Lecture 12:

Ray Tracing

Computer Graphics 2025

Fuzhou University - Computer Science

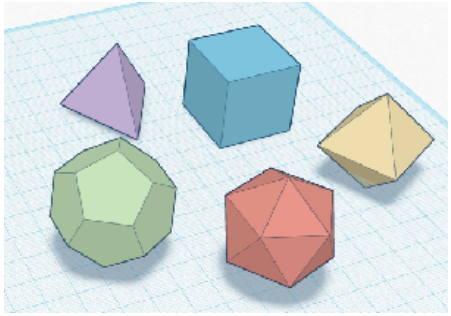
Photorealistic Rendering — Our Goal

照片级真实感渲染

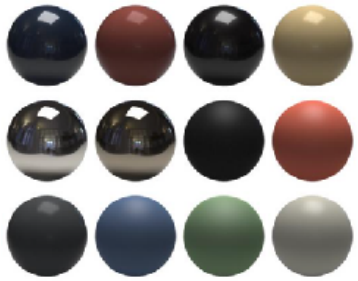
输入各要素

camera

geometry

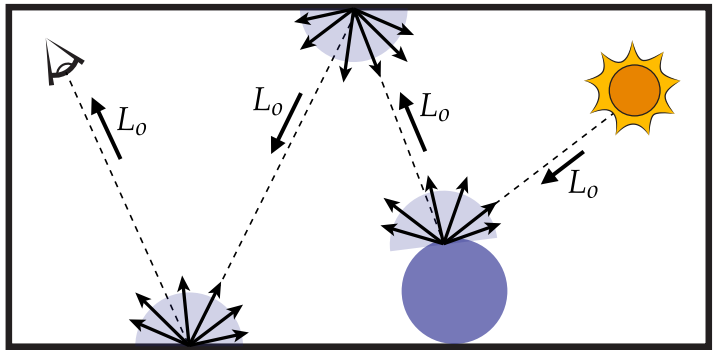


materials



lights

("scene")



Ray Tracer

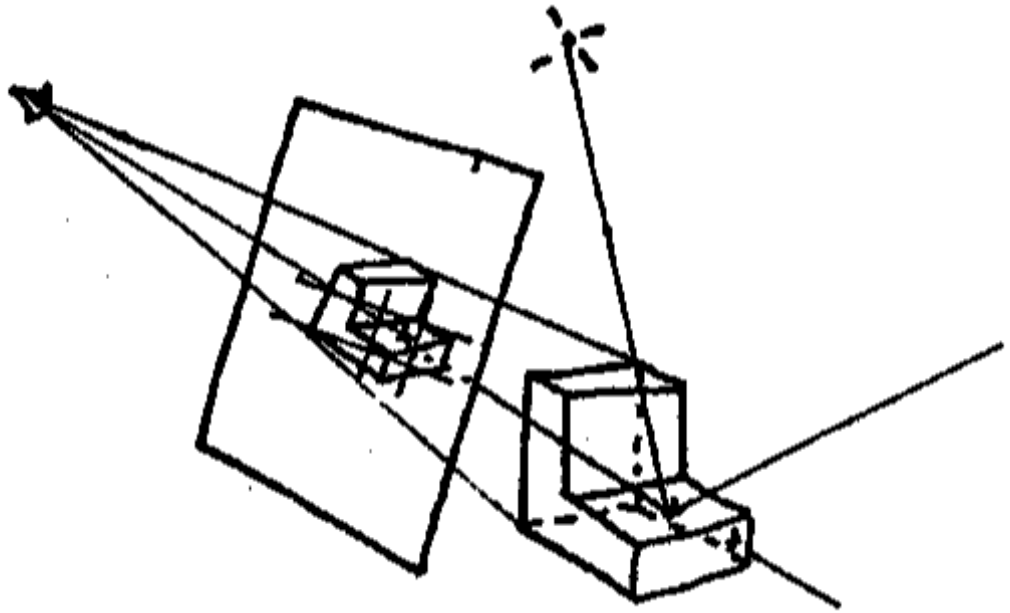
输出图像

Basic Ray-Tracing Algorithm

Ray Casting 光线投射

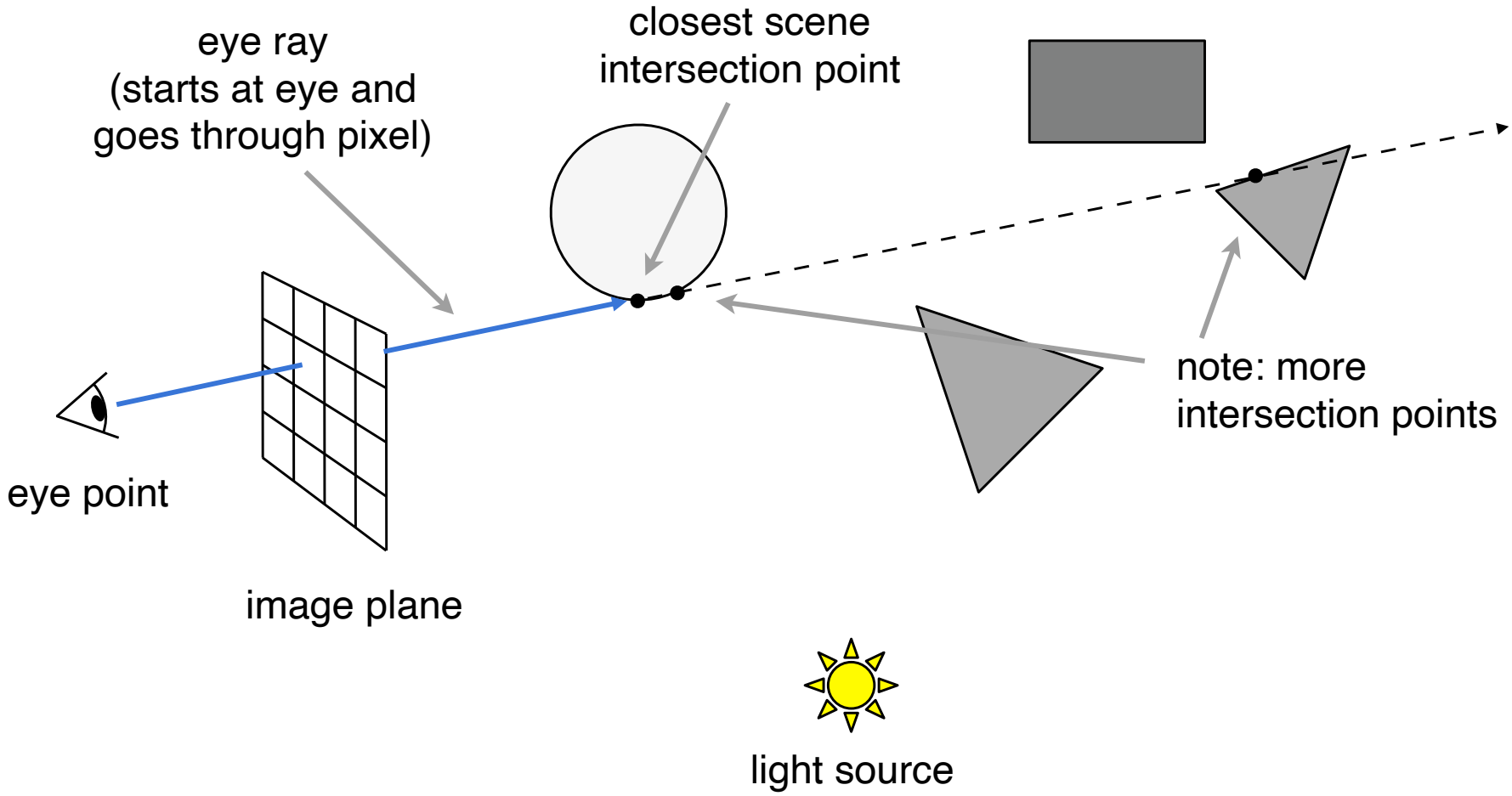
Appel 1968 - Ray casting

1. Generate an image by casting one ray per pixel
2. Check for shadows by sending a ray to the light



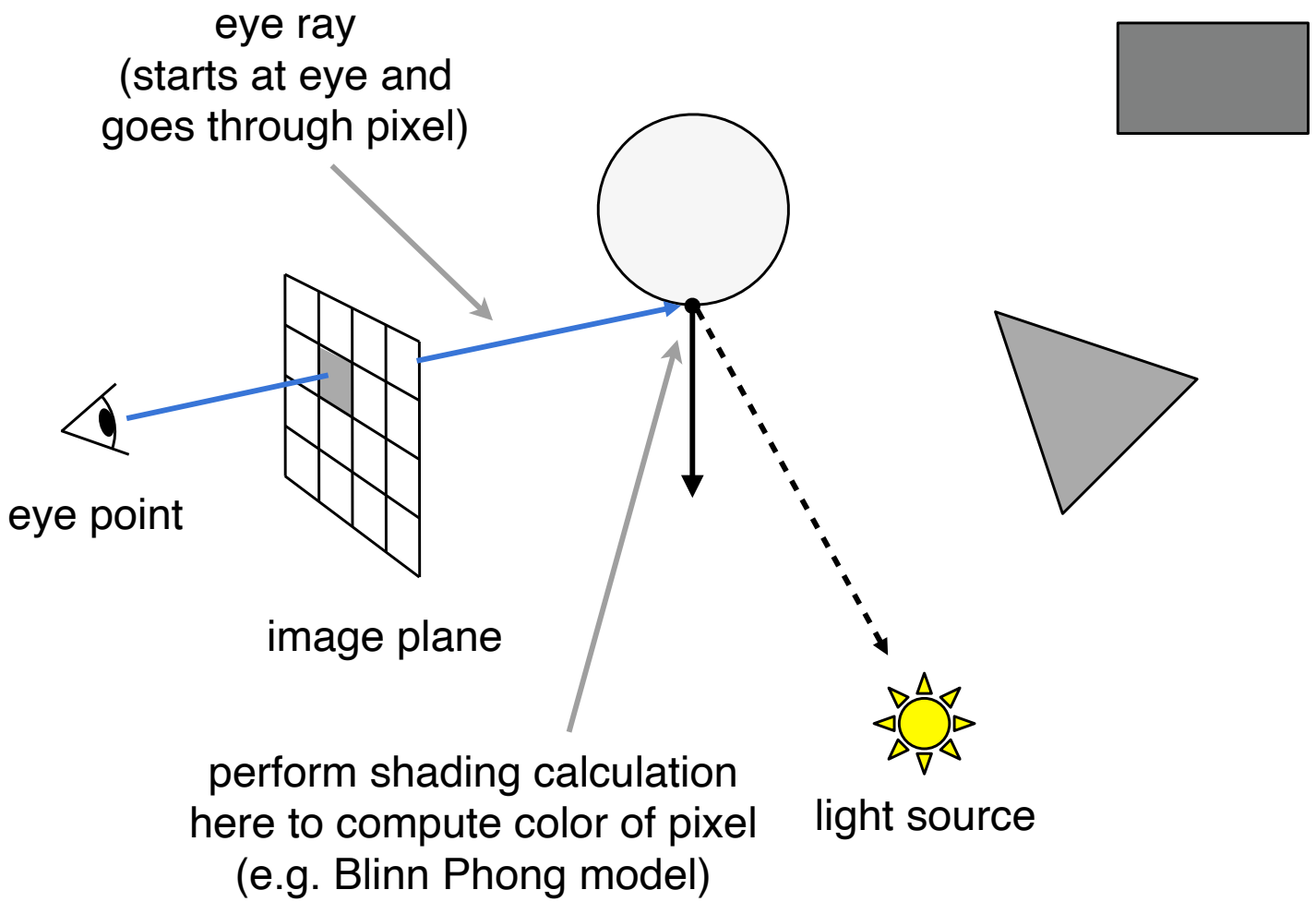
Ray Casting - Generating Eye Rays

Pinhole Camera Model



Ray Casting - Shading Pixels (Local Only)

Pinhole Camera Model

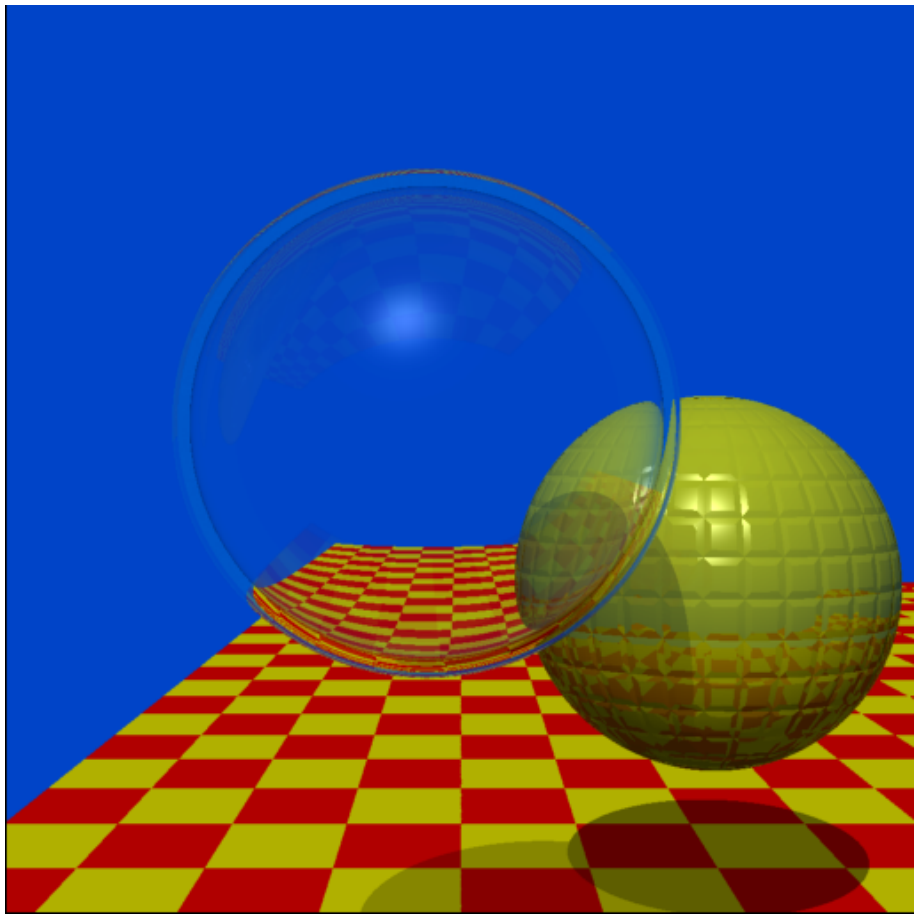


Recursive Ray Tracing 光线追踪（光追）

"An improved Illumination model for shaded display"
T. Whitted, CACM 1980

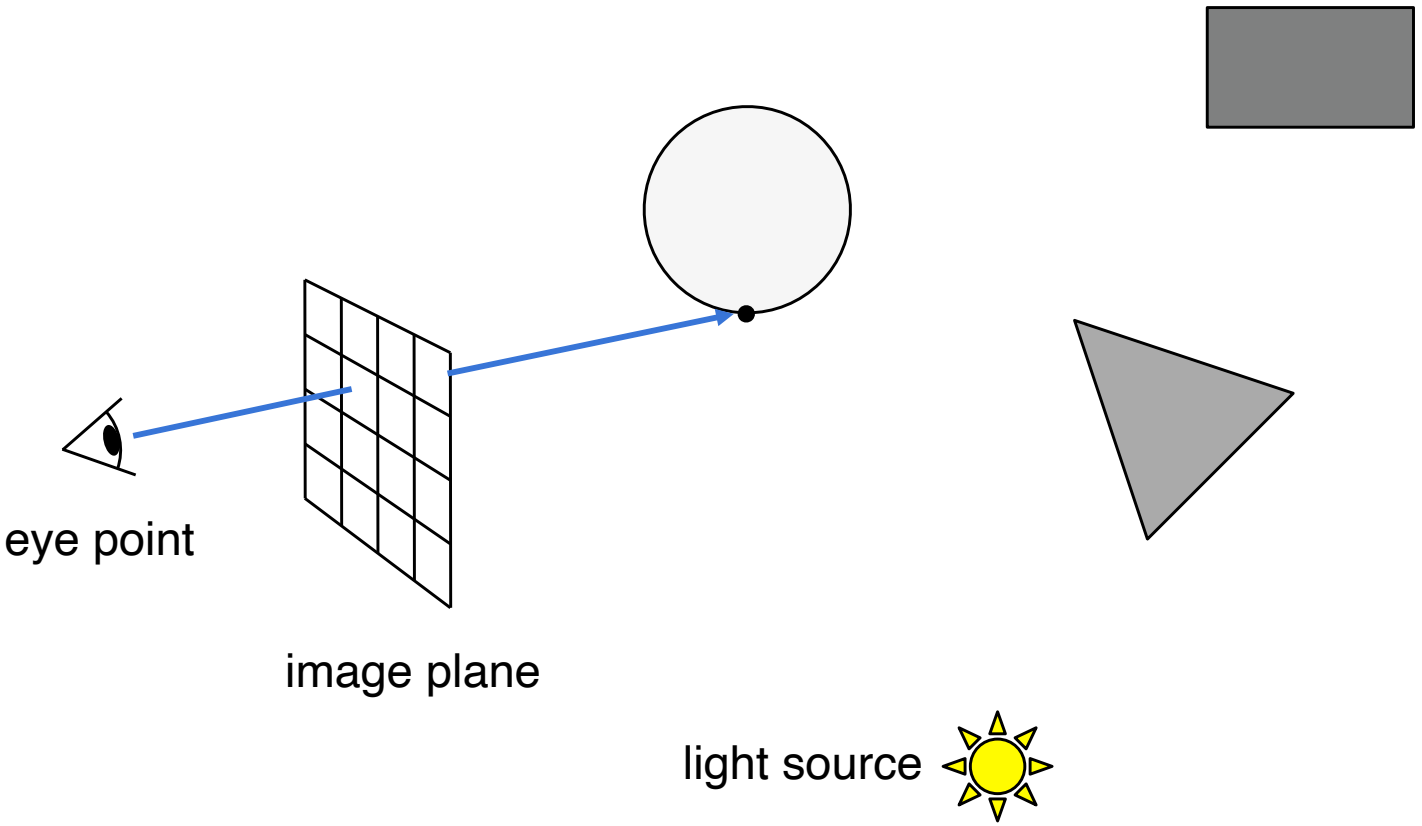
Time:

