Ray Tracing

Ray Tracing 1

- Basic algorithm
- Ray-surface intersection (triangles, ...)

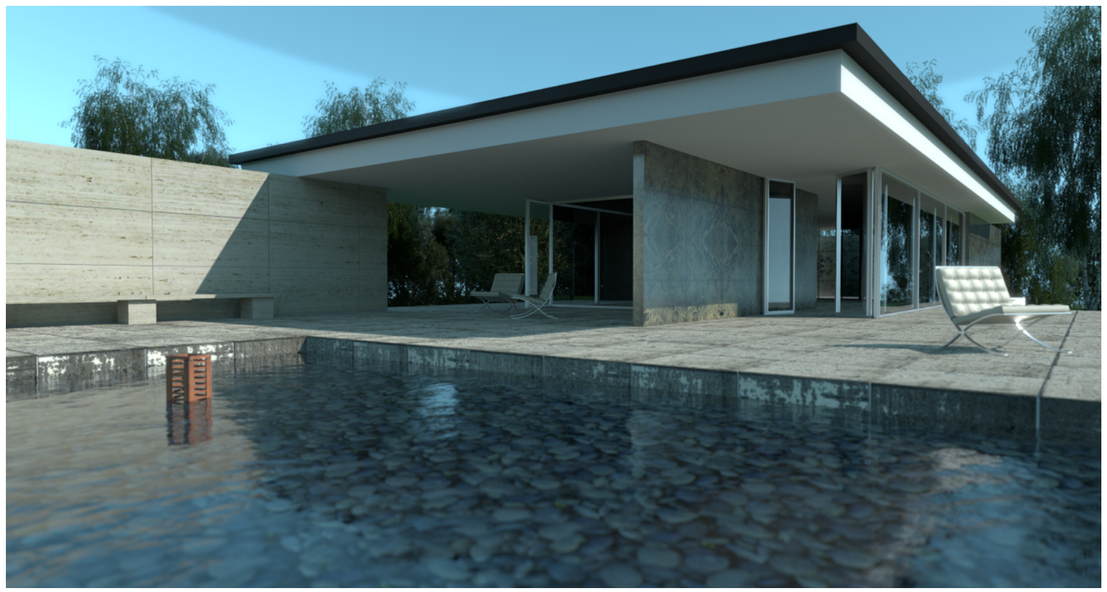
Ray Tracing 2

- Problem: Brute force = |Image|×|Objects|
- Solution: Acceleration ≈ |Image|×log|Objects|

Later

- How to choose the best rays to trace?
- How to minimize the number of rays traced?

300k tris
Time: Accel 27.4%, Tri 10.2%



5.3M tris
Time: Accel 42.7%, Tri 13.3%

3.1B tris
Time: Accel 74.7%, Tri 13.3%

Primitives

pbrt Primitive base class

- Shape
- Material (emission, reflection and transmission)

```
class Primitive {
  public:
    virtual Bounds3f WorldBound() const = 0;
    virtual bool Intersect(const Ray &r,
      SurfaceInteraction *) const = 0;
    virtual bool IntersectP(const Ray &r) const = 0;
    virtual const AreaLight *GetAreaLight() const = 0;
    virtual const Material *GetMaterial() const = 0;
    virtual void ComputeScatteringFunctions(…) const = 0;
};
```

Primitives

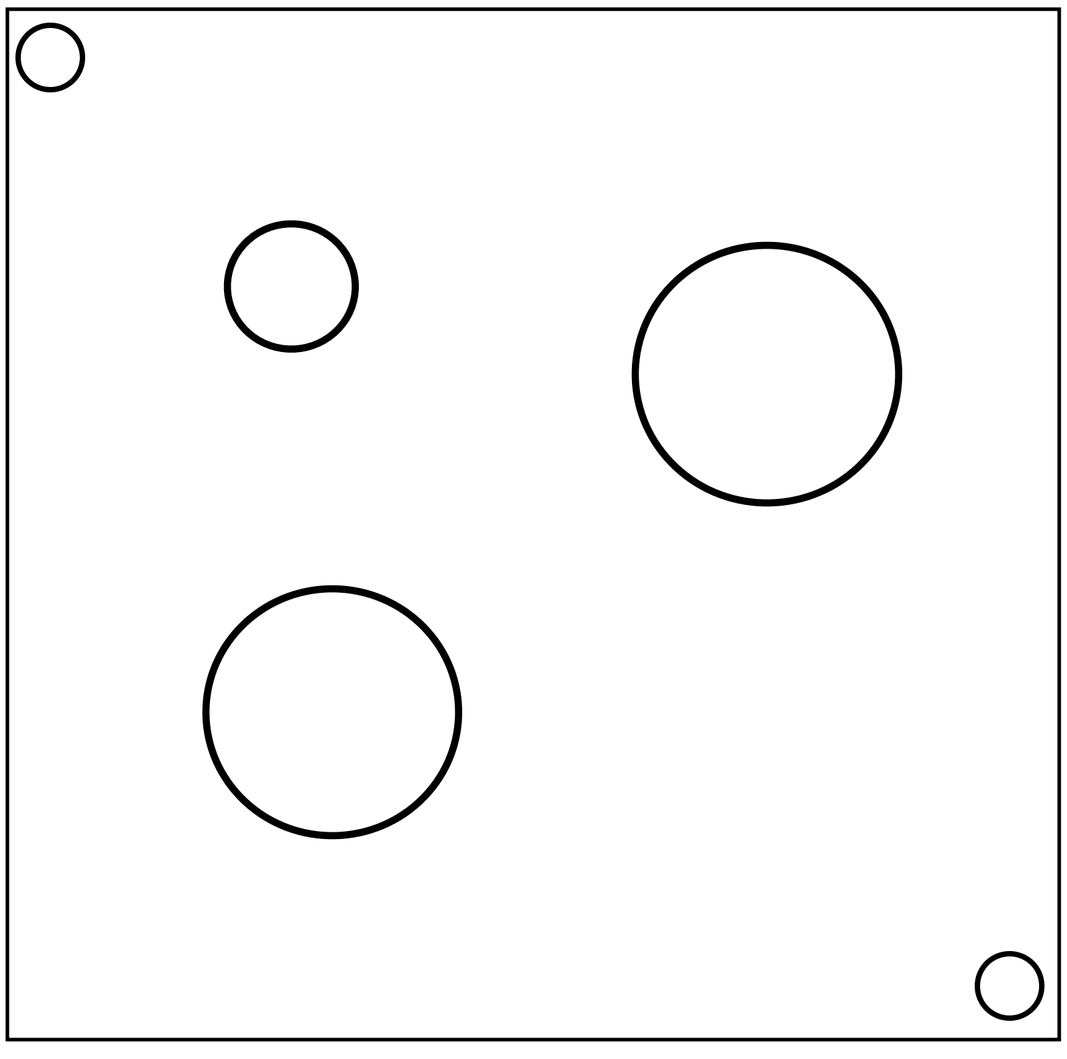
Collections

- TransfomedPrimitive: Transformation + primitive
- Aggregate
  - Treat acceleration data structures as primitives
  - Two types of accelerators: kdtree.cpp, and bvh.cpp
- May nest accelerators of different types

```
class Scene {
   // …
   bool Intersect(const Ray &ray,
                  SurfaceInteraction *isect) const {
       return aggregate->Intersect(ray, isect);
   }
   std::shared_ptr<Primitive> aggregate;
};
```

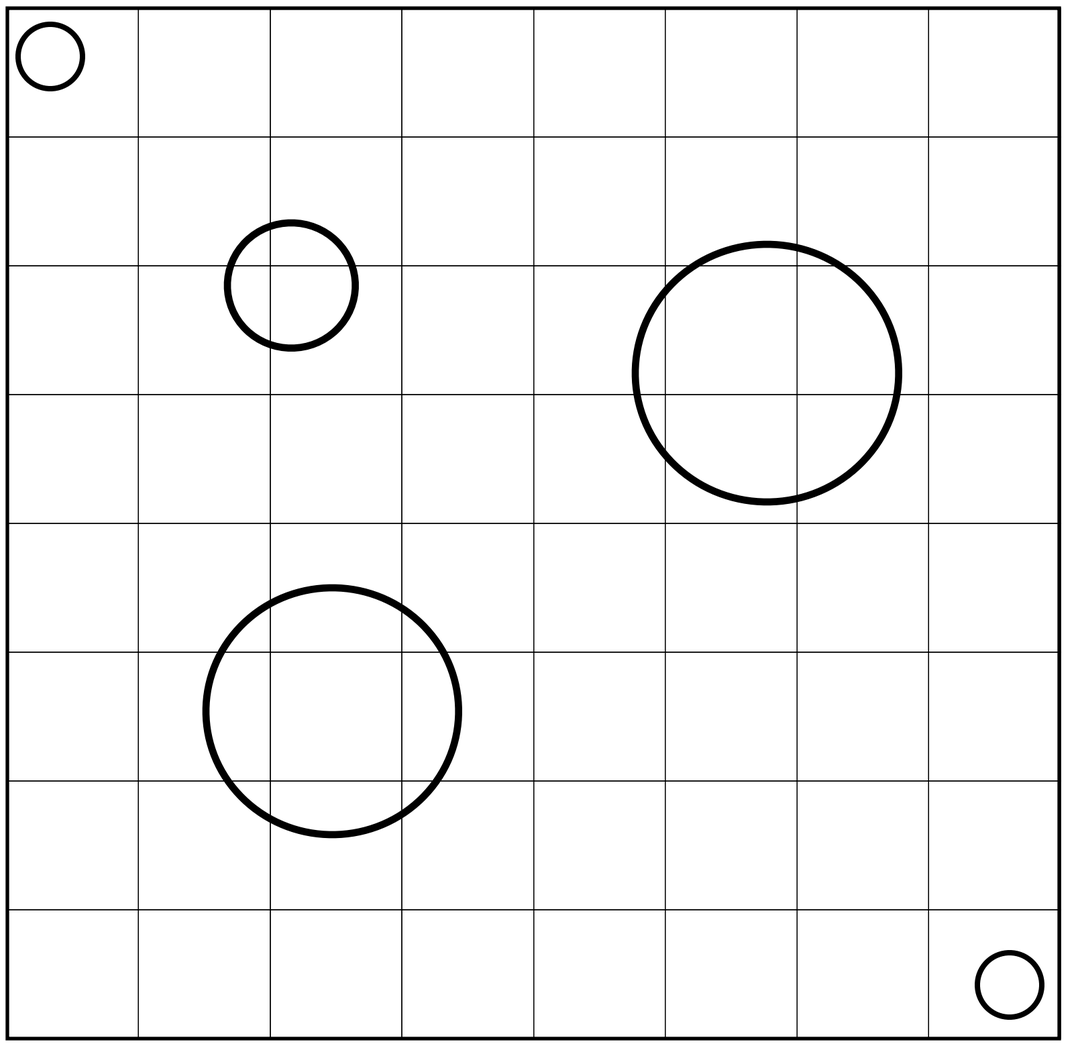
Uniform Grids

Preprocess Scene



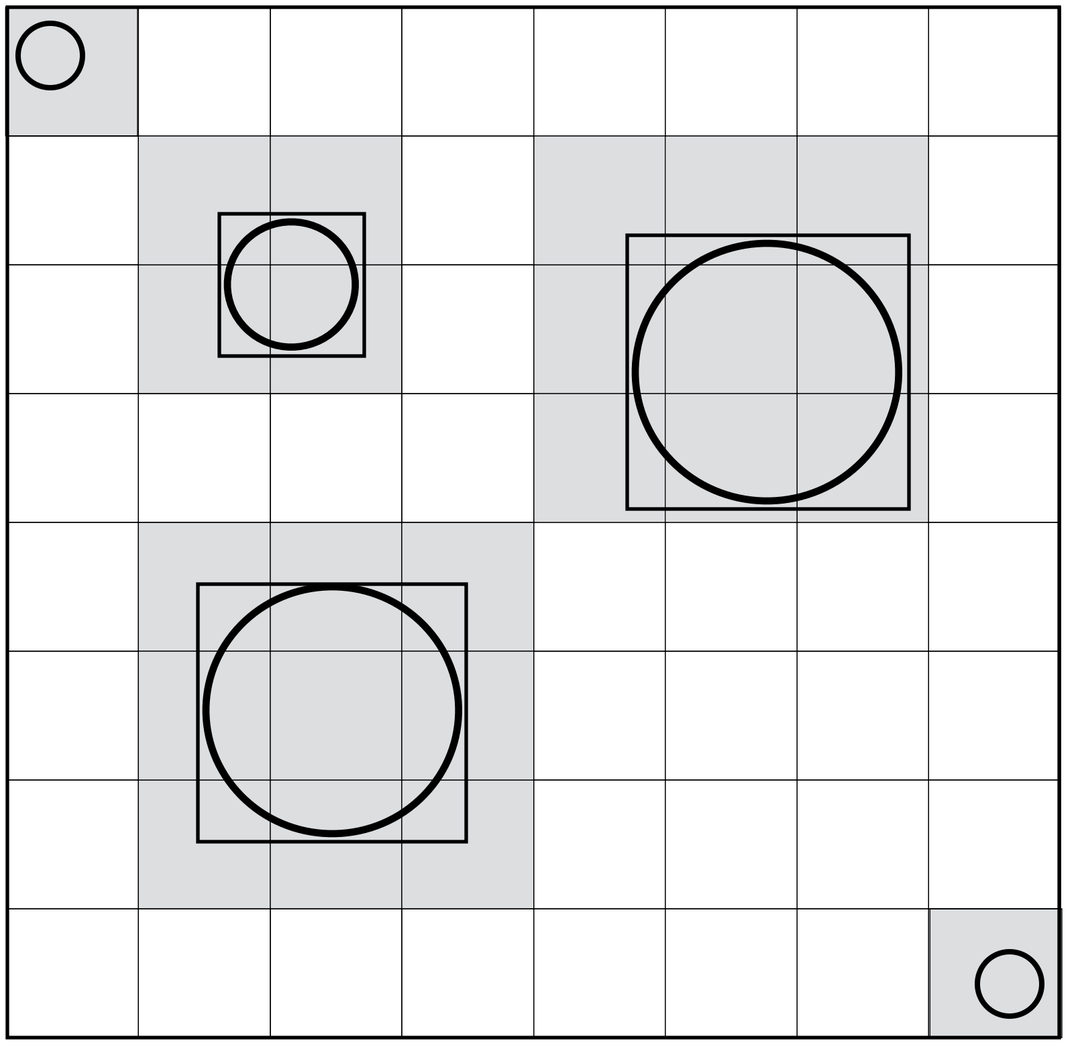
1. Find bounding box

Preprocess Scene



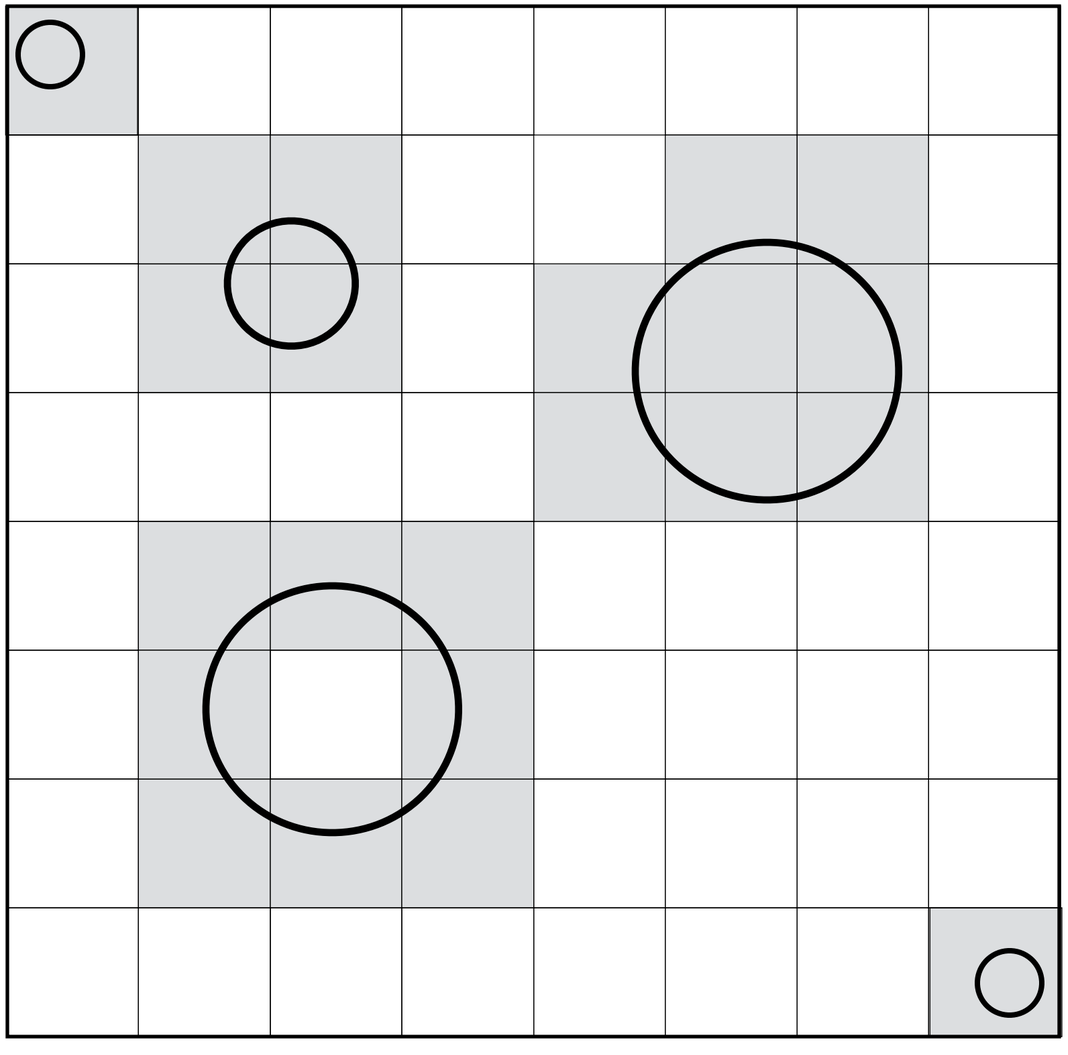
1. Find bounding box
2. Determine resolution

