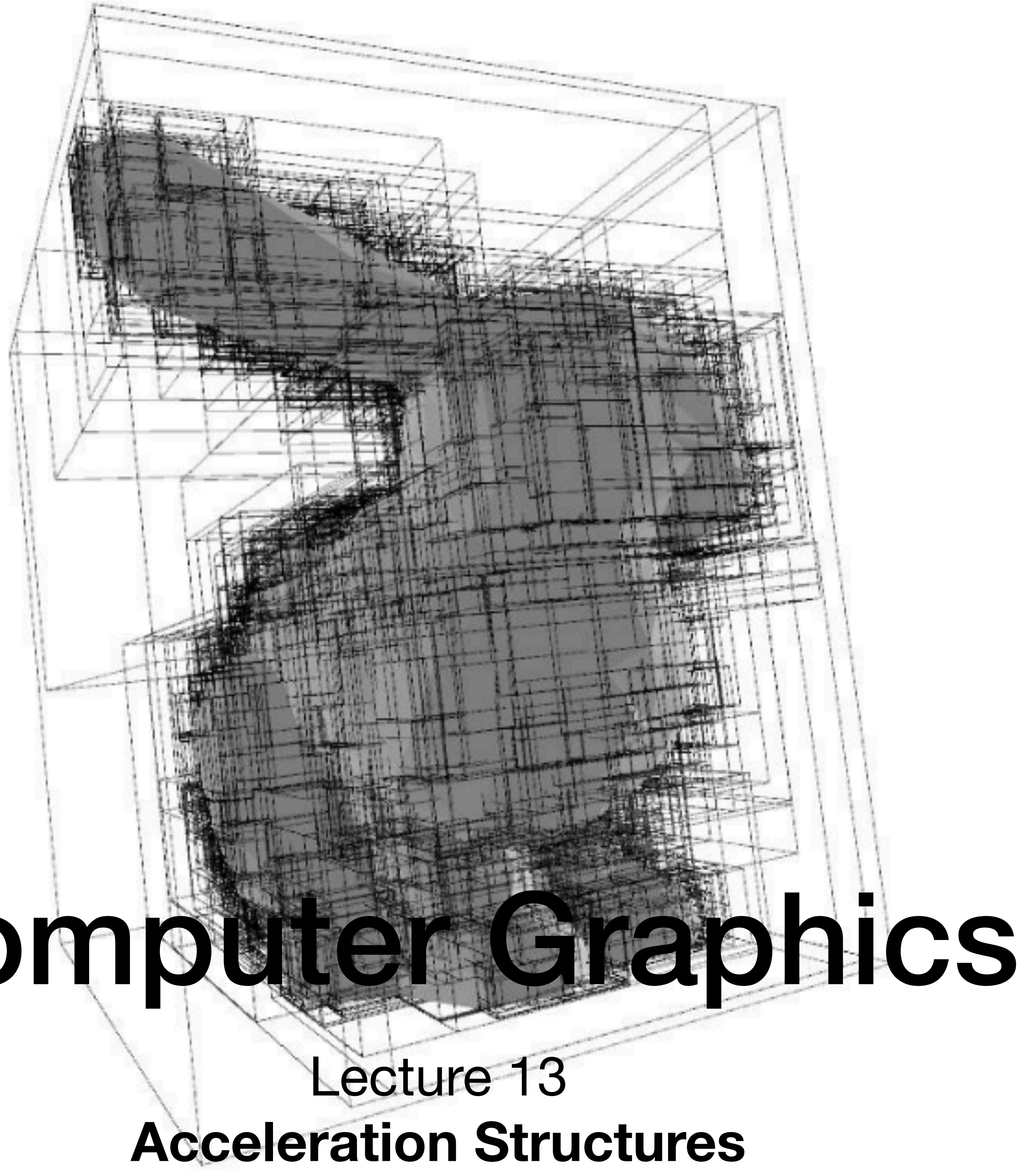


Computer Graphics

Lecture 13

Acceleration Structures

Ray tracing is expensive.

```
for each pixel:
    for each triangle:
        compute barycentric intersection
```

How expensive? Let's (informally) count some FLOPs.