- VAX 11/780 (1979) 74m
- PC (2006) 6s
- GPU (2012) 1/30s

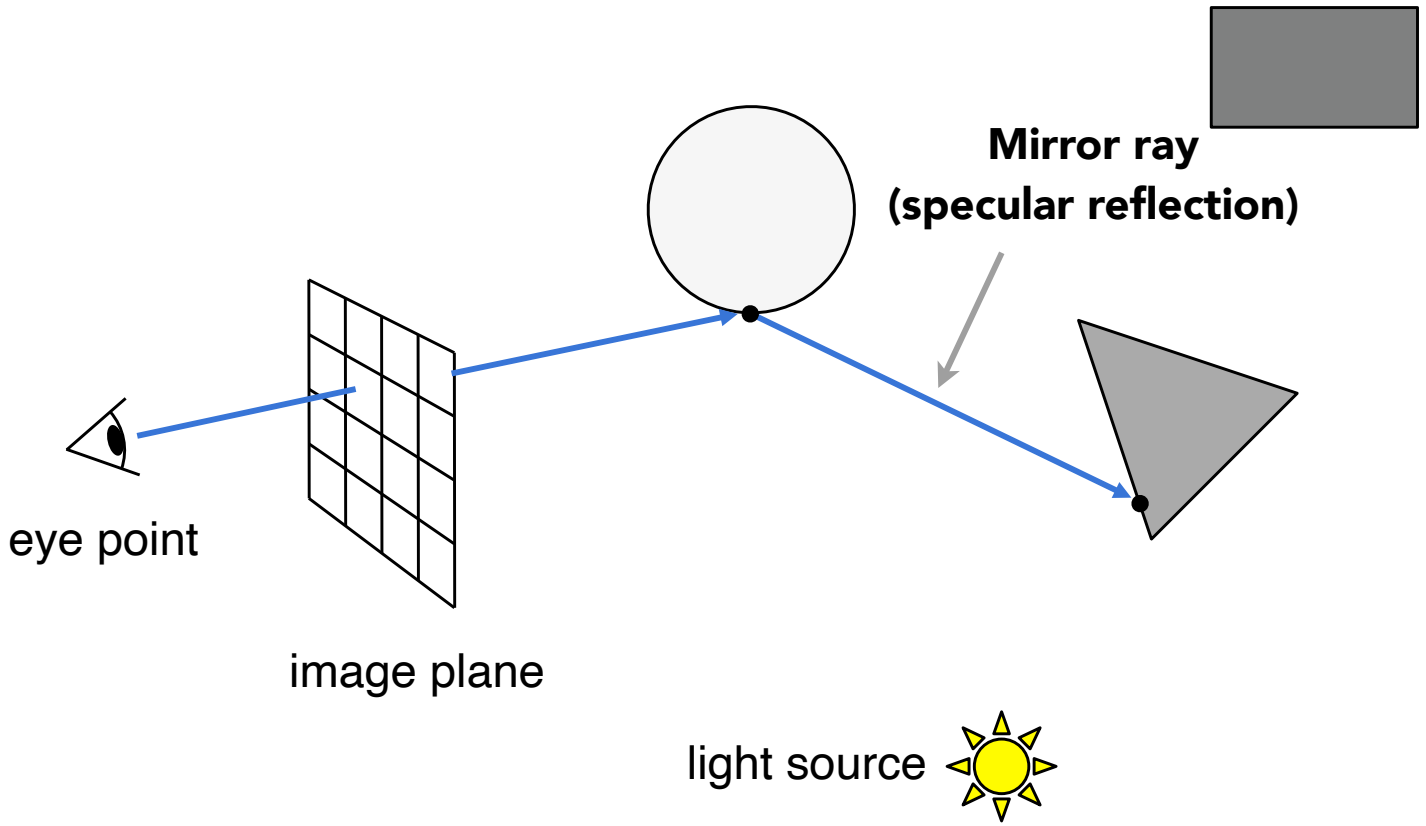


Spheres and Checkerboard, T. Whitted, 1979

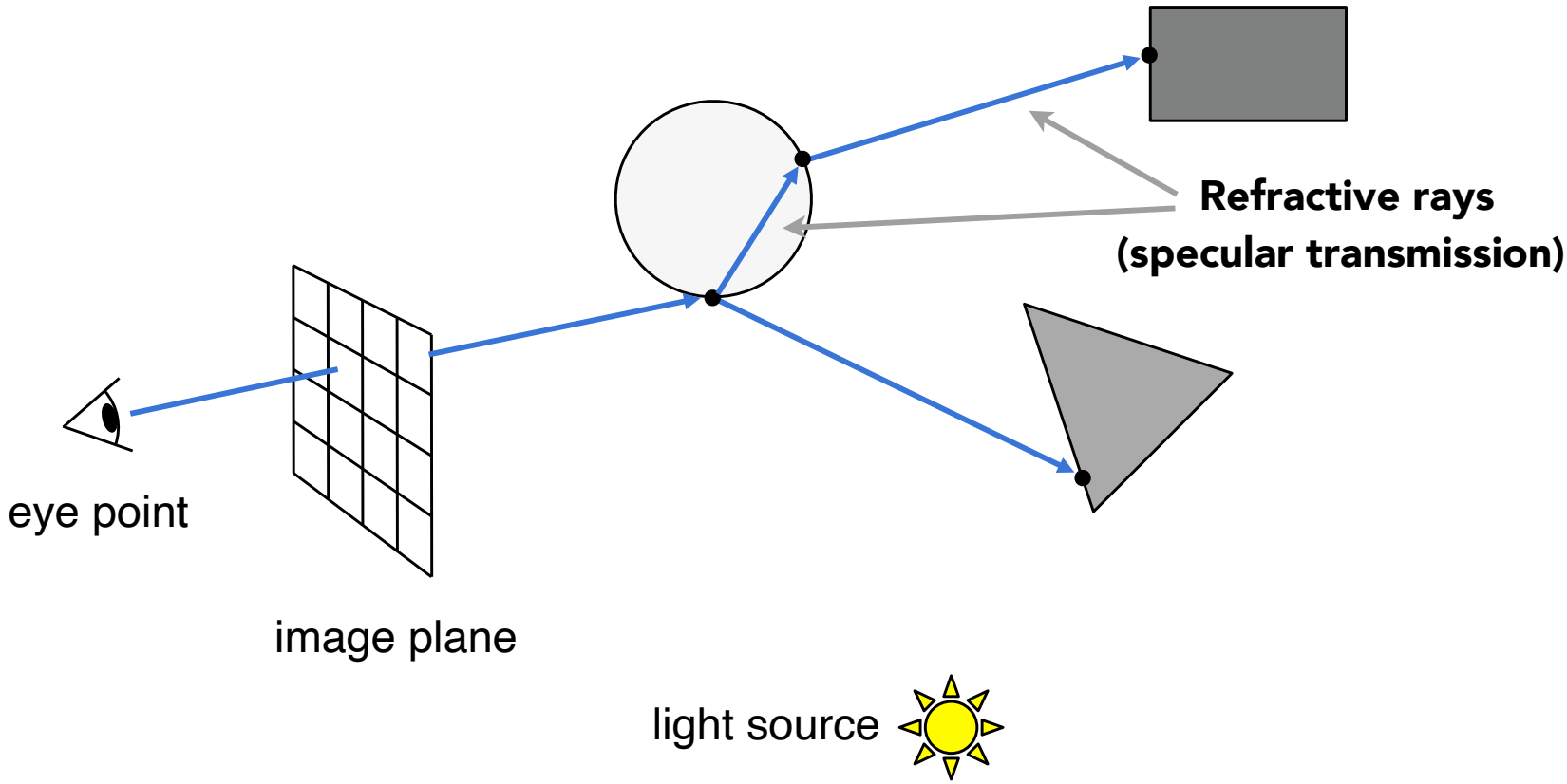
Recursive Ray Tracing



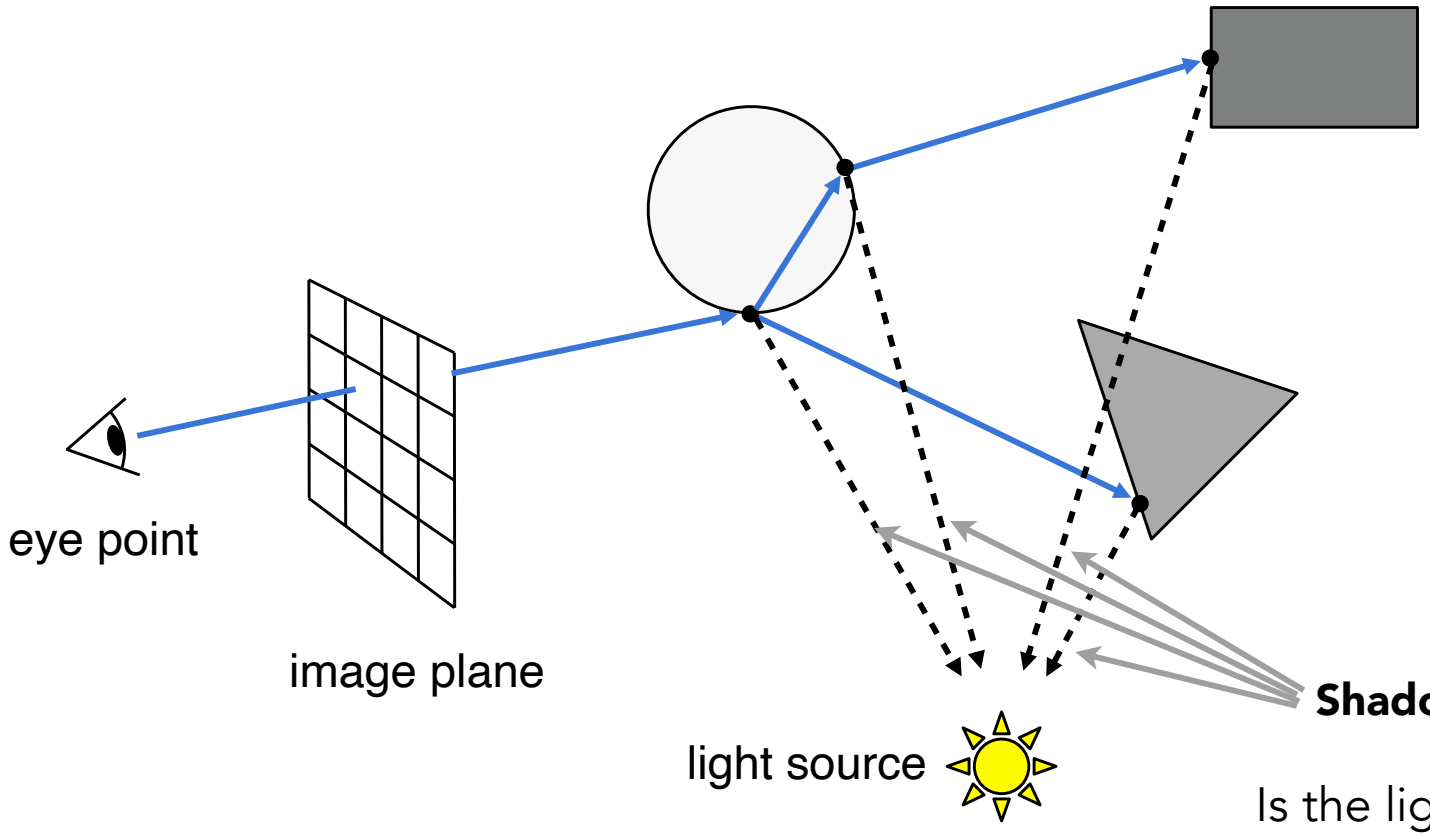
Recursive Ray Tracing



Recursive Ray Tracing

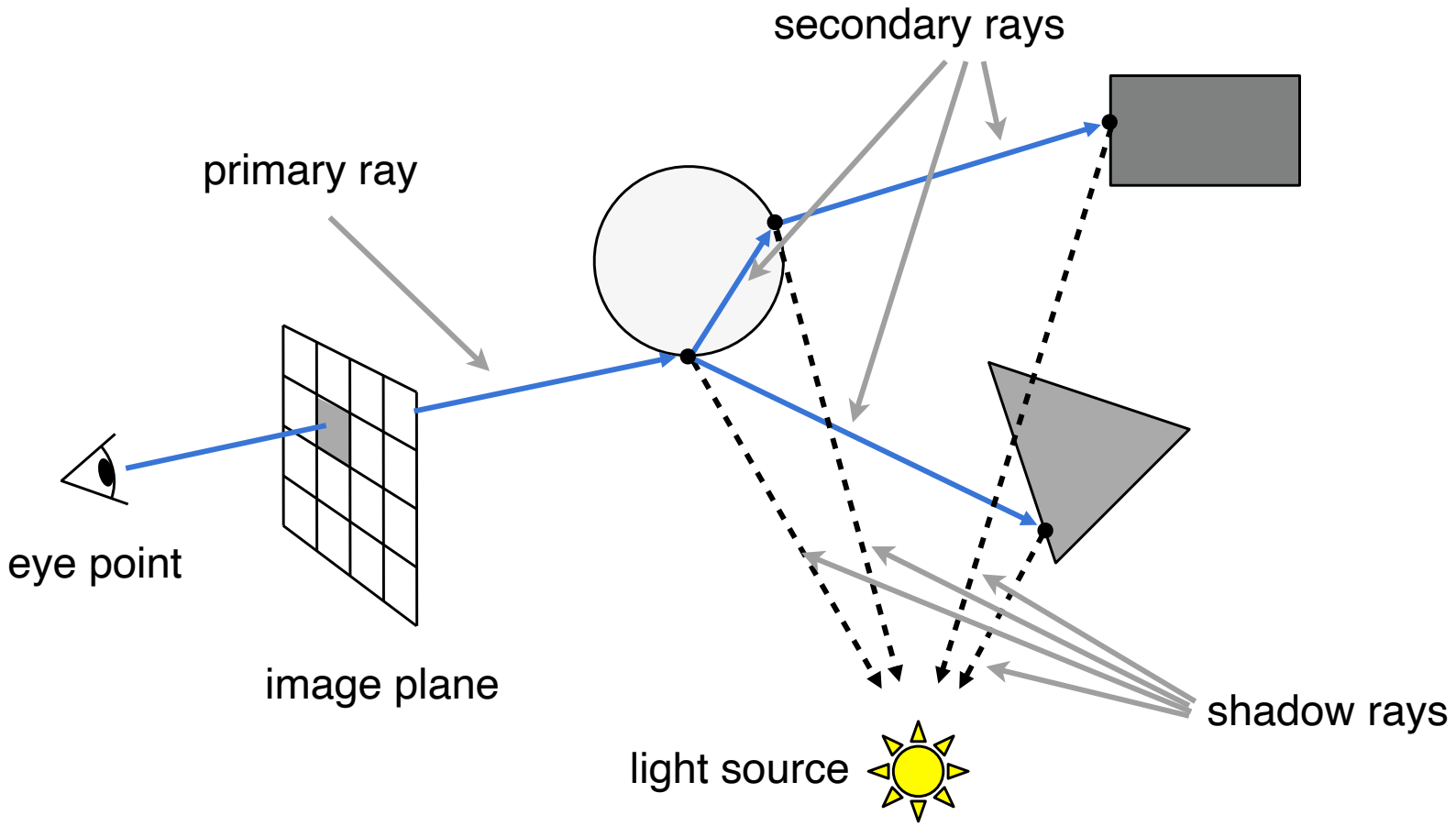


Recursive Ray Tracing



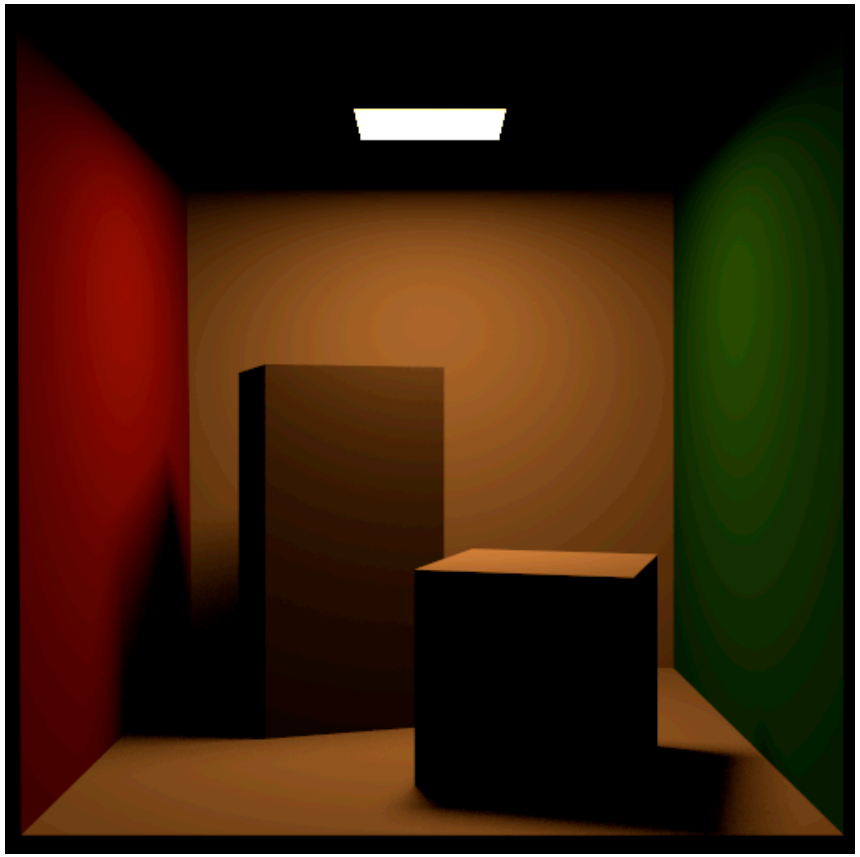
Is the light source visible from the surface point?

Recursive Ray Tracing



- Trace secondary rays recursively until hit a non-specular surface (or max desired levels of recursion)
- At each hit point, trace shadow rays to test light visibility (no contribution if blocked)
- Final pixel color is weighted sum of contributions along rays, as shown

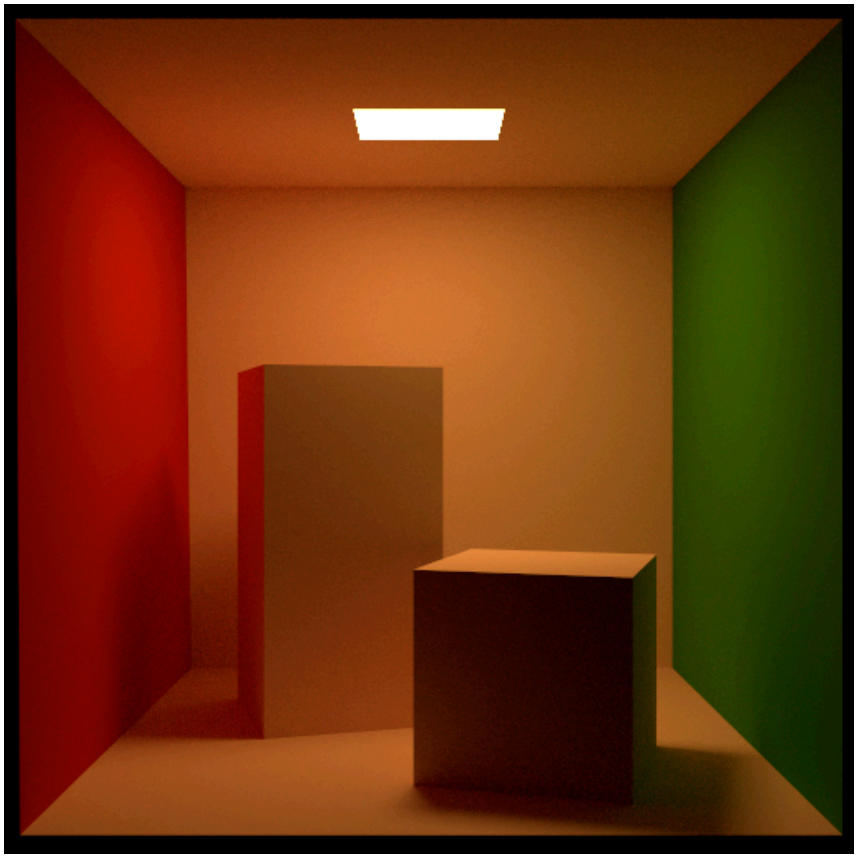
Ray Tracing: Examples



Direct/Local Illumination

直接/局部 光照

Ray Casting



Global Illumination

全局光照

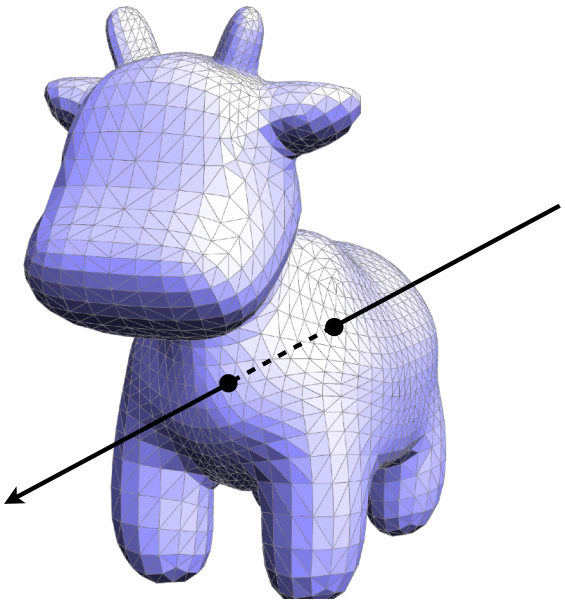
Ray Tracing

Ray-Surface Intersection

Ray Intersection With Triangle Mesh

Why?

- Rendering: visibility, shadows, lighting …
- Geometry: inside/outside test



How to compute?

