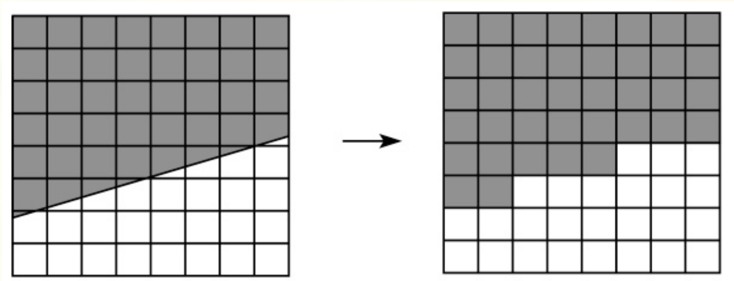
Anti-aliased and accelerated ray tracing

Reading

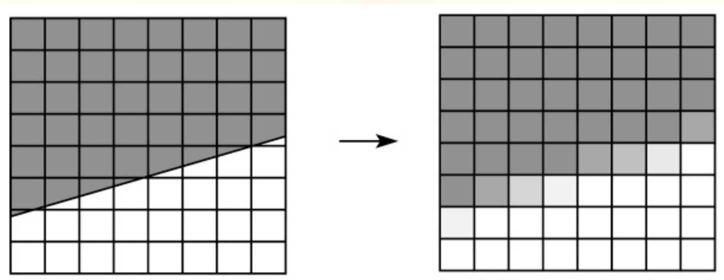
- Required:
  - Watt, sections 12.5.3 – 12.5.4, 14.7
- Further reading:
  - A. Glassner. An Introduction to Ray Tracing. Academic Press, 1989. [In the lab.]

Aliasing in rendering

- One of the most common rendering artifacts is the "jaggies". Consider rendering a white polygon against a black background:



- We would instead like to get a smoother transition:

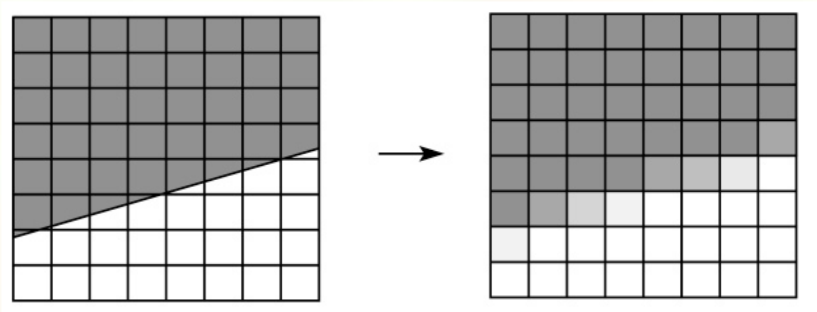


Anti-aliasing

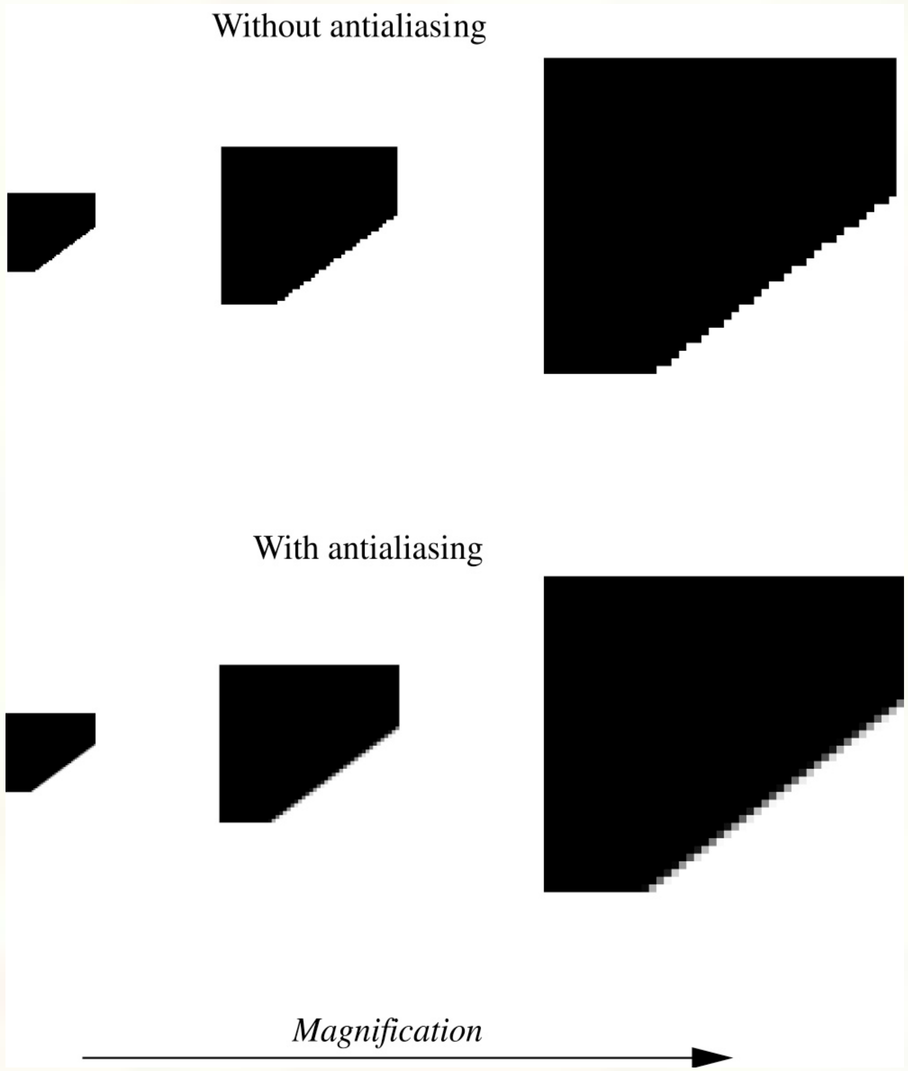
- **Q**: How do we avoid aliasing artifacts?

1. Sampling:
2. Pre-filtering:
3. Combination:

- Example - polygon:

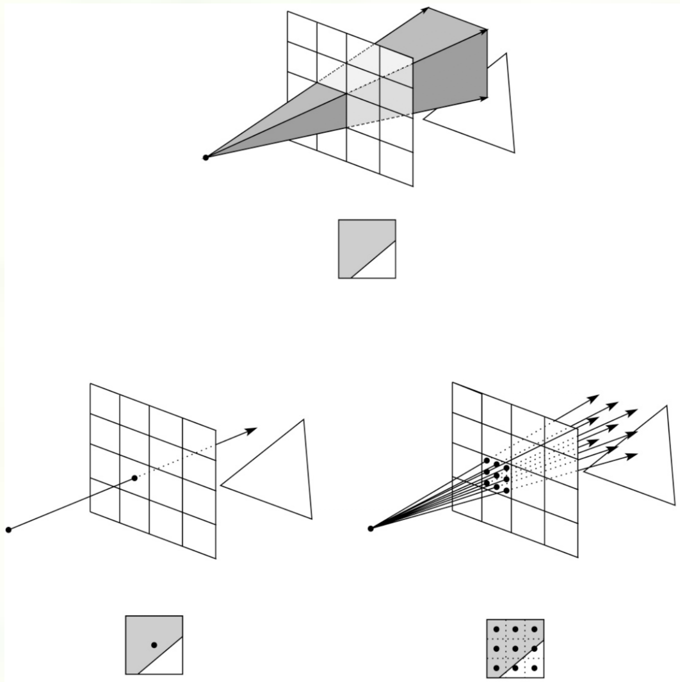


Polygon anti-aliasing



Antialiasing in a ray tracer

- We would like to compute the average intensity in the neighborhood of each pixel.