Preprocess Scene



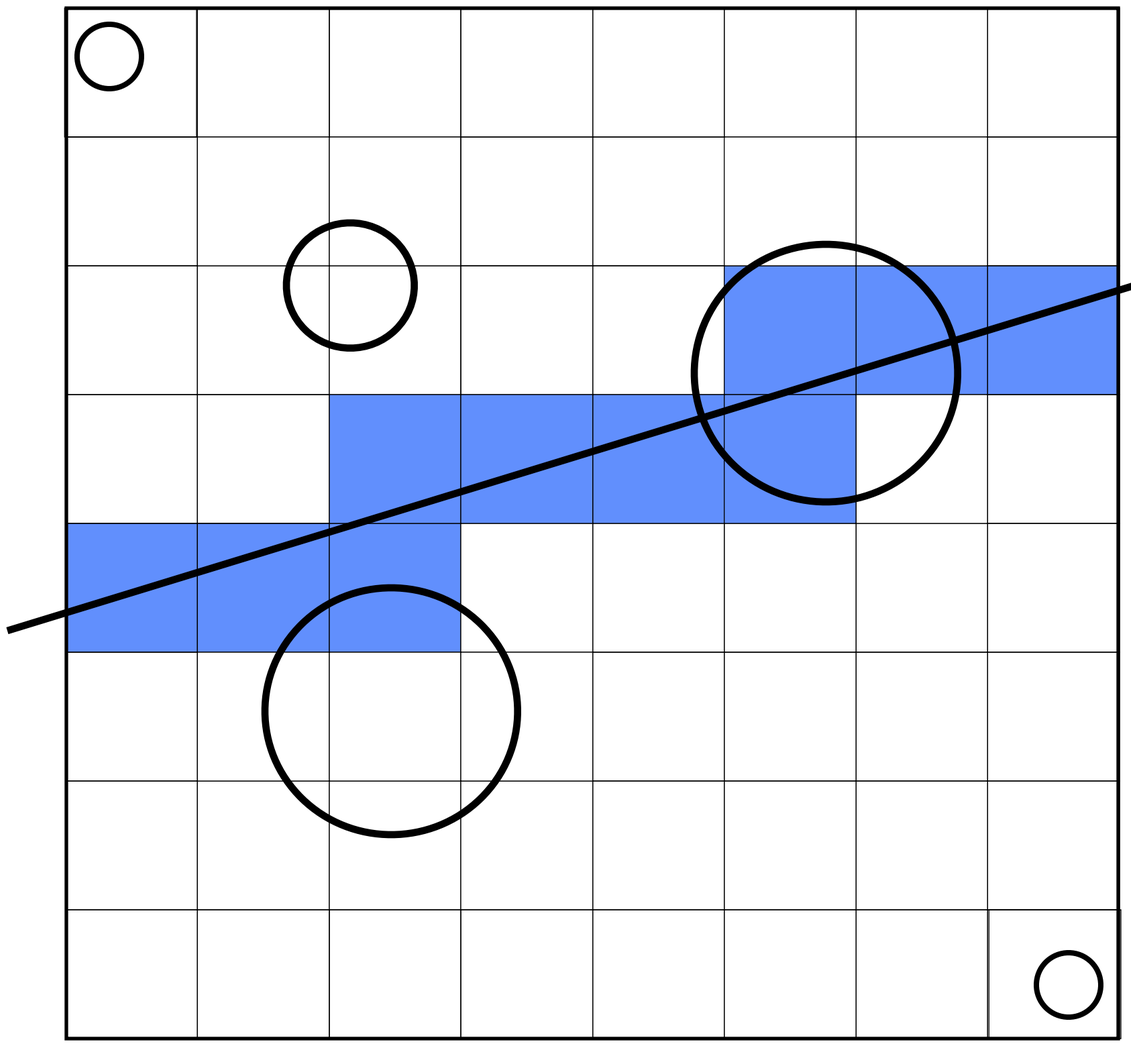
1. **Find bounding box**
2. **Determine resolution**
3. **Place objects in cells if bbox overlaps**

Preprocess Scene



1. Find bounding box
2. Determine resolution
3. Place objects in cells
   - if bbox overlaps
   - if surface intersects

Ray-Intersection



For each ray,

step through grid

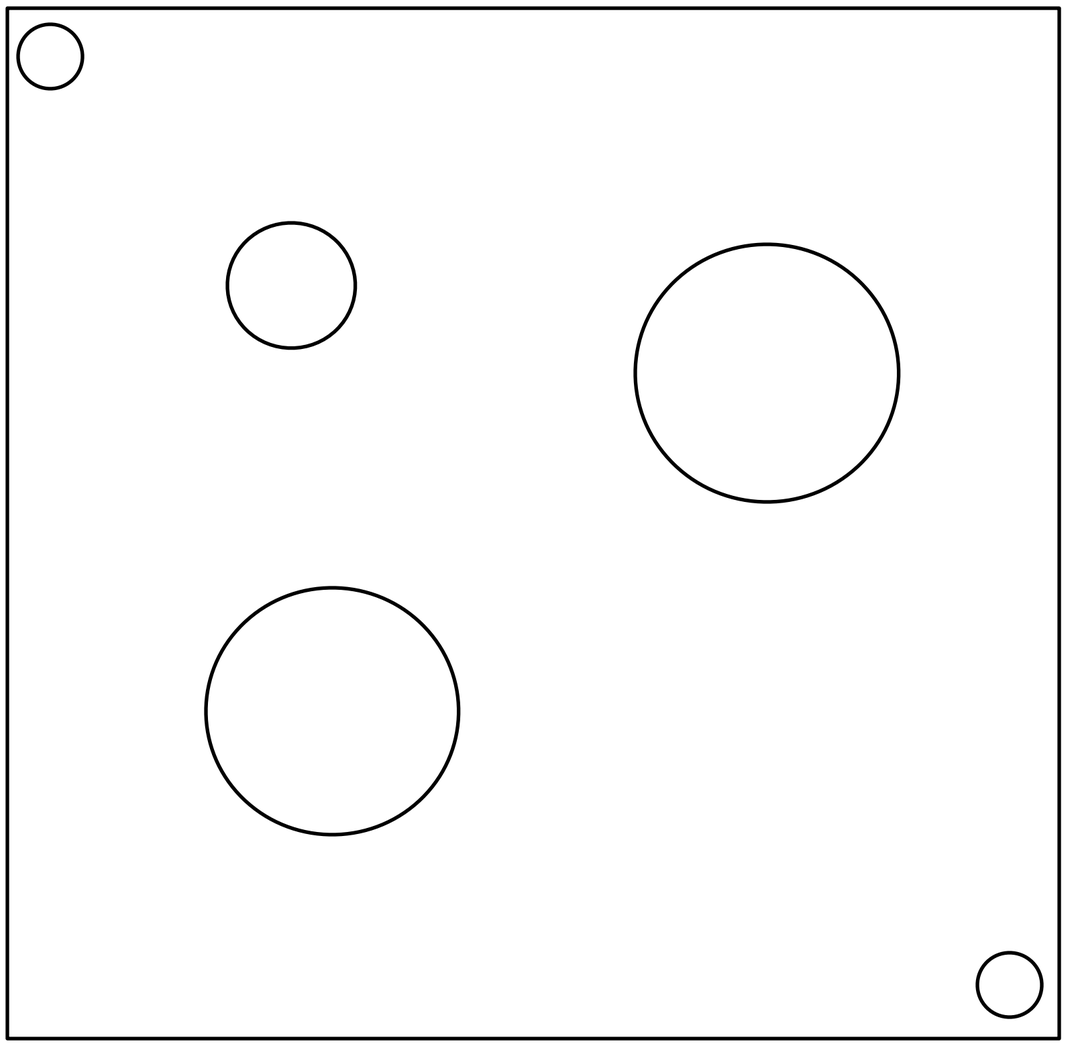
3D line – 3D-DDA

Output all grid cells

the ray pierces

In 3D, 6-connected line

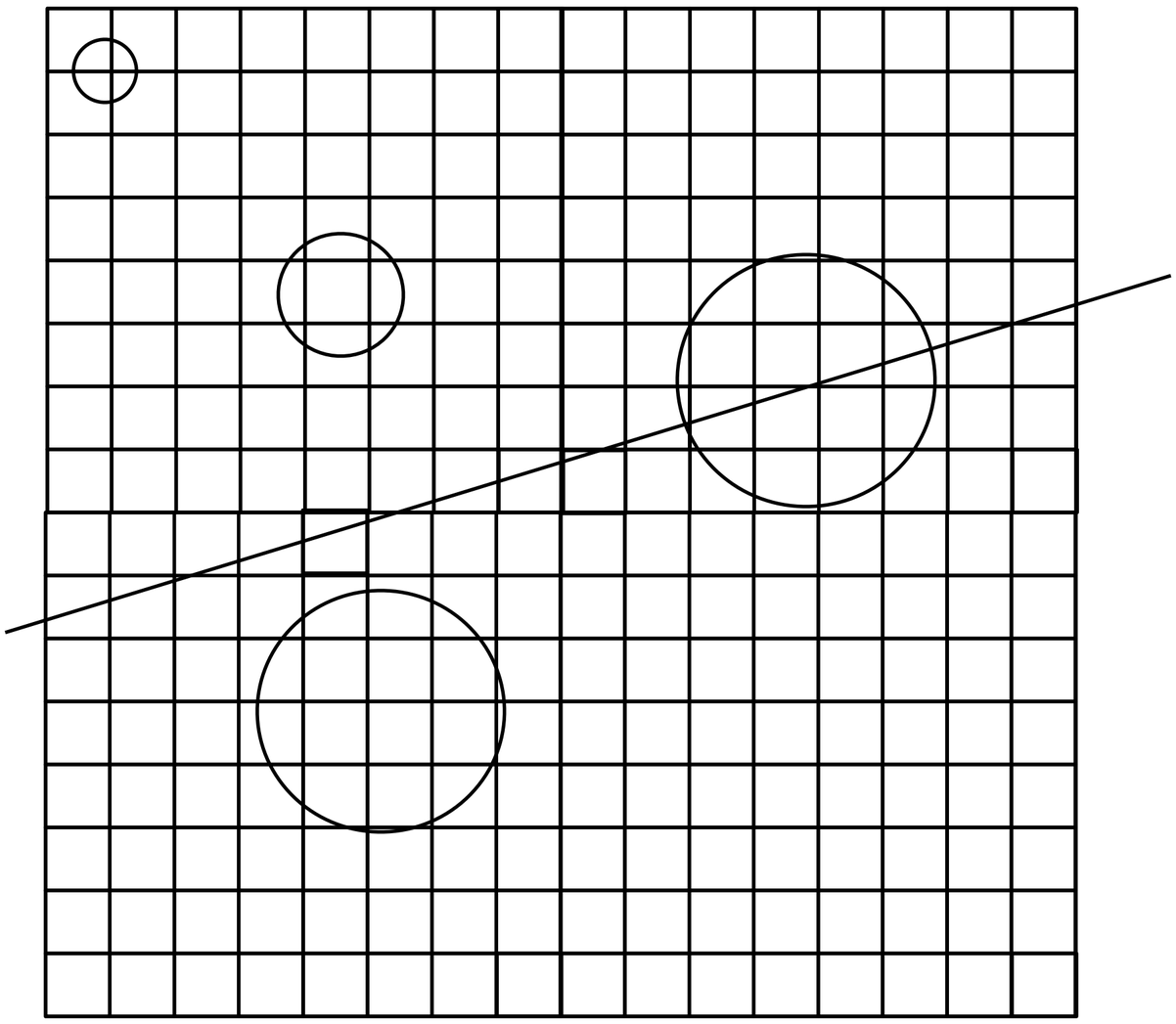
Resolution?



1 cell

No speedup

Resolution?

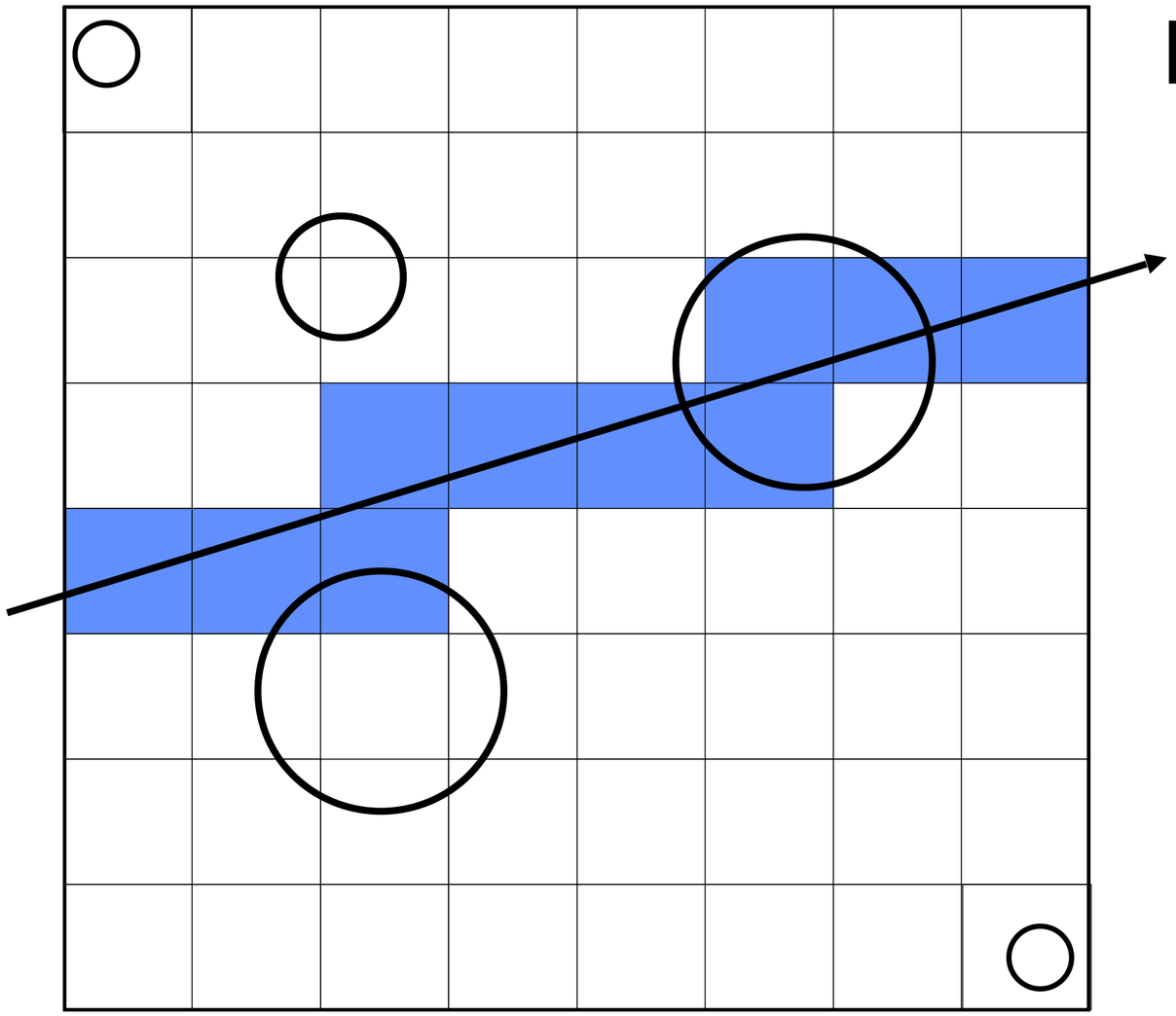


Large number of cells

Too many empty cells

Extra cost!

Resolution?



Heuristic

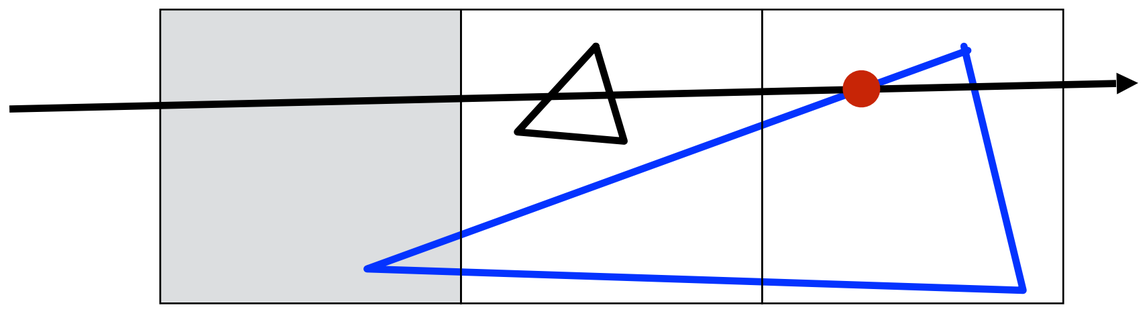
$$n_v = n_x n_y n_z \propto n_o$$

$$\max(n_x, n_y, n_z) = d\sqrt[3]{n_o}$$

$$d \approx 3$$

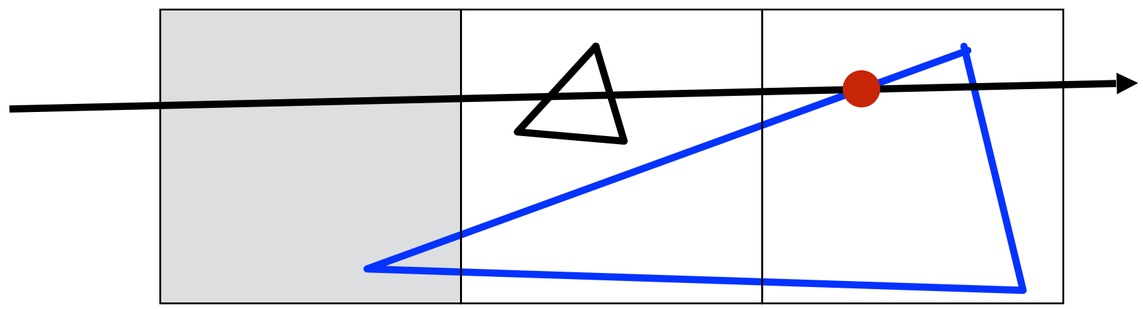
Objects Overlapping Multiple Cells

Mistake: Output first intersection found



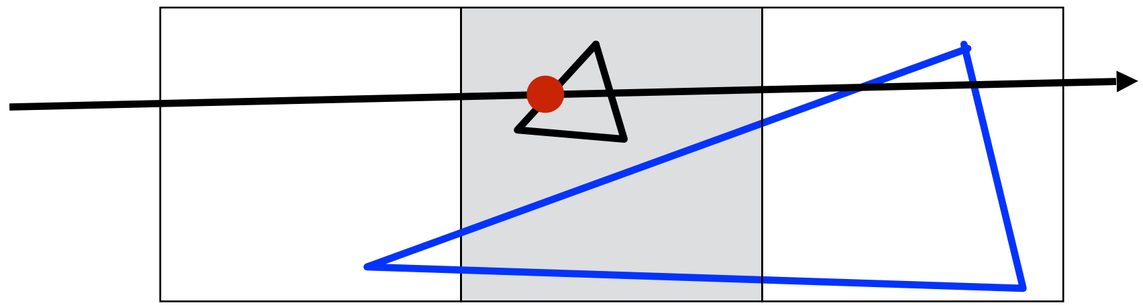
Objects Overlapping Multiple Cells

Solution: Check whether intersection is inside cell



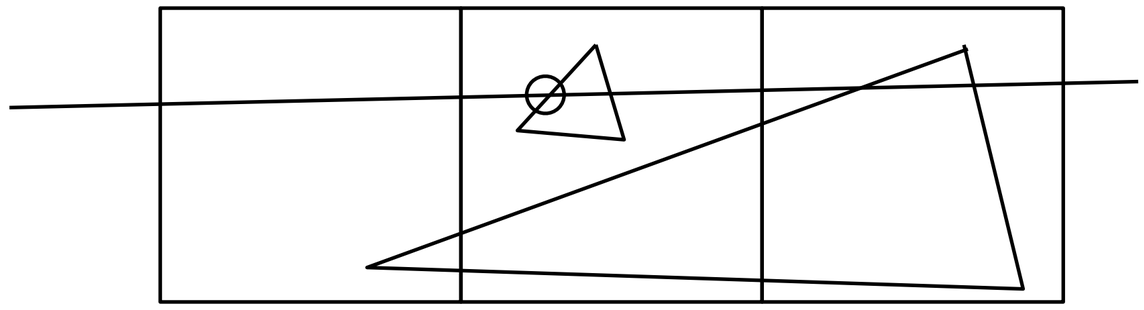
Objects Overlapping Multiple Cells

Solution: Check whether intersection is inside cell



Mailboxes

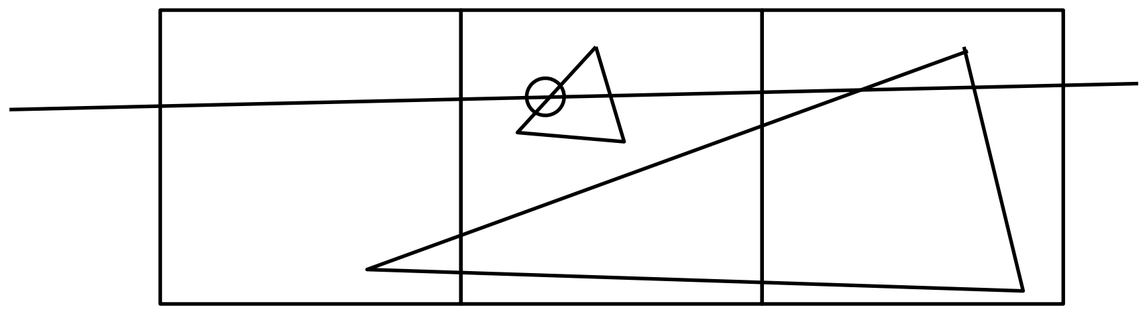
Solution: Check whether intersection is inside cell



