## Ray Tracing

Ray Tracing 1

- Basic algorithm
- Overview of pbrt
- Ray-surface intersection (triangles, …)

Ray Tracing 2

- Problem: brute force = |Image| x |Objects|
- Acceleration data structures



## Primitives

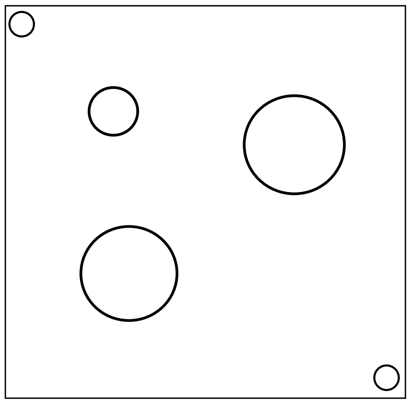
`pbrt` primitive base class

- Shape
- Material (reflection and emission)

Subclasses

- Primitive instance
  - Transformation and pointer to a primitive
- Aggregate (collection)
  - Treat collections just like single primitives
  - Incorporate acceleration structures into collections
  - May nest accelerators of different types
  - Types: grid.cpp and kdtree.cpp

## Uniform Grids

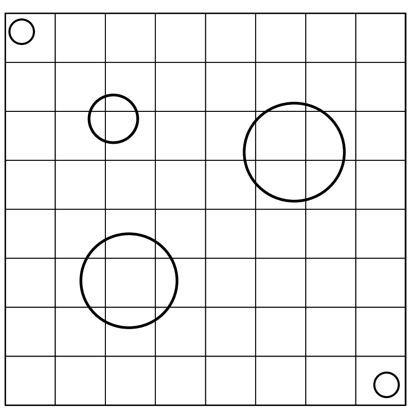


**Preprocess scene**

**1. Find bounding box**



## Uniform Grids



**Preprocess scene**

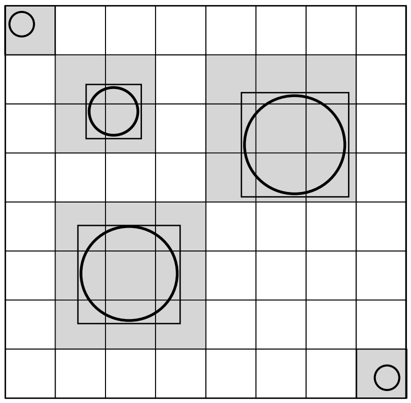
**1. Find bounding box**

**2. Determine resolution**

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

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

## Uniform Grids



**Preprocess scene**

**1. Find bounding box**

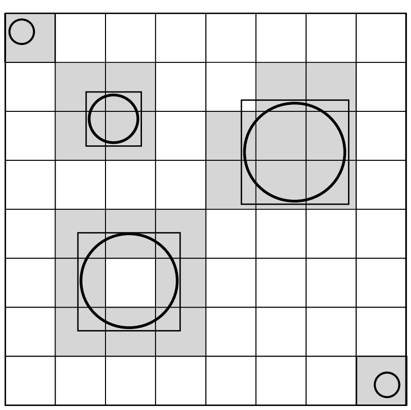
**2. Determine resolution**

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

**3. Place object in cell, if object overlaps cell**



## Uniform Grids



**Preprocess scene**

**1. Find bounding box**

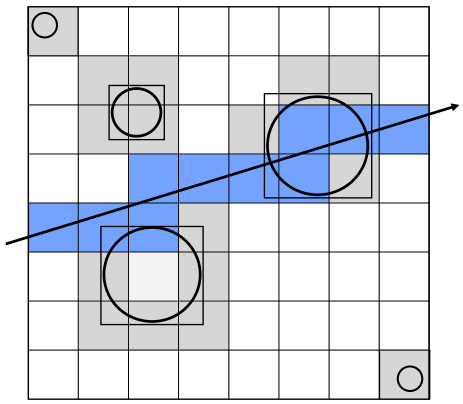
**2. Determine resolution**

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

**3. Place object in cell, if object overlaps cell**

**4. Check that object's surface intersects cell**

## Uniform Grids



**Preprocess scene**

**Traverse grid**

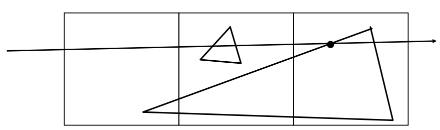
- **3D line – 3D-DDA**
- **6-connected line**