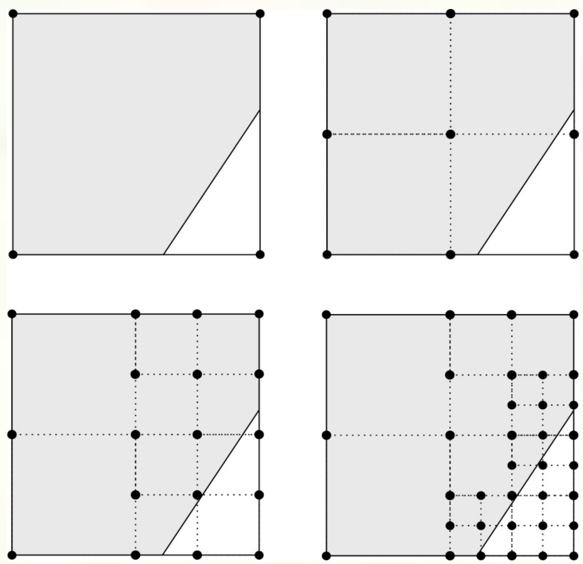
- When casting one ray per pixel, we are likely to have aliasing artifacts.
- To improve matters, we can cast more than one ray per pixel and average the result.
- A.k.a., **super-sampling and averaging down**.

Speeding it up

- Vanilla ray tracing is really slow!
- Consider: m x m pixels, k x k supersampling, and n primitives, average ray path length of d, with 2 rays cast recursively per intersection.
- Complexity =
- For m=1,000,000, k = 5, n = 100,000, d=8…very expensive!!
- In practice, some acceleration technique is almost always used.
- We've already looked at reducing d with adaptive ray termination.
- Now we look at reducing the effect of the k and n terms.

Antialiasing by adaptive sampling

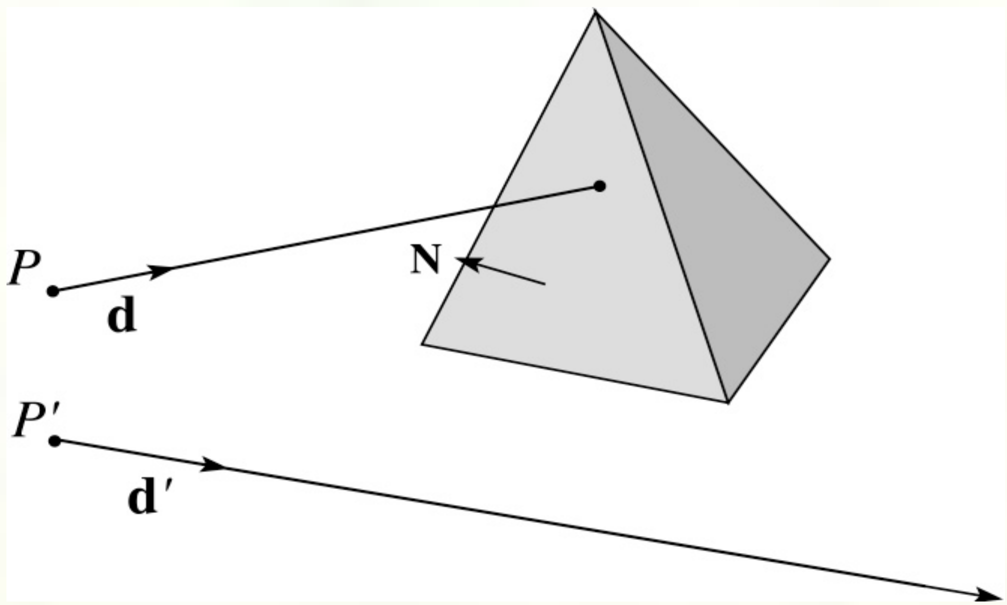
- Casting many rays per pixel can be unnecessarily costly.
- For example, if there are no rapid changes in intensity at the pixel, maybe only a few samples are needed.
- Solution: **adaptive sampling**.



- **Q**: When do we decide to cast more rays in a particular area?

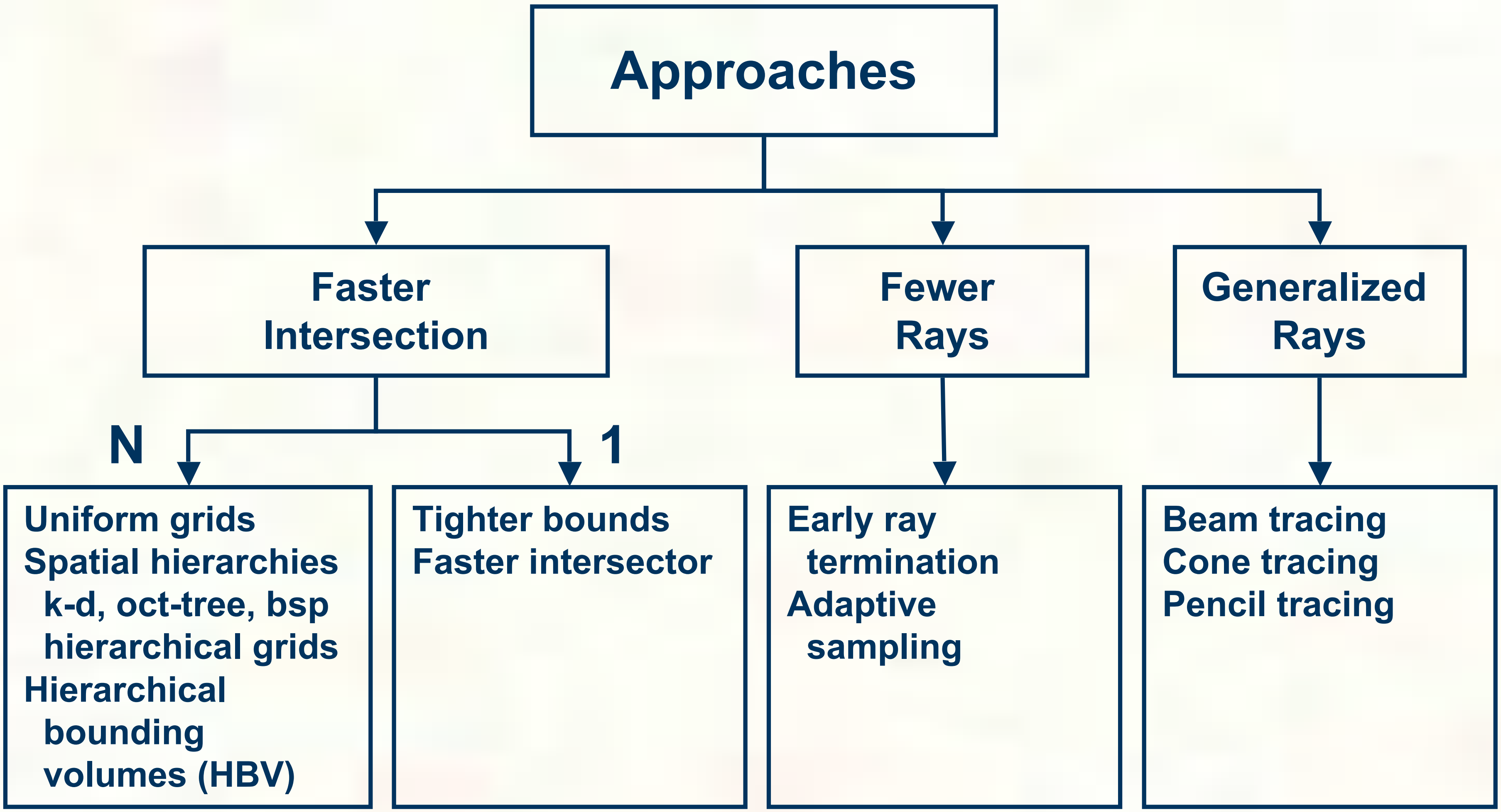
Faster ray-polyhedron intersection

- Let's say you were intersecting a ray with a polyhedron:



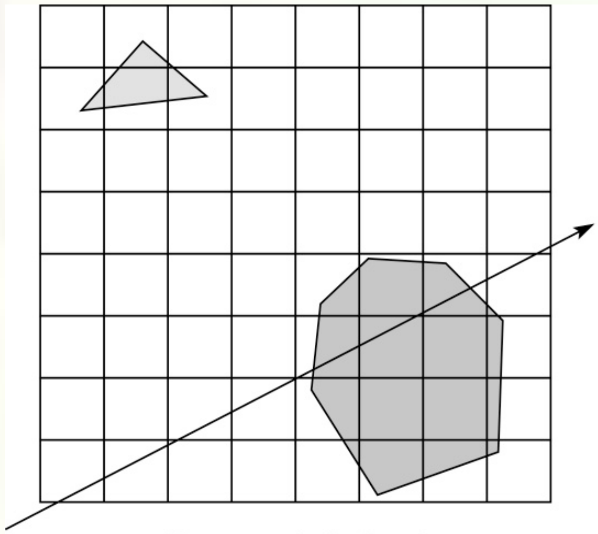
- Straightforward method
  - intersect the ray with each triangle
  - return the intersection with the smallest *t*-value.
- **Q**: How might you speed this up?

Ray Tracing Acceleration Techniques

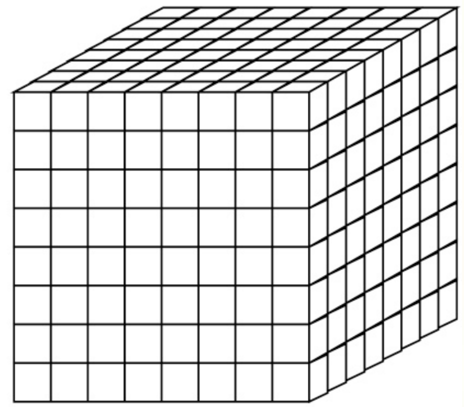


Uniform spatial subdivision

- Another approach is **uniform spatial subdivision**.



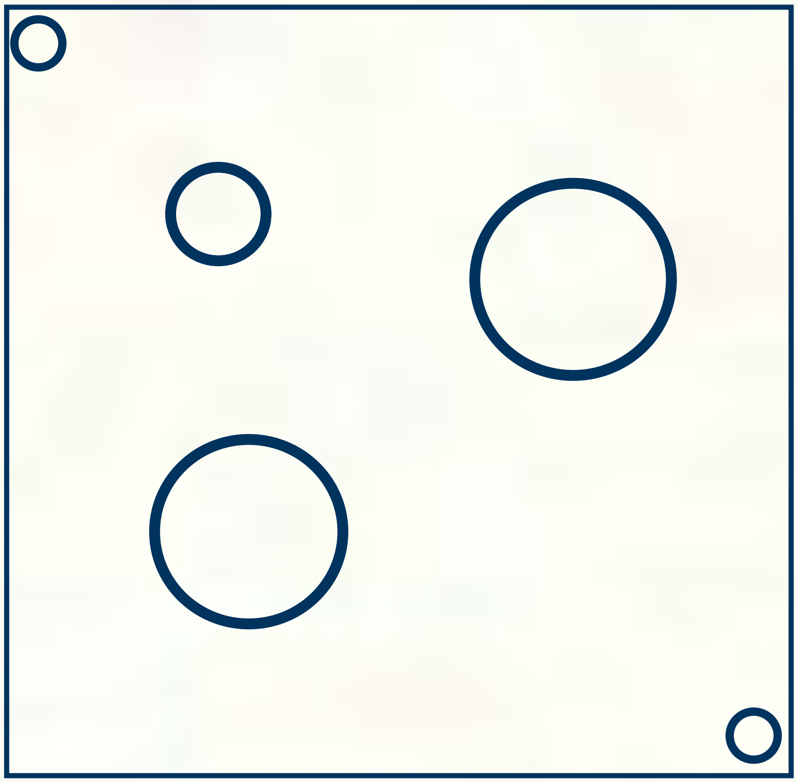
Uniform subdivion in 2D



Uniform subdivion in 3D

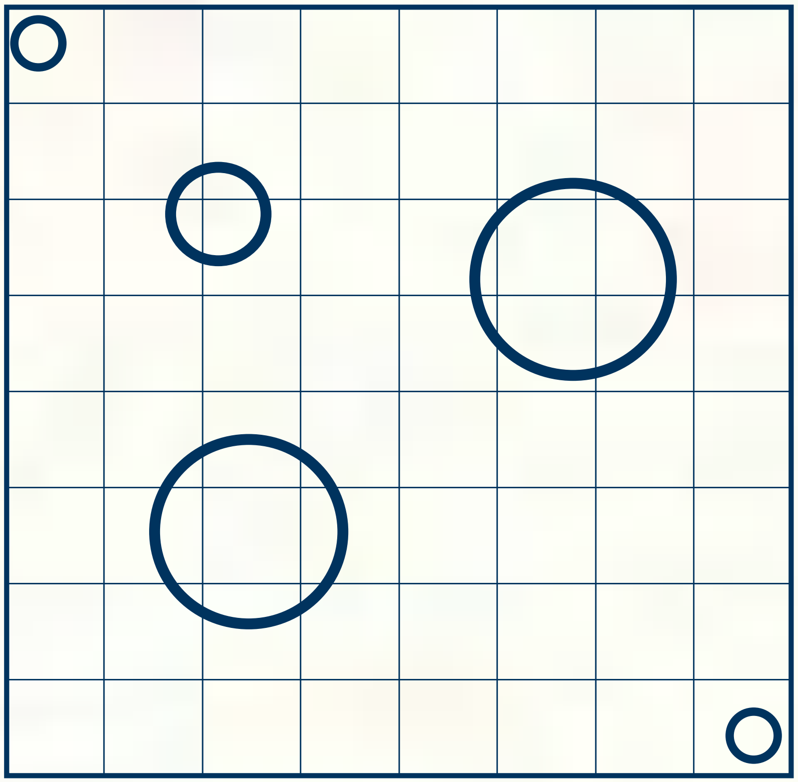
- Idea:
  - Partition space into cells (voxels)
  - Associate each primitive with the cells it overlaps
  - Trace ray through voxel array *using fast incremental arithmetic* to step from cell to cell

Uniform Grids



- Preprocess scene
  - Find bounding box

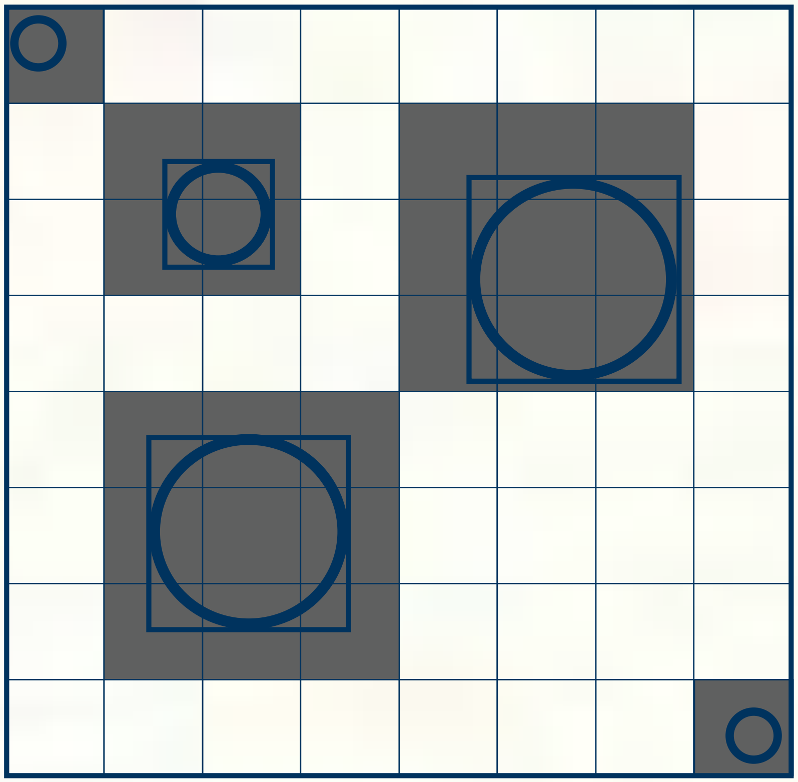
Uniform Grids



- Preprocess scene
  - Find bounding box
  - 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