floating-point operations

Reminder:
Barycentric ray-triangle intersection

$$\mathbf{p} + t\mathbf{d} = \mathbf{a} + \beta(\mathbf{b} - \mathbf{a}) + \gamma(\mathbf{c} - \mathbf{a})$$

$$\beta(\mathbf{a} - \mathbf{b}) + \gamma(\mathbf{a} - \mathbf{c}) + t\mathbf{d} = \mathbf{a} - \mathbf{p}$$

$$\begin{bmatrix} \mathbf{a} - \mathbf{b} & \mathbf{a} - \mathbf{c} & \mathbf{d} \end{bmatrix} \begin{bmatrix} \beta \\ \gamma \\ t \end{bmatrix} = \begin{bmatrix} \mathbf{a} - \mathbf{p} \end{bmatrix}$$

$$\begin{bmatrix} x_a - x_b & x_a - x_c & x_d \\ y_a - y_b & y_a - y_c & y_d \\ z_a - z_b & z_a - z_c & z_d \end{bmatrix} \begin{bmatrix} \beta \\ \gamma \\ t \end{bmatrix} = \begin{bmatrix} x_a - x_p \\ y_a - y_p \\ z_a - z_p \end{bmatrix}$$

lin. sys.

- This is a linear system: Ax = b
- Various ways to solve, but a fast one uses *Cramer's rule*.
- See 4.4.2 for the TL;DR formula
- See 5.3.2 for an explanation of Cramer's rule

Reminder:
Barycentric ray-triangle intersection

$$\mathbf{p} + t\mathbf{d} = \mathbf{a} + \beta(\mathbf{b} - \mathbf{a}) + \gamma(\mathbf{c} - \mathbf{a})$$

$$\beta(\mathbf{a} - \mathbf{b}) + \gamma(\mathbf{a} - \mathbf{c}) + t\mathbf{d} = \mathbf{a} - \mathbf{p}$$

$$\begin{bmatrix}\mathbf{a} - \mathbf{b} & \mathbf{a} - \mathbf{c} & \mathbf{d}\end{bmatrix} \begin{bmatrix}\beta \\ \gamma \\ t\end{bmatrix} = \begin{bmatrix}\mathbf{a} - \mathbf{p}\end{bmatrix}$$

$$\begin{bmatrix} x_a - x_b & x_a - x_c & x_d \\ y_a - y_b & y_a - y_c & y_d \\ z_a - z_b & z_a - z_c & z_d \end{bmatrix} \begin{bmatrix}\beta \\ \gamma \\ t\end{bmatrix} = \begin{bmatrix} x_a - x_p \\ y_a - y_p \\ z_a - z_p \end{bmatrix}$$

9 subtractions

Pre-calculate entries and rename:

$$\begin{bmatrix} a & d & g \\ b & e & h \\ c & f & i \end{bmatrix} \begin{bmatrix}\beta \\ \gamma \\ t\end{bmatrix} = \begin{bmatrix} j \\ k \\ l \end{bmatrix}$$

Barycentric Ray-Triangle Intersection

Cramer's rule gives us

5 add/sub
10 mult/div

Total: 27 FLOPs

$$\beta = \frac{j(ei-hf)+k(gf-di)+l(dh-eg)}{M},$$

$$\gamma = \frac{i(ak-jb)+h(jc-al)+g(bl-kc)}{M},$$

$$t = -\frac{f(ak-jb)+e(jc-al)+d(bl-kc)}{M},$$

where

Reusing from above:
3 mult

$$M = a(ei-hf)+b(gf-di)+c(dh-eg).$$

Assume (conservatively) that on average, we calculate β and determine that it doesn't intersect (because $\beta < 0$ or $\beta > 1$)

Ray tracing is expensive.

```
for each pixel:
  for each triangle:
    compute barycentric intersection
```

720p = 1280×720 = 921600 pixels

bunny: 114 triangles

27 flops

= 2,836,684,800

= 2.8 GFLOPs

Ray tracing is expensive.

```
for each pixel:
  for each triangle:
    compute barycentric intersection
```

720p = 1280×720 = 921600 pixels

bunny: 114 triangles

27 flops

= 2,836,684,800
= 2.8 GFLOPs

A typical laptop can currently can do about 100-200 GFLOPS

gigaflops per second

so what's the problem?

https://polycount.com/discussion/comment/2742856/#Comment_2742856

Ray tracing is expensive.

```
for each pixel:
  for each triangle:
    compute barycentric intersection
```

720p = 1280×720 = 921600 pixels

computer game model: 40k triangles

27 flops

= 995,328,000,000

= 995 GFLOPs

~= 1 TFLOP

Want to render this for an interactive game?
Simply do this 30+ times per second.