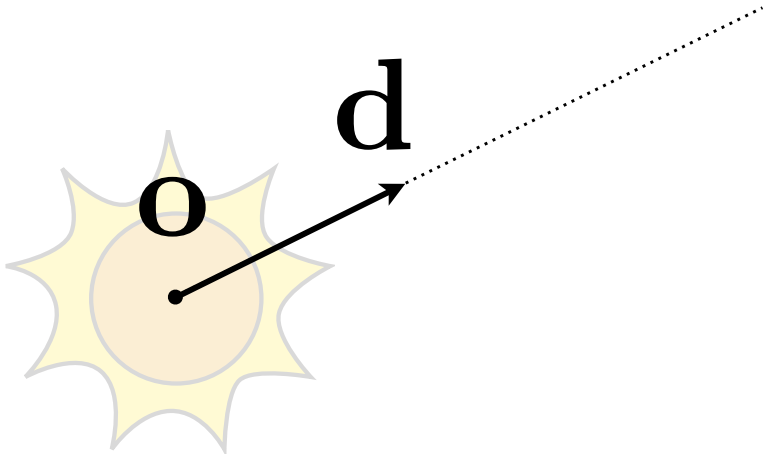
Let's break this down:

- Simple idea: just intersect ray with each triangle
- Simple, but slow (study acceleration later)
- Note: can have 0, 1 or multiple intersections

Ray Equation

Ray is defined by its origin and a direction vector

Example:



Ray equation:

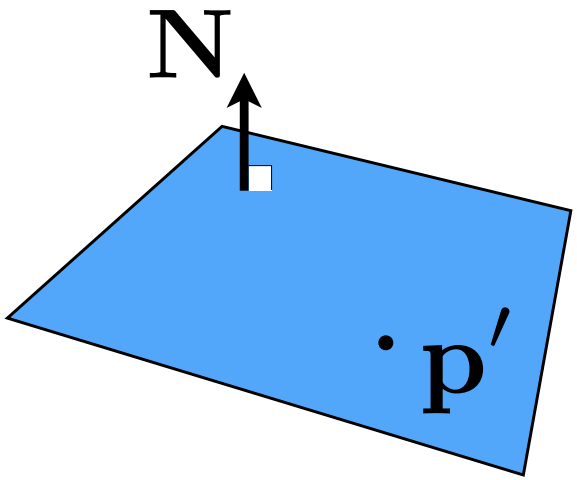
$$\mathbf{r}(t) = \mathbf{o} + t\mathbf{d} \qquad 0 \leq t < \infty$$

point along ray "time" origin unit direction

Plane Equation

Plane is defined by normal vector and a point on plane

Example:



Plane Equation:

$$\mathbf{p} : (\mathbf{p} - \mathbf{p}') \cdot \mathbf{N} = 0$$

all points on plane — point on plane — normal vector

$$ax + by + cz + d = 0$$

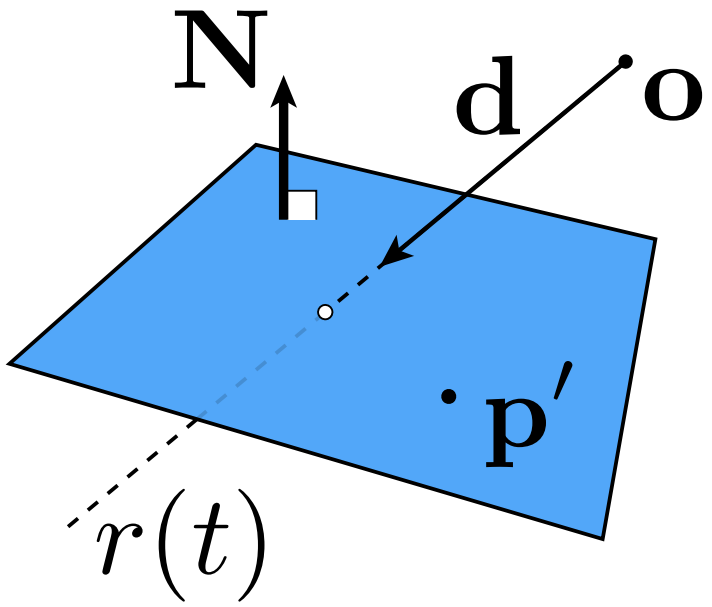
Ray Intersection With Plane

Ray equation:

$$\mathbf{r}(t) = \mathbf{o} + t\,\mathbf{d},\ 0 \le t < \infty$$

Plane equation:

$$\mathbf{p} : (\mathbf{p} - \mathbf{p}') \cdot \mathbf{N} = 0$$



Solve for intersection

Set $\mathbf{p} = \mathbf{r}(t)$ and solve for t

$$(\mathbf{p} - \mathbf{p}') \cdot \mathbf{N} = (\mathbf{o} + t\,\mathbf{d} - \mathbf{p}') \cdot \mathbf{N} = 0$$

$$t = \frac{(\mathbf{p}' - \mathbf{o}) \cdot \mathbf{N}}{\mathbf{d} \cdot \mathbf{N}}$$

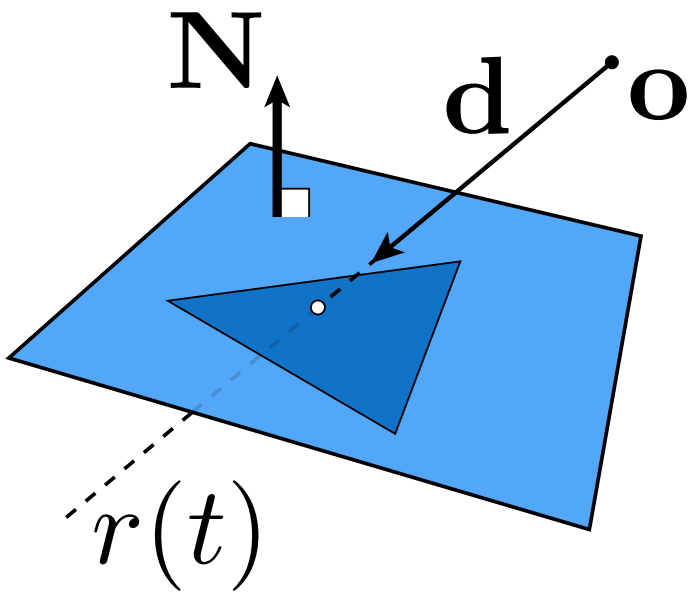
Check: $0 \le t < \infty$

Ray Intersection With Triangle

Triangle is in a plane

- Ray-plane intersection
- Test if hit point is inside tria

Many ways to optimize…



Can Optimize: e.g. Möller Trumbore Algorithm

$$\vec{\mathbf{O}} + t\vec{\mathbf{D}} = (1 - b_1 - b_2)\vec{\mathbf{P}}_0 + b_1\vec{\mathbf{P}}_1 + b_2\vec{\mathbf{P}}_2$$

Where:

$$\begin{bmatrix} t \\ b_1 \\ b_2 \end{bmatrix} = \frac{1}{\vec{\mathbf{S}}_1 \bullet \vec{\mathbf{E}}_1} \begin{bmatrix} \vec{\mathbf{S}}_2 \bullet \vec{\mathbf{E}}_2 \\ \vec{\mathbf{S}}_1 \bullet \vec{\mathbf{S}} \\ \vec{\mathbf{S}}_2 \bullet \vec{\mathbf{D}} \end{bmatrix}$$

$$\vec{\mathbf{E}}_1 = \vec{\mathbf{P}}_1 - \vec{\mathbf{P}}_0$$

$$\vec{\mathbf{E}}_2 = \vec{\mathbf{P}}_2 - \vec{\mathbf{P}}_0$$

$$\vec{\mathbf{S}} = \vec{\mathbf{O}} - \vec{\mathbf{P}}_0$$

$$\vec{\mathbf{S}}_1 = \vec{\mathbf{D}} \times \vec{\mathbf{E}}_2$$

$$\vec{\mathbf{S}}_2 = \vec{\mathbf{S}} \times \vec{\mathbf{E}}_1$$

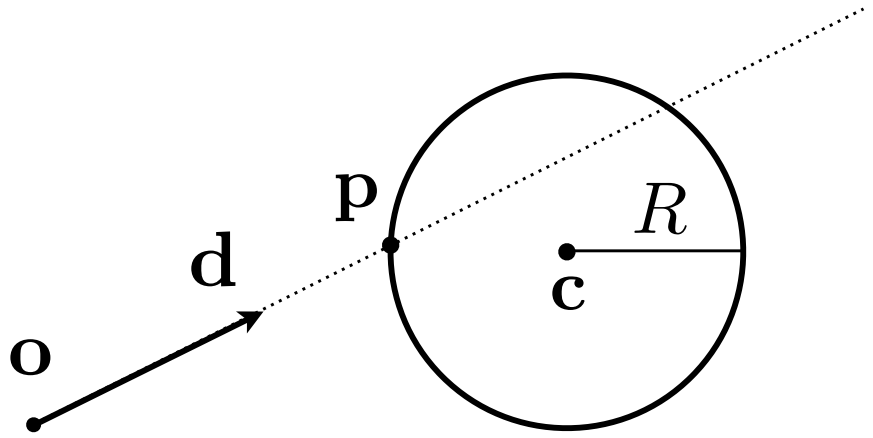
Cost = (1 div, 27 mul, 17 add)

https://en.wikipedia.org/wiki/M%C3%B6ller%E2%80%93Trumbore_intersection_algorithm

Ray Intersection With Sphere

Ray: $\mathbf{r}(t) = \mathbf{o} + t\,\mathbf{d},\ 0 \le t < \infty$

Sphere: $\mathbf{p} : (\mathbf{p} - \mathbf{c})^2 - R^2 = 0$



Solve for intersection:

$$(\mathbf{o} + t\,\mathbf{d} - \mathbf{c})^2 - R^2 = 0$$

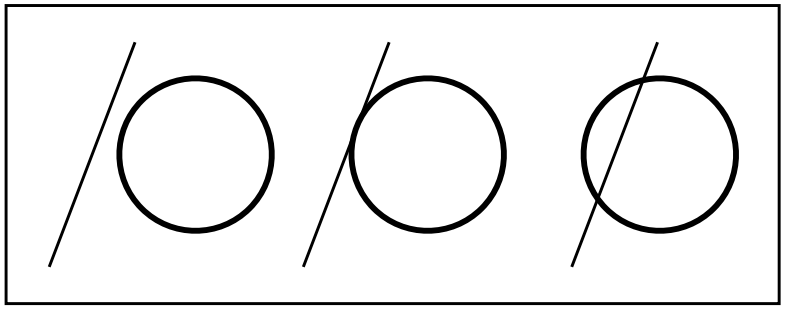
$$a\,t^2 + b\,t + c = 0, \text{ where}$$

$$a = \mathbf{d} \cdot \mathbf{d}$$

$$b = 2(\mathbf{o} - \mathbf{c}) \cdot \mathbf{d}$$

$$c = (\mathbf{o} - \mathbf{c}) \cdot (\mathbf{o} - \mathbf{c}) - R^2$$

$$t = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a}$$



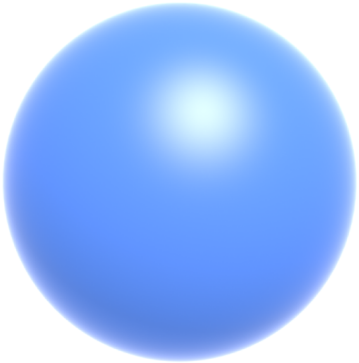
Ray Intersection With Implicit Surface

Ray: $\mathbf{r}(t) = \mathbf{o} + t\,\mathbf{d},\ 0 \leq t < \infty$

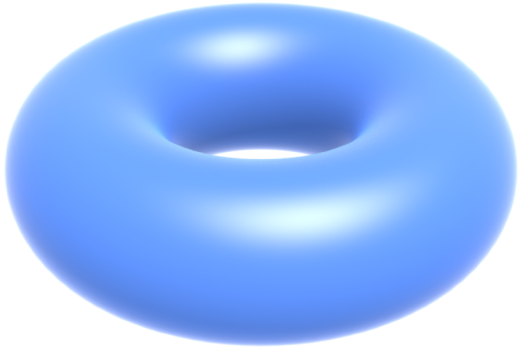
General implicit surface: $\mathbf{p} : f(\mathbf{p}) = 0$

Substitute ray equation: $f(\mathbf{o} + t\,\mathbf{d}) = 0$

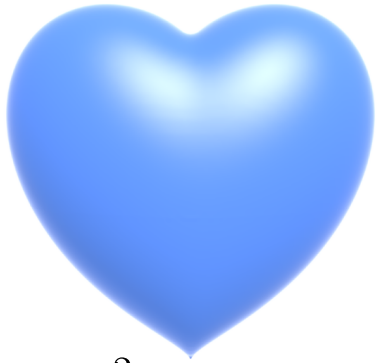
Solve for real, positive roots



$x^2+y^2+z^2=1$



$(R - \sqrt{x^2 + y^2})^2 + z^2 = r^2$



$(x^2 + \frac{9y^2}{4} + z^2 - 1)^3 = x^2z^3 + \frac{9y^2z^3}{80}$

Accelerating Ray-Surface Intersection

Why care about performance?

Simple ray-scene intersection

- Exhaustively test ray-intersection with every object

Problem:

- Exhaustive algorithm = #pixels $\times$ #objects
- Very slow!

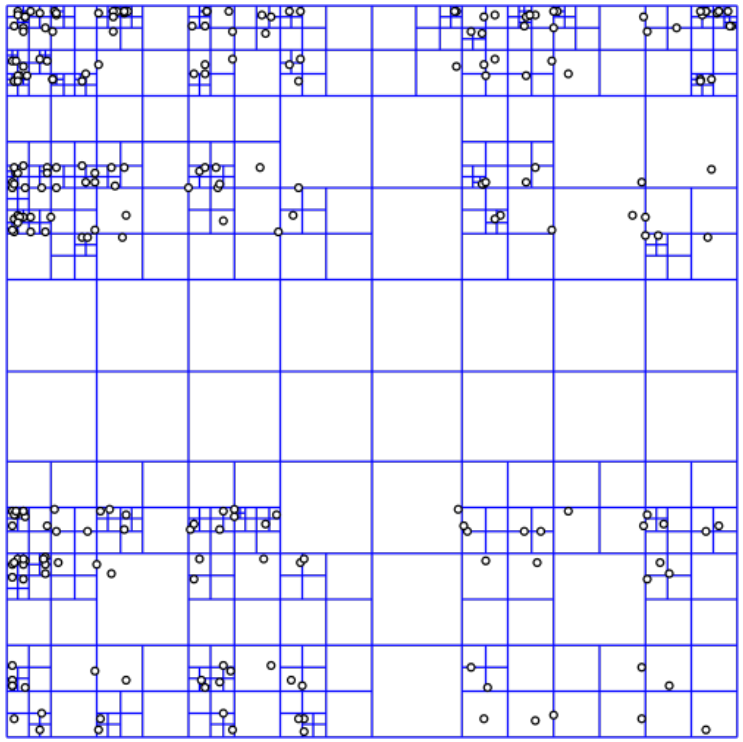
Ray Tracing – Performance Challenges

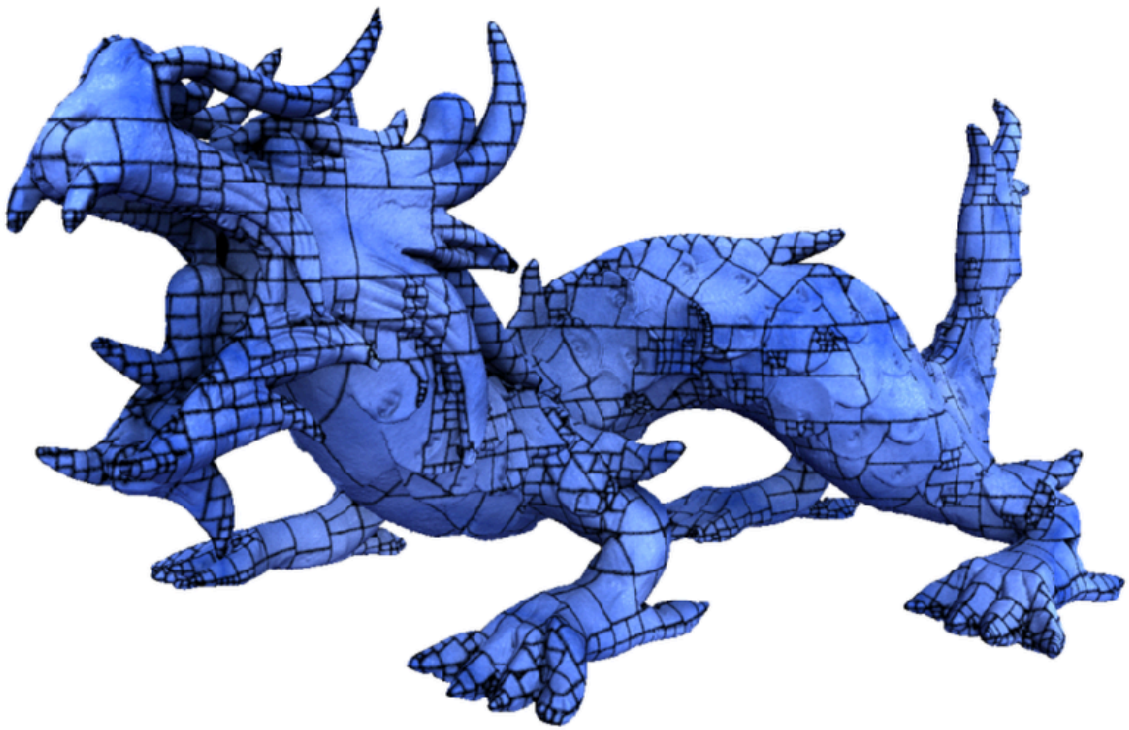
San Miguel Scene, 10.7M triangles

Ray Tracing – Performance Challenges

Plant Ecosystem, 20M triangles

Spatial Acceleration Data Structures





Bounding Volumes

Bounding Volumes

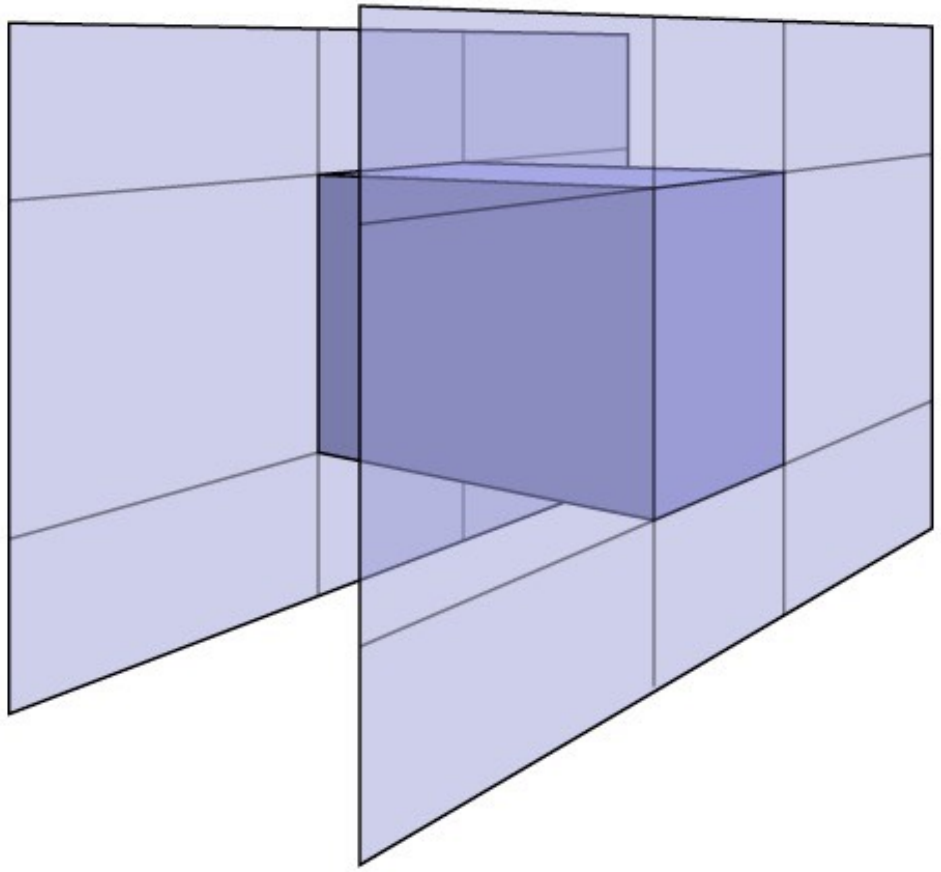
Quick way to avoid intersections: bound complex object with a simple volume