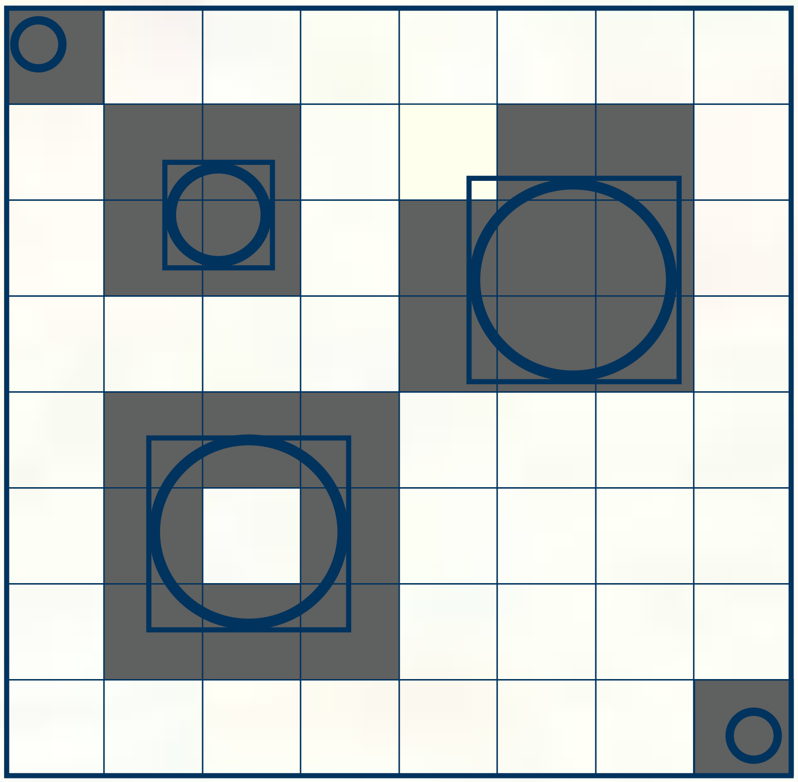


- Preprocess scene
  - Find bounding box
  - Determine resolution

  - Place object in cell, if object overlaps cell

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

Uniform Grids

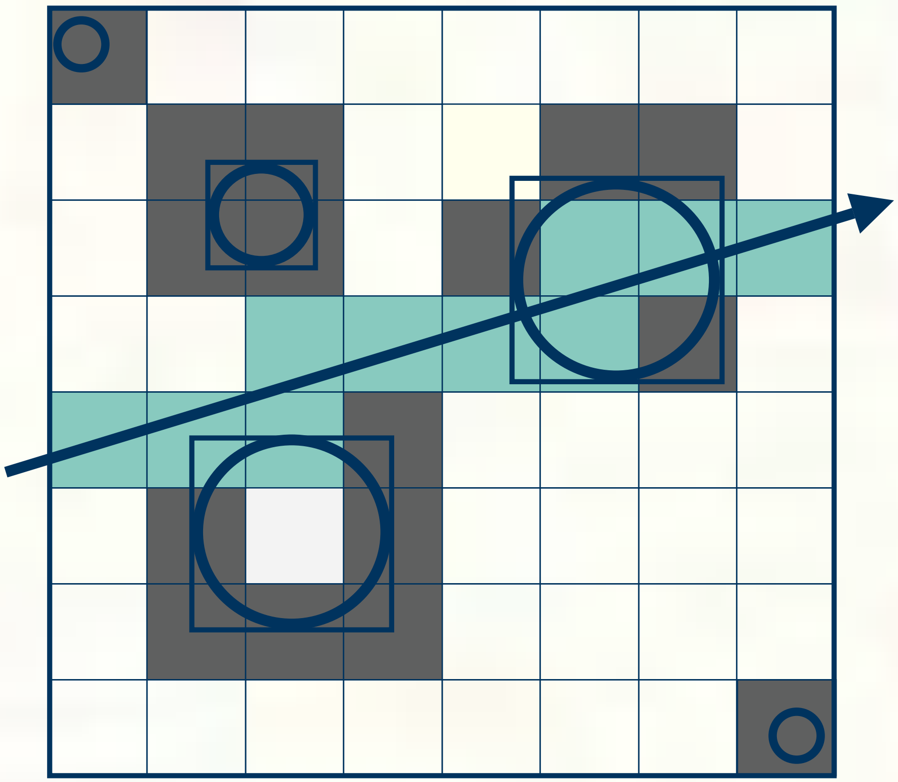


- Preprocess scene
  - Find bounding box
  - Determine resolution

  - Place object in cell, if object overlaps cell
  - Check that object intersects cell