What can we do?

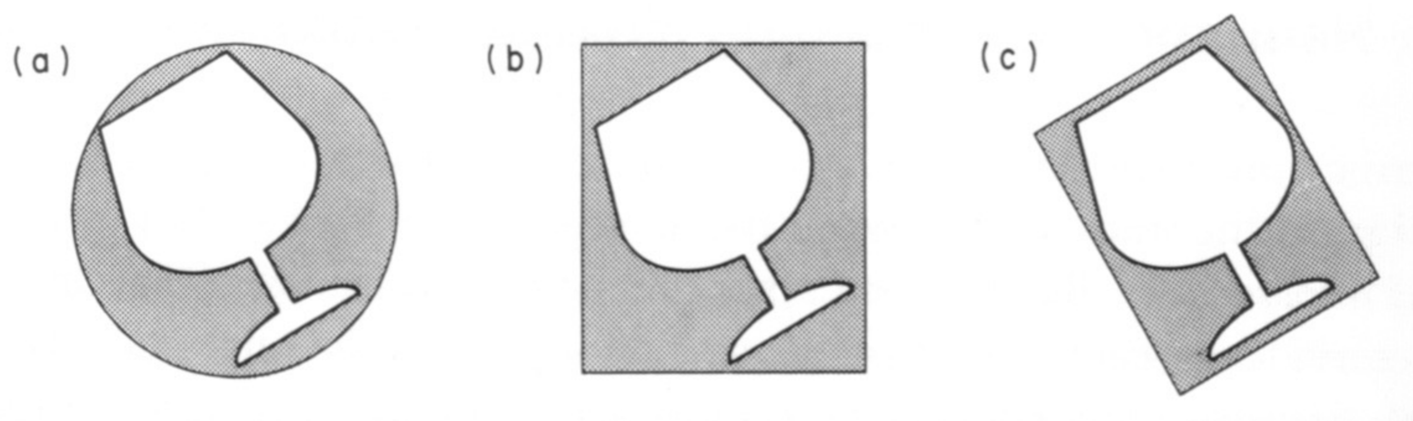
- ✓ Optimize the inner-inner loop: more efficient intersection routines
- ✓ Carefully reduce triangle count

these only go so far...

- Intersect fewer things
  - Most ray intersections don't hit the object!
  - Basic strategy: efficiently find big chunks of the scene that definitely **don't** intersect your ray

Bounding Volumes

- Quick way to avoid intersections: 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