- 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

Ray-Intersection With Box

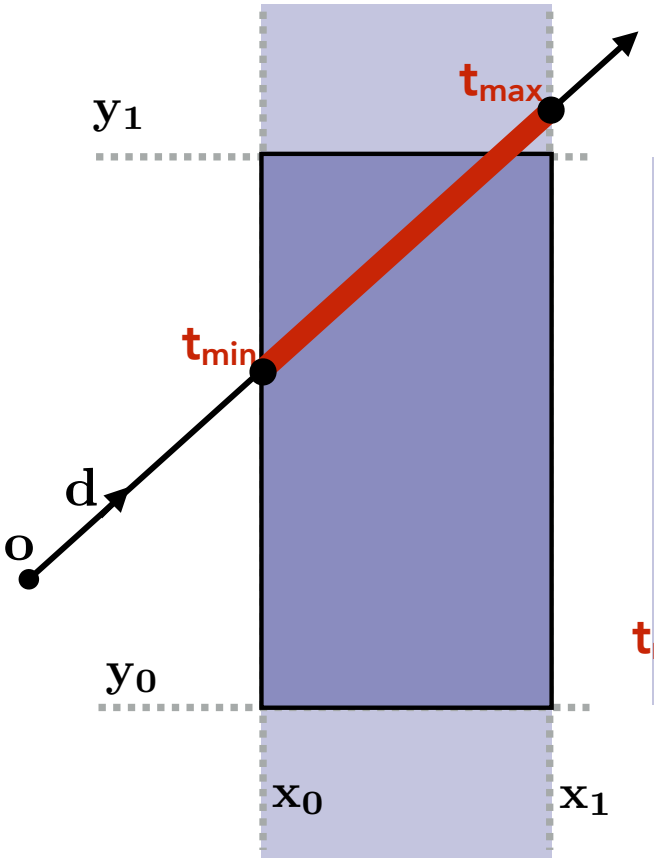
Could intersect with 6 faces individually

Better way: box is the intersection of 3 slabs

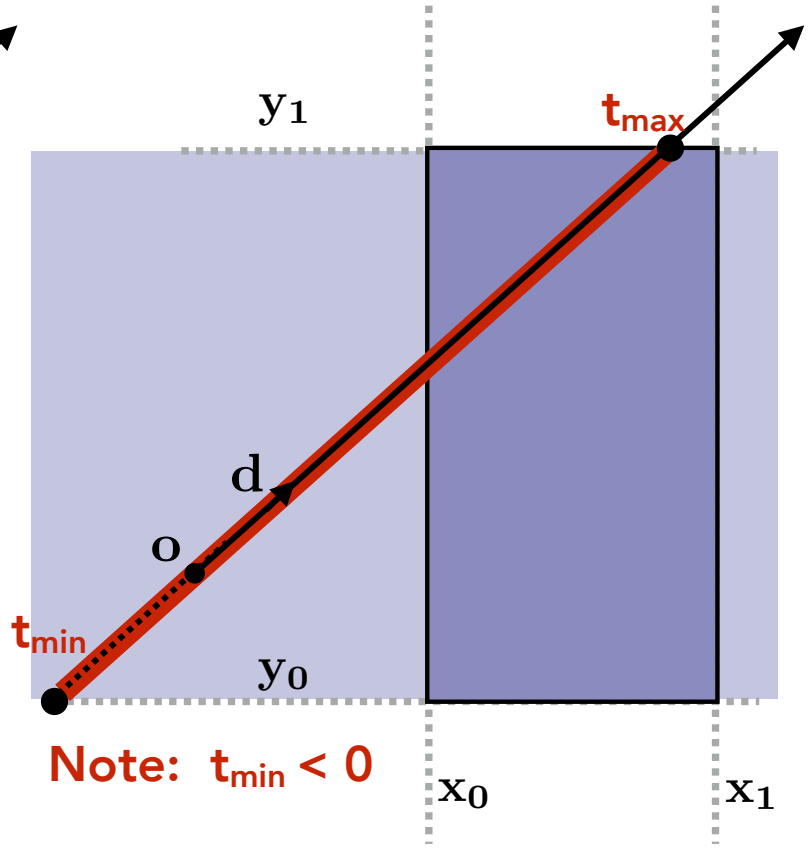


Ray Intersection with Axis-Aligned Box

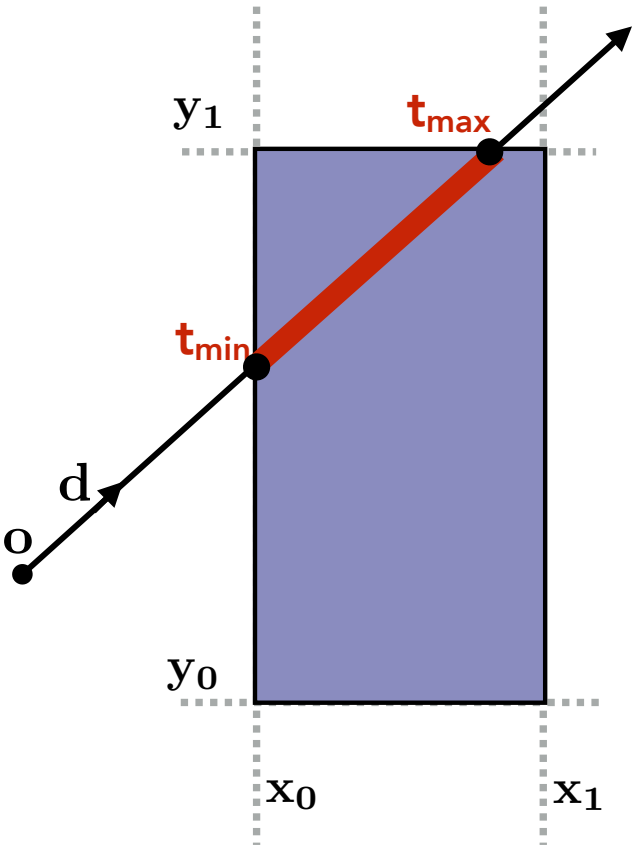
2D example; 3D is the same! Compute intersections with slabs and take intersection of t_{min}/t_{max} intervals



Intersections with x planes



Intersections with y planes

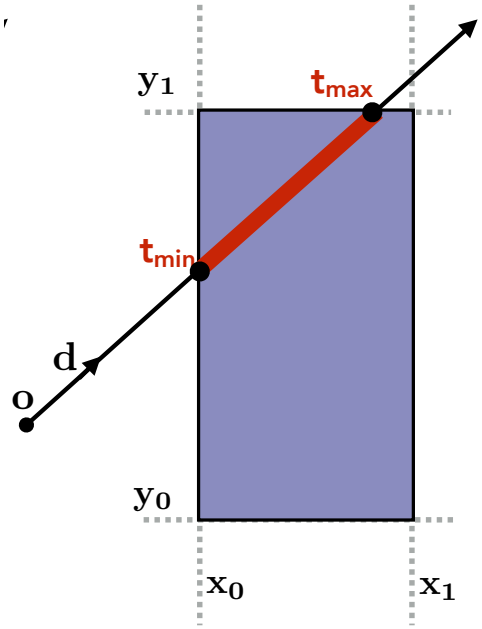


Final intersection result

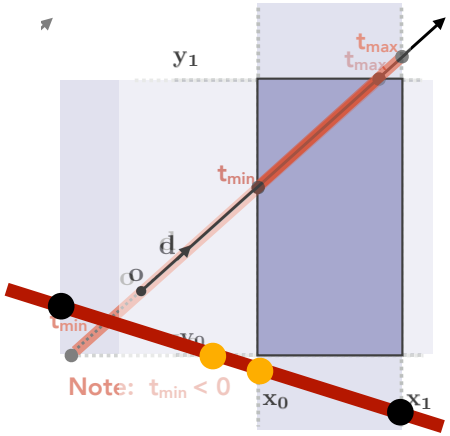
$\max\{t_{min}\}$, $\min\{t_{max}\}$

Ray Intersection with Axis-Aligned Box

- Recall: a box (3D) = three pairs of infinitely large slabs
- For each pair, calculate the t_{min} and t_{max}
- For the 3D box, t_{enter}=max$\{t_{min}\}$, t_{exit}=min$\{t_{max}\}$
- If $t_{enter} \leq t_{exit}$, we know the ray stays a while in the box (so they must intersect!)
  - $t_{enter} > t_{exit}$ means the ray misses the box completely

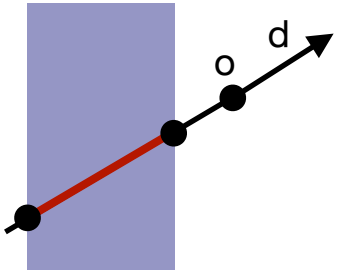


Final intersection result

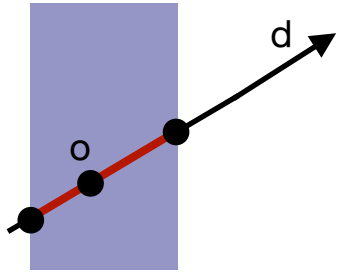


Ray Intersection with Axis-Aligned Box

- However, ray is not a line
  - Should check whether t is negative for physical correctness!
- What if $t_{exit} < 0$?
  - The box is "behind" the ray — no intersection!

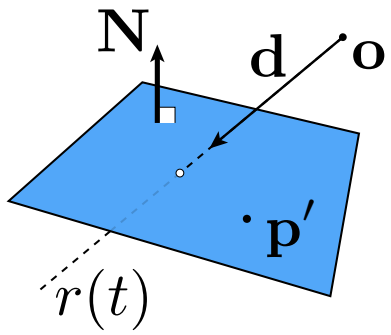


- What if $t_{exit} \geq 0$ and $t_{enter} < 0$
  - The ray's origin is inside the box — have intersection!
- In summary, ray and AABB intersect iff
  - $t_{enter} < t_{exit}$ && $t_{exit} \geq 0$



Optimize Ray-Plane Intersection For Axis-Aligned Planes?

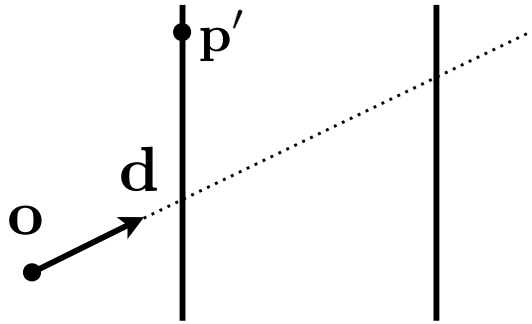
General



$$t = \frac{(\mathbf{p}' - \mathbf{o}) \cdot \mathbf{N}}{\mathbf{d} \cdot \mathbf{N}}$$

3 subtractions, 6 multiplies, 1 division

Perpendicular to x-axis



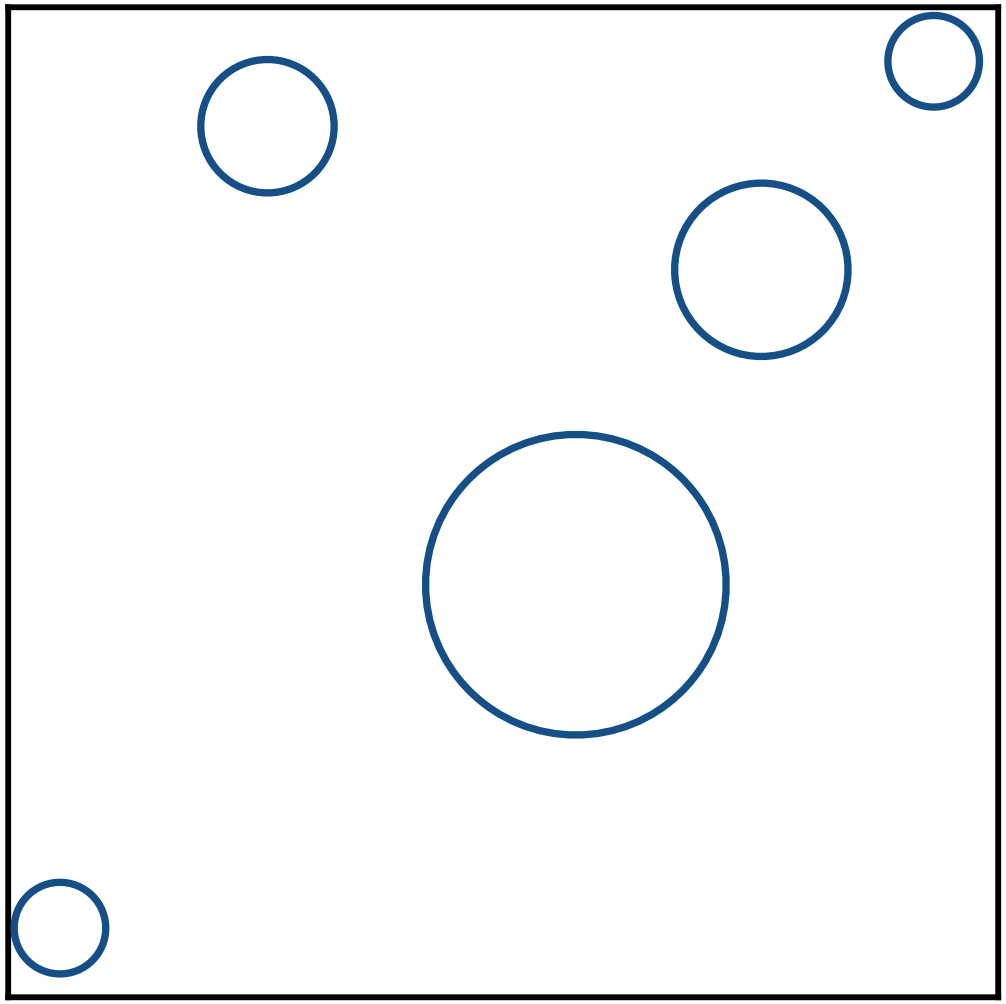
$$t = \frac{\mathbf{p}'_x - \mathbf{o}_x}{\mathbf{d}_x}$$

1 subtraction, 1 division

Uniform Spatial Partitions (Grids)

Subdivision of AABB into smaller, equally sized boxes

Preprocess – Build Acceleration Grid



1. Find bounding box

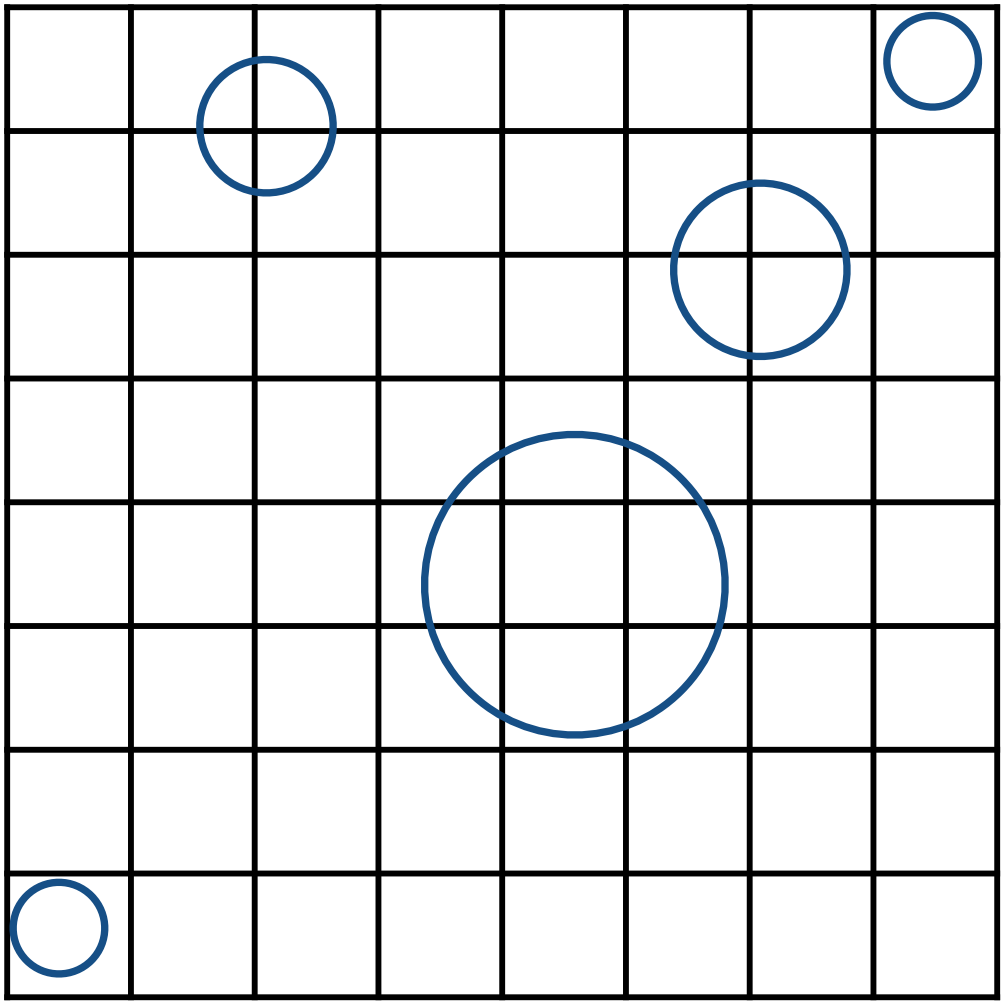
An AABB can be defined by two points:

$$\mathbf{b_{min}} = (x_{min}, y_{min}, z_{min})$$

$$\mathbf{b_{max}} = (x_{max}, y_{max}, z_{max})$$

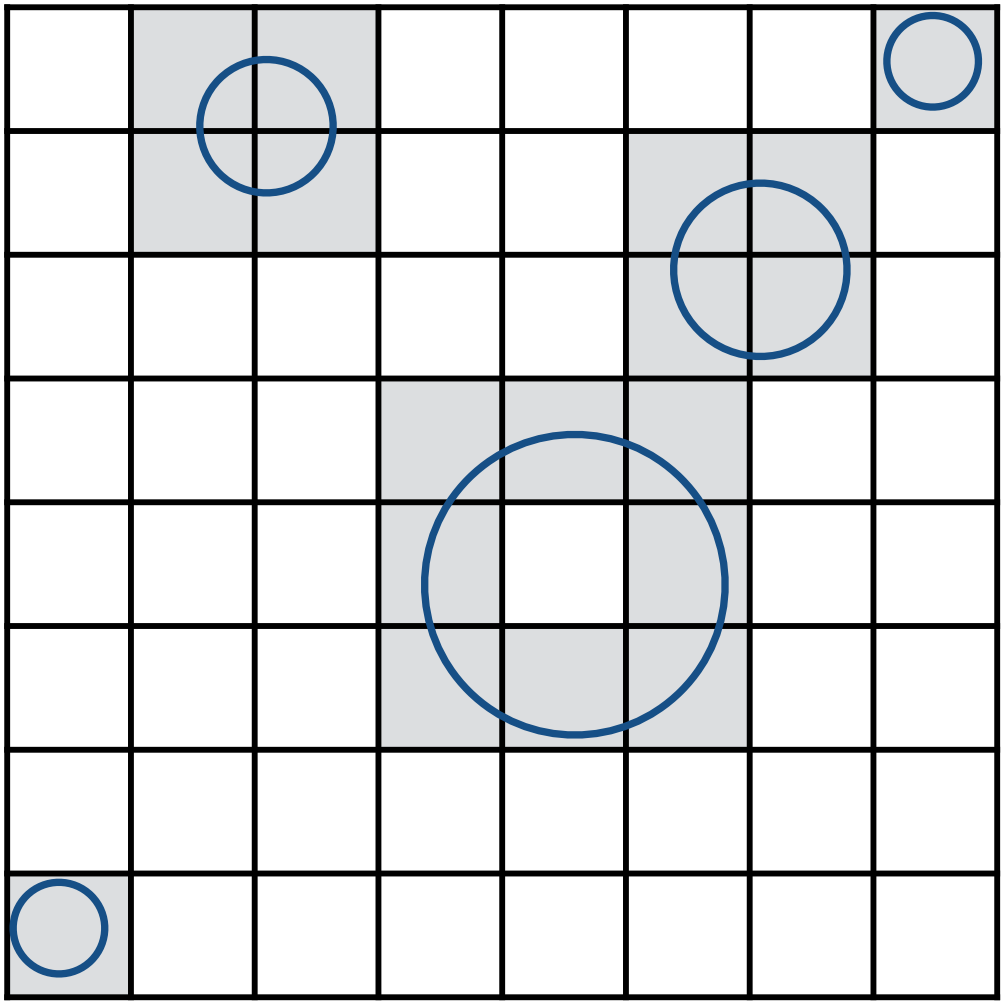
$$B = [x_{min}, x_{max}] \times [y_{min}, y_{max}] \times [z_{min}, z_{max}]$$