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

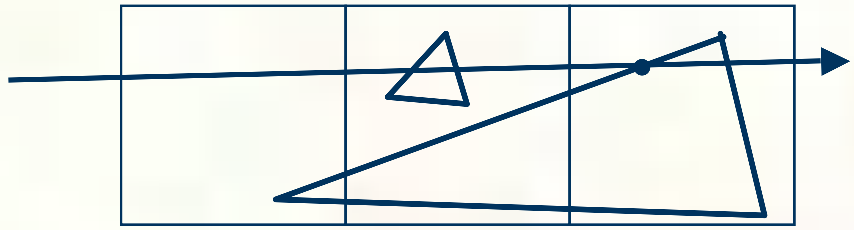
Uniform Grids



- Preprocess scene
- Traverse grid
  - 3D line – 3D-DDA
  - 6-connected line

Caveat: Overlap

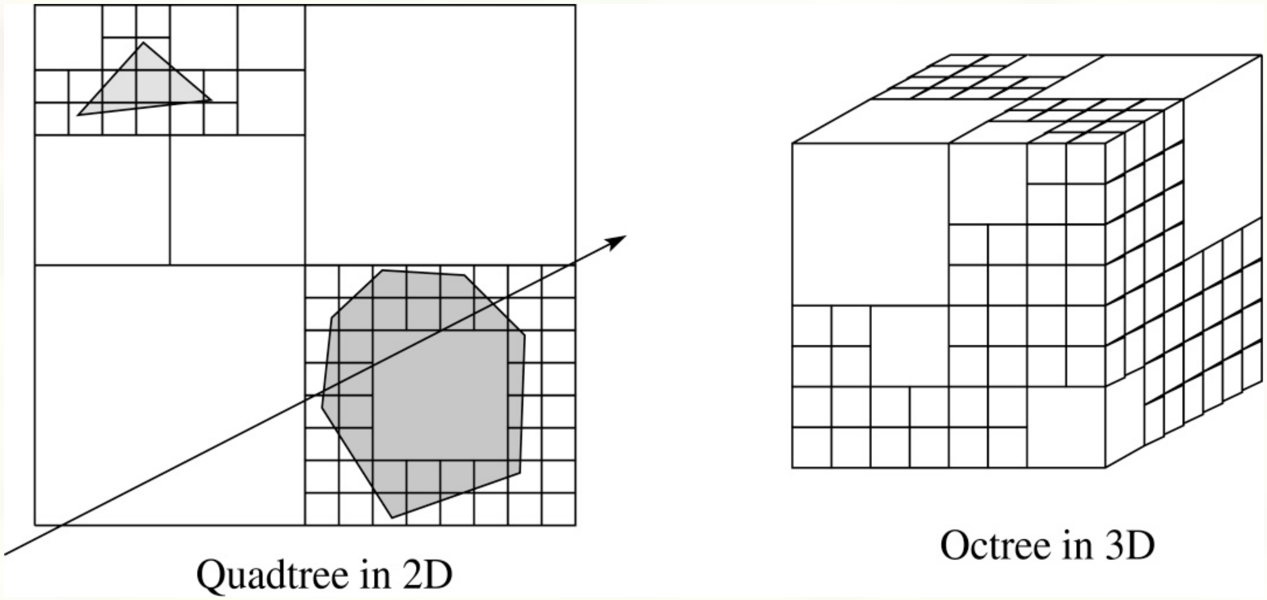
- *Optimize for objects that overlap multiple cells*



- Traverse until tmin(cell) > tmax(ray)
- 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

Non-uniform spatial subdivision

- Still another approach is **non-uniform spatial subdivision**.

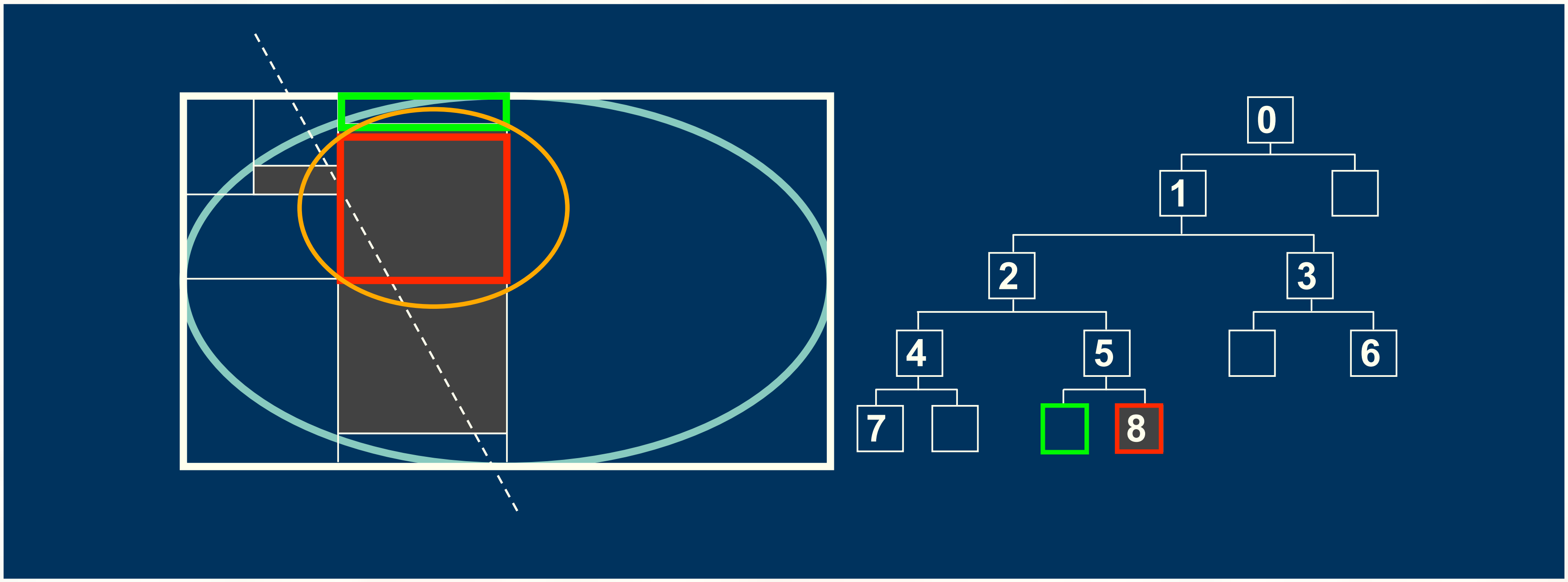


- Other variants include k-d trees and BSP trees.

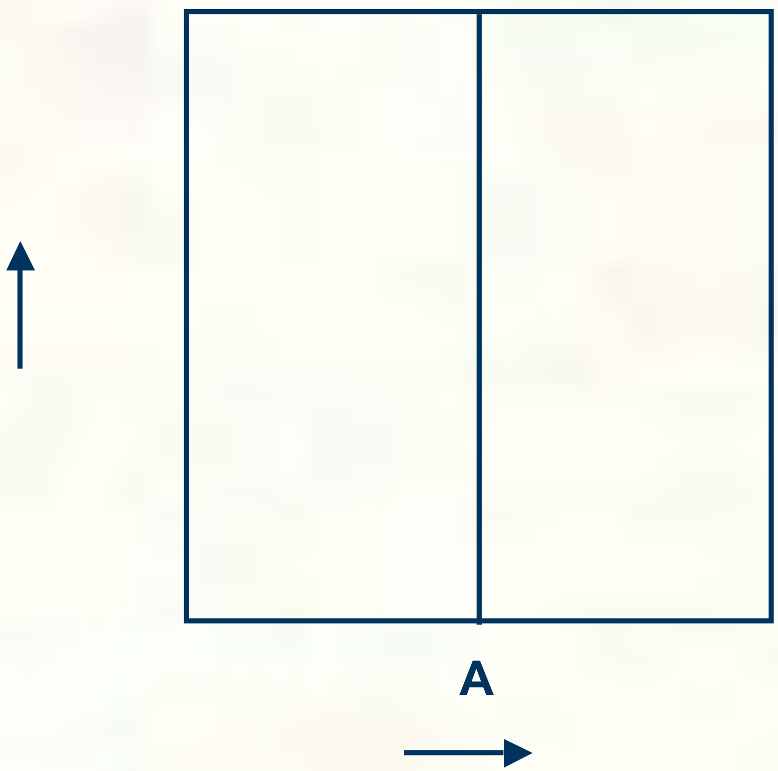
- Various combinations of these ray intersections techniques are also possible. See Glassner and pointers at bottom of project web page for more.