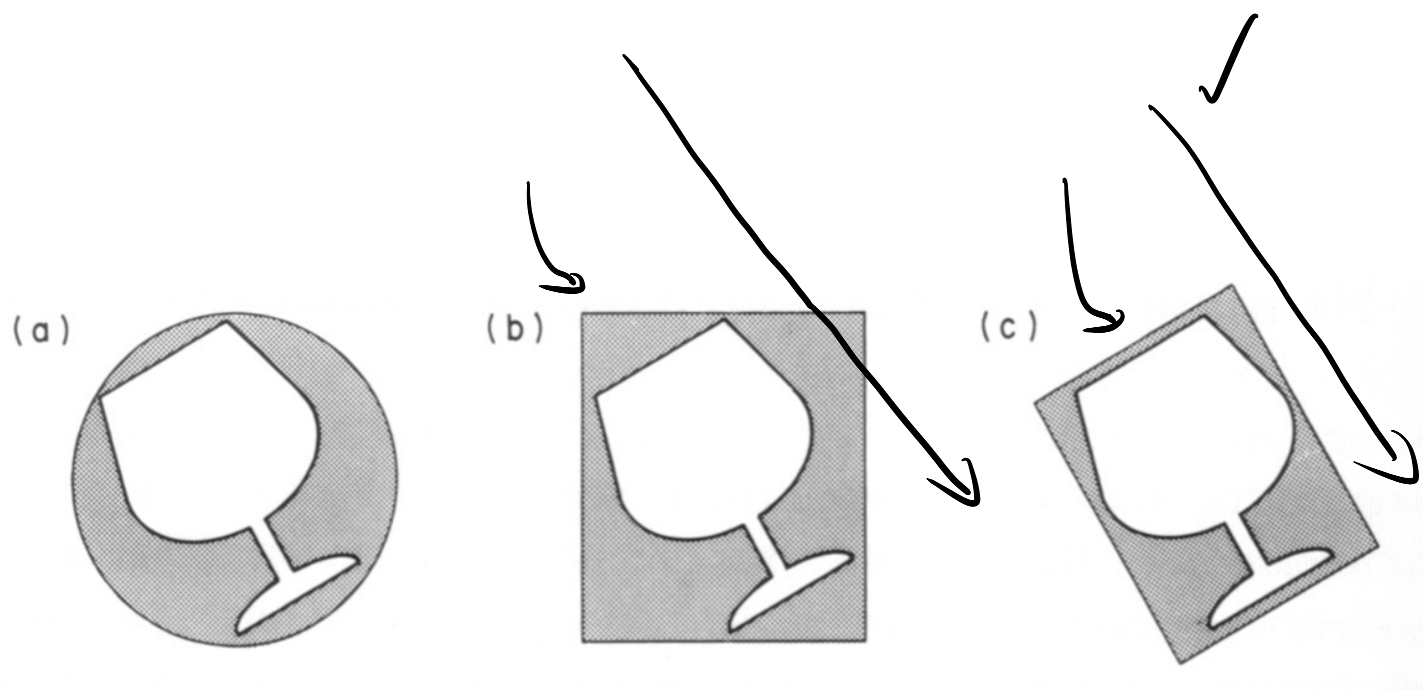
[Glassner 89, Fig 4.5]

sphere axis-aligned box oriented box

Bounding Volumes

Algorithm:

```
if ray intersects bounding volume:
    if ray intersects object:
        do stuff
```



[Glassner 89, Fig 4.5]

Bounding Volumes

Algorithm:

```
if ray intersects bounding volume:
    if ray intersects object:
        do stuff
```

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

Is this worth it?

- bvol intersection should be much cheaper than object intersection
  - works best for simple bvols, complicated objects
- bvol should bound object as tightly as possible

