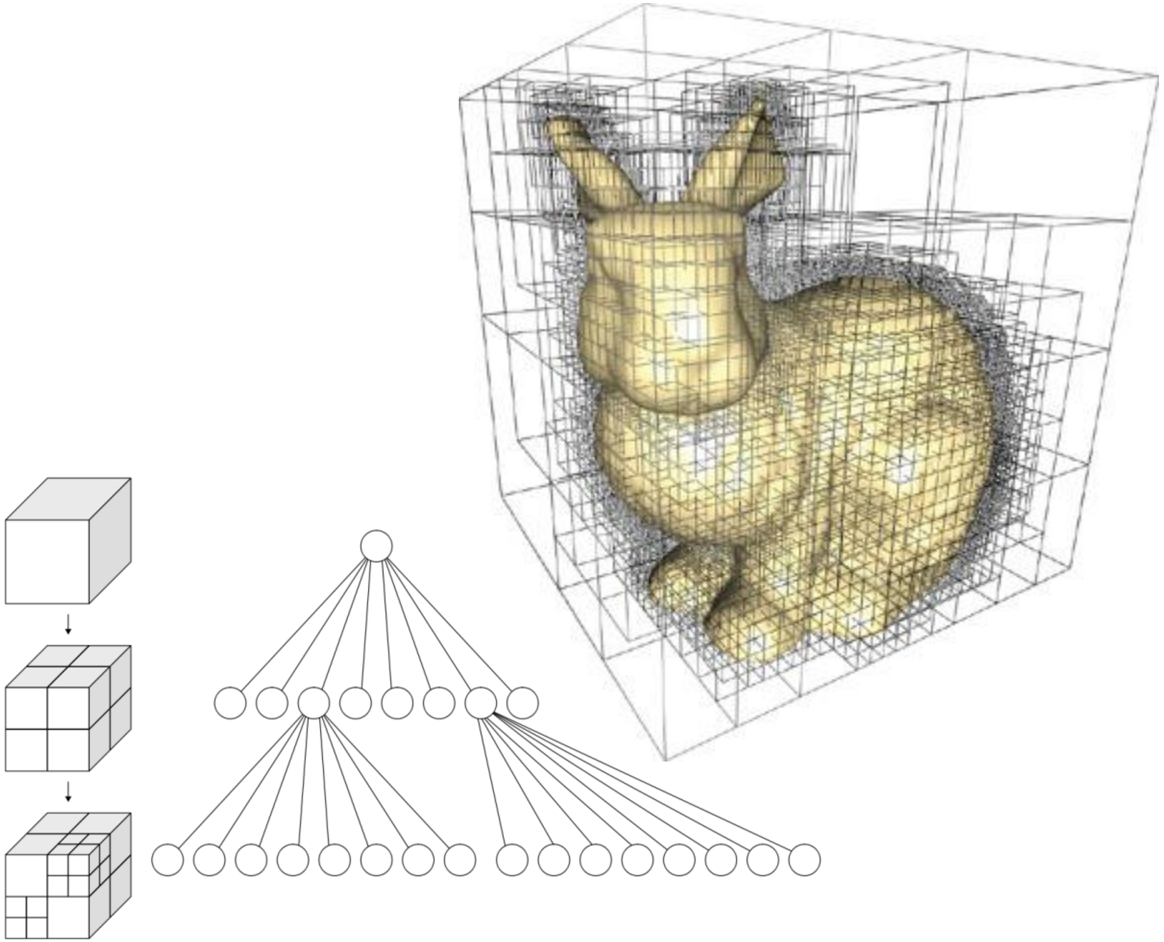
Grid divides space, not objects

Non-regular space subdivision

k-d Tree



Octree



Next Lecture: meshes

So far triangles have been individual and unconnected…

Meshes allow us to explicitly connect them to form more complex surfaces and volumes.