Preprocess – Build Acceleration Grid



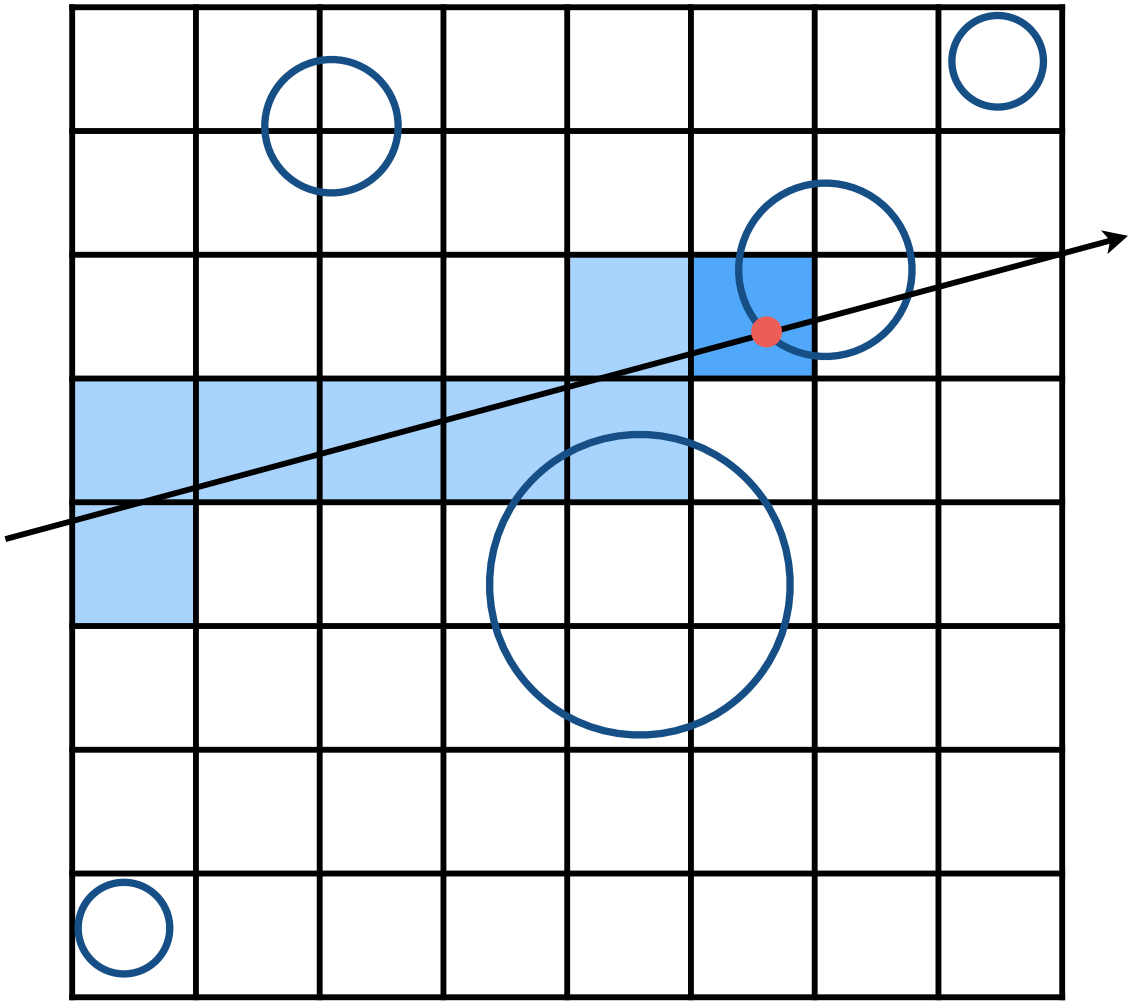
1. Find bounding box
2. Create grid

Preprocess – Build Acceleration Grid



1. Find bounding box
2. Create grid
3. Store each object in overlapping cells

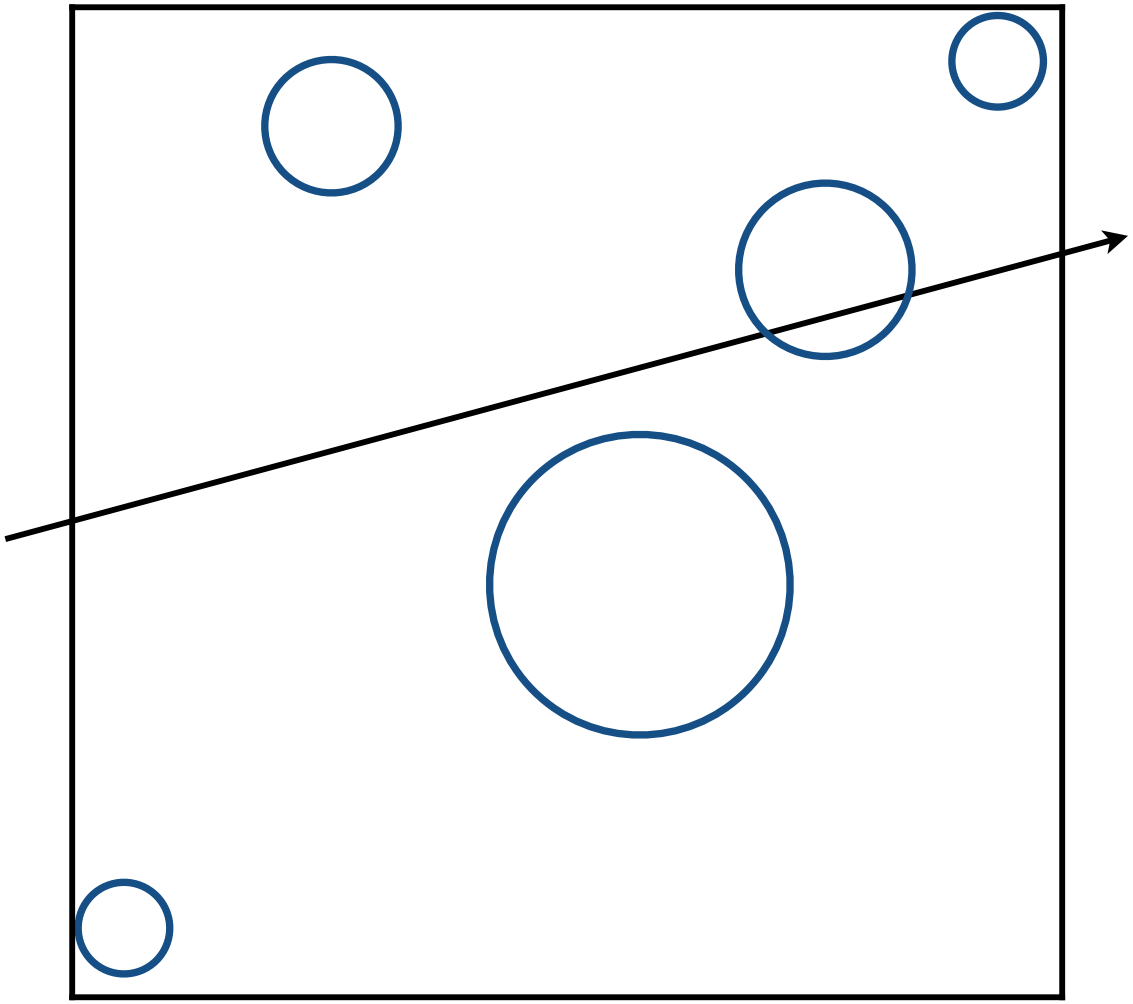
Ray-Scene Intersection



Step through grid in ray traversal order (3D line - 3D DDA)

For each grid cell
Test intersection with all objects stored at that cell

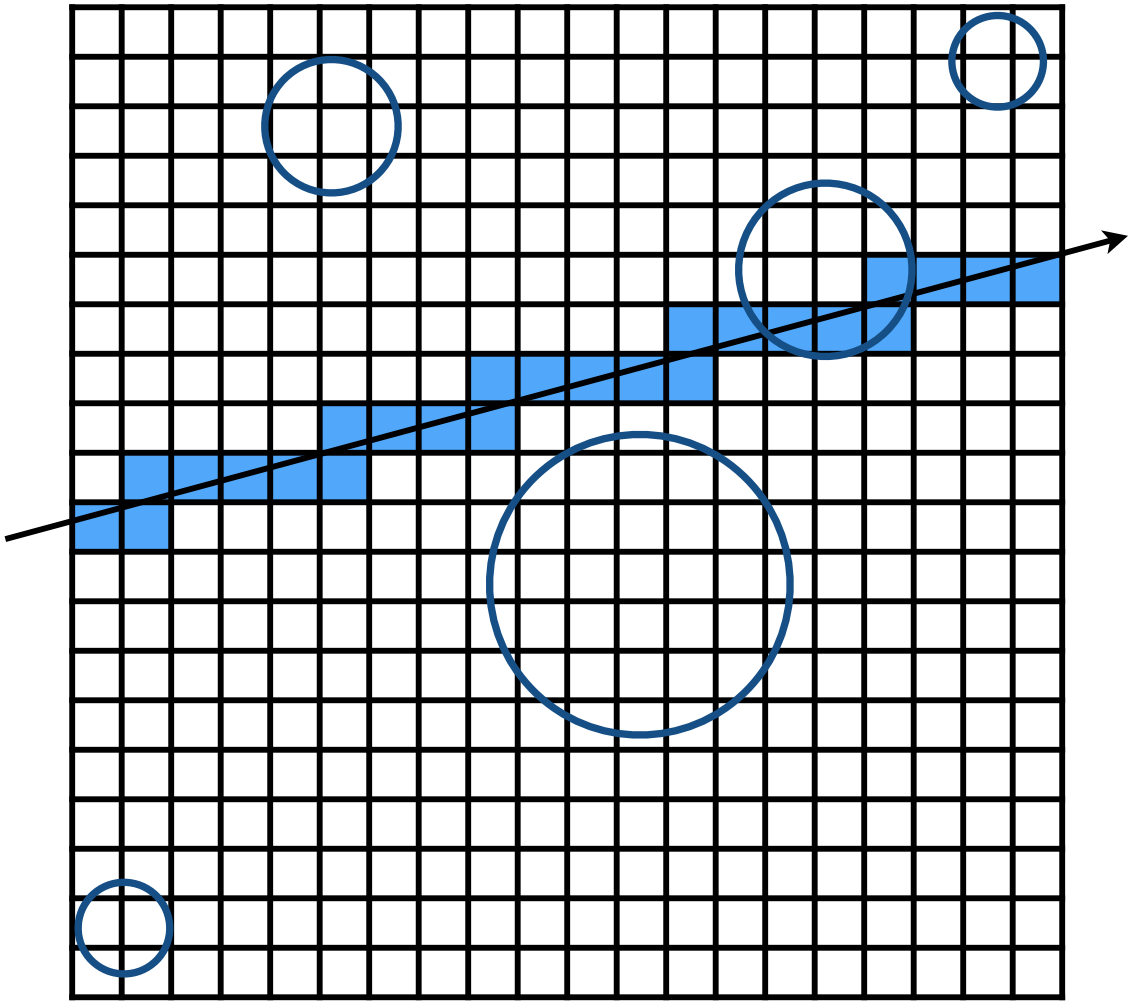
Grid Resolution?



One cell

- No speedup

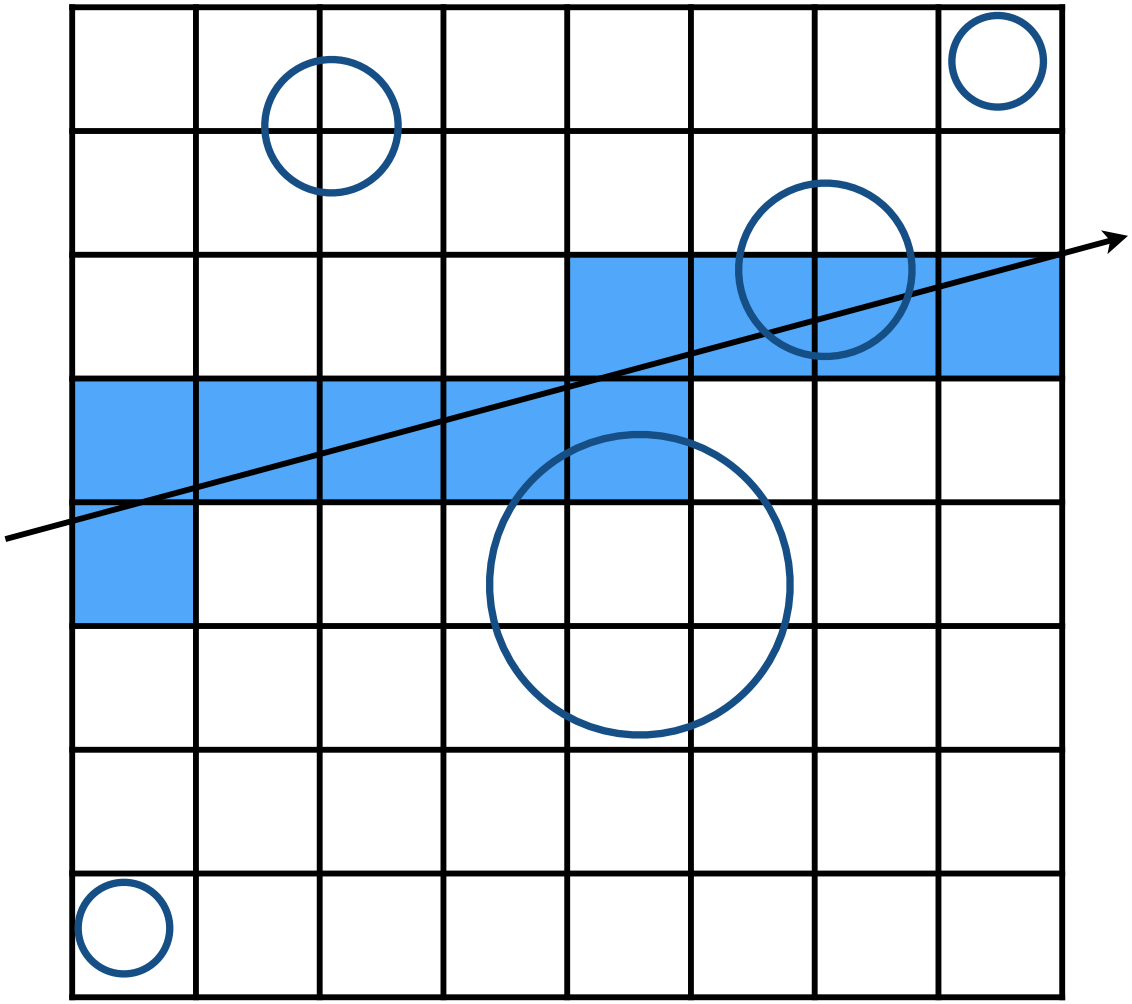
Grid Resolution?



Too many cells

- Inefficiency due to extraneous grid traversal

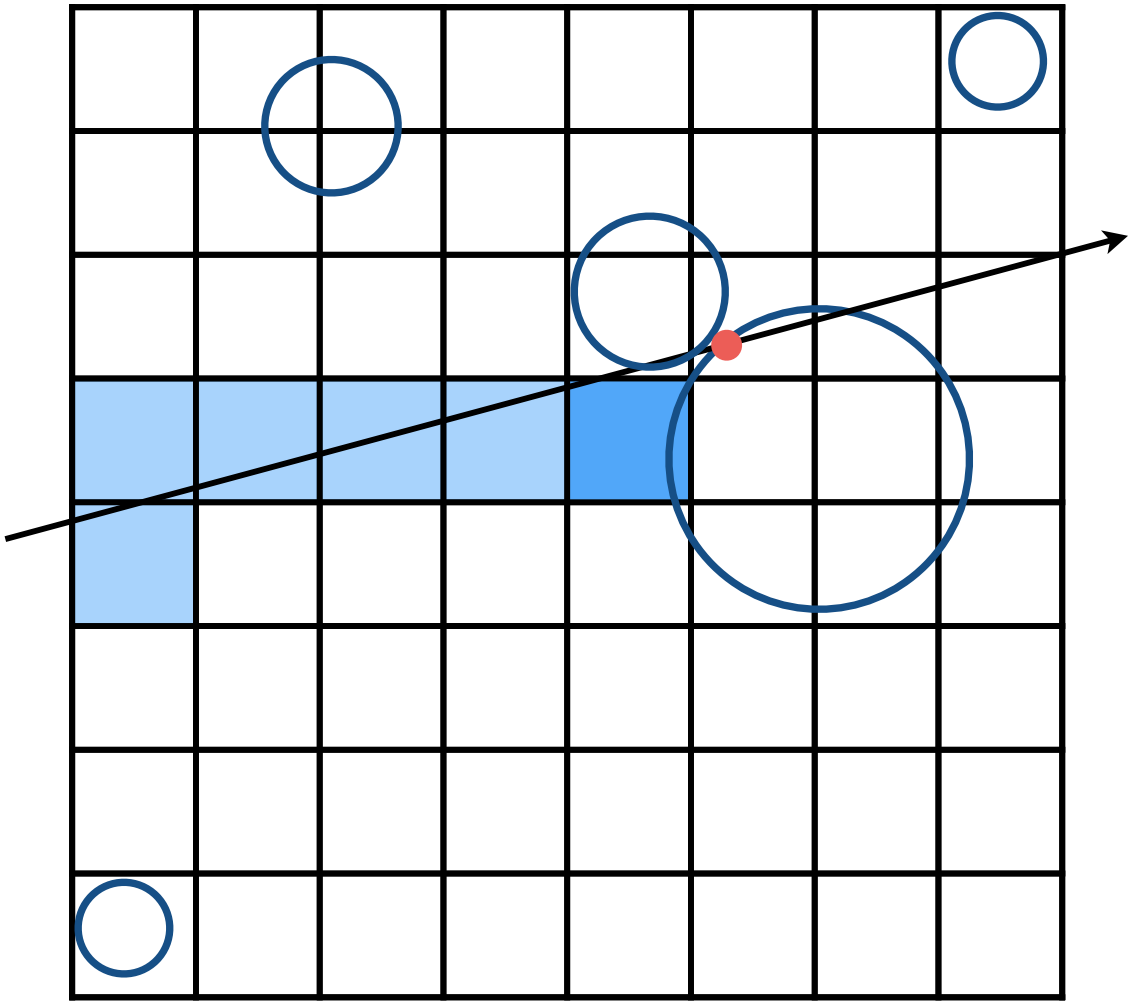
Grid Resolution?



Heuristic:

- #cells = C * #objs
- C ≈ 27 in 3D

Careful! Objects Overlapping Multiple Cells



What goes wrong here?

- First intersection found (red) is not the nearest!

Solution?