Problem: Objects tested for intersection multiple times

Mailboxes

Solution: Check whether intersection is inside cell



Problem: Objects tested for intersection multiple times

Solution: Mailboxes

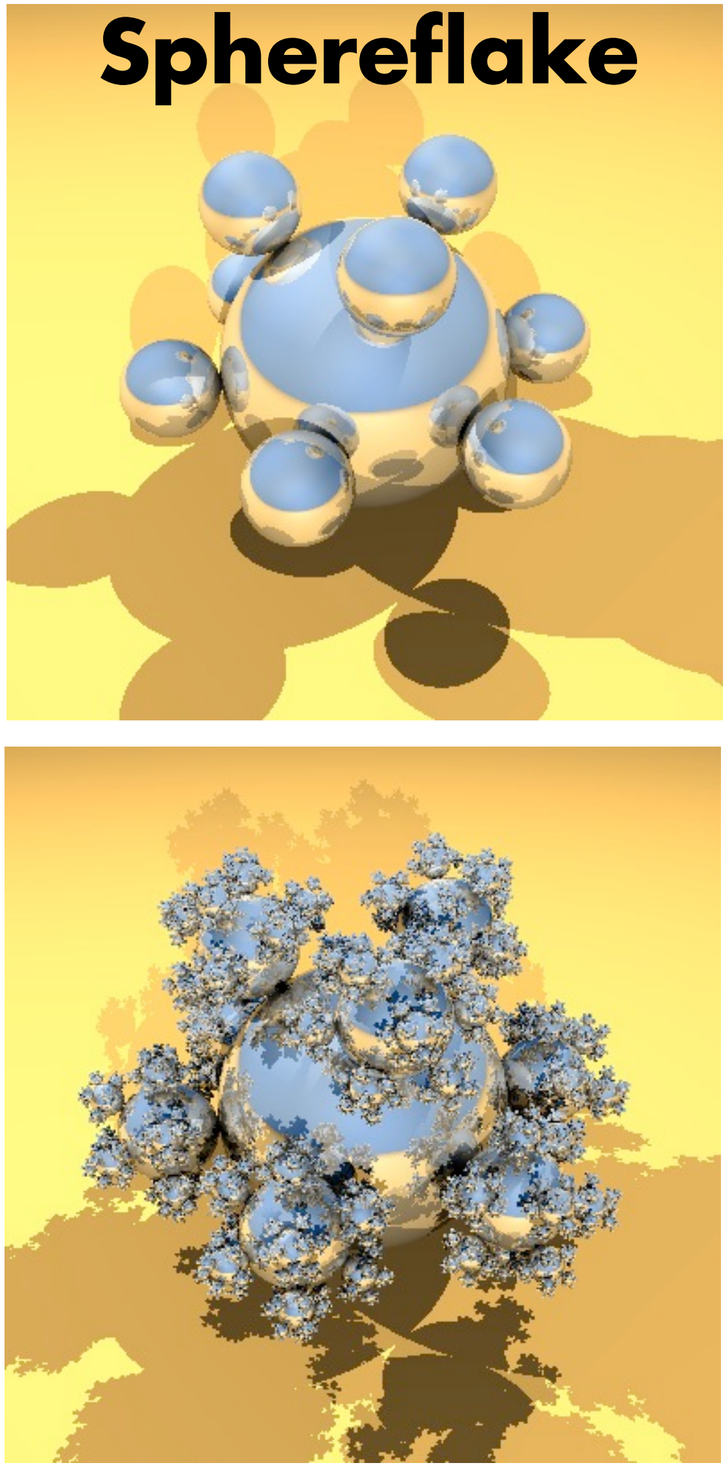
- Assign each ray an increasing number
- Primitive intersection cache (mailbox)
  - Give each ray a number n
  - Store intersection point and ray n w/ each primitive
  - Only re-intersect if ray n is greater than last ray n

Uniform Grids: When They Work Well

Uniform grids work well for large collections of objects that are uniform in size and distribution

http://www.kevinboulanger.net/grass.html

Uniform Grids: When They Work Poorly



Teapot in the Stadium Problem

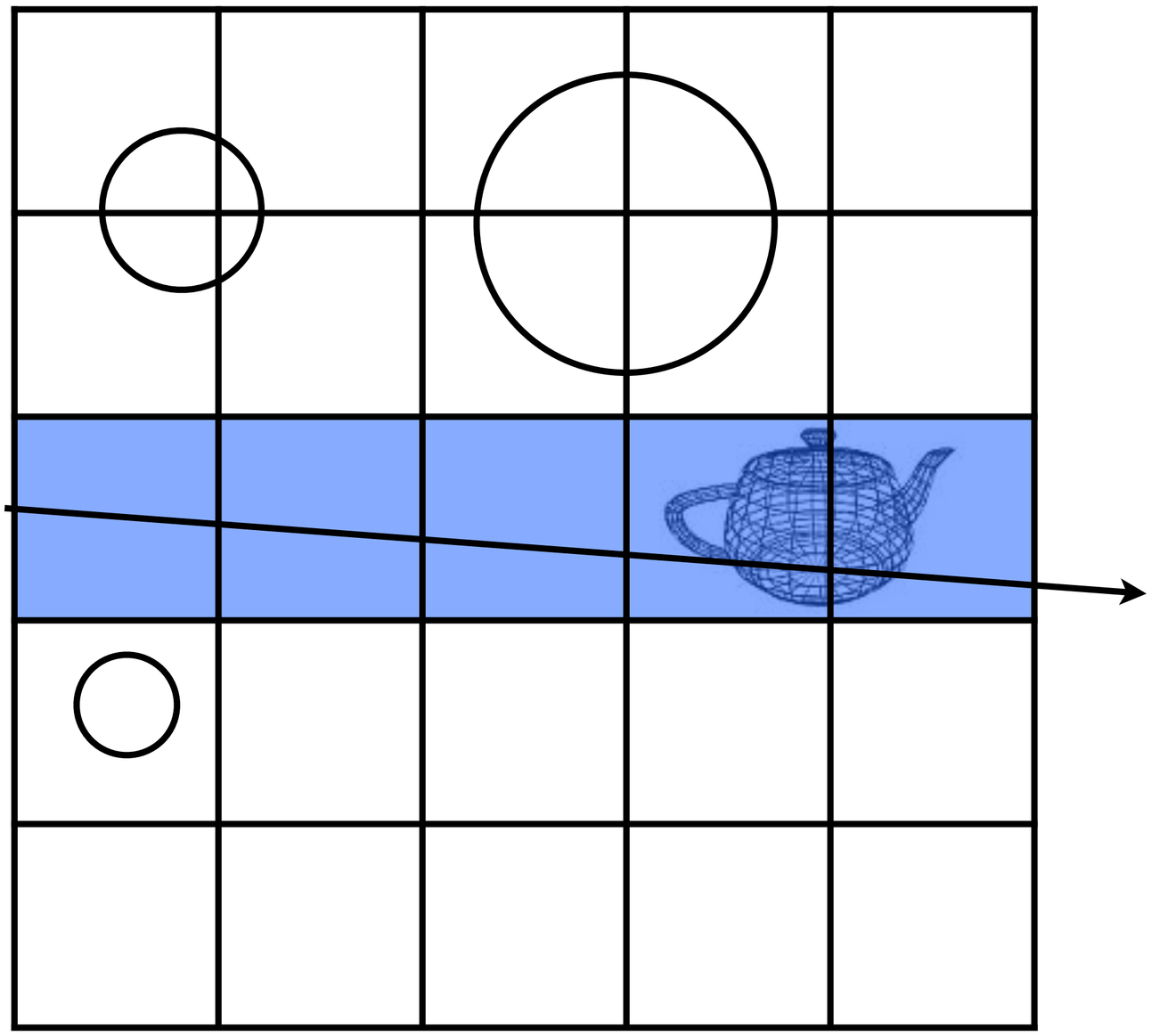
"For example, imagine you have a football stadium made of, say, 5K primitives. Sitting on a goal line is a shiny polygonalized teapot of 5K quadrilaterals (note that the teapot is teapot sized compared to the stadium)."

Eric Haines

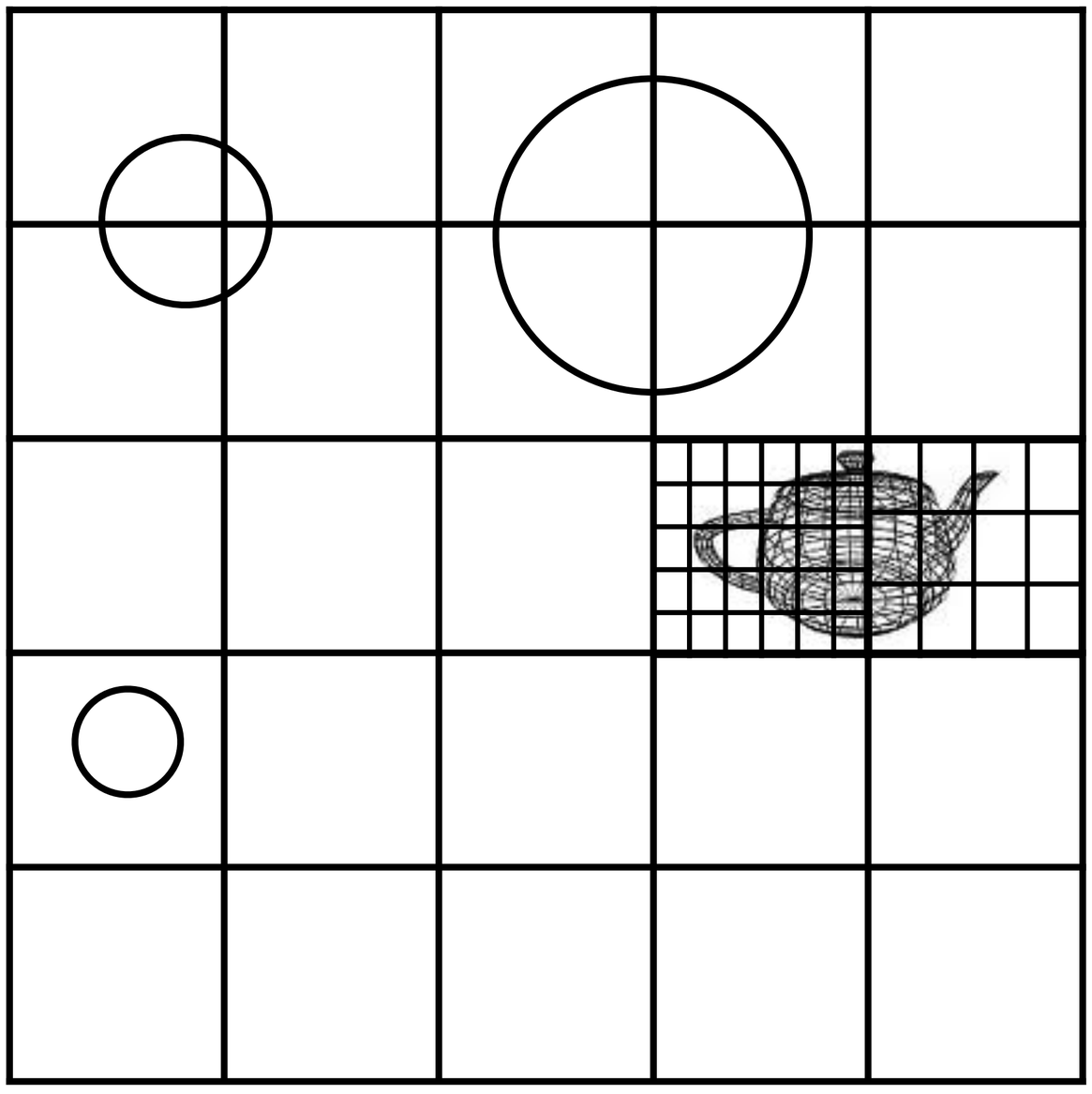
http://tog.acm.org/resources/RTNews/html/rtnews1b.html#art4