**Section 4.3**



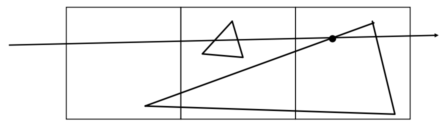
## Caveat: Overlap

**Problem: Don't output first intersection found!**



## Caveat: Overlap

**Problem: Don't output first intersection found!**

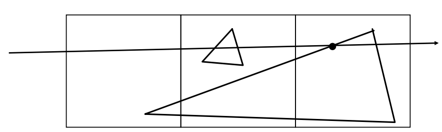


**Problem: Redundant intersection tests**

CS348B Lecture 3 Pat Hanrahan, Spring 2010


## Caveat: Overlap

**Problem: Don't output first intersection found!**

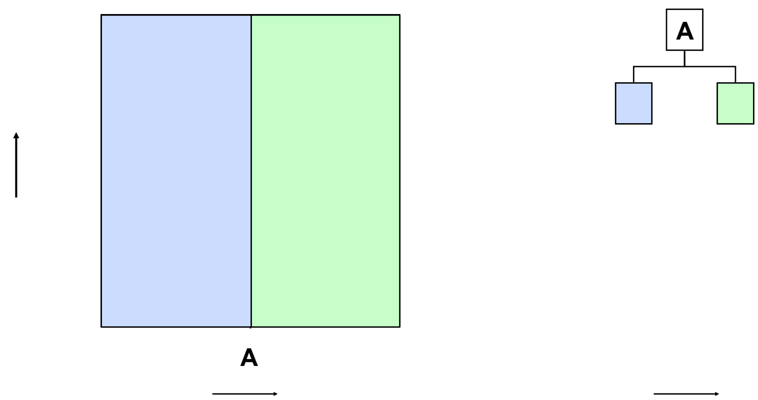
**Problem: Redundant intersection tests**

**Solution: Mailboxes**

- **Assign each ray an increasing number**
- **Primitive intersection cache (mailbox)**
  - **Store last ray number tested in mailbox**
  - **Only intersect if ray number is greater**

CS348B Lecture 3 Pat Hanrahan, Spring 2010

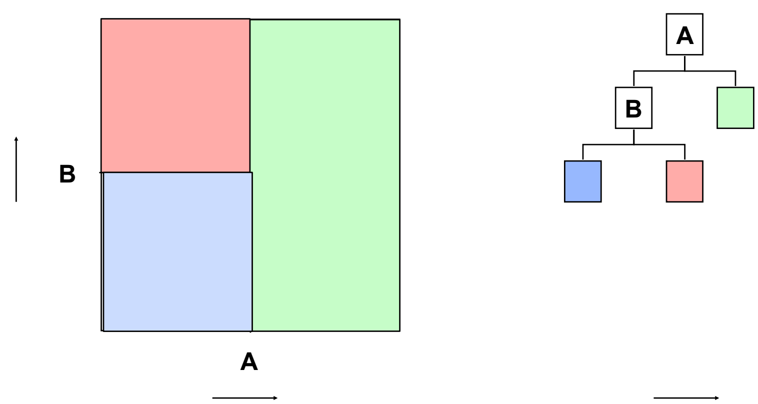
## Spatial Hierarchies




Letters correspond to planes (A)



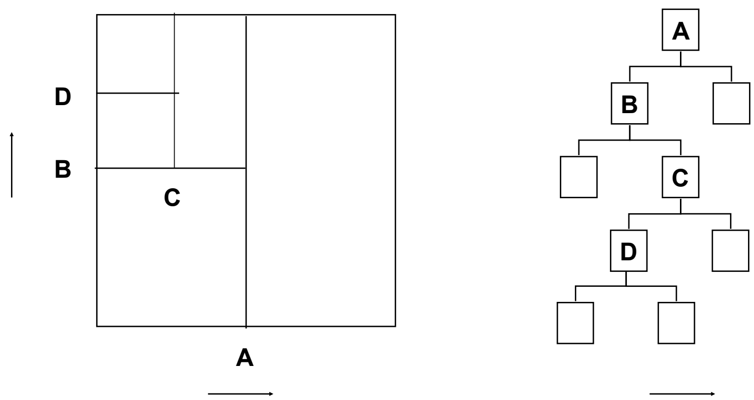
## Spatial Hierarchies




Letters correspond to planes (A, B)