- Check intersection point is inside cell

Optimize

- Cache intersection to avoid re-testing (mailboxing)

Uniform Grids – When They Work Well

Grids work well on large collections of objects that are distributed evenly in size and space

Uniform Grids – When They Fail

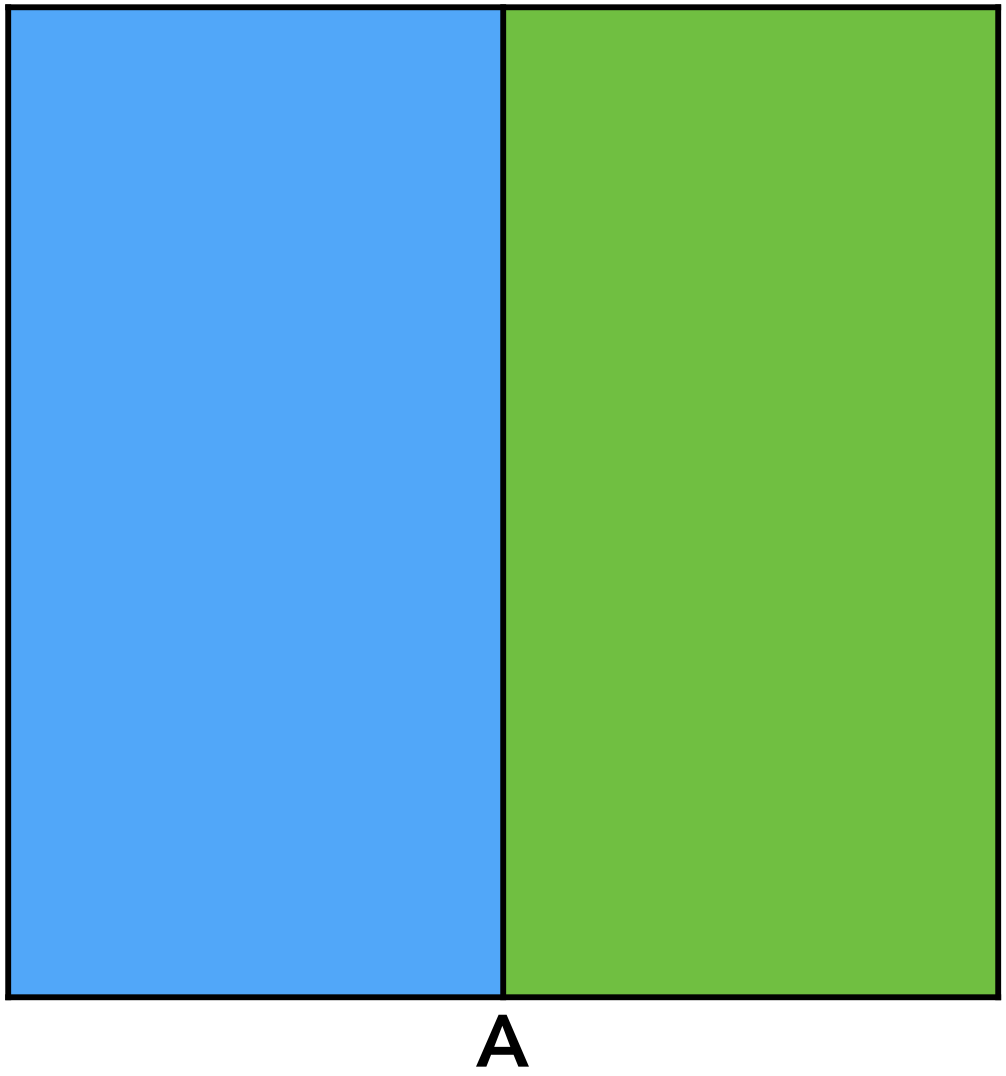
are inefficient

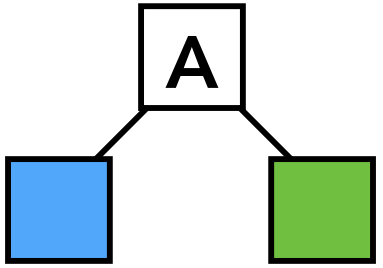
“Teapot in a stadium” problem

Almost every cell in that grid is empty, and the grid still allocates memory for every voxel. A ray traversing the scene must step through thousands or millions of empty voxels before it reaches the teapot.

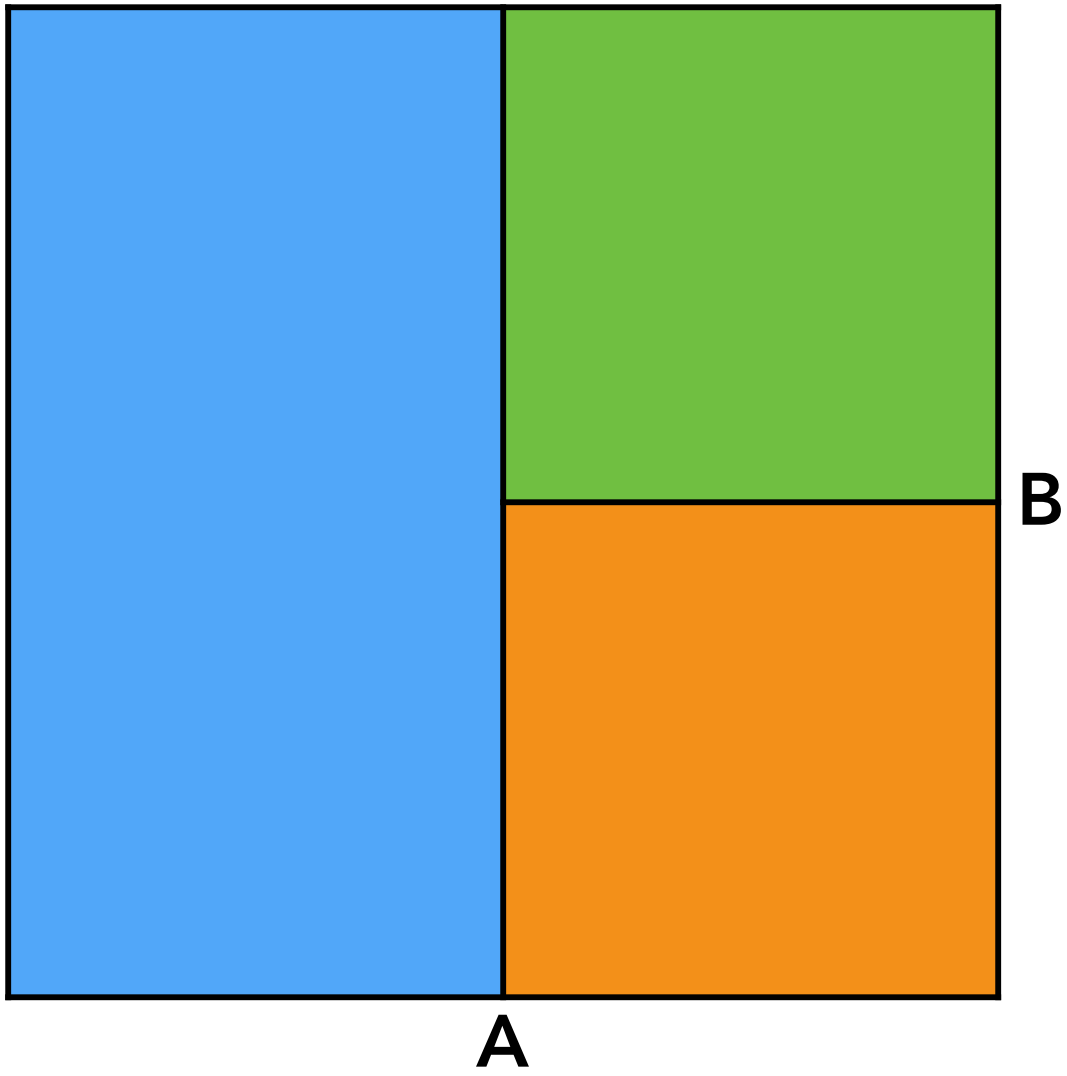
Non-Uniform Spatial Partitions: Spatial Hierarchies

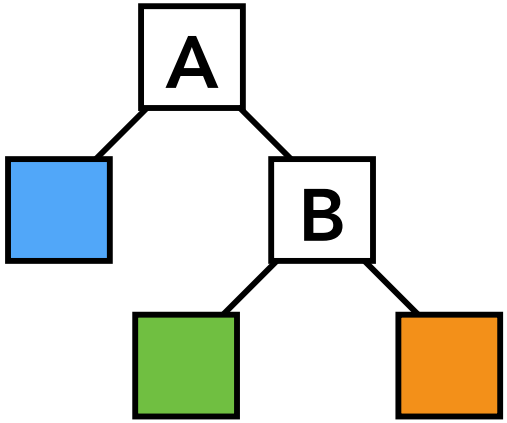
Spatial Hierarchies



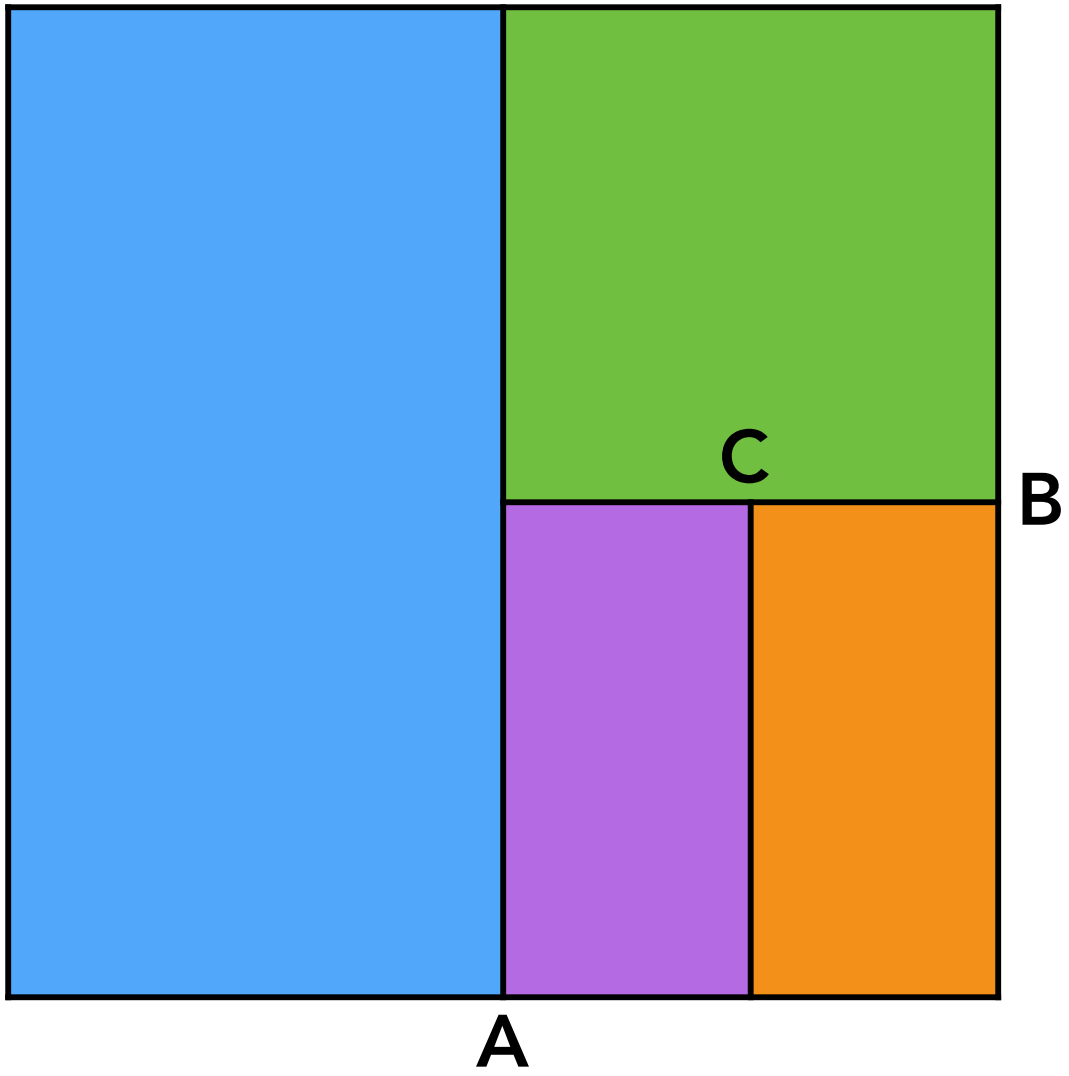


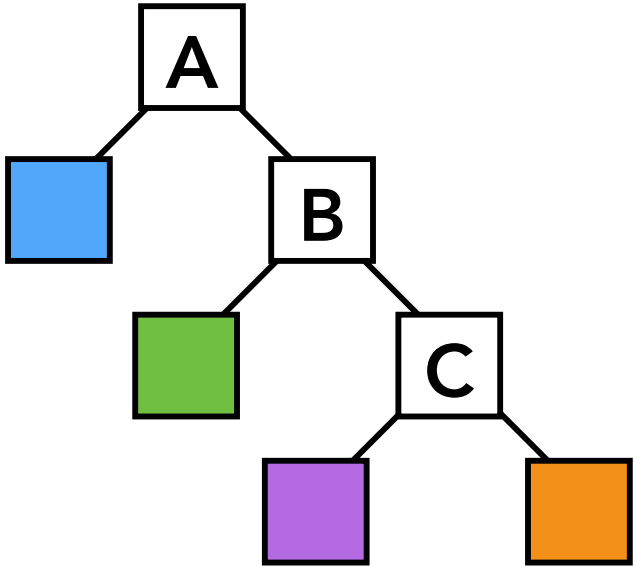
Spatial Hierarchies



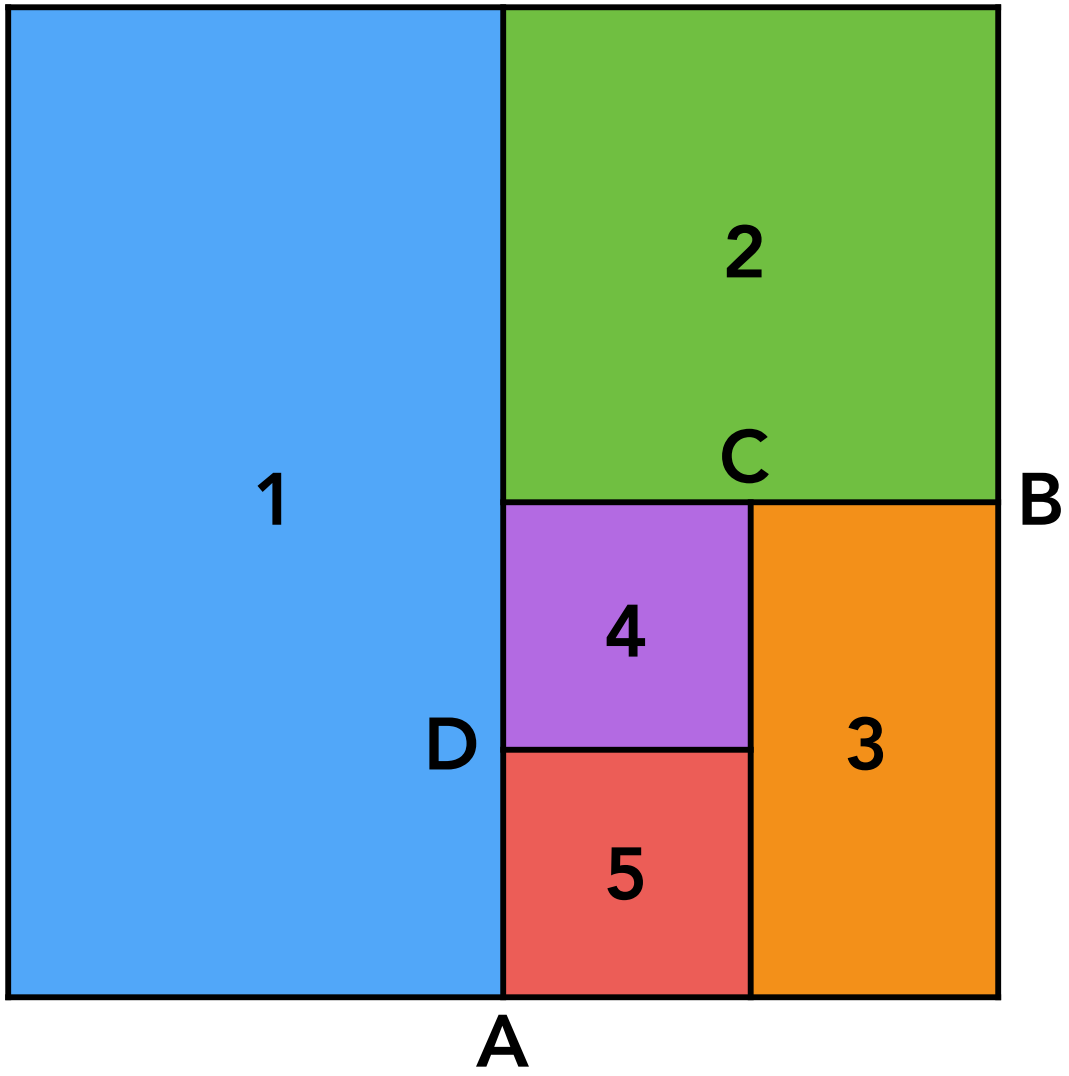


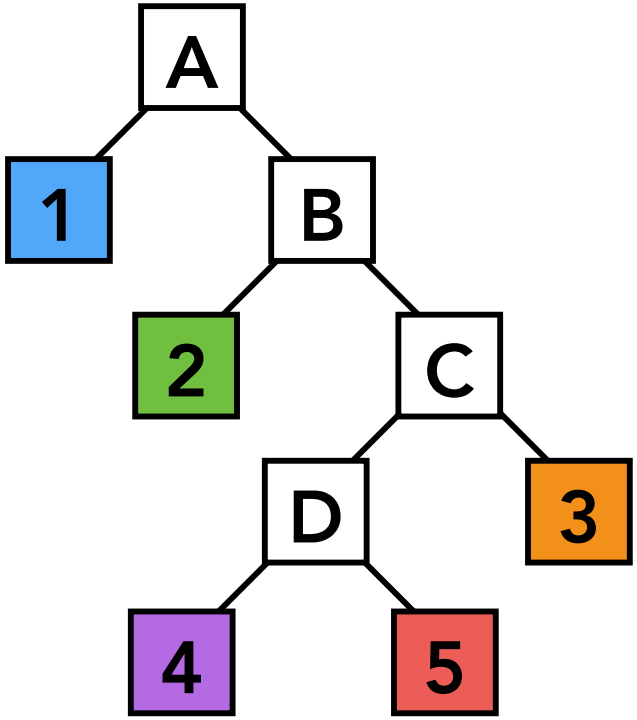
Spatial Hierarchies



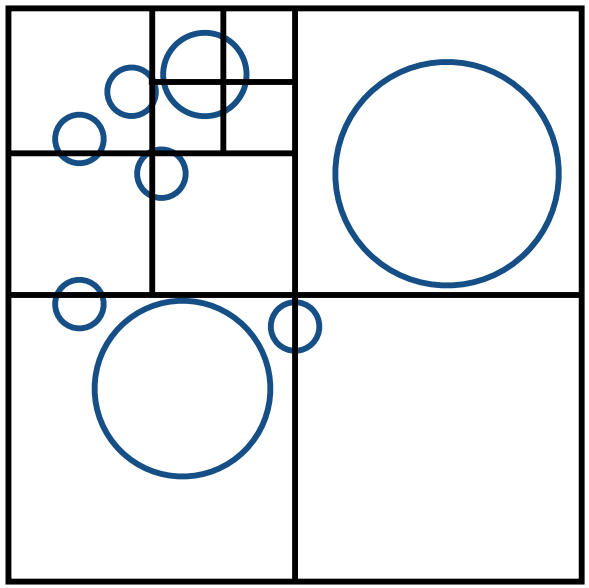


Spatial Hierarchies

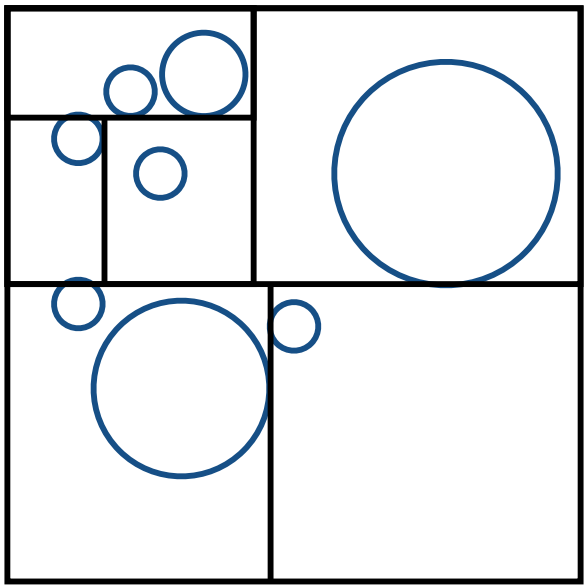




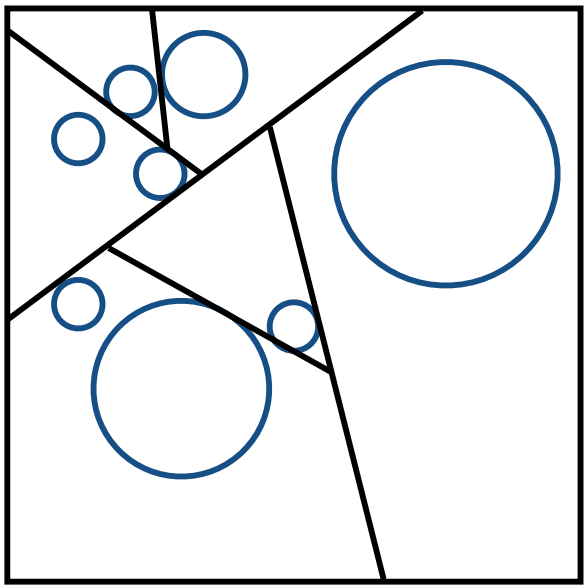
Spatial Partitioning Variants



Oct-Tree



KD-Tree



BSP-Tree

Note: you could have these in both 2D and 3D. In lecture we will illustrate principles in 2D, but for assignment you will implement 3D versions.