Tradeoff: **efficient intersection** vs **tightness**

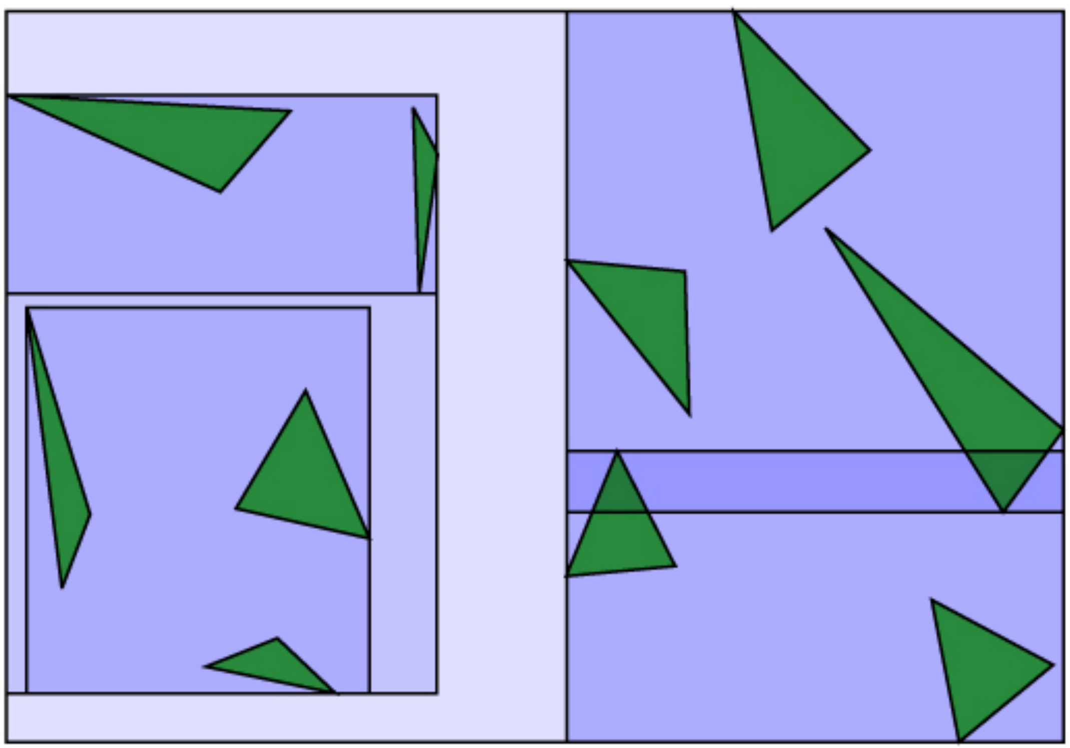
Bounding Volume Hierarchy

- Bvols around objects *might* help
- Bvols around groups of objects **will** help
- Bvols around parts of complex objects will help
- Idea: use bounding volumes all the way from the whole scene down to groups of a few objects

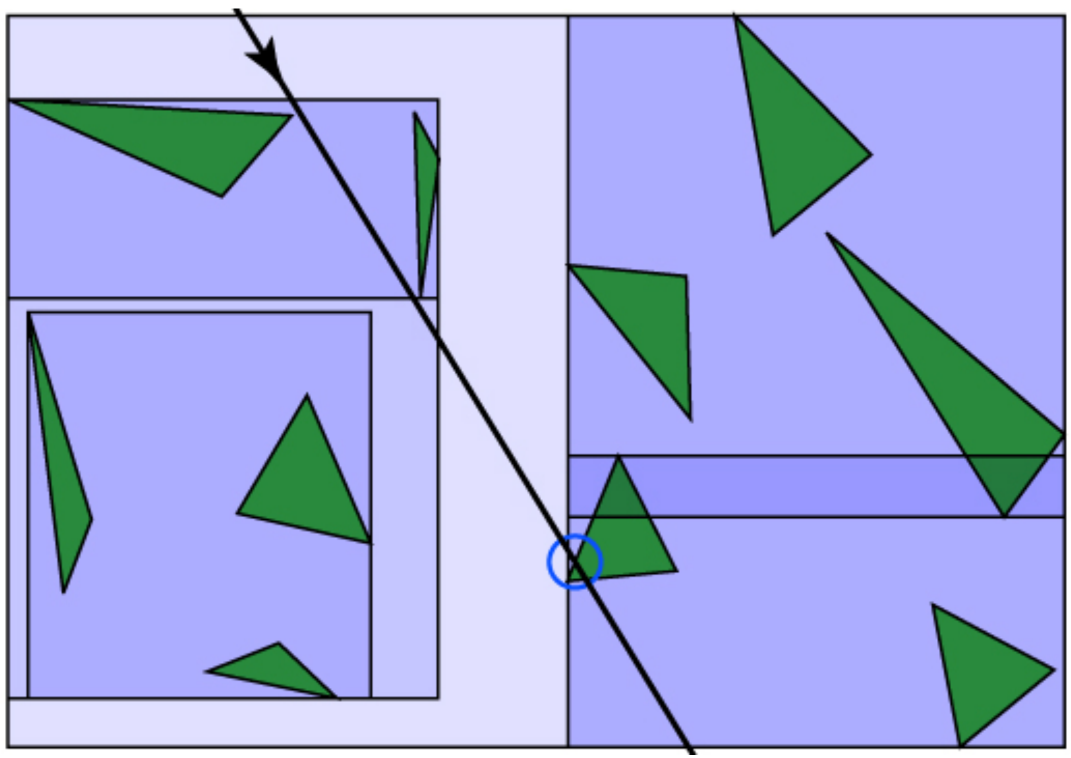
Building the Hierarchy

- Ideally: bound nearby clusters of objects
- Practical solution: partition along axis