Point Location by recursive search

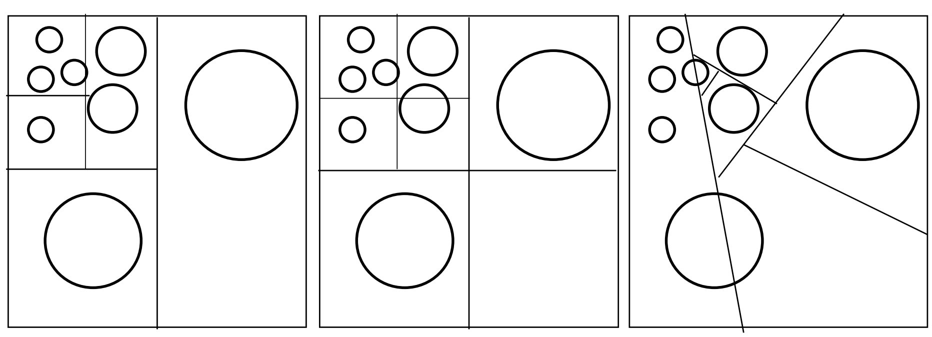
## Spatial Hierarchies




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



## Variations



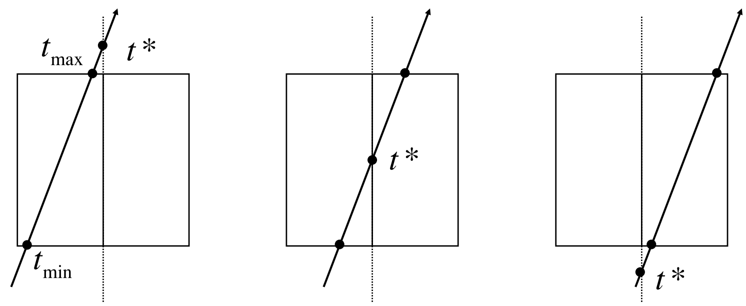
**kd-tree** **oct-tree** **bsp-tree**

## Ray Traversal Algorithms

**Recursive inorder traversal**

**[Kaplan, Arvo, Jansen]**

$$t^* = (S - \mathbf{O}[a]) / \mathbf{D}[a]$$




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

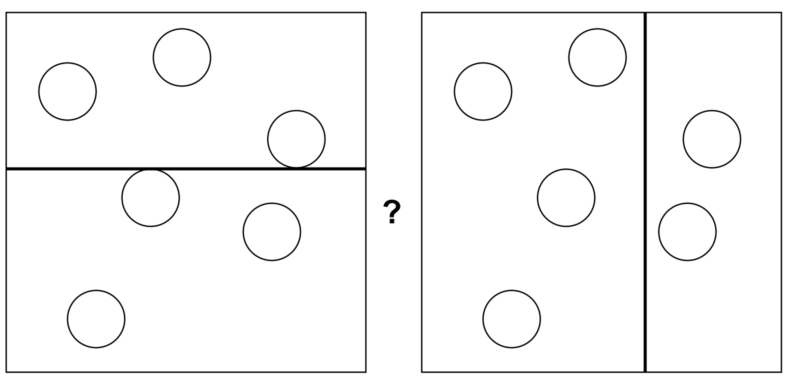
```
Intersect(L,tmin,tmax)
```

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

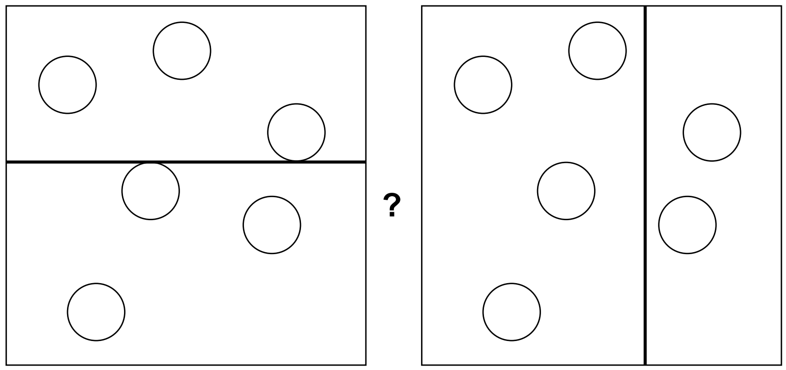
```
Intersect(R,tmin,tmax)
```



## How to Build the Hierarchy?



## Build Hierarchy Top-Down




**Methods to choose axis and splitting plane**

- **Midpoint**
- **Median cut (balanced)**
- **Surface area heuristic**