KD-Trees

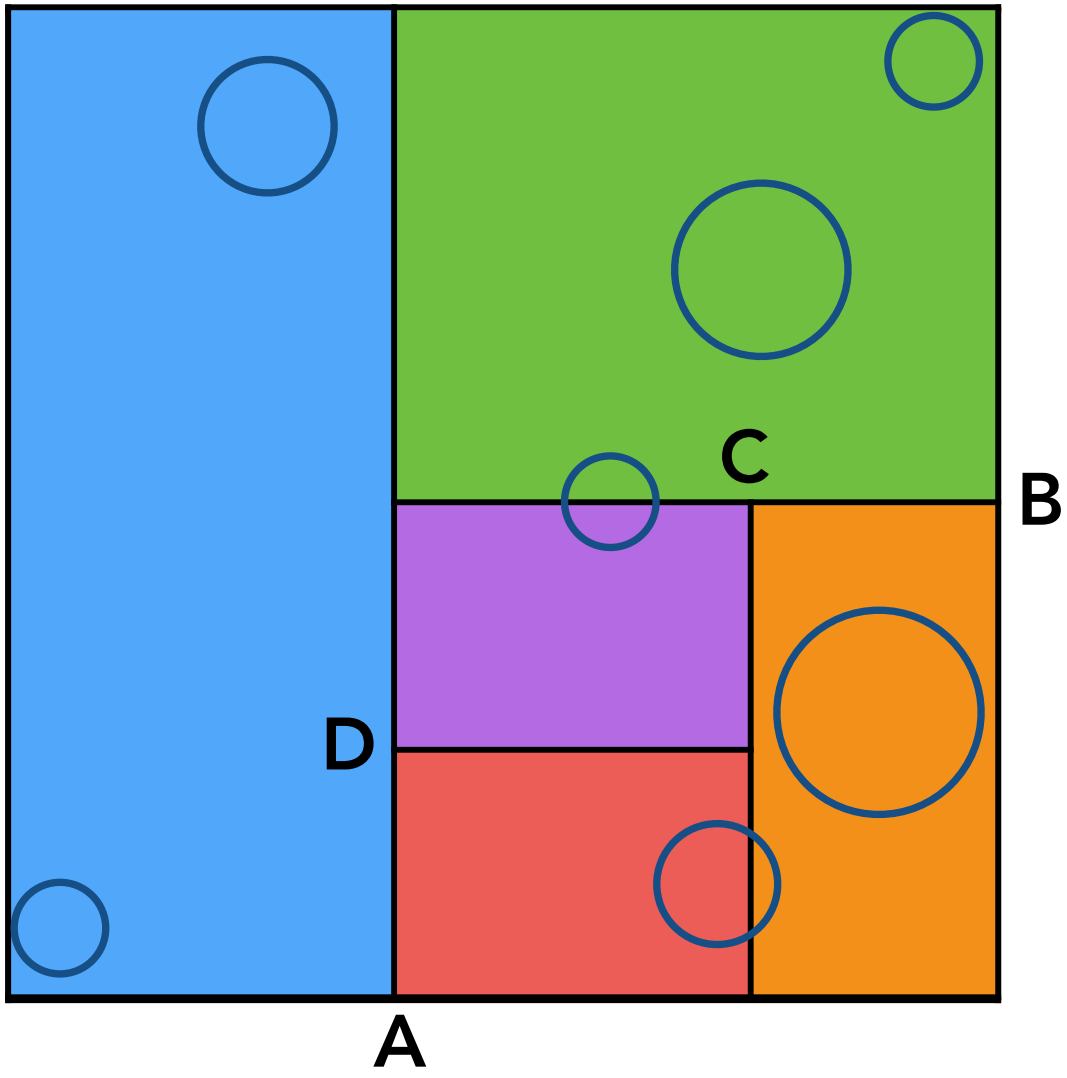
Internal nodes store

- split axis: x-, y-, or z-axis
- split position: coordinate of split plane along axis
- children: reference to child nodes

Leaf nodes store

- list of objects
- mailbox information

KD-Tree Pre-Processing



- Find bounding box
- Recursively split cells, axis-aligned planes
- Until termination criteria met (e.g. max #splits or min #objs)
- Store obj references with each leaf node

KD-Tree Pre-Processing

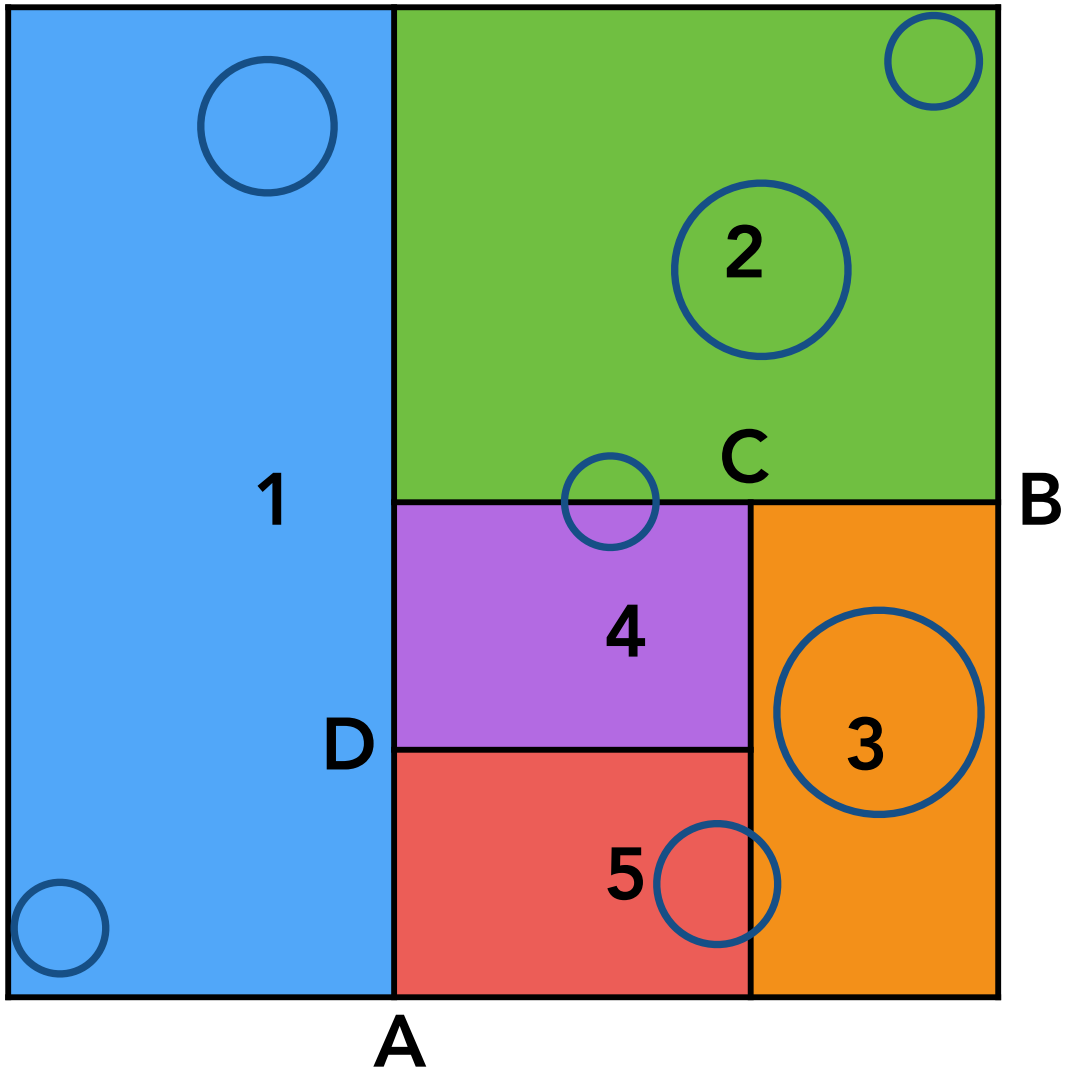
Choosing the split plane

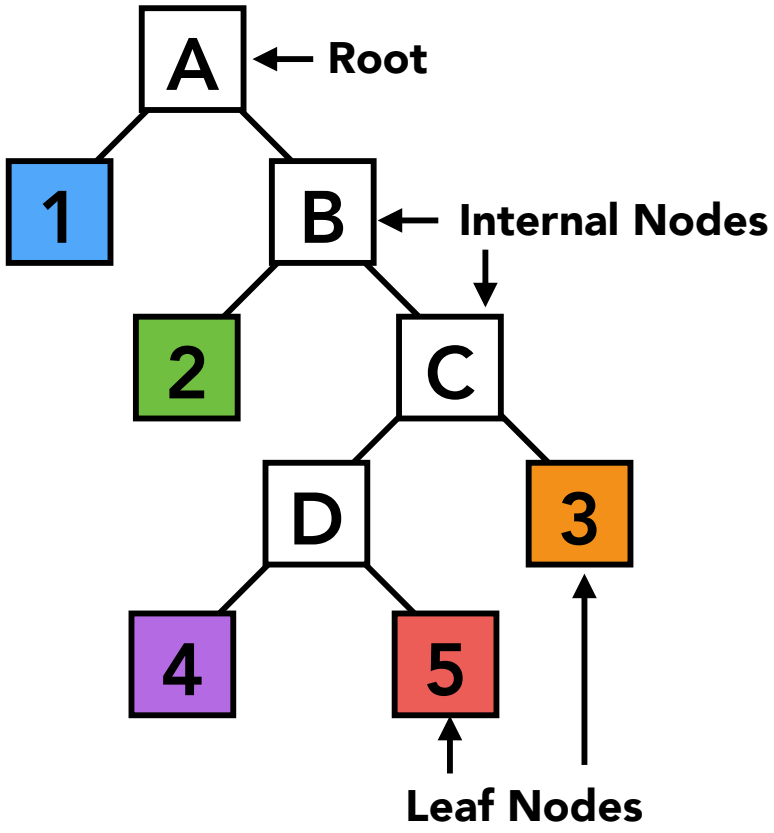
- Simple: midpoint, median split
- Ideal: split to minimize expected cost of ray intersection

Termination criteria?

- Simple: common to prescribe maximum tree depth (empirical 8 + 1.3 log N, N = #objs) [PBRT]
- Ideal: stop when splitting does not reduce expected cost of ray intersection

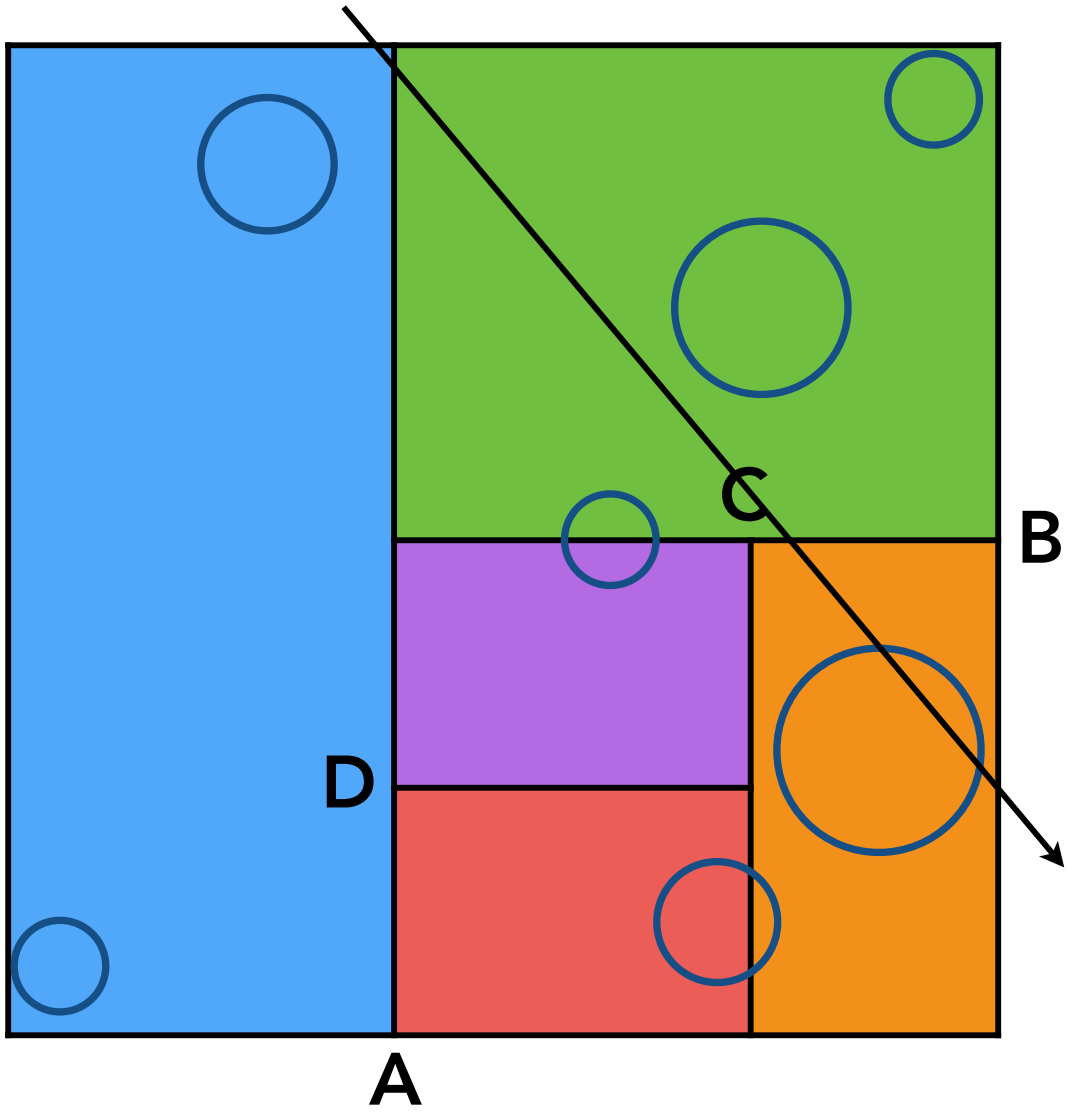
KD-Tree Pre-Processing

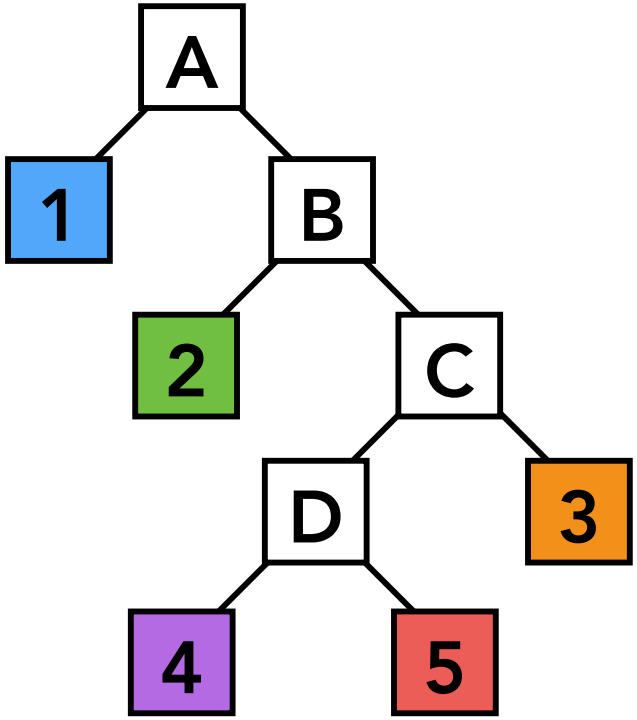




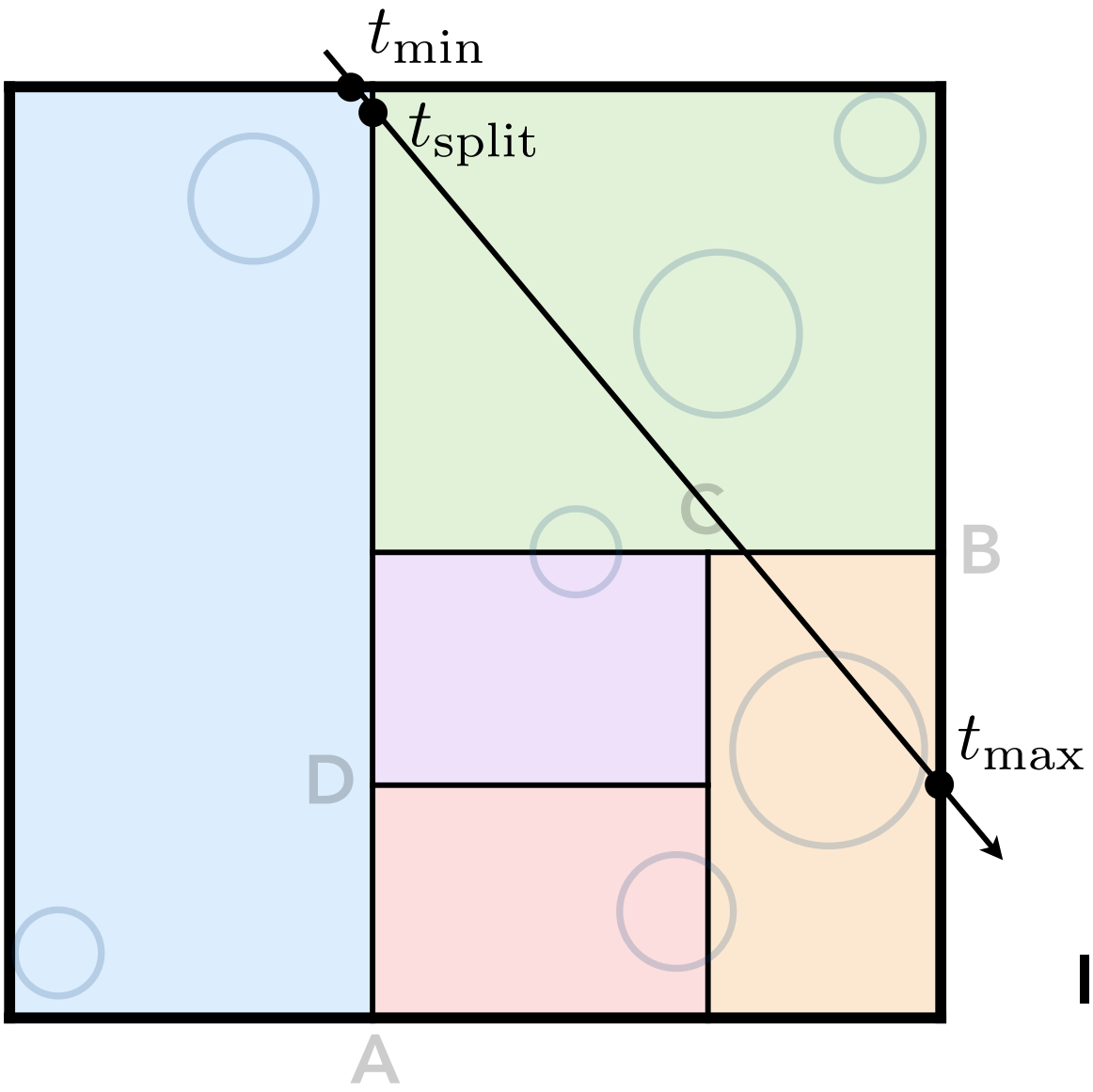
Only leaf nodes store references to geometry