BVH construction example



BVH ray-tracing example

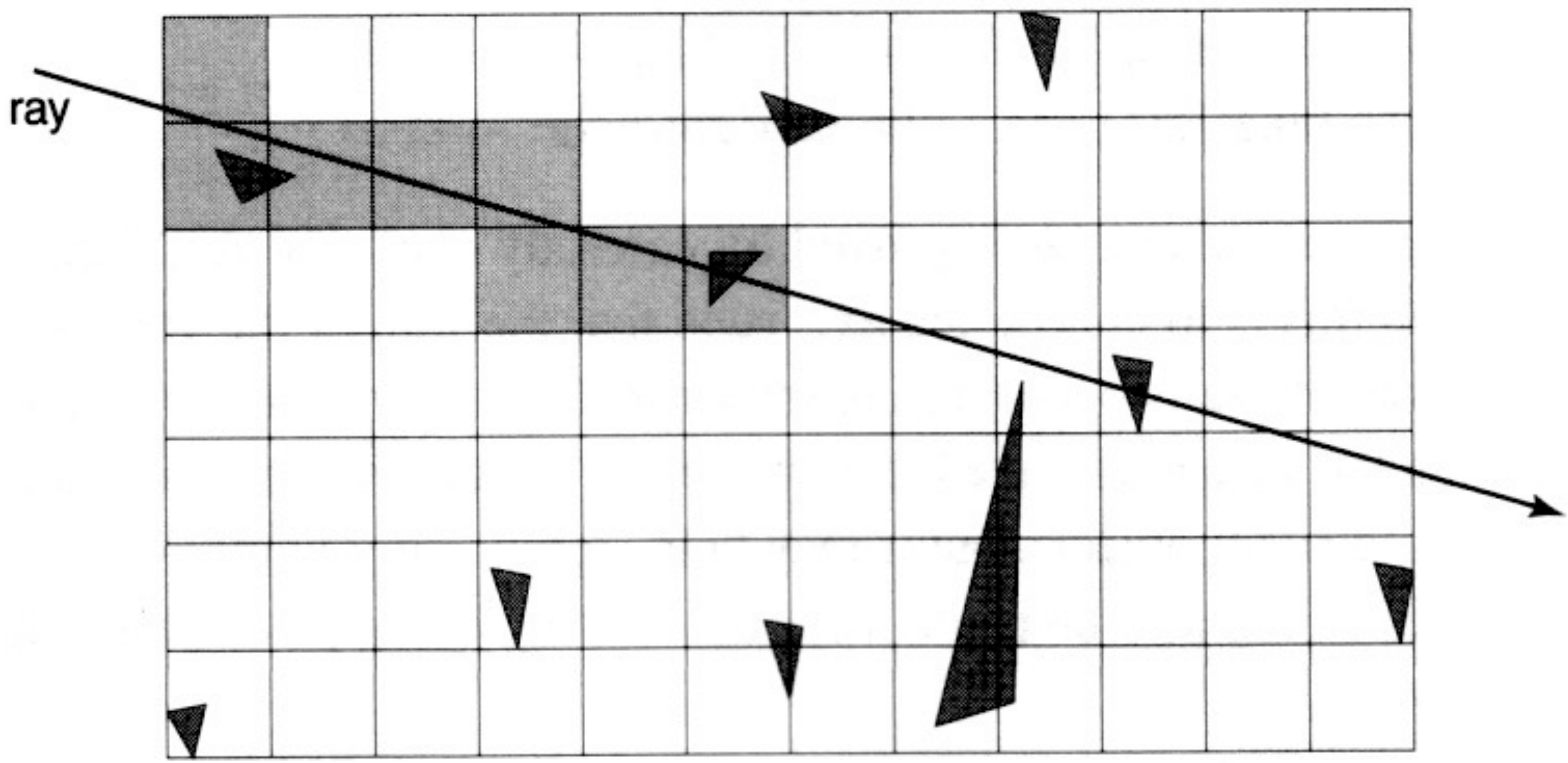


Implementation

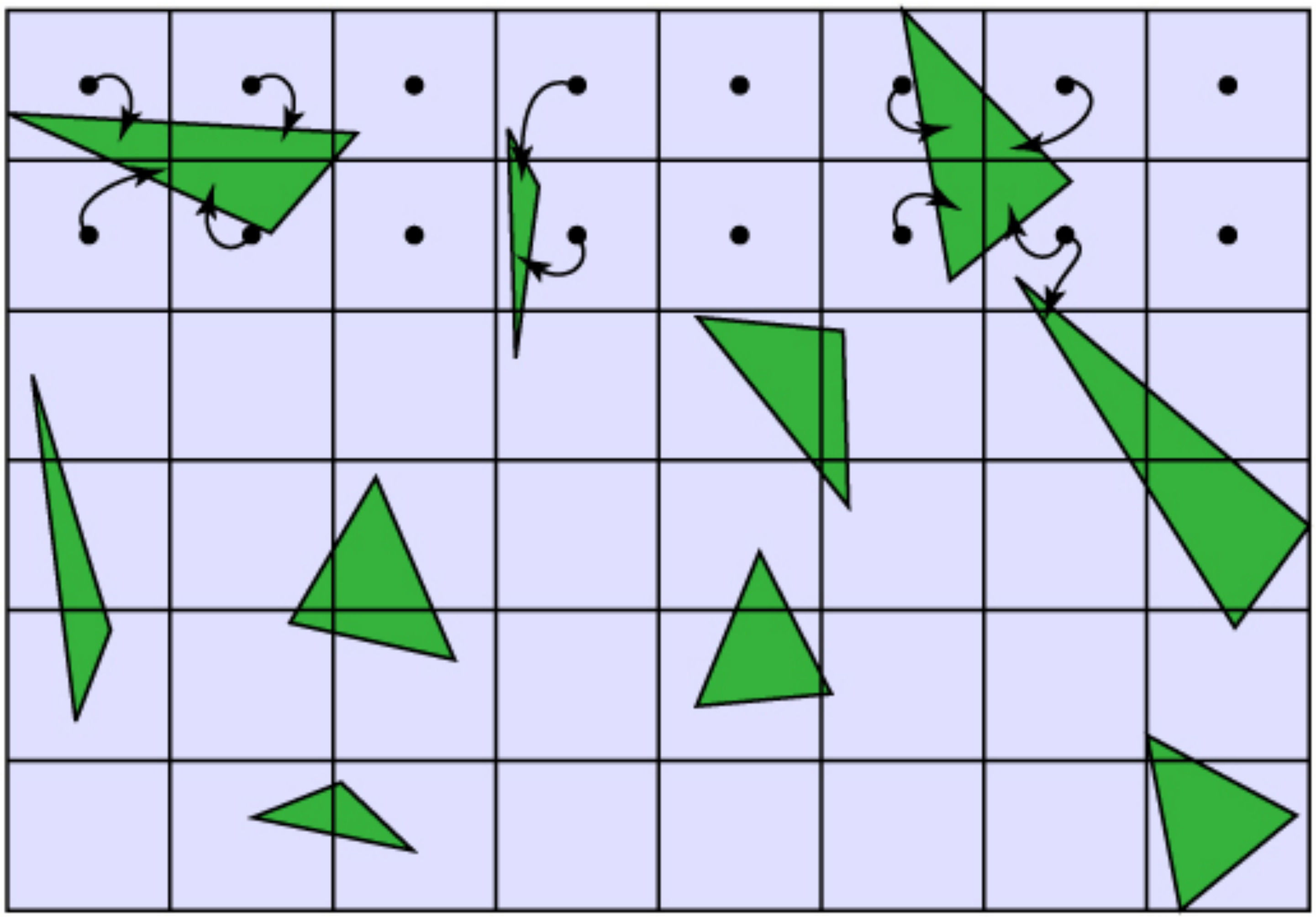
- New kind of object: BoundedObject
  - stores references to contained objects
    (may be BoundedObjects themselves!)
- New `ray_intersect` routine for BoundedObject:
  - Intersect with each child; if any, return closest.