Non-uniform spatial subdivision

- Best approach - k-d trees or perhaps BSP trees
  - More adaptive to actual scene structure
  - BSP vs. k-d tradeoff between speed from simplicity and better adaptability



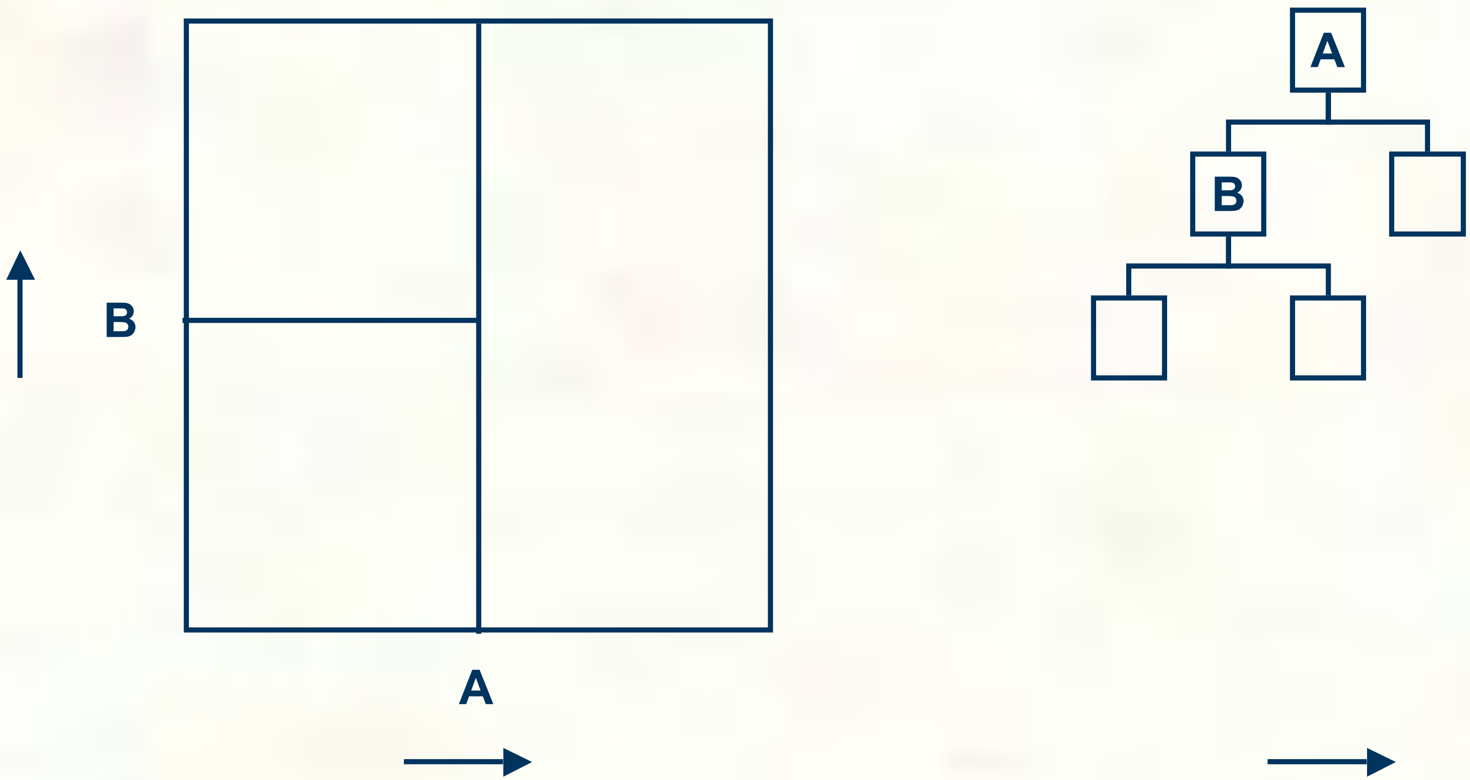
Spatial Hierarchies



Letters correspond to planes (A)

Point Location by recursive search

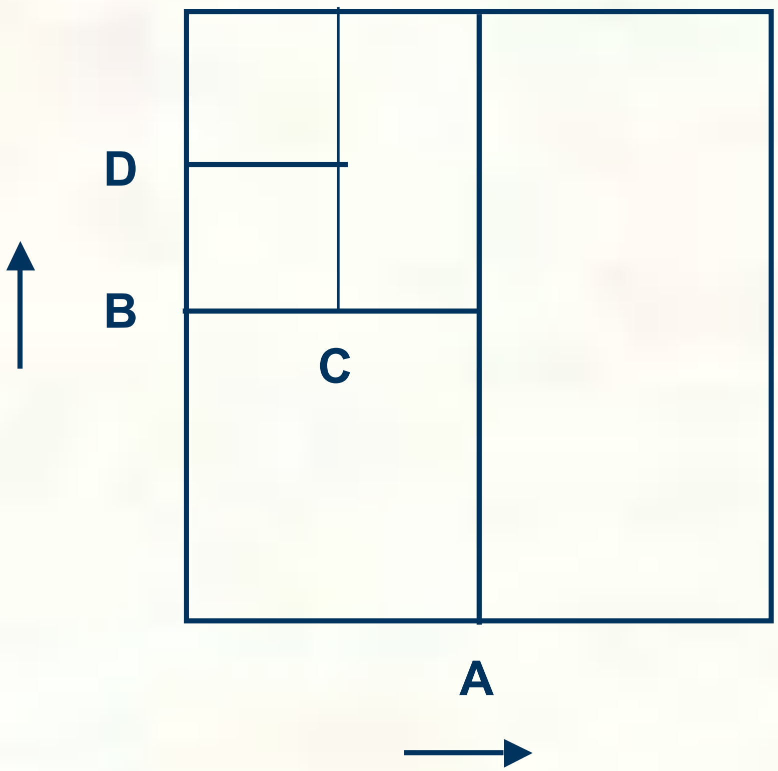
Spatial Hierarchies

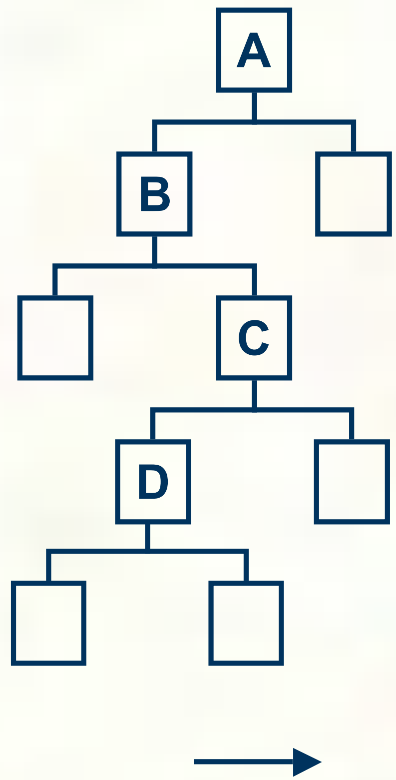


Letters correspond to planes (A, B)

Point Location by recursive search

Spatial Hierarchies

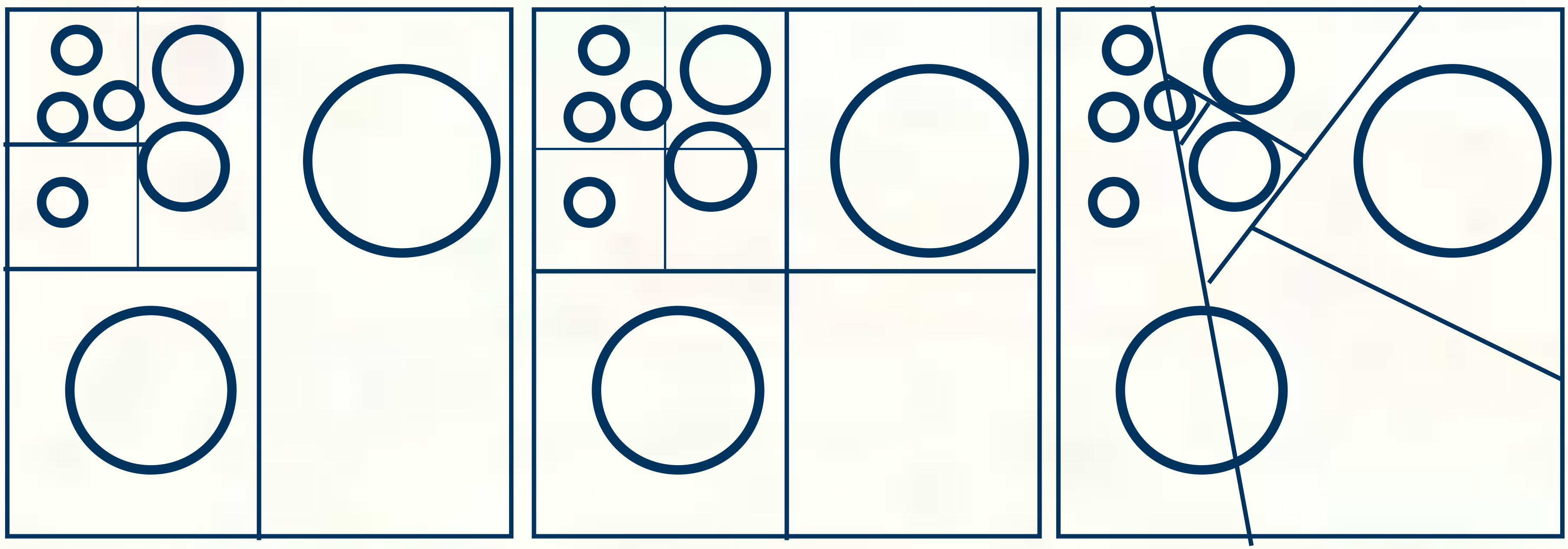




Letters correspond to planes (A, B, C, D)