Problem: non-uniform size distribution

Nested Grids

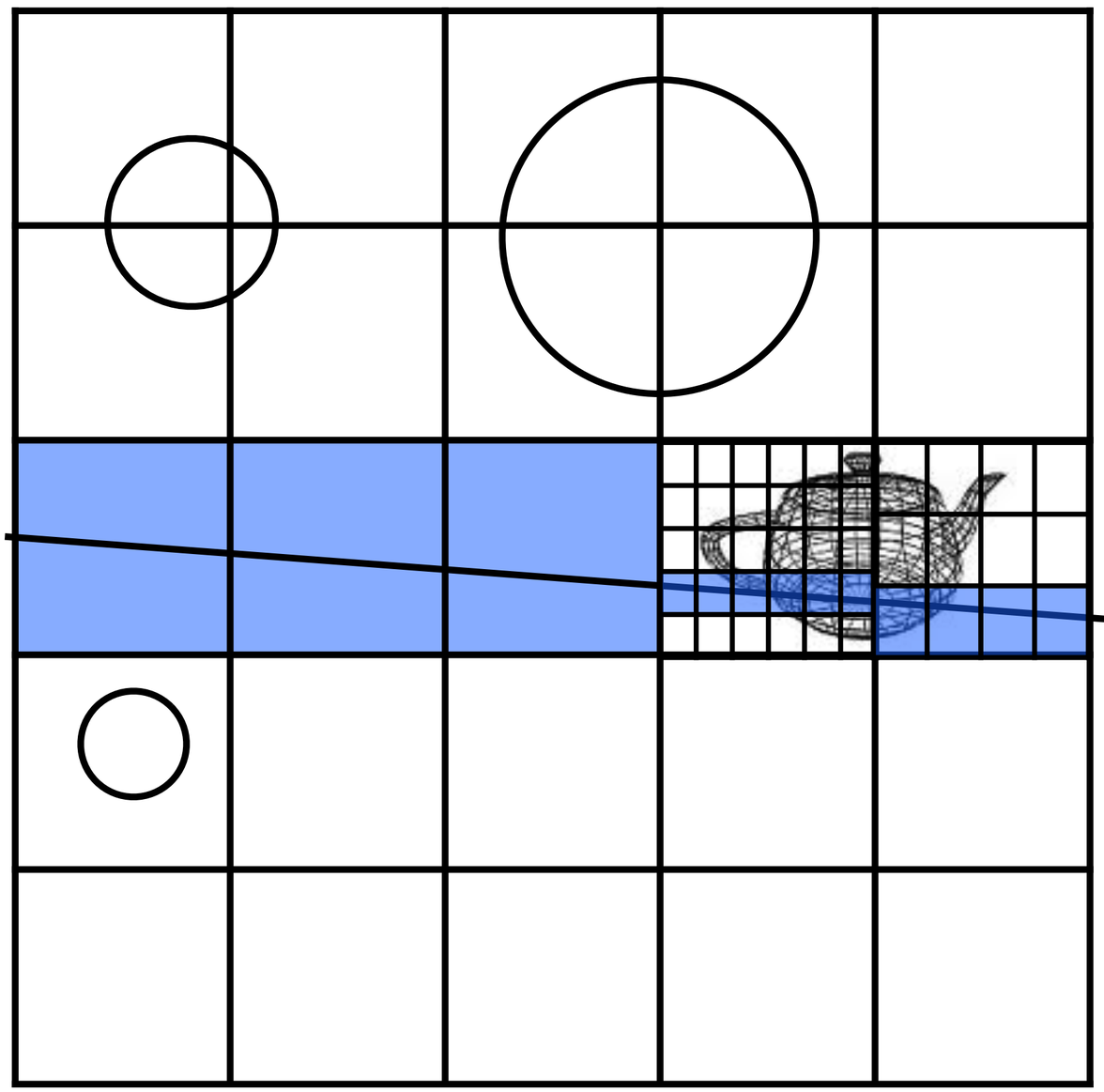


Nested Grids



Refine grid cells with many primitives to have second-level grids in those cells

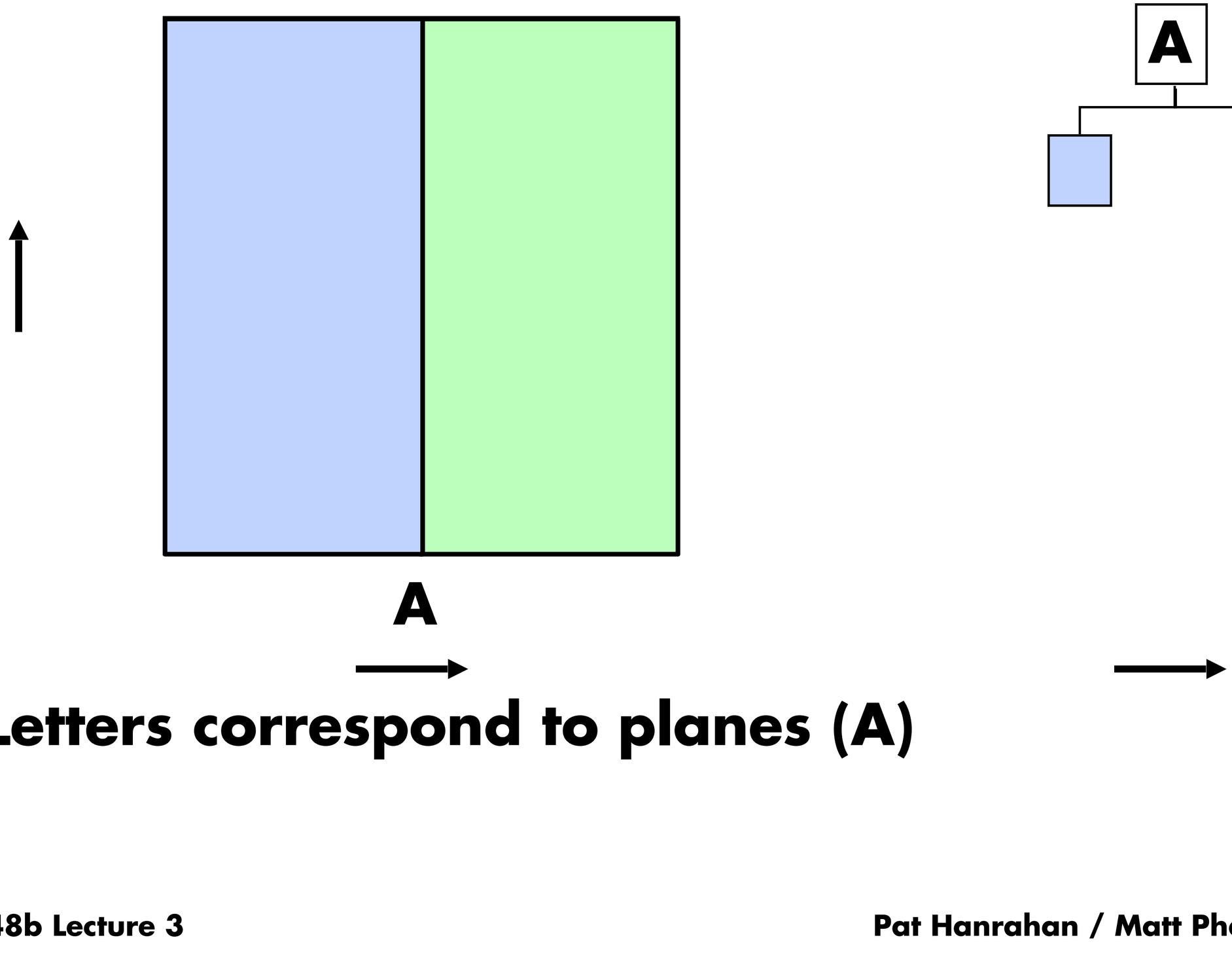
Nested Grids



Refine grid cells with many primitives to have second-level grids in those cells

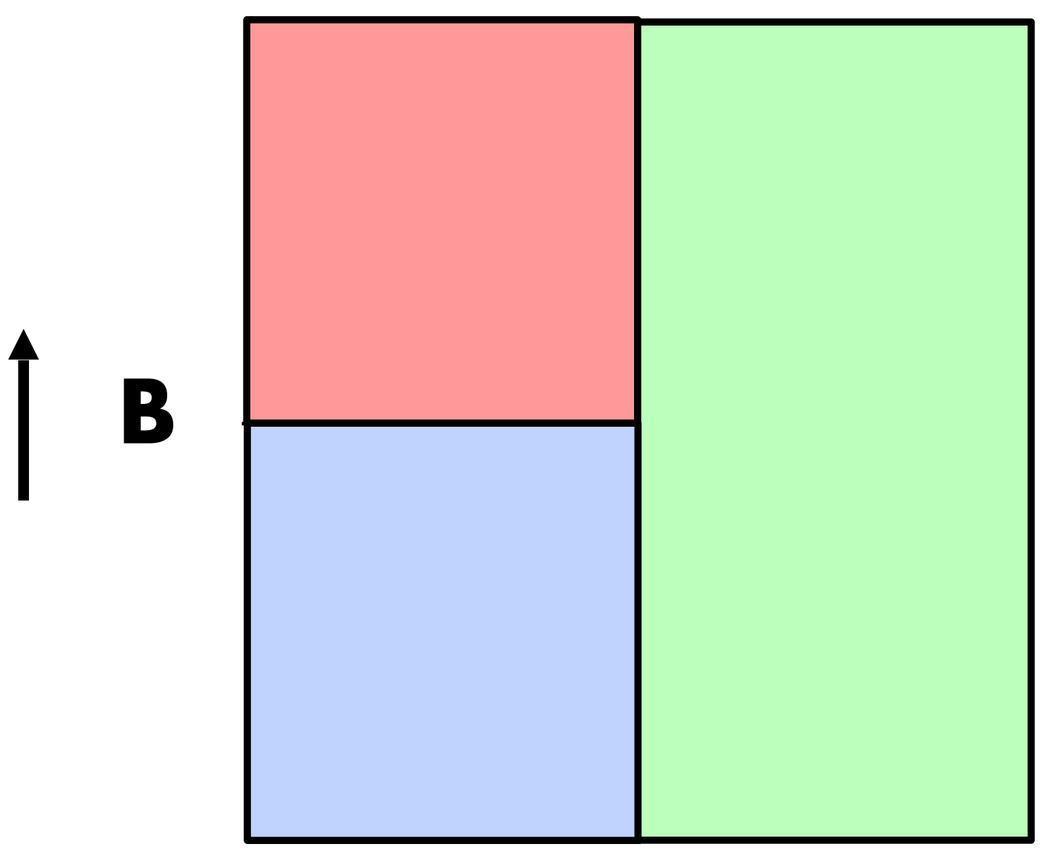
Spatial Hierarchies

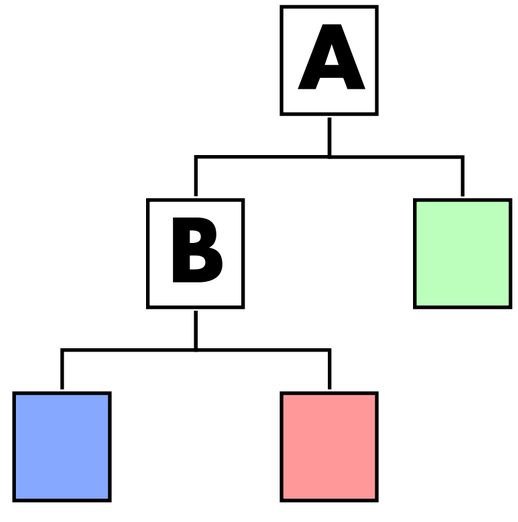
Spatial Hierarchies



Letters correspond to planes (A)

Spatial Hierarchies

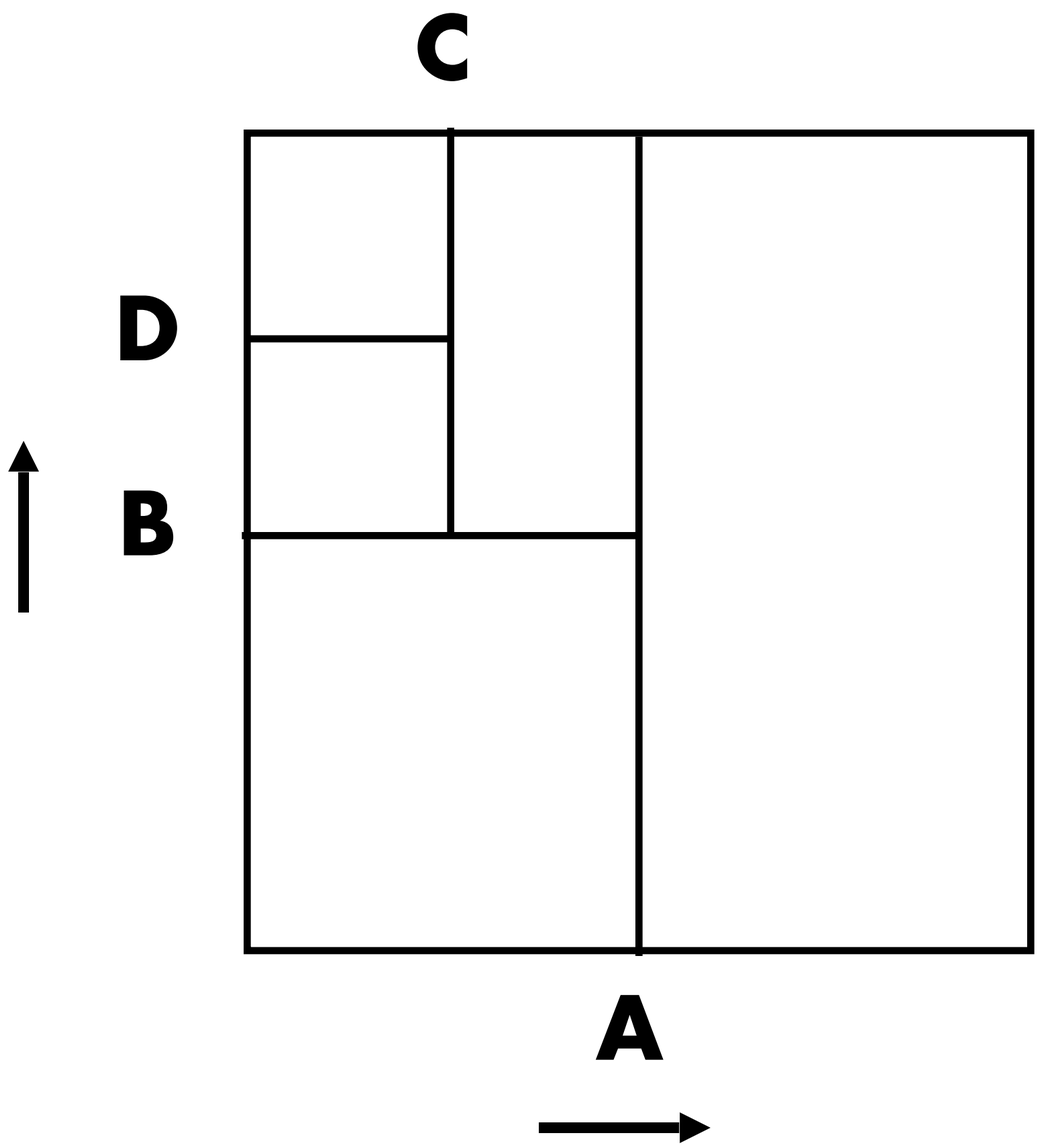


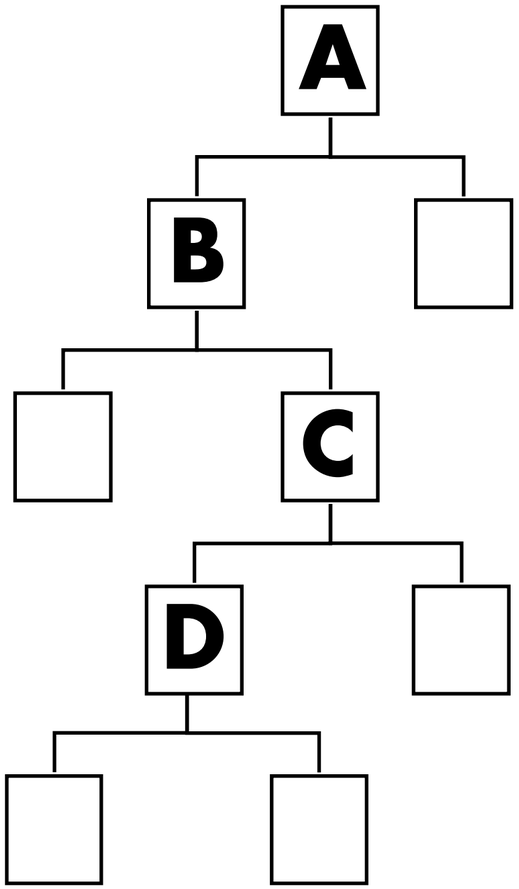


Letters correspond to planes (A, B)

Point Location by recursive search

Spatial Hierarchies

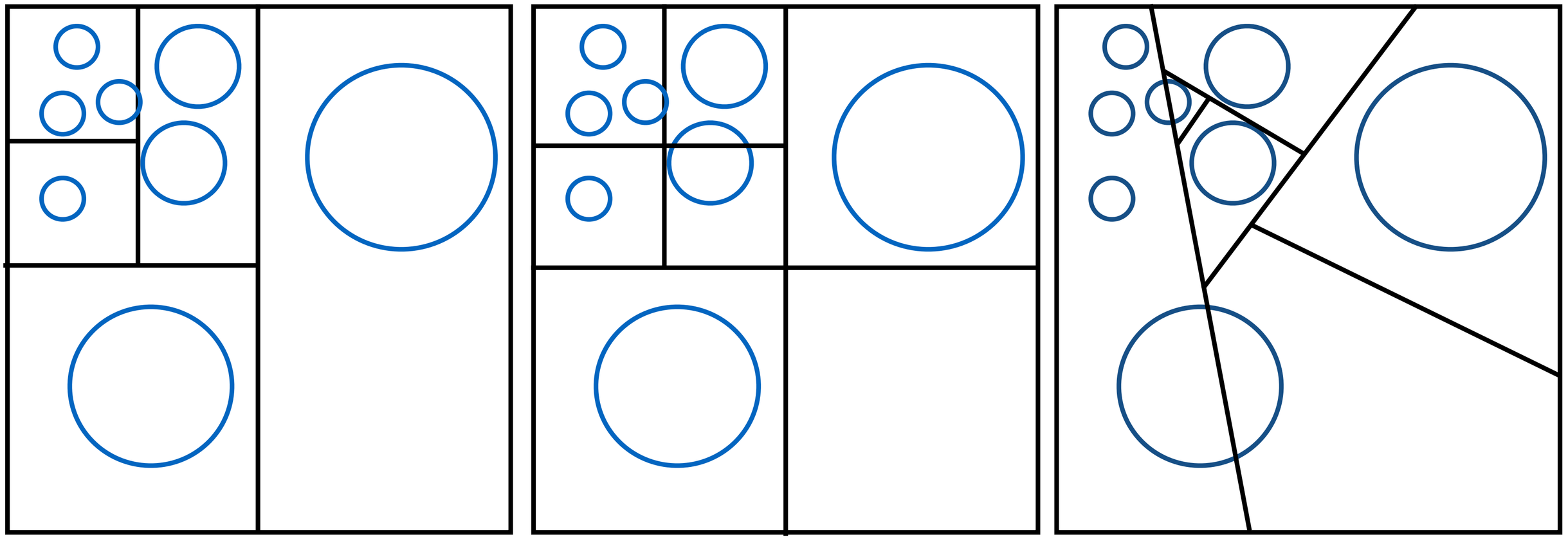




Letters correspond to planes (A, B, C, D)

Boxes at leaves correspond to regions

Variations



kd-tree

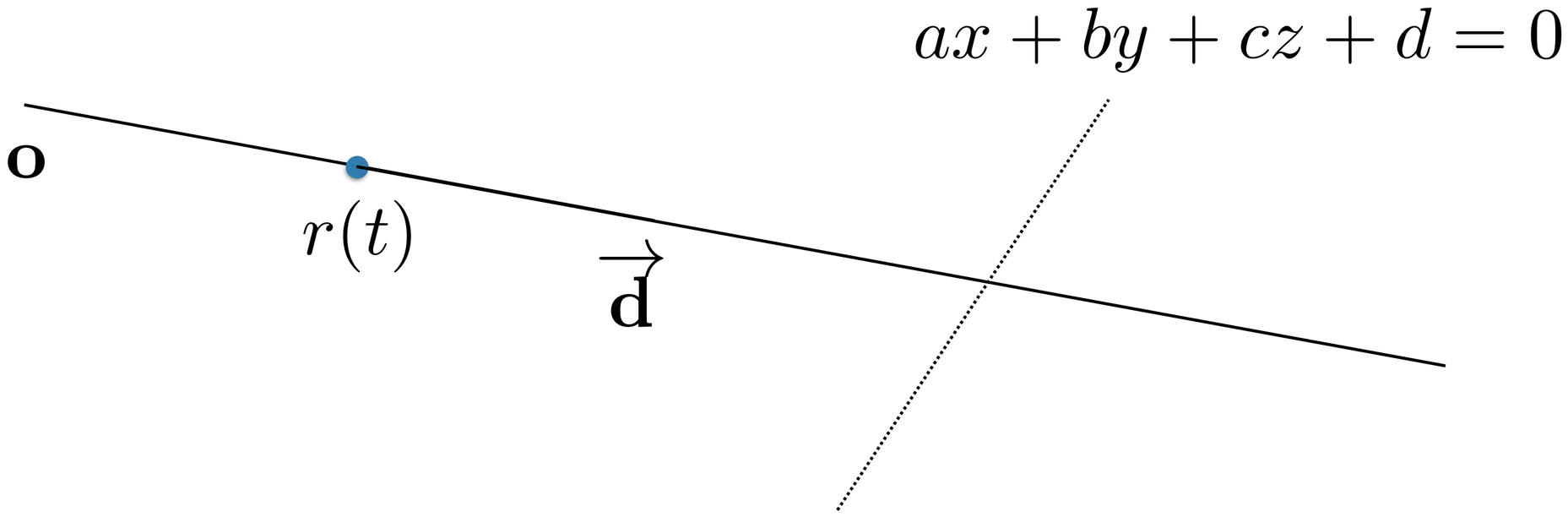
oct-tree

bsp-tree

Review: Ray-Plane Intersection

Ray: $r(t) = \mathbf{o} + t\,\overrightarrow{\mathbf{d}}$

Plane: $(\mathbf{p} - \mathbf{p}') \cdot \overrightarrow{\mathbf{n}} = 0$



$$t = -\frac{(\mathbf{o} - \mathbf{p}) \cdot \overrightarrow{\mathbf{n}}}{\overrightarrow{\mathbf{d}} \cdot \overrightarrow{\mathbf{n}}}$$

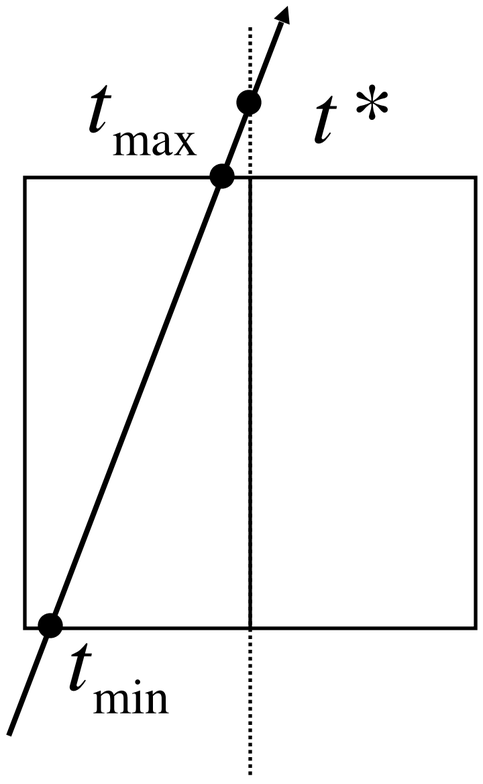
General case

$$t = -\frac{\mathbf{o_x} - \mathbf{p_x}}{\overrightarrow{\mathbf{d_x}}}$$

Axis-aligned

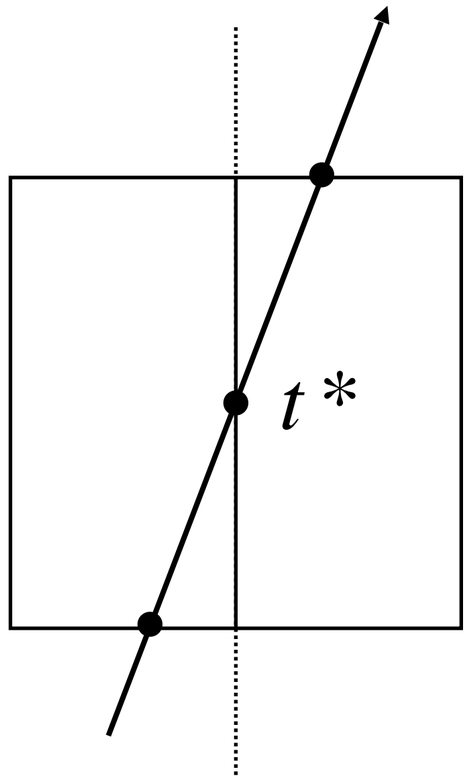
Recursive Inorder Traversal

[Kaplan, Arvo, Jansen]