## Cost

**What is the cost of tracing a ray through a node?**

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

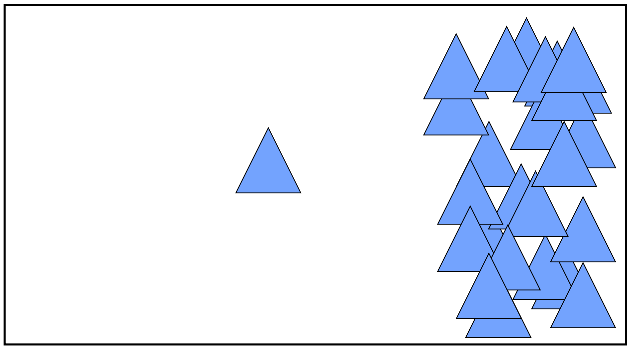
C_trav = cost of traversing a cell

Cost(L) = cost of traversing left child

Cost(R) = cost of traversing right child

## Splitting with Cost in Mind

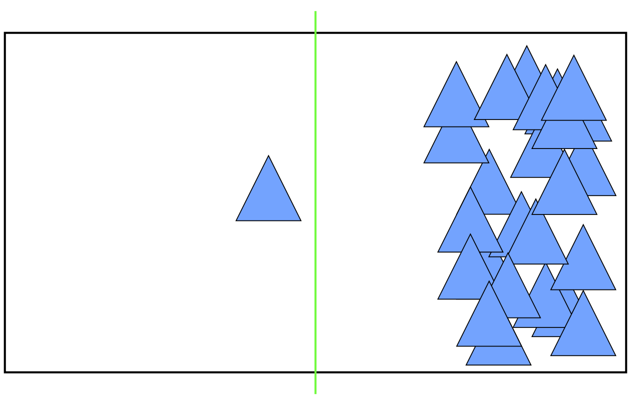
From Gordon Stoll





## Split in the Middle = Bad!

From Gordon Stoll

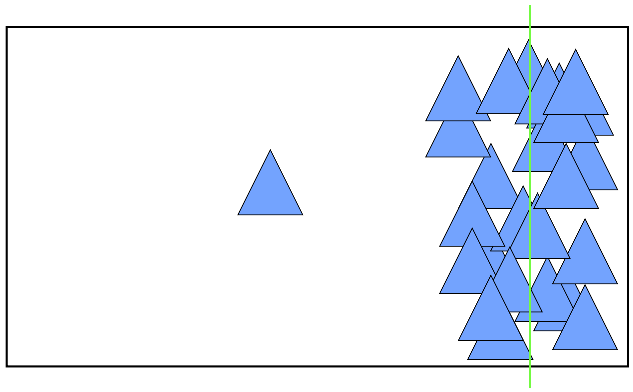


**Makes the L & R probabilities equal**

**Cost of R greater than cost of L**

## Split at the Median = Bad!

From Gordon Stoll



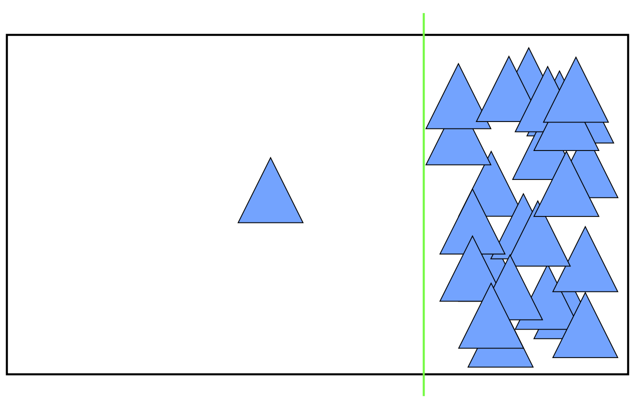
Makes the L & R costs equal

Probability of hitting L greater than R



## Cost-Optimized Split = Good!

From Gordon Stoll



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 is the ratio of the cost to traverse to the cost to intersect