Other Approaches:

- Uniform Space Subdivision



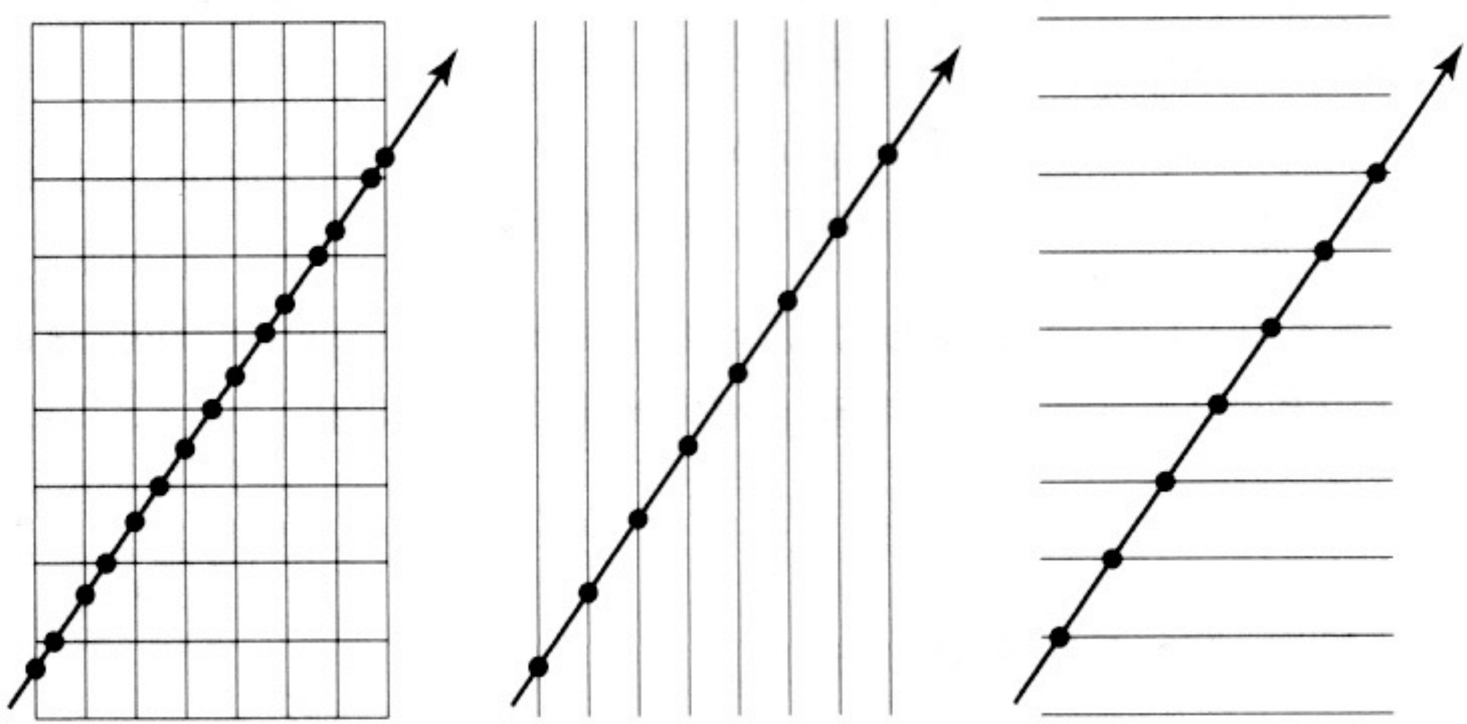
Uniform Space Subdivision



- Grid cells store references to overlapping objects

Compute the grid cells intersected by a ray

Constant offset between cell edge intersections in each dimension:



Problems: AABB Construction and Intersection

How do we intersect with an axis-aligned bounding box (AABB)?

Construction:

- AABB for a sphere
- AABB for a triangle
- AABB for a collection of AABBs

Intersection:

- 1D: intersect a slab
- 3D: intersect the intersection of 3 slabs