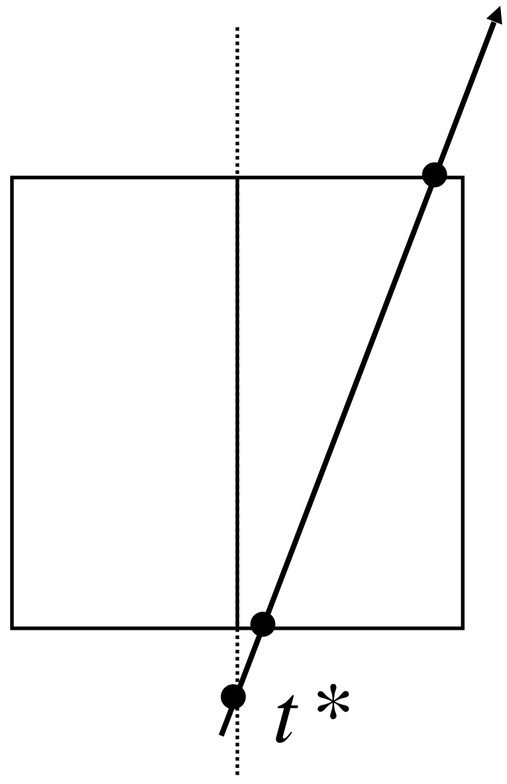
$t_{max} < t*$

Intersect(L,tmin,tmax)



$t_{min} < t* < t_{max}$

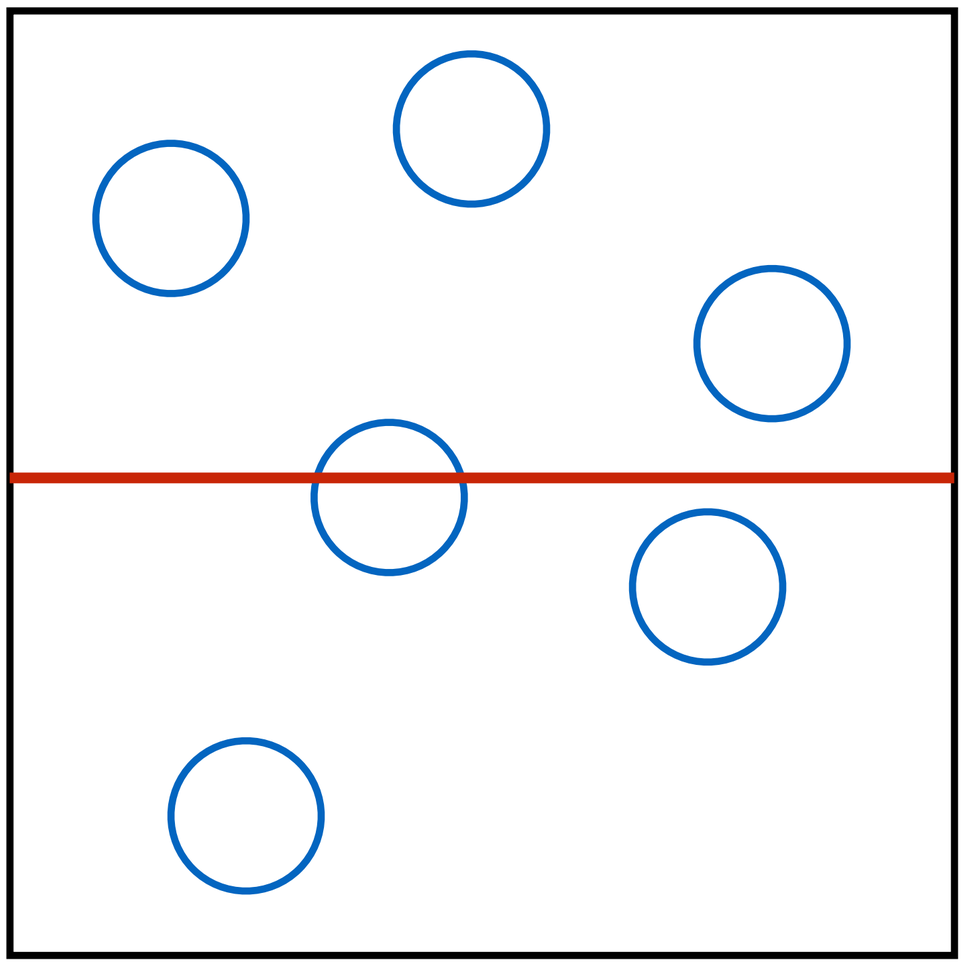
Intersect(L,tmin,t*)
Intersect(R,t*,tmax)



$t* < t_{min}$

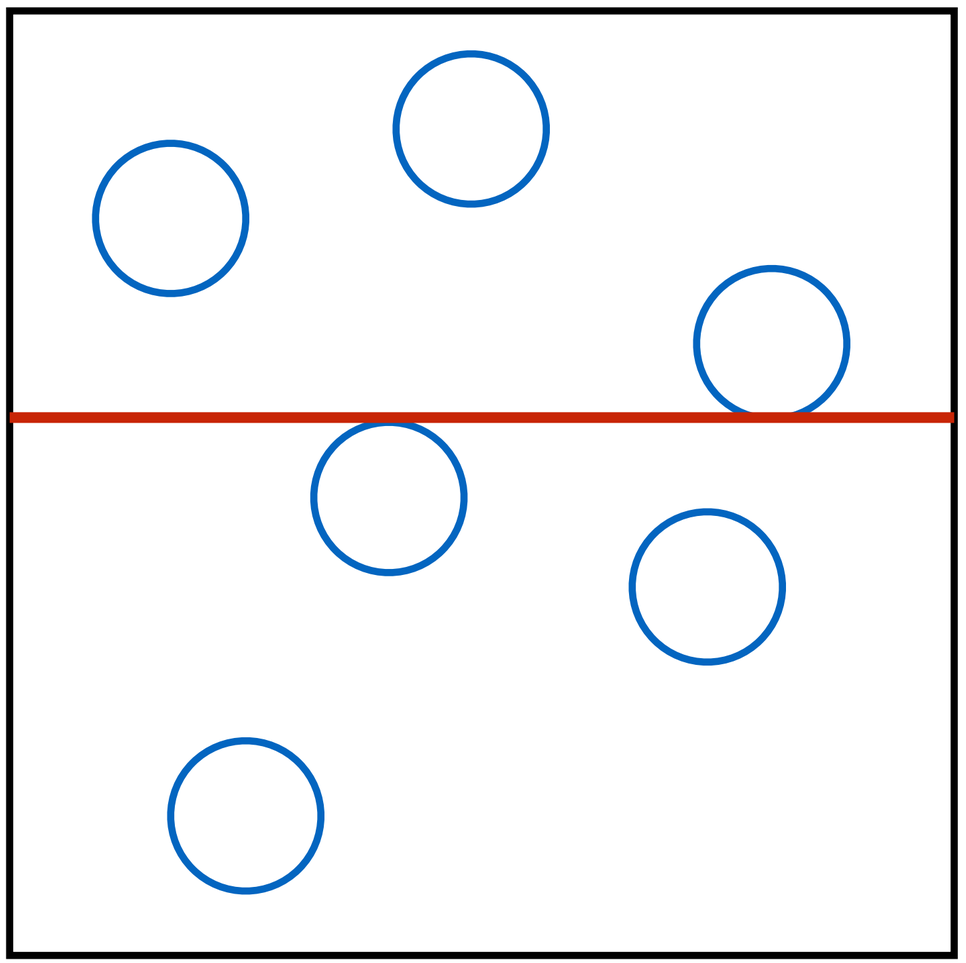
Intersect(R,tmin,tmax)

Building the Hierarchy



Midpoint?

?



Median?

Which Hierarchy is Fastest?

What is the cost of tracing a ray?

Cost(node)= C_trav

+ Prob(hit L)*Cost(L)

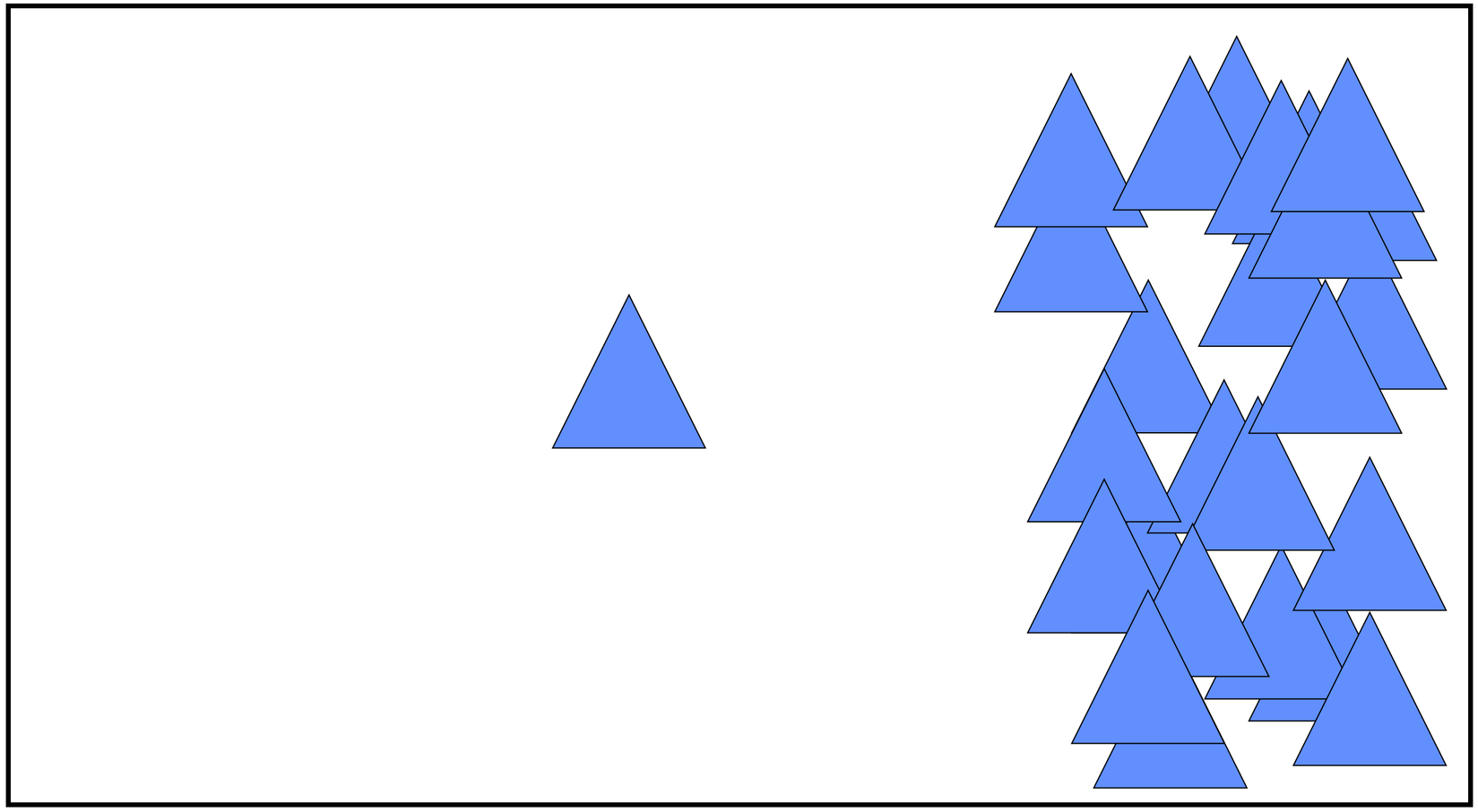
+ Prob(hit R)*Cost(R)

C_trav = cost of traversing a cell

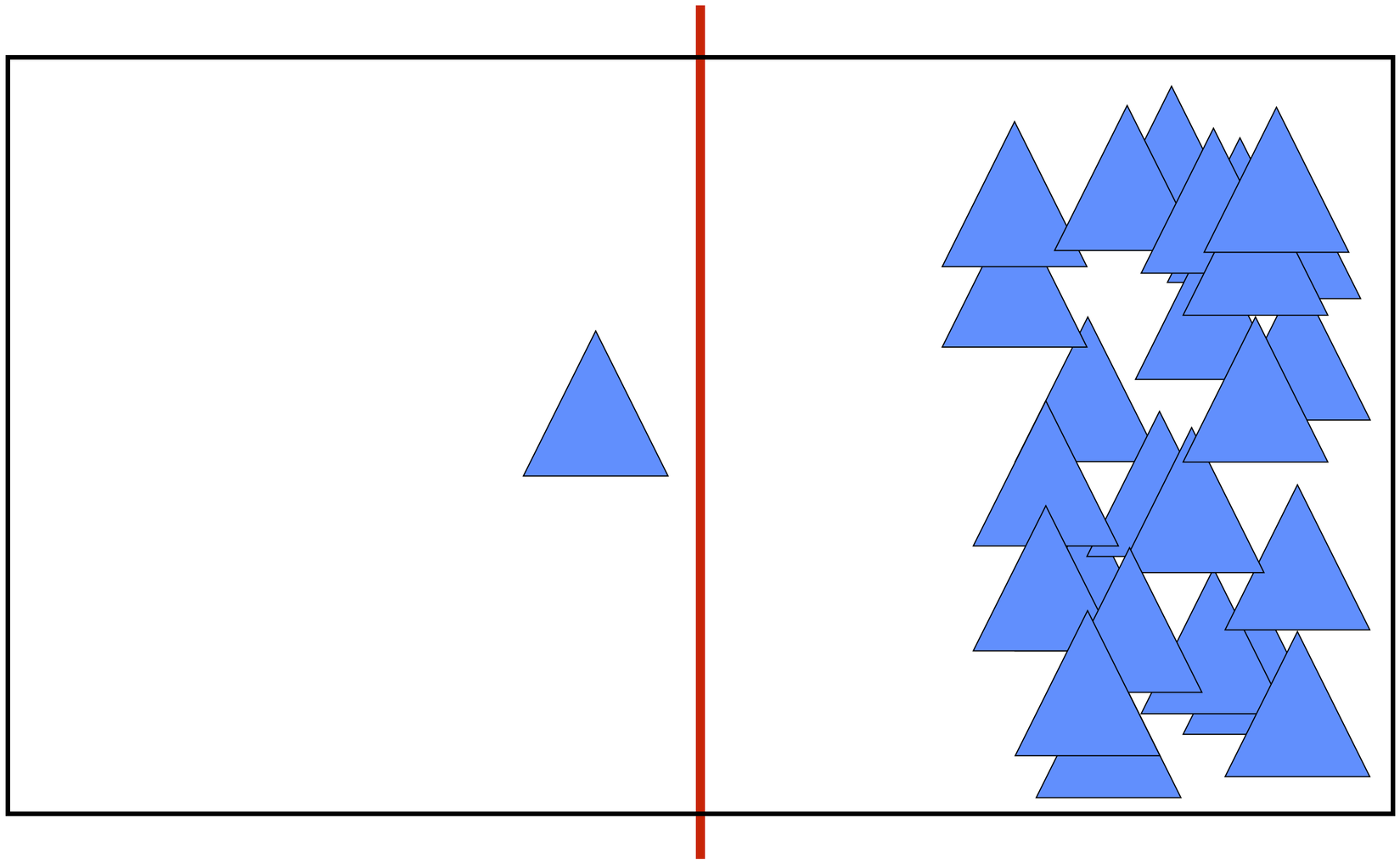
Cost(L) = cost of traversing left child

Cost(R) = cost of traversing right child

Splitting with Cost in Mind



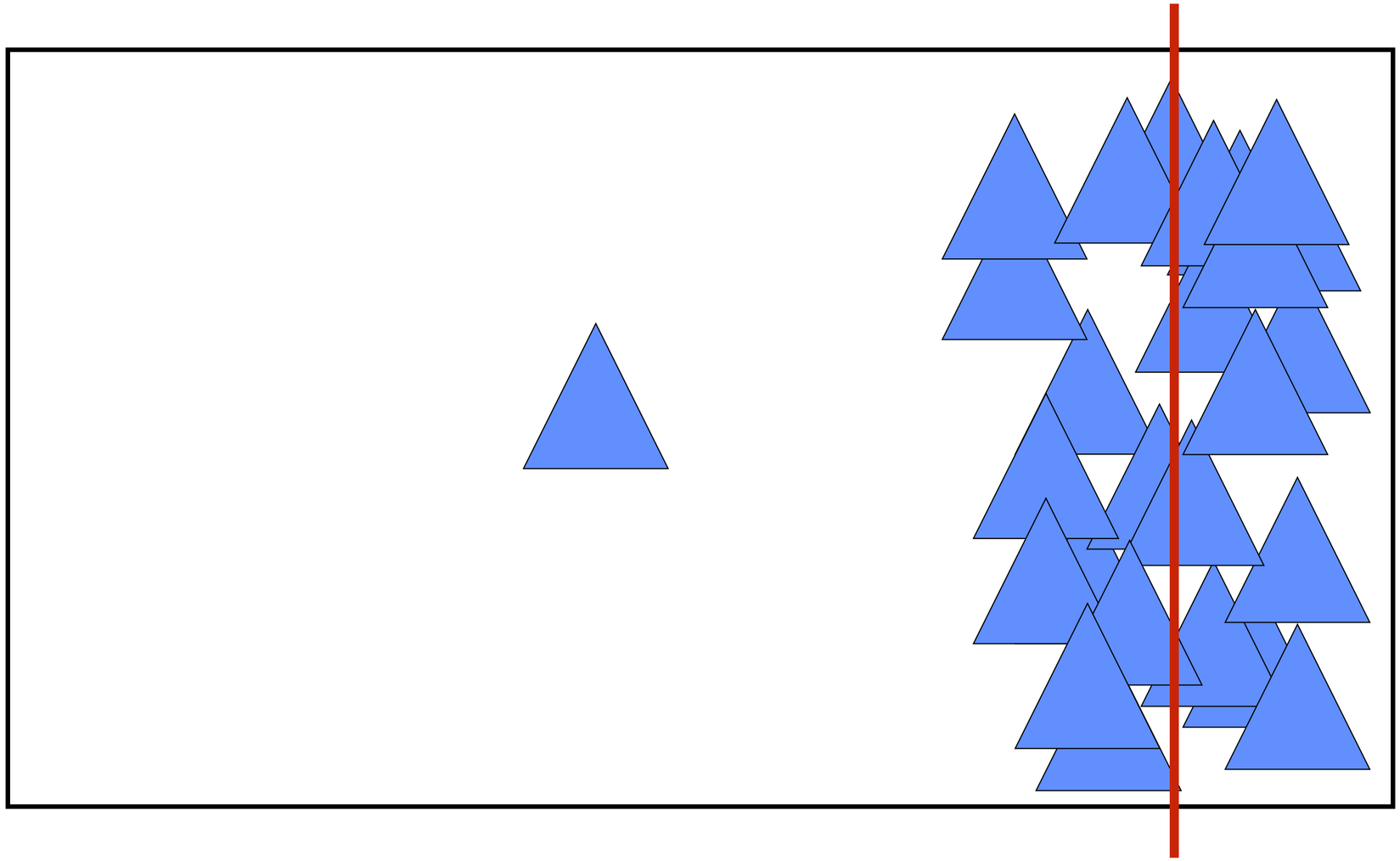
Split in the Middle = Bad!