Point Location by recursive search

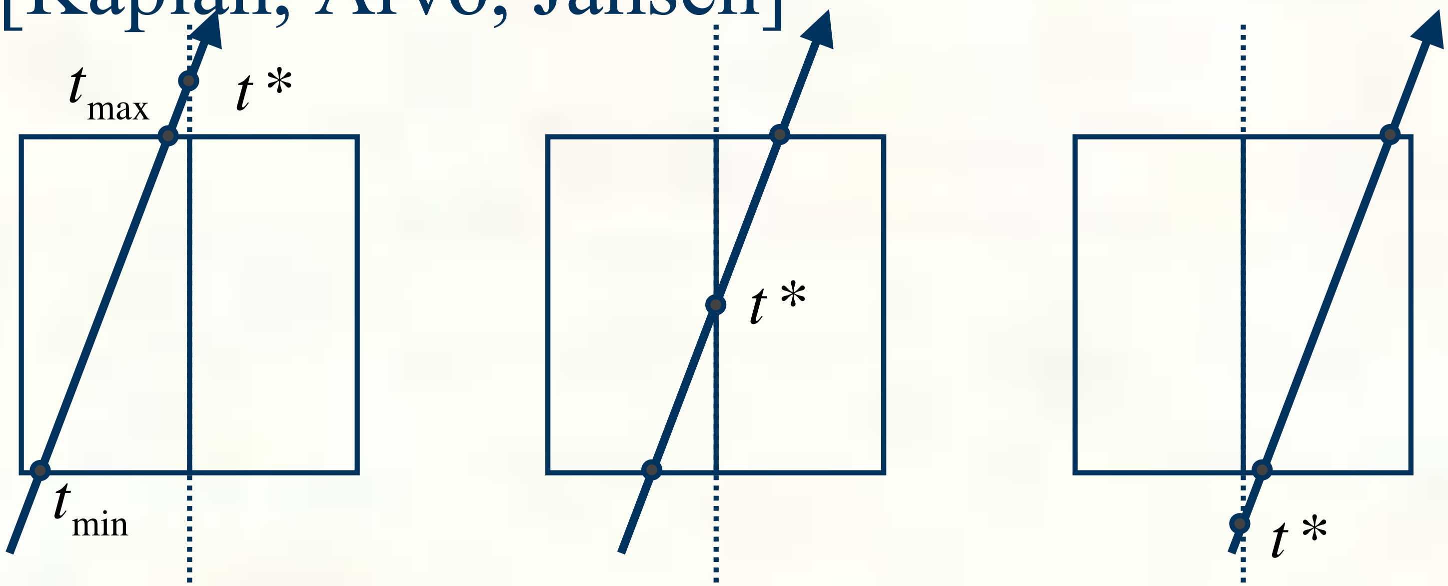
Variations



kd-tree   oct-tree   bsp-tree

Ray Traversal Algorithms

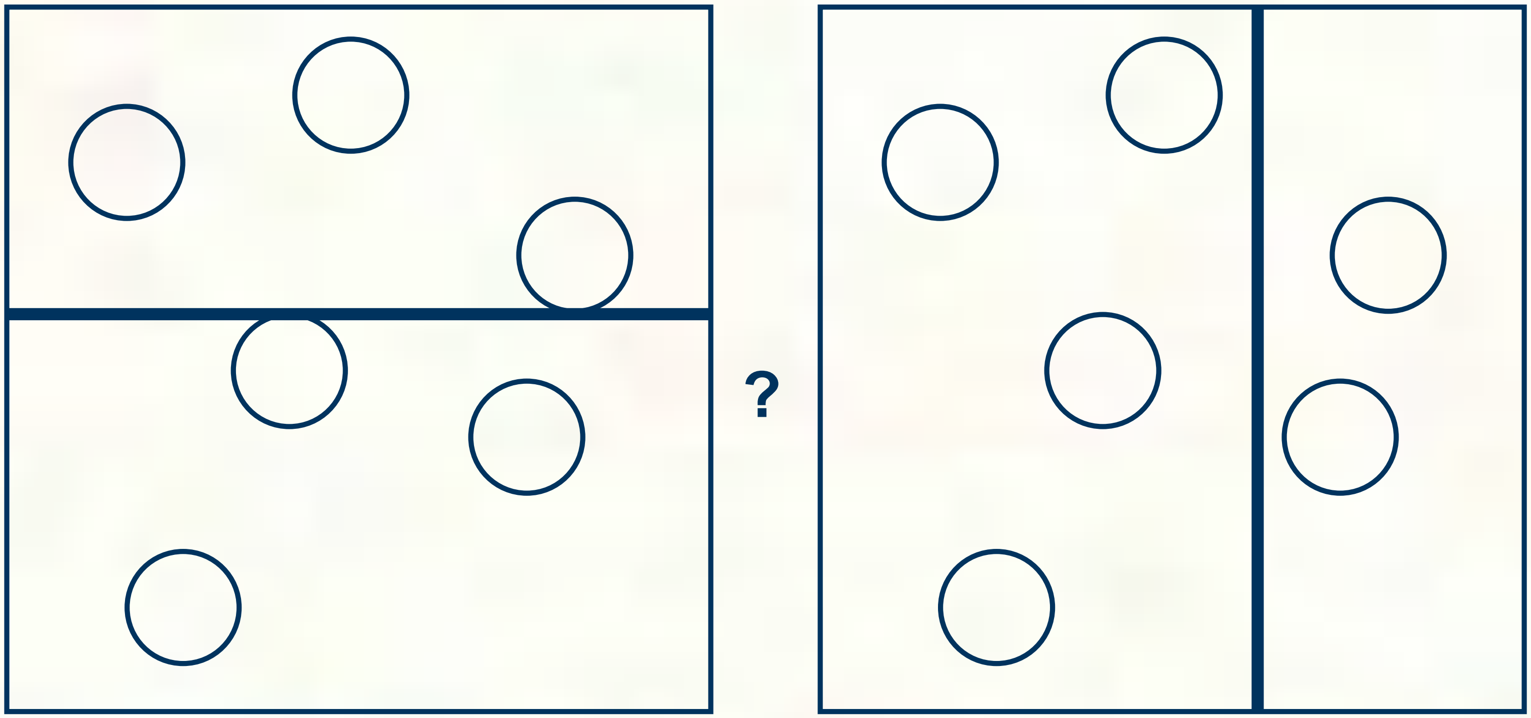
- Recursive inorder traversal
- [Kaplan, Arvo, Jansen]