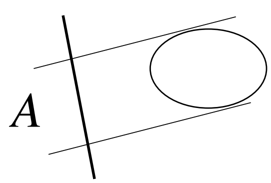
C_trav = 1:80 in pbrt

C_trav = 1:1.5 in a highly optimized version



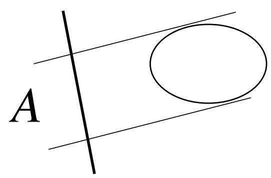
## Projected Area and Ray Intersection

**Number of rays in a given direction that hit an object is proportional to its projected area**



## Projected Area and Surface Area

**Number of rays in a given direction that hit an object is proportional to its projected area**




**The total number of rays hitting an object is** $4\pi \overline{A}$

**Crofton's Theorem:**

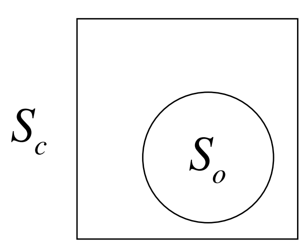
**For a convex body** $\overline{A} = \frac{S}{4}$

**For a sphere** $S = 4\pi r^2$ **and** $\overline{A} = A = \pi r^2$



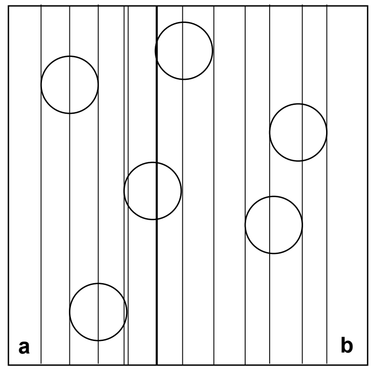
## Surface Area and Ray Intersection

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




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

## Sweep Build Algorithm




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

$$N_a \qquad N_b$$



## Basic Build Algorithm (Triangles)

1. Pick an axis, or optimize across x, y, z
2. Build a set of "candidate" split locations

   Note: Cost extrema must be at bbox vertices

   - Vertices of triangle
   - Vertices of triangle clipped to node bbox
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 cost improvement

- Stretch over multiple levels
- For example, if cost doesn't go down after three splits in a row, terminate

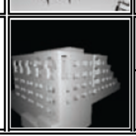
Threshold of cell size

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



# Best Reported Timings

**Millions of Rays per Second**

| scene # of triangles and shader (+/-) \ Framerate (FPS) @ 1024x1024 resolution | | | OpenRT @ 2.5 GHz P4 1 thread | MLRTA @ 2.4 GHz P4 1 thread | MLRTA @ 3.2 GHz P4 with HT 2 threads |
|---|---|---|---|---|---|
| Erw6 804 |  | – shader | 7.1 | 70.2 | 109.8 |
| | | + shader | 2.3 | 37.8 | 50.7 |
| Confe-rence 274K |  | – shader | 4.55 | 11.2 | 19.5 |
| | | + shader | 1.93 | 9.5 | 15.6 |
| Soda Hall 2195K |  | – shader | 4.12 | 21.1 | 35.5 |
| | | + shader | 1.8 | 15.3 | 24.1 |

Reshetov, Soupikov, Hurley, SIGGRAPH 2005

## Superoptimizations

Lots of optimizations

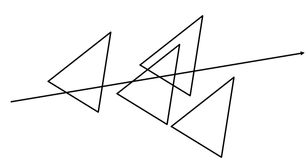
- Carefully written inner loop (no recursion)
- Use vector instructions SSE2
- 64 bits per kd-tree node
  - 32 bit position
  - 32 bit pointer to pair of child nodes
  - 2 bits for split plane direction (x, y, or z)
- Trace packet of rays
  - 4 or more rays at a time
- Intersect beam at top of tree
- Encourage empty nodes
- Special case axis-aligned triangles
- …



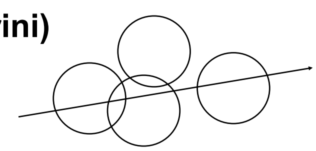
## Theoretical Nugget 1

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: $O(n^{5+\varepsilon})$

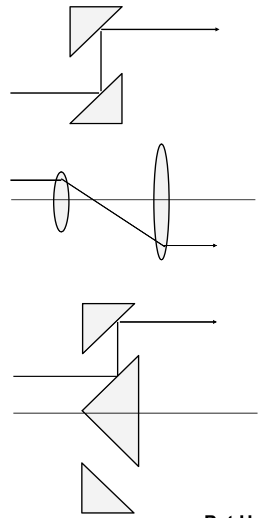


# Theoretical Nugget 2

**Optical computer = Turing machine**

**Reif, Tygar, Yoshida**

**Determining if a ray starting at y0 arrives at yn is undecidable**



**y = y+1**

**y = -2*y**

**if( y>0 )**