Midpoint: Makes the L & R probabilities equal

Cost of R greater than cost of L

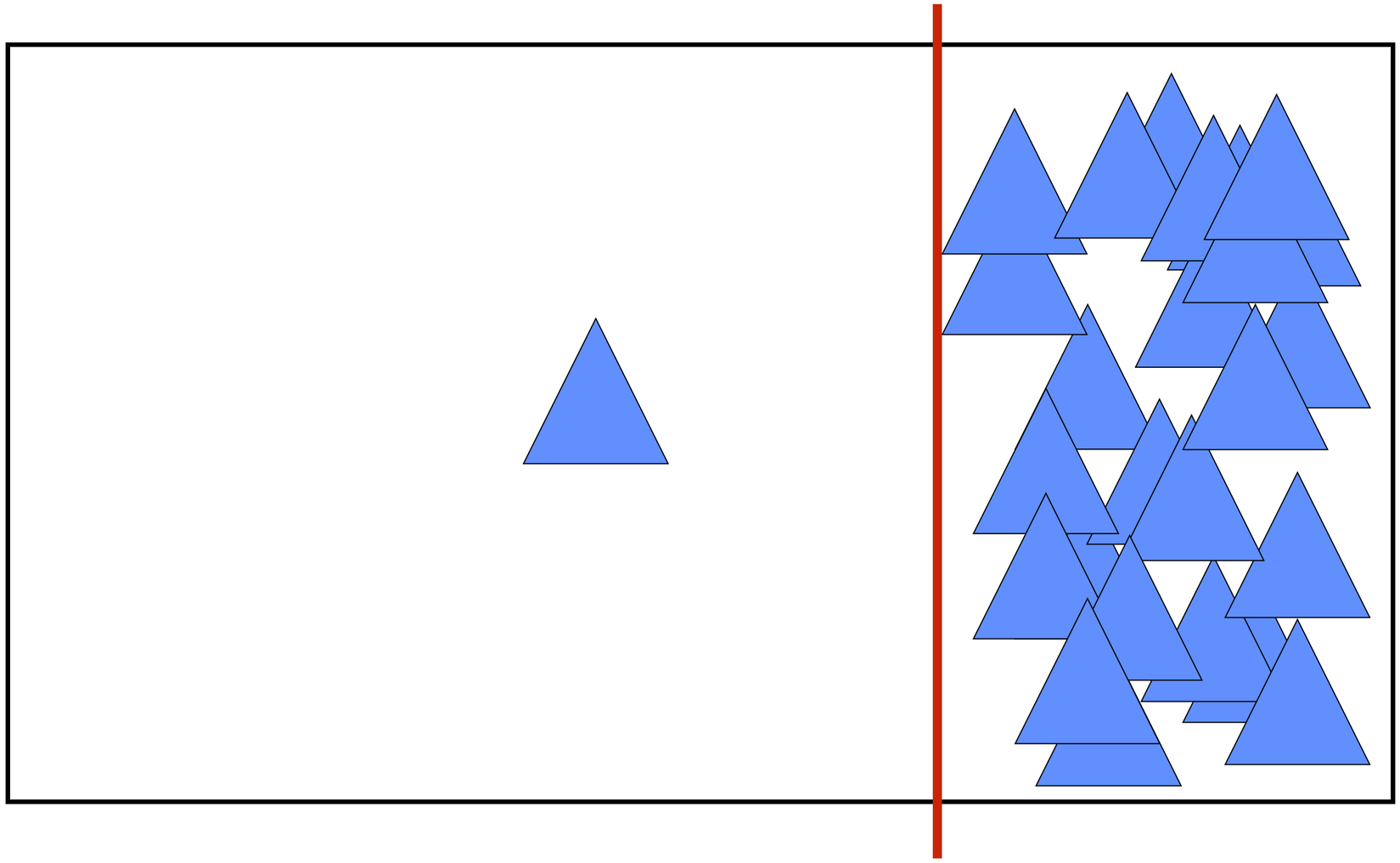
Split at the Median = Bad!



Median: Makes the L & R costs equal

Probability of hitting L greater than R

Cost-Optimized Split = Best!



Cost(cell)=C_trav+Prob(hit L)*Cost(L)+Prob(hit R)*Cost(R)

Cost

Need the probabilities

- **Turns out to be proportional to surface area**

Need the child cell costs

- **Triangle count is a good approximation**

Cost(cell) = C_trav + SA(L)*TriCount(L) + SA(R)*TriCount(R)

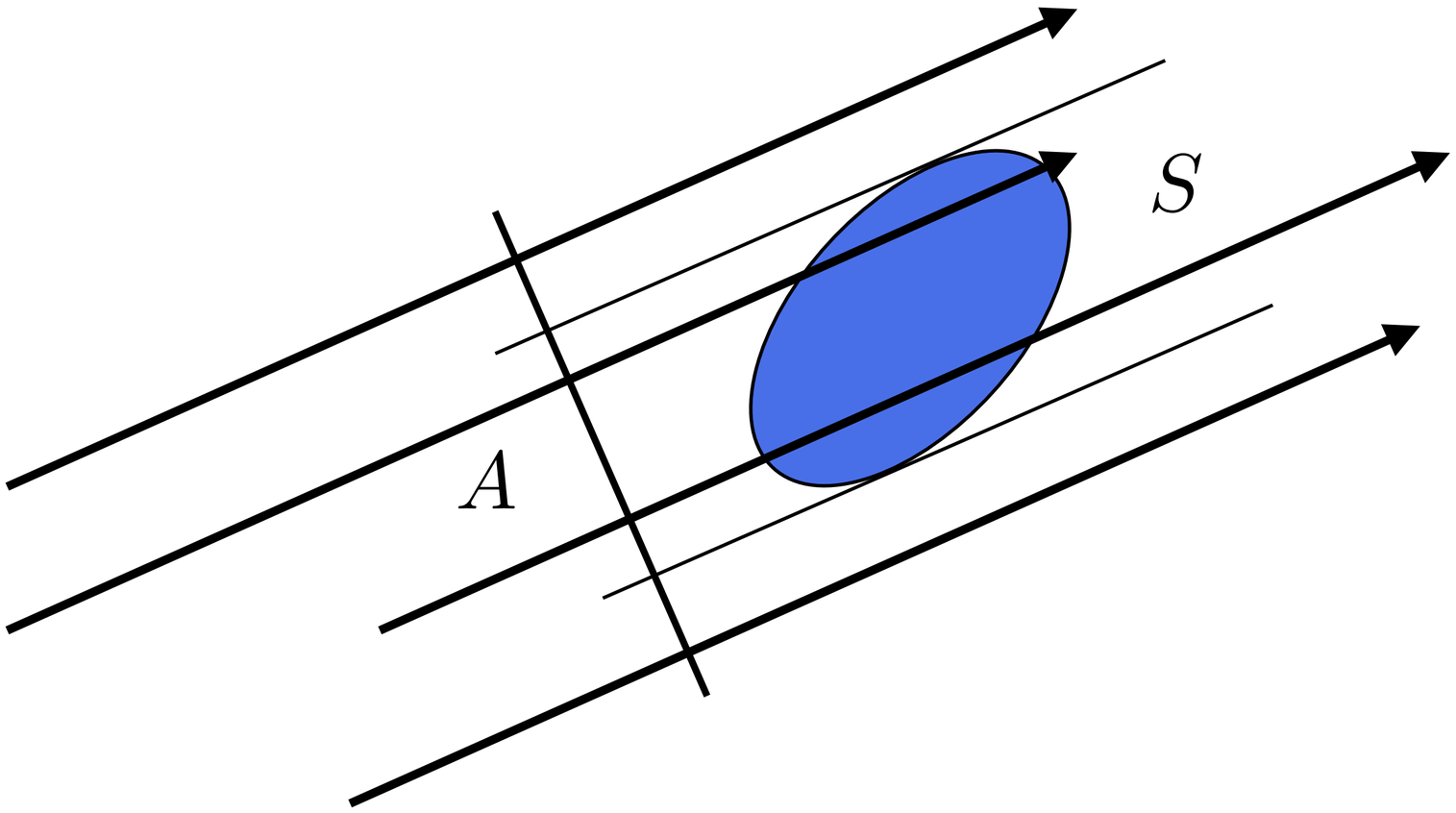
C_trav: the ratio of cost to traverse to cost to intersect

C_trav = 1:80 in pbrt

C_trav = 1:1.5 in a highly optimized version

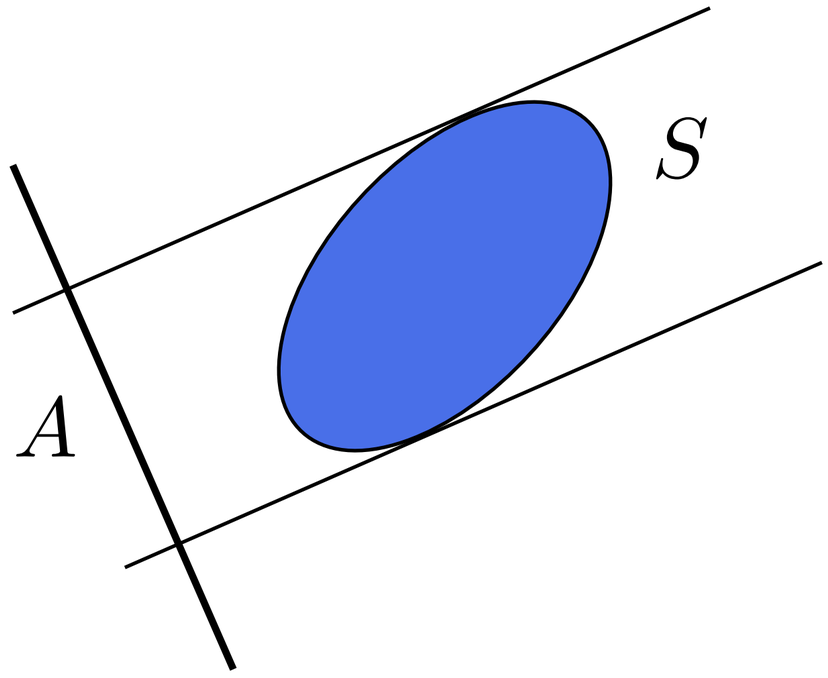
Projected Area and Surface Area

The probability of ray in a given direction hitting an object with surface area *S* is proportional to its projected area A in the direction of the ray



Average Proj Area and Surface Area

The probability of ray in a any direction hitting an object with surface area *S* is proportional to its average projected area

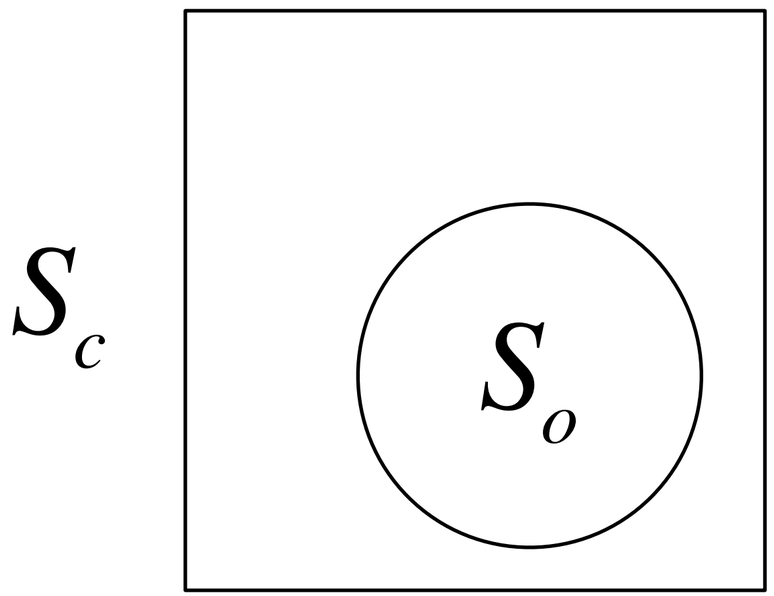


Average projected area: $\bar{A} = \frac{1}{4\pi} \int A \, \mathrm{d}\omega$

Crofton's theorem: $\bar{A} = \frac{S}{4}$ **(Convex shapes only)**

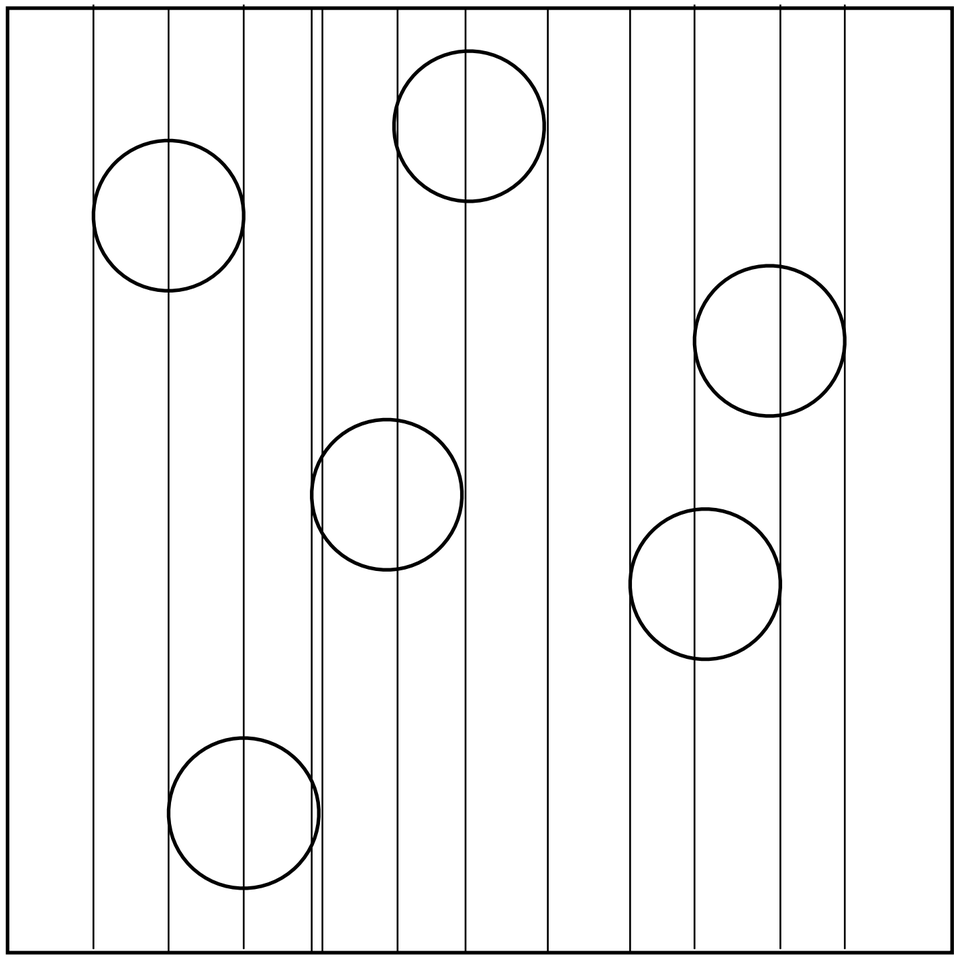
Ray Intersection Probability

The probability of a ray hitting a convex shape enclosed by another convex shape is



$$\Pr[r \cap S_o \,|\, r \cap S_c] = \frac{S_o}{S_c}$$

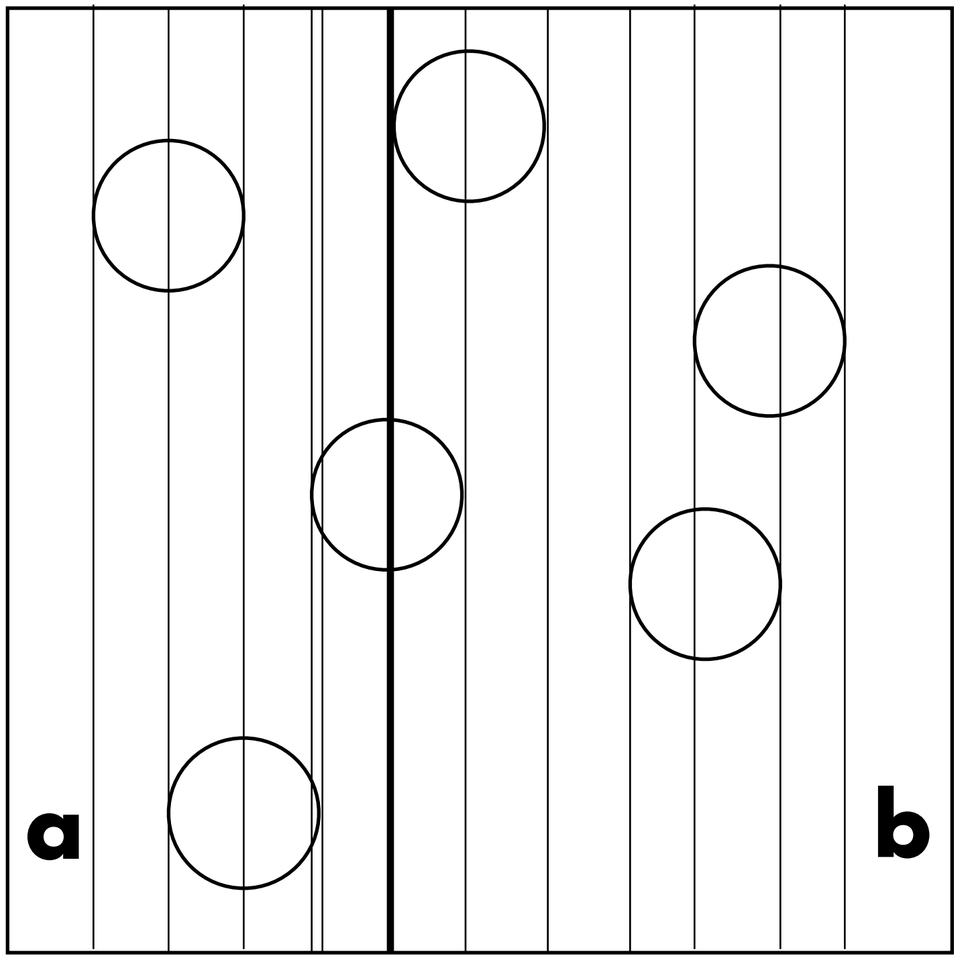
Build Algorithm



2n candidate split locations

Note: Cost minimum must be at extrema

Build Algorithm



2n candidate split locations

$$p_a = \frac{S_a}{S} \qquad p_b = \frac{S_b}{S}$$

$$N_a \qquad N_b$$

Sweep Build Algorithm (Triangles)

1. Pick an axis, or optimize across x, y, z
2. Build a set of "candidate" split locations
3. Sort the triangles into intervals
4. Sweep to incrementally track L/R counts, cost
5. Output position of minimum cost split