$t_{max} < t^*$ $\quad$ $t_{min} < t^* < t_{max}$ $\quad$ $t^* < t_{min}$

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

Build Hierarchy Top-Down

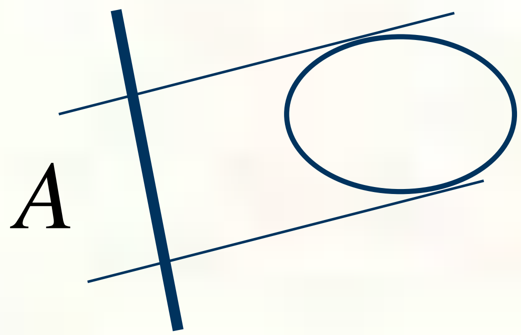


Choose splitting plane

- **Midpoint**
- **Median cut**
- **Surface area heuristic**

Surface Area and Rays

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

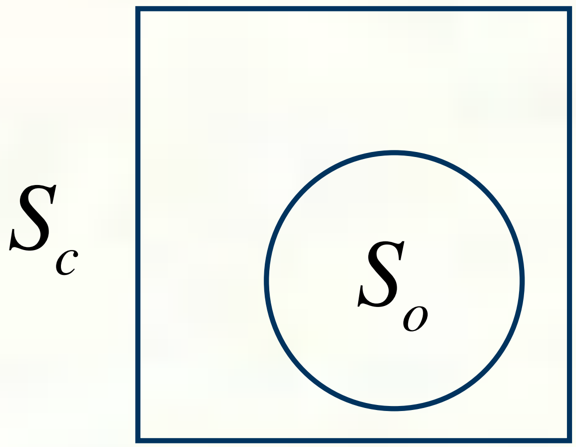


- The total number of rays hitting an object is $4\pi \bar{A}$
- Crofton's Theorem:
  - For a convex body $\bar{A} = \frac{S}{4}$
- For example: sphere

$$S = 4\pi r^2 \qquad \bar{A} = A = \pi r^2$$

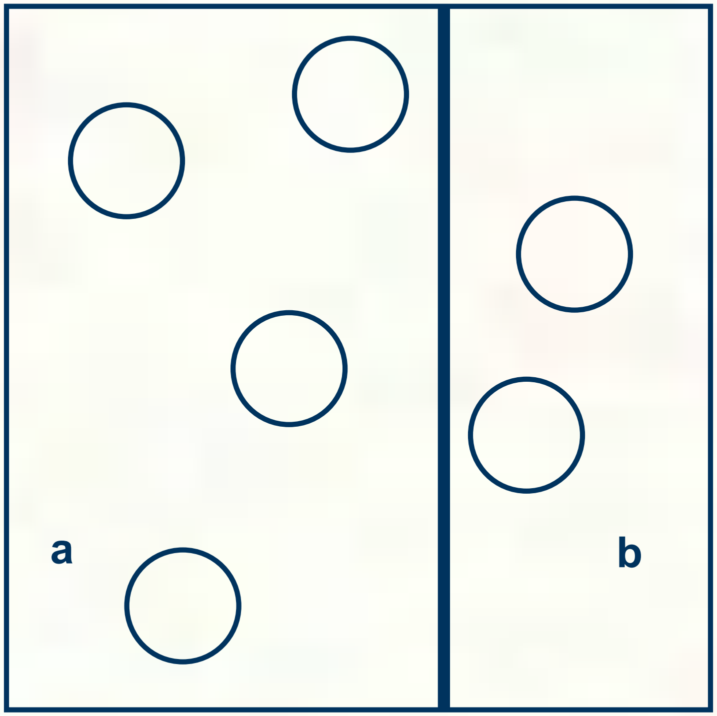
Surface Area and Rays

- The probability of a ray hitting a convex shape
- that is completely inside a convex cell equals



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

Surface Area Heuristic



Intersection time

$$t_i$$

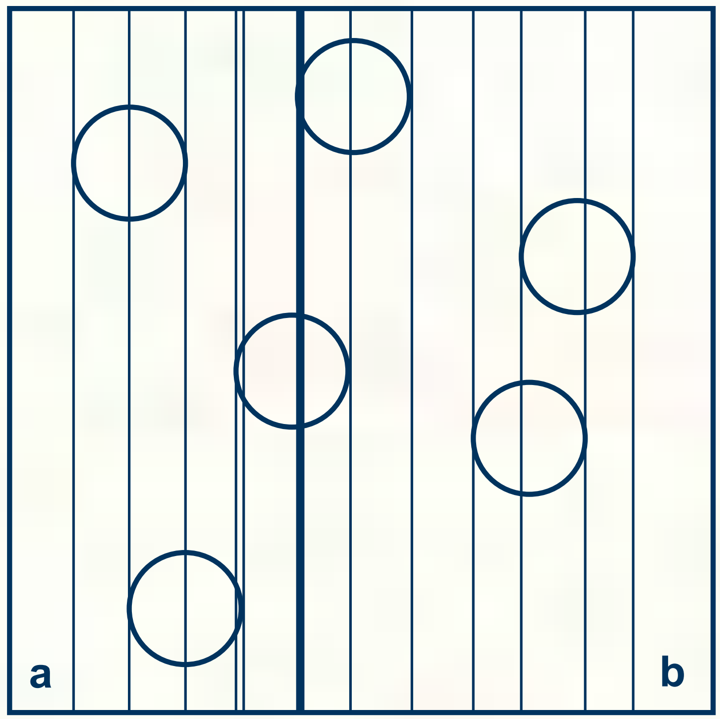
Traversal time

$$t_t$$

$$t_i = 80t_t$$

$$C = t_t + p_a N_a t_i + p_b N_b t_i$$

Surface Area Heuristic

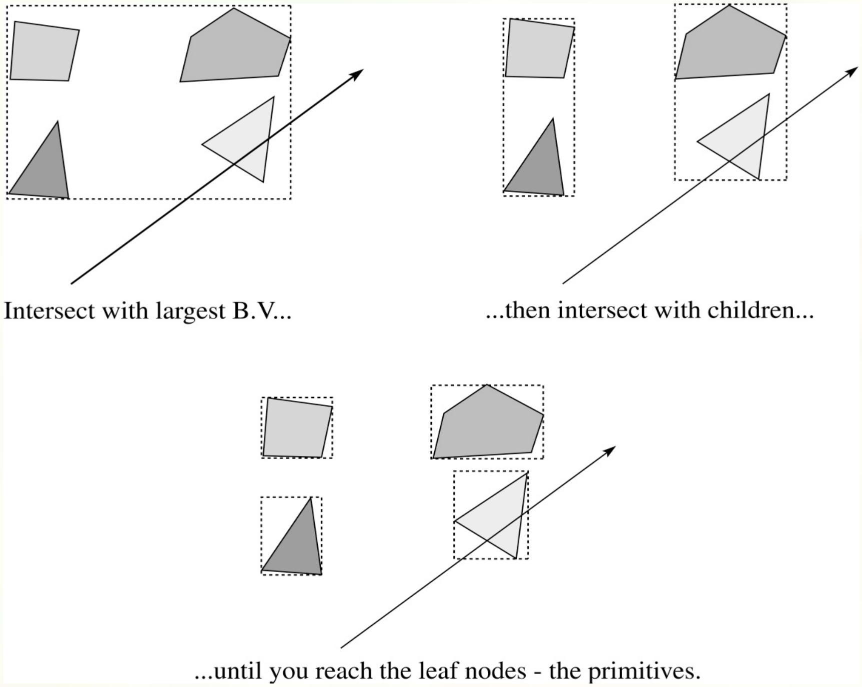


2n splits

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

Hierarchical bounding volumes

- We can generalize the idea of bounding volume acceleration with **hierarchical bounding volumes**.



- Key: build balanced trees with *tight bounding volumes*.

  Many different kinds of bounding volumes.
  Note that bounding volumes can overlap.