Running time: $T(N) = N \log N + 2T(N/2)$

$$T(N) = N \log^2 N$$

Termination Criteria

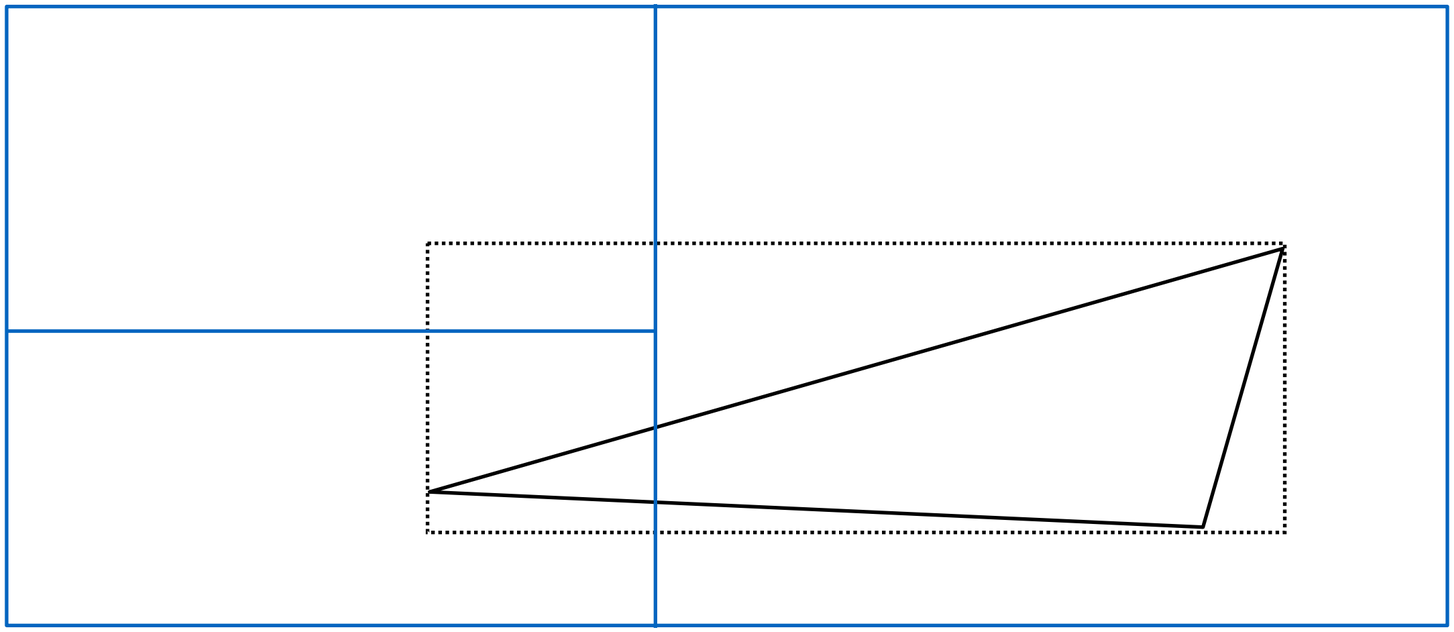
When should we stop splitting?

- Bad: depth limit, number of triangles
- Good: When split does not lower the cost

Threshold of cell size

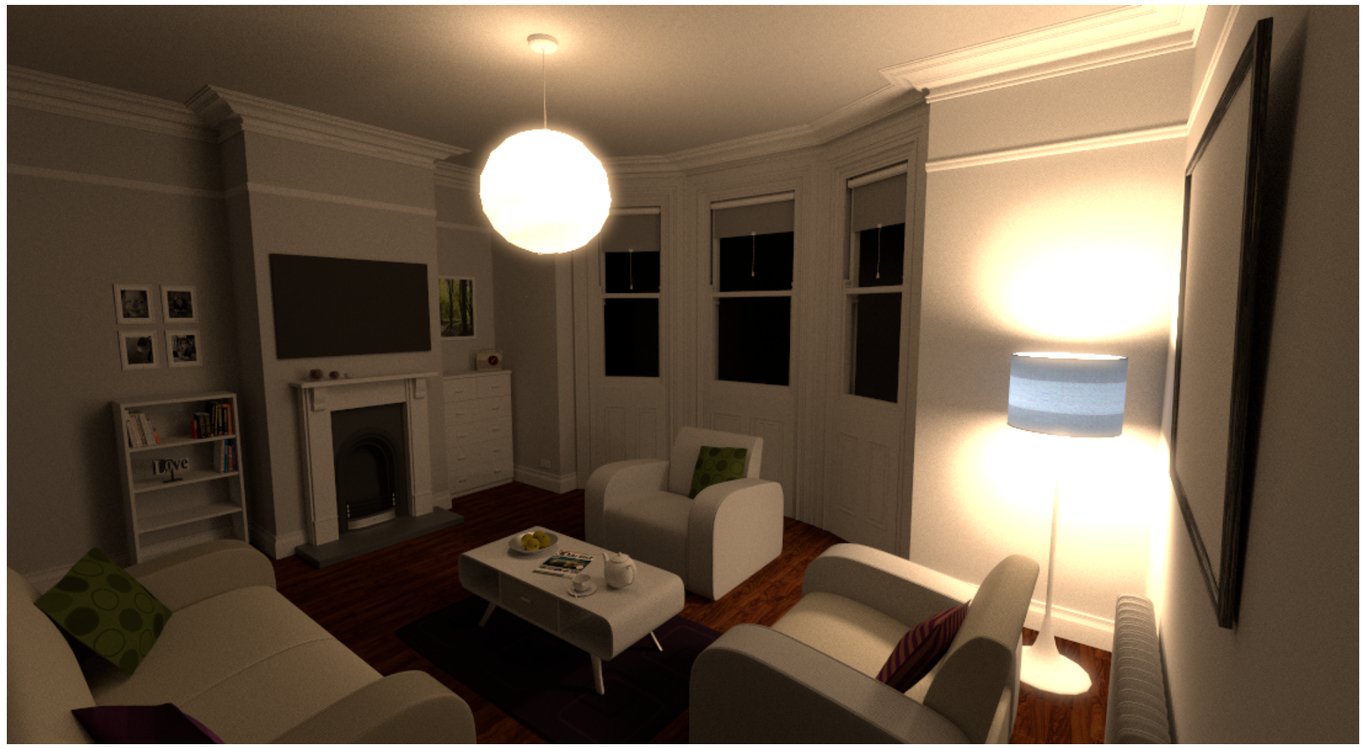
- Absolute probability SA(node)/SA(scene) small

"Split Clipping"



Bounding box is conservative:
may incorrectly report primitive overlap

Results



# Triangles	600,127
Render time	19328s
kd-tree Build	0.43s (~0%)
BVH Traverse	7775s (40.0%)
Tri Intersect	6279s (32.5%)
Ray/Tri tests	107.4B
Ray/Tri hits	4.4B (4.1%)

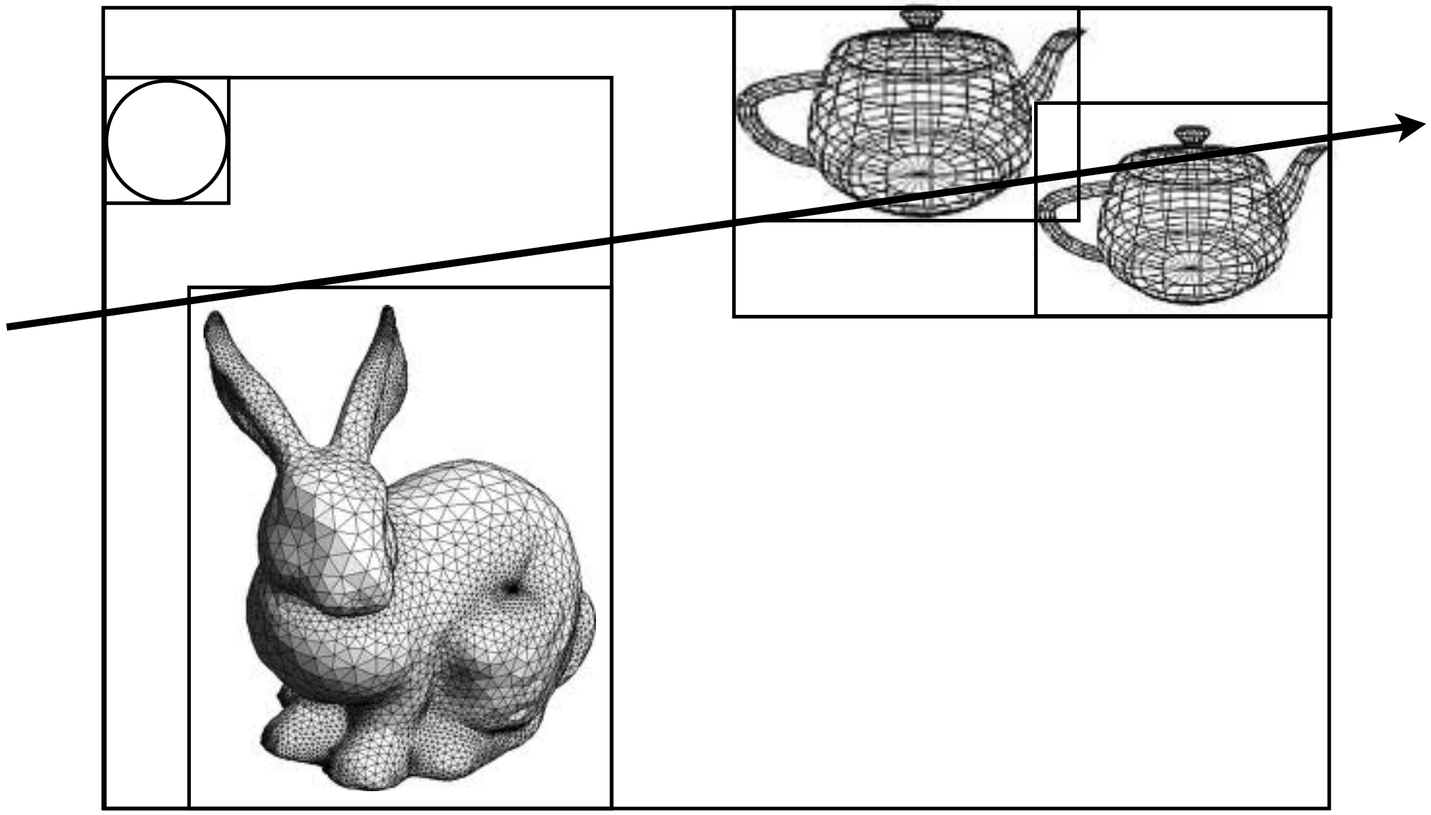
Primitive Hierarchies

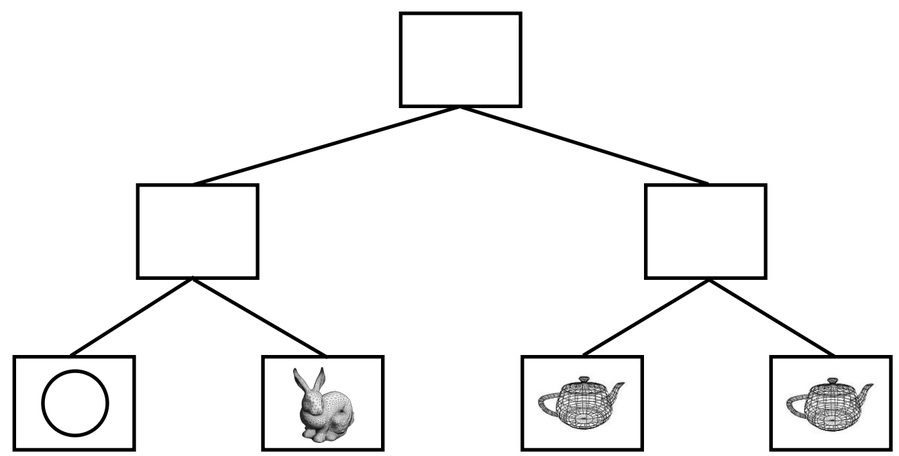
Bounding Volume Hierarchies (BVH)

Bounding Volume Hierarchies

Spatial subdivision (grids, kd-trees): subdivide 3D space into non-overlapping cells

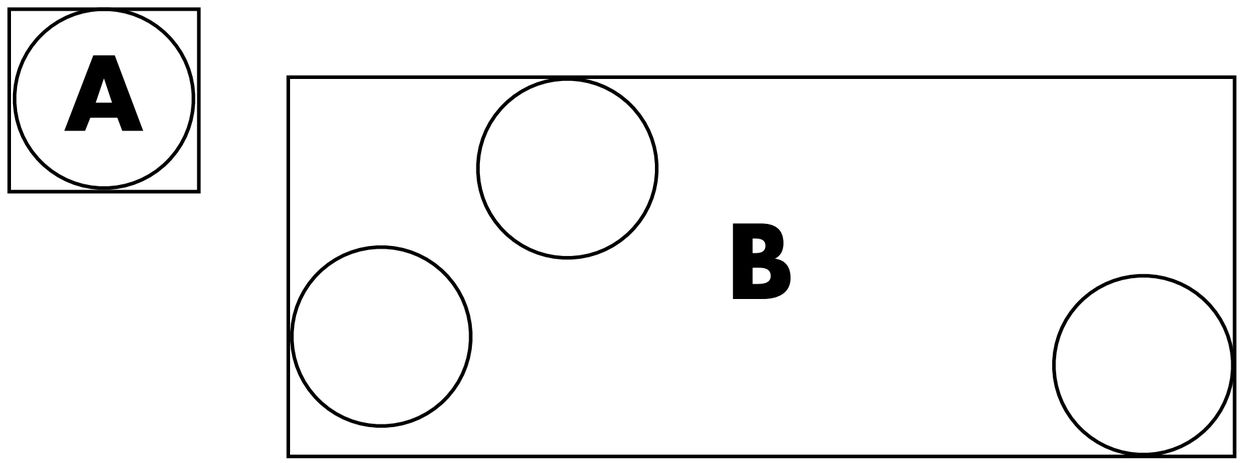
Primitive subdivision (e.g. bounding volume hierarchies): partition scene primitives into disjoint sets (note bounding volumes may overlap)





Building BVH

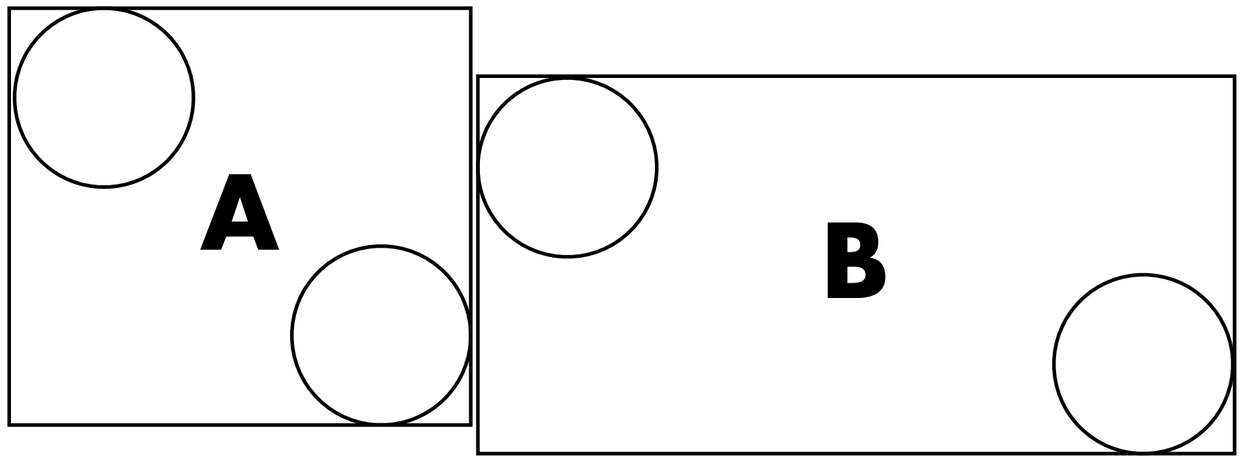
Can also apply sweep build



$$\text{Cost} = N_A * S_A + N_B * S_B$$

Building BVH

Can also apply sweep build



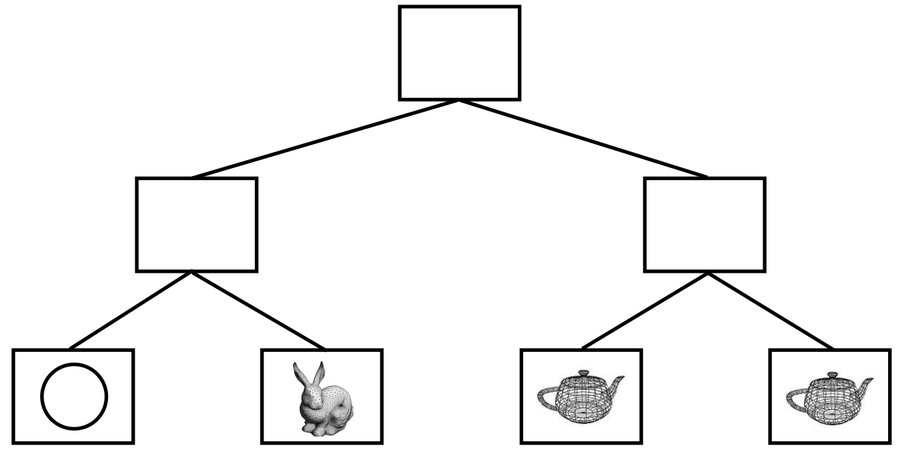
$$\text{Cost} = N_A * S_A + N_B * S_B$$

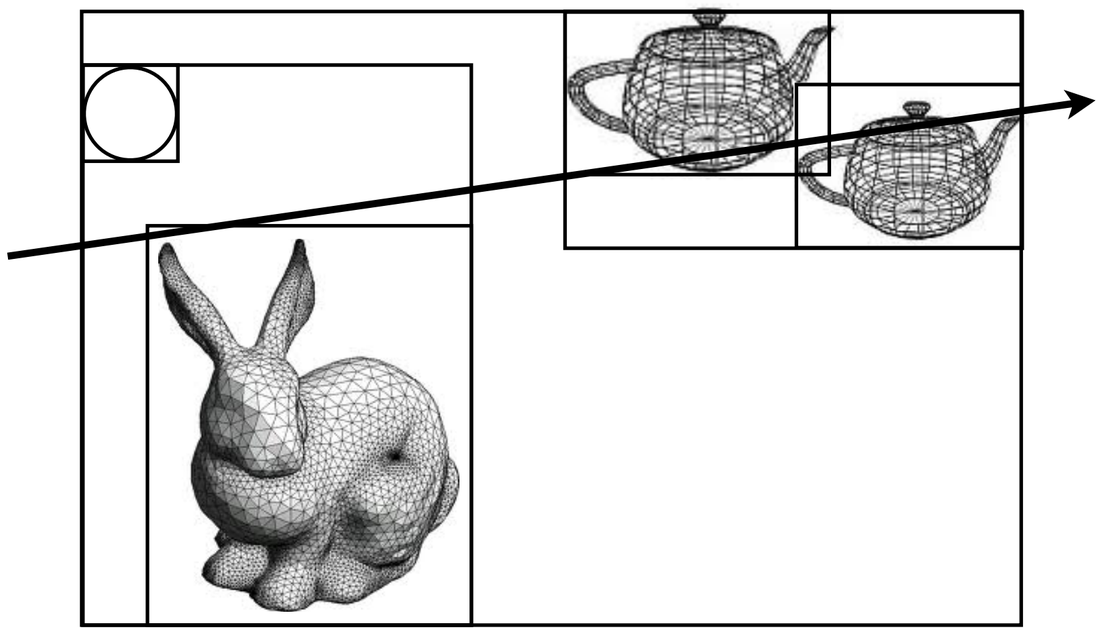
**Consider all three axes during partitioning,
Choose lowest cost**

**Caveat: “greedy” algorithm;
not guaranteed to give global optimum**

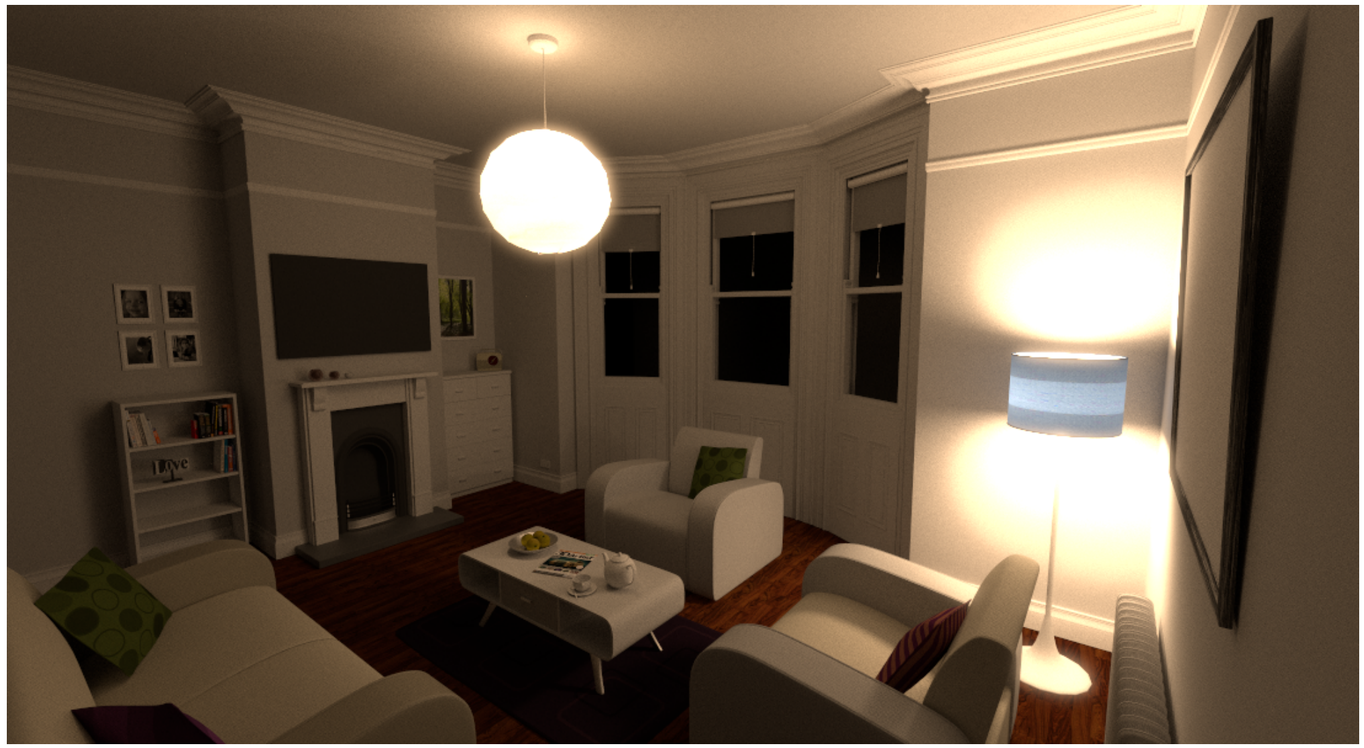
BVH Traversal

```
stack<Node *> toVisit;

toVisit.push(root);

while (Node *cur = toVisit.pop()) {

   if (cur->bounds->Intersect(ray)) {

     if (cur->isLeaf) {

       /* intersect with primitives */

     } else {

       toVisit.push(cur->near);

       toVisit.push(cur->far);

   }

}
```



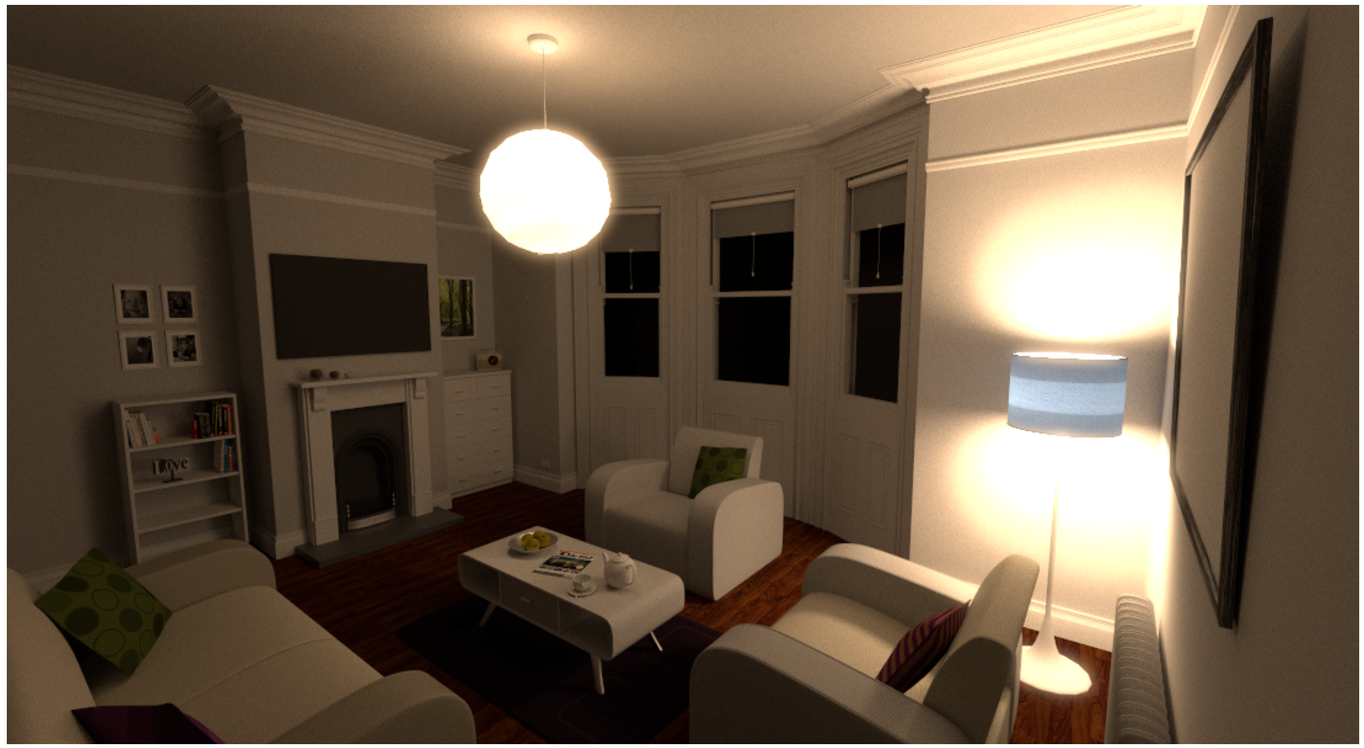


Results



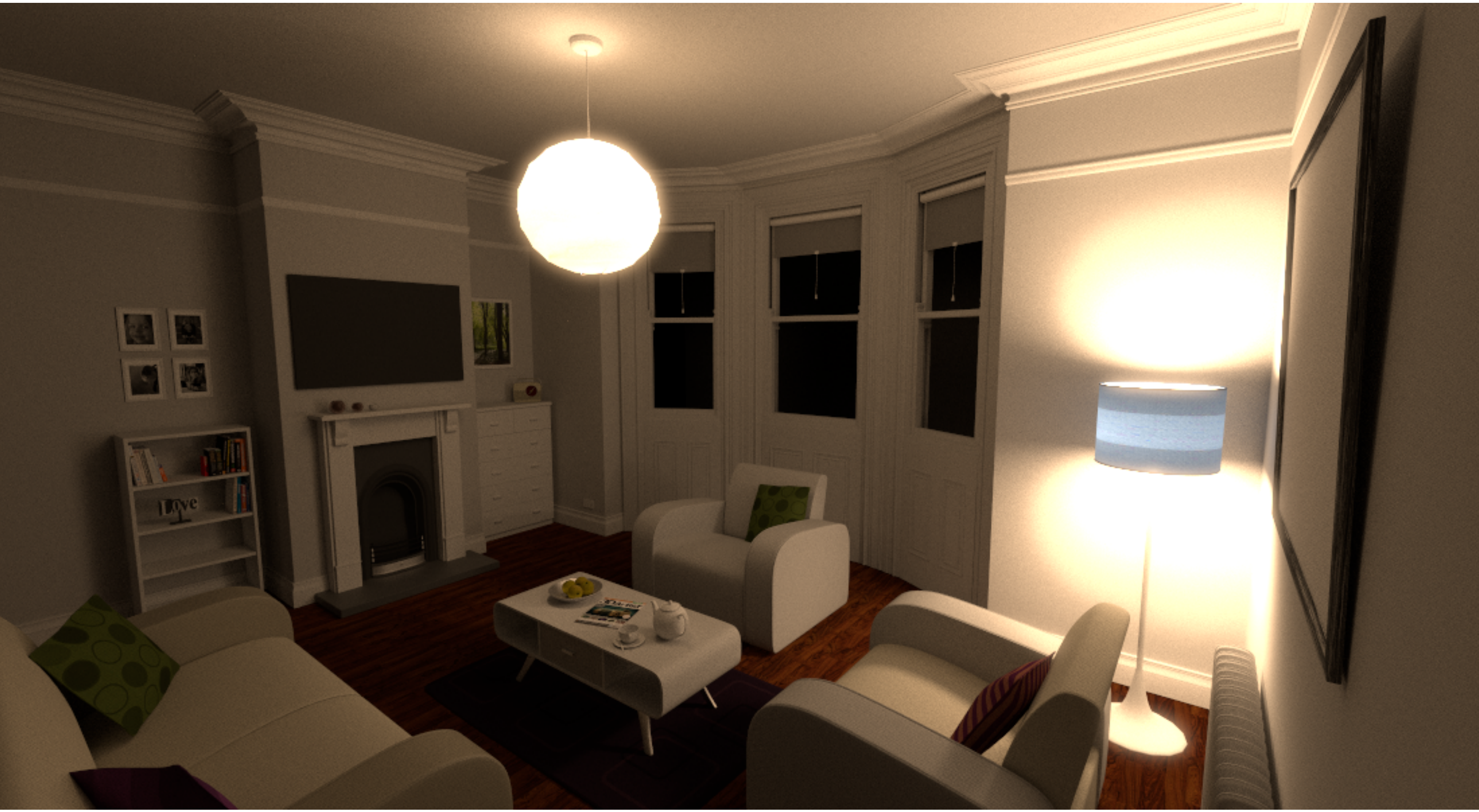
# Triangles	600,127
Render time	12905s
BVH Build	0.17s (~0%)
BVH Traverse	6155s (47.7%)
Tri Intersect	2178s (16.9%)
Ray/Tri tests	22.6B
Ray/Tri hits	3.6B (16%)

Comparison



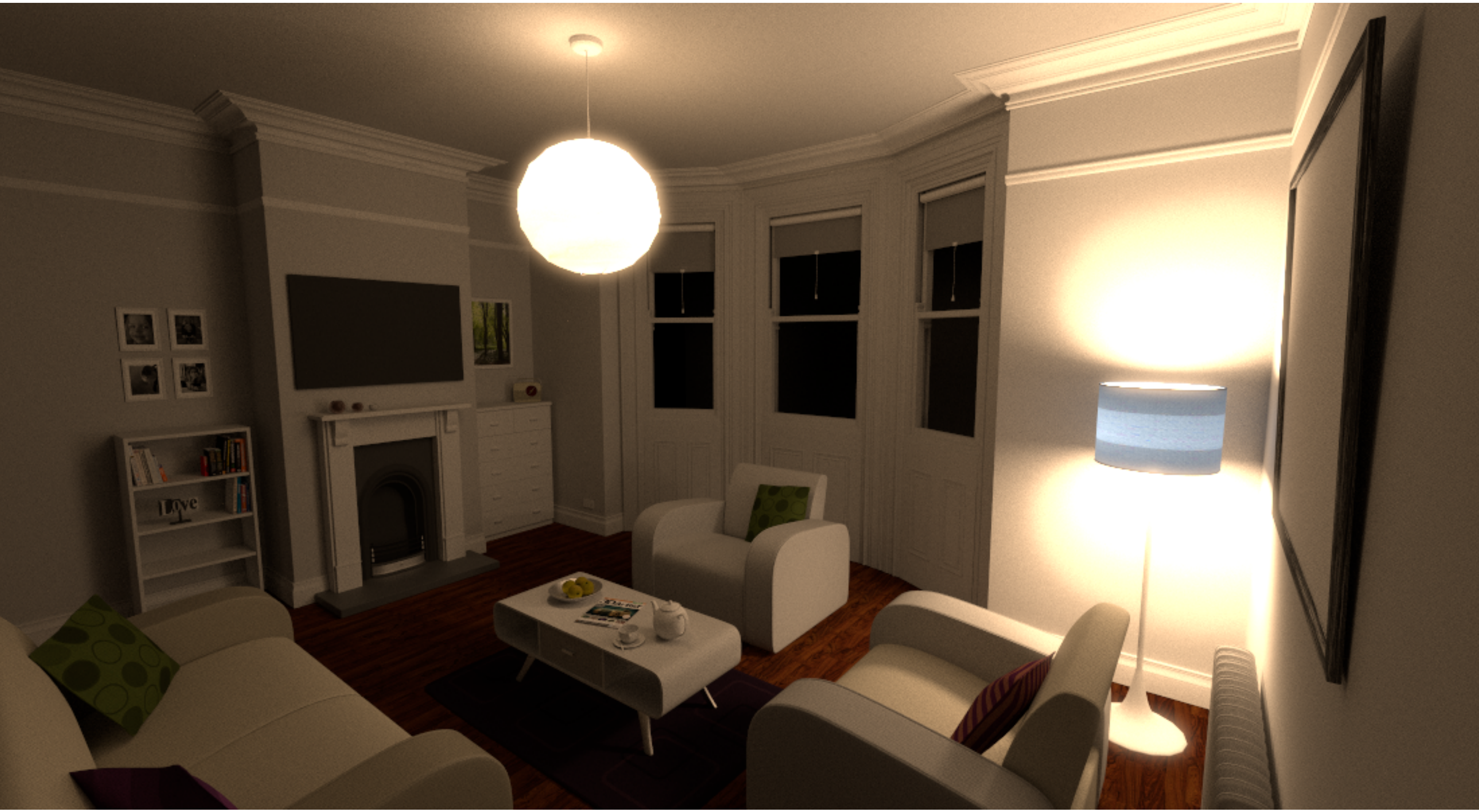
	BVH	kd-tree
Render time	12905s	19328s
Build	0.17s (~0%)	0.43s (~0%)
Traverse	6155s (47.7%)	7775s (40.0%)
Tri Intersect	2178s (16.9%)	6279s (32.5%)
Ray/Tri tests	22.6B	107.4B
Ray/Tri hits	3.6B (16%)	4.4B (4.1%)

What's the difference?



Rendering time: 27m 38s

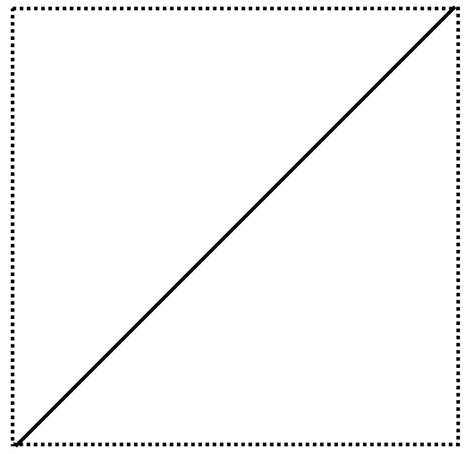
What's the difference?



Rendering time: 1h 55m 45s

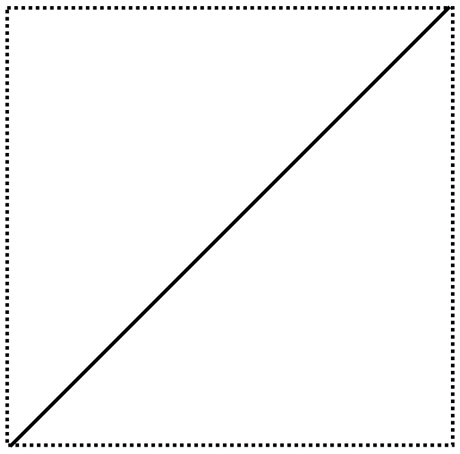
Axis-alignment and performance

Wall and its bounding box

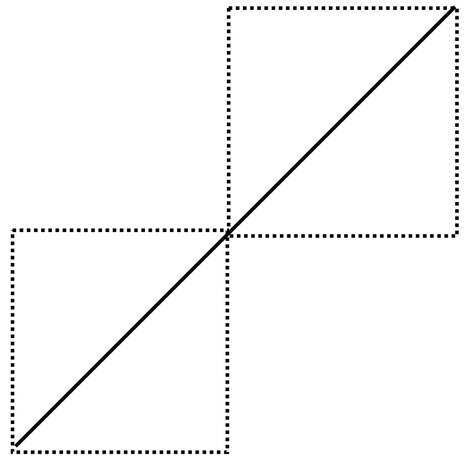


Rotated wall and its bounding box

Axis-alignment and performance



Rotated wall and its bounding box



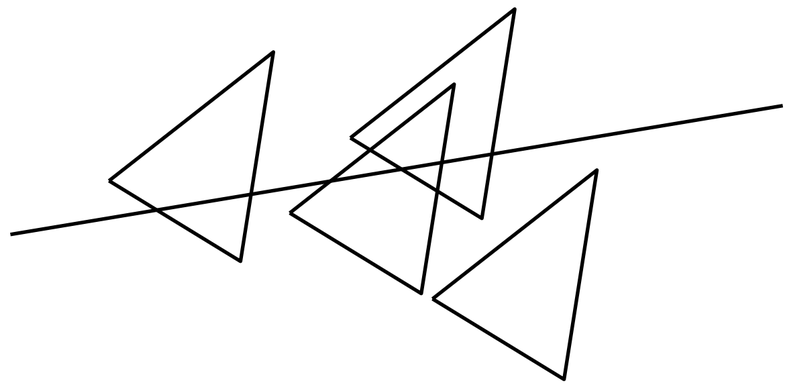
Work-around: refine bounding boxes

Theory

Computational geometry of ray shooting

1. Triangles (Pellegrini)

- **Time:** $O(\log n)$
- **Space:** $O(n^{5+\varepsilon})$



2. Sphere (Guibas and Pellegrini)

- **Time:** $O(\log^2 n)$
- **Space:** $O(n^{5+\varepsilon})$