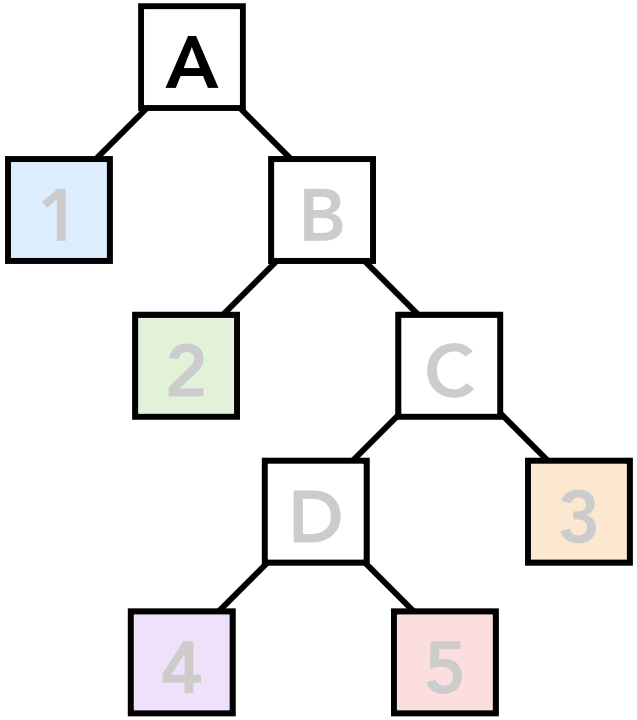
Top-Down Recursive In-Order Traversal





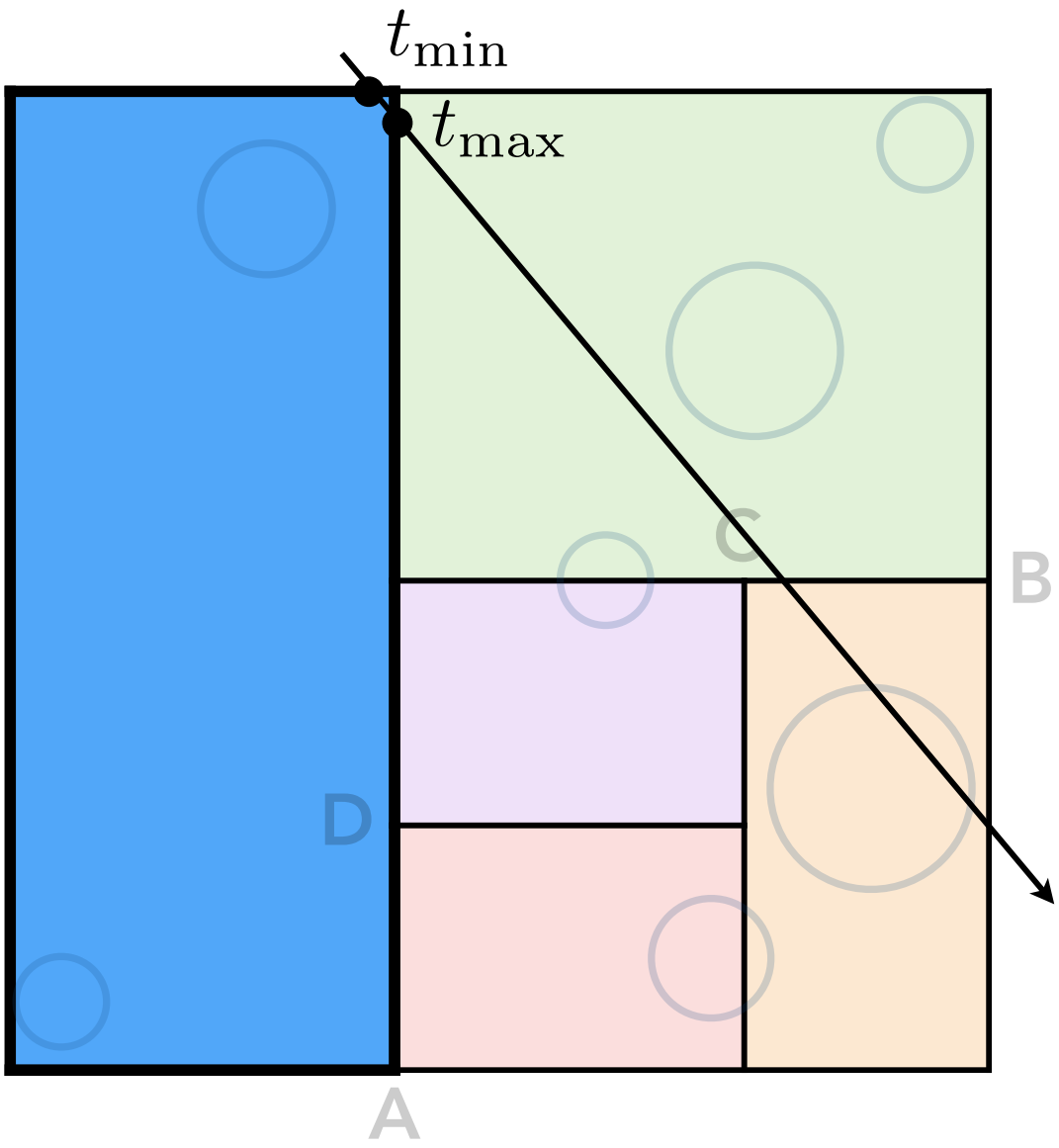
Top-Down Recursive In-Order Traversal

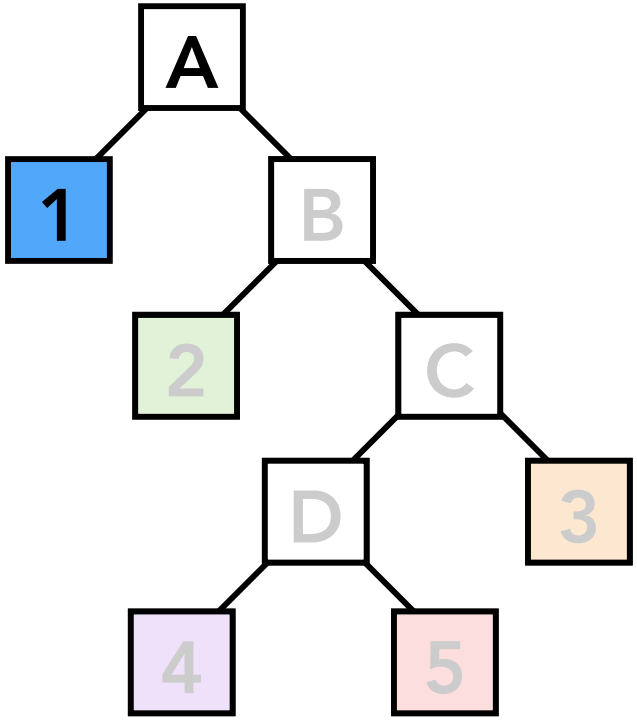




Internal node: split

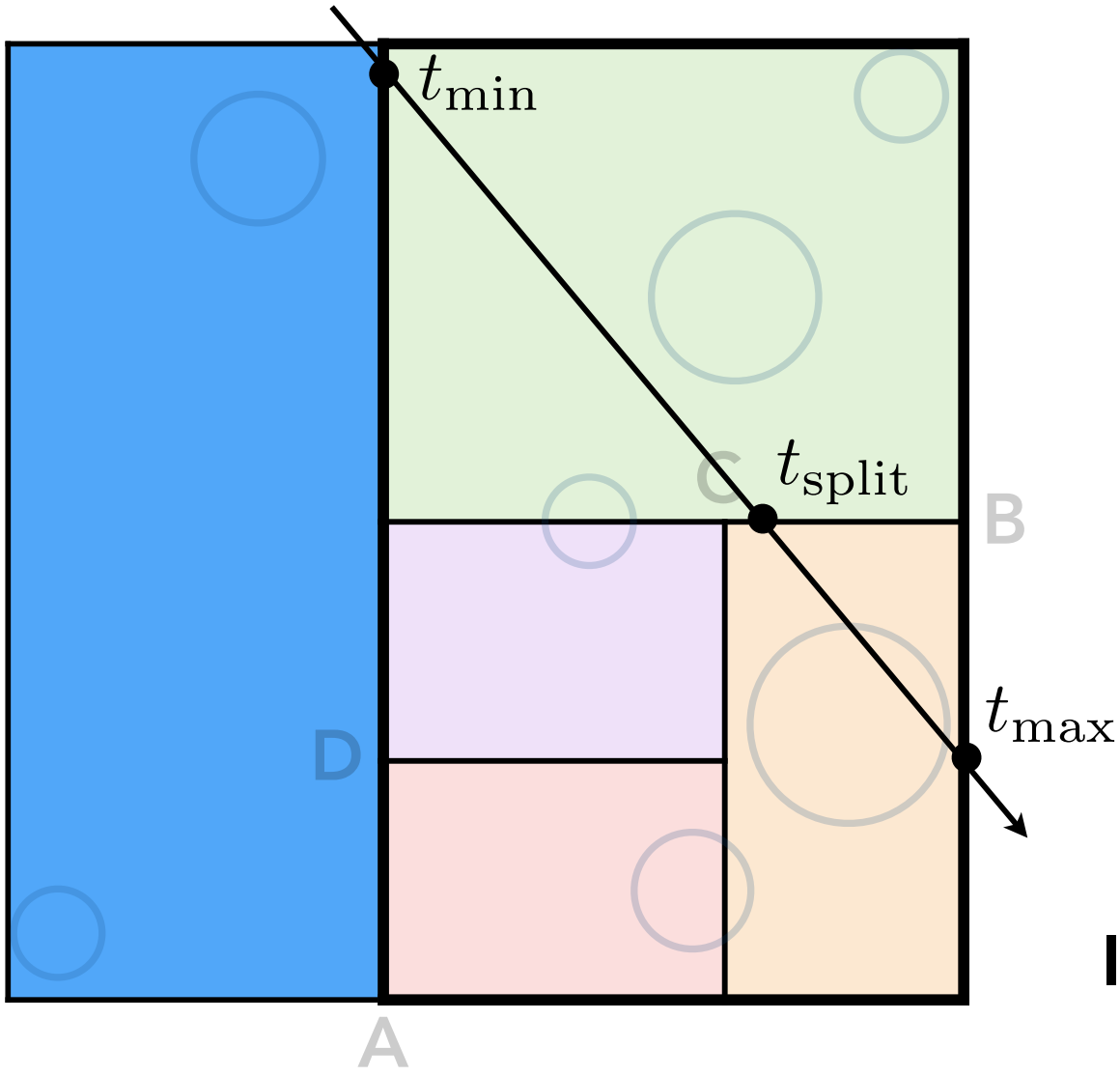
Top-Down Recursive In-Order Traversal

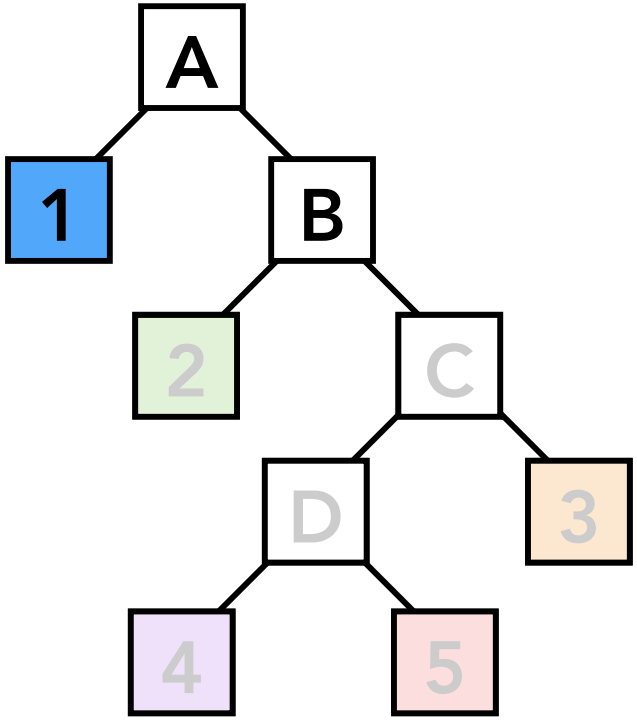




Leaf node: intersect all objects

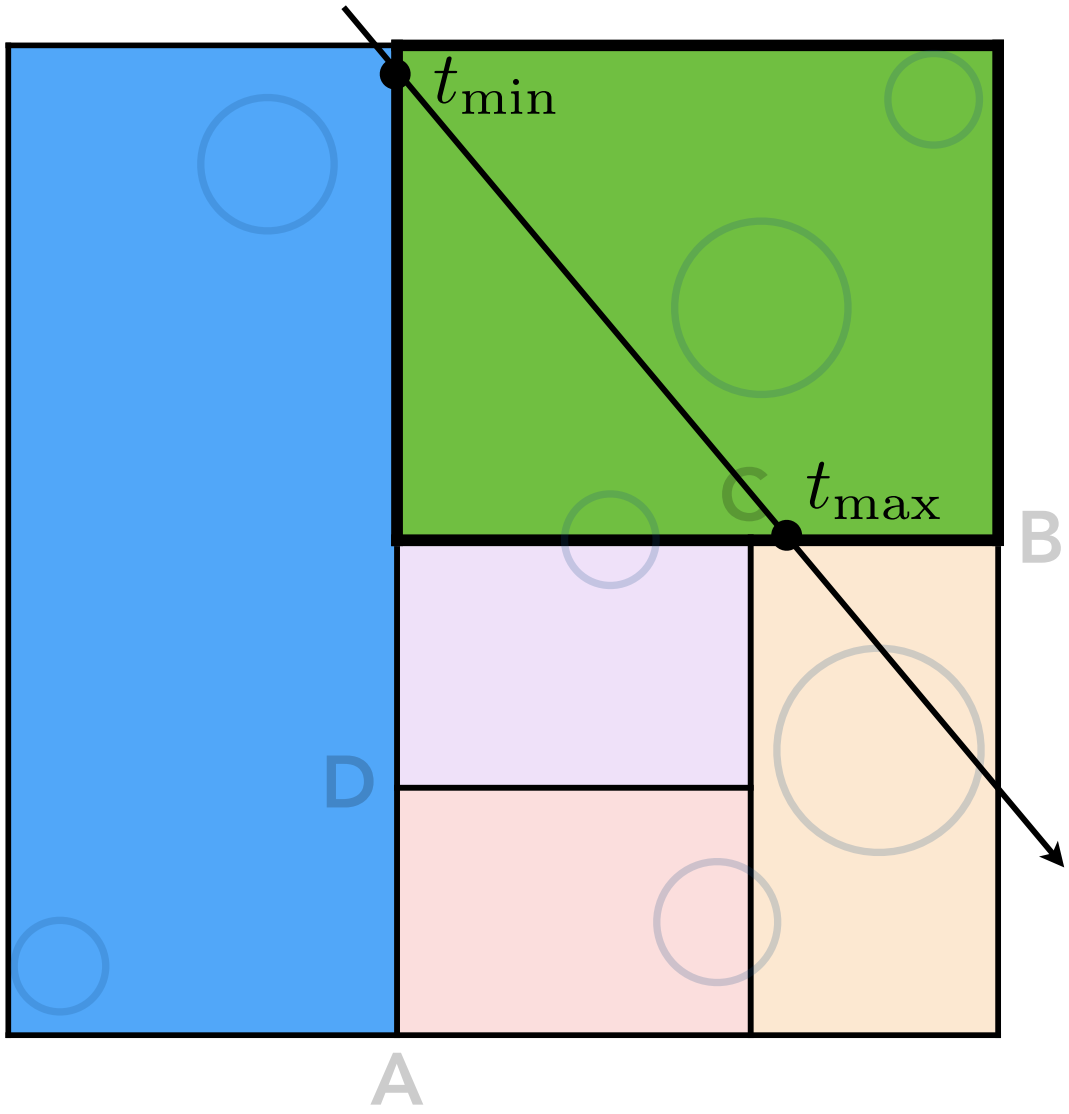
Top-Down Recursive In-Order Traversal

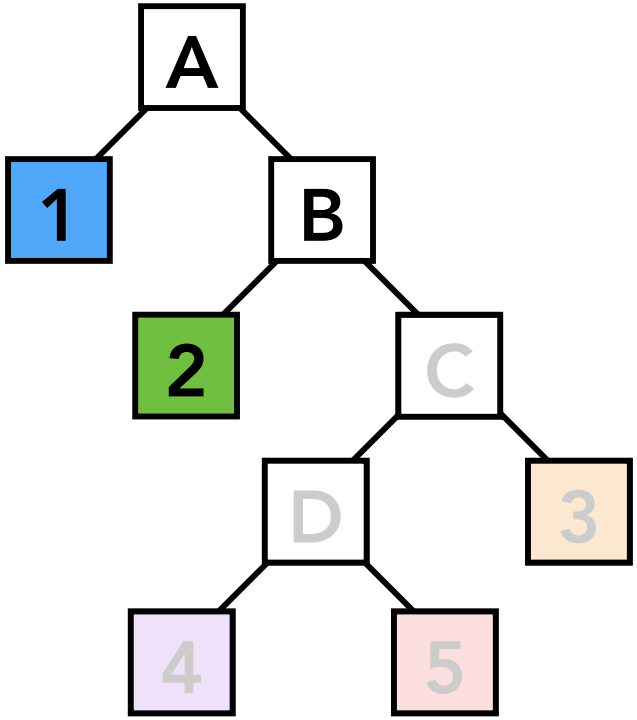




Internal node: split

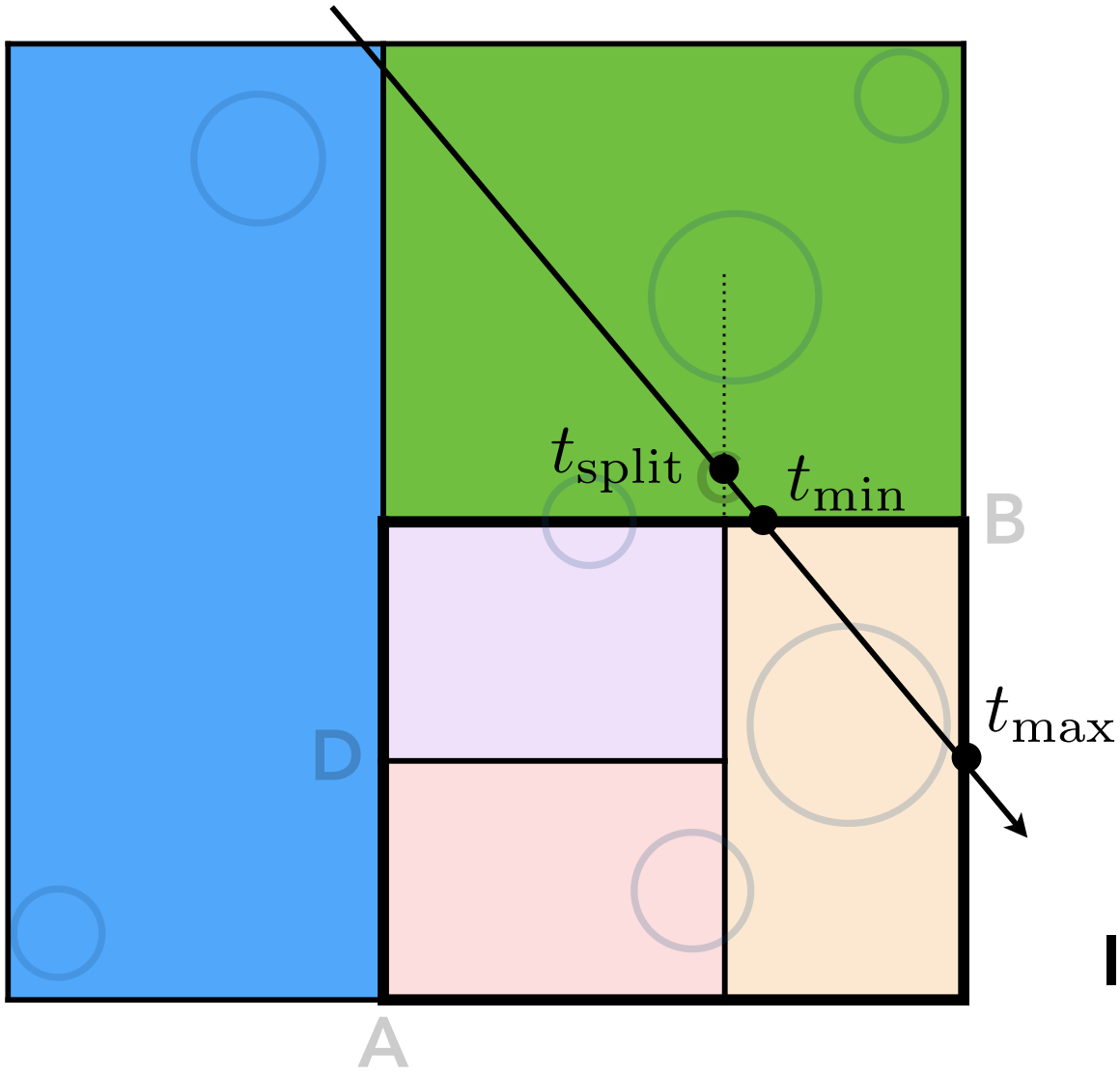
Top-Down Recursive In-Order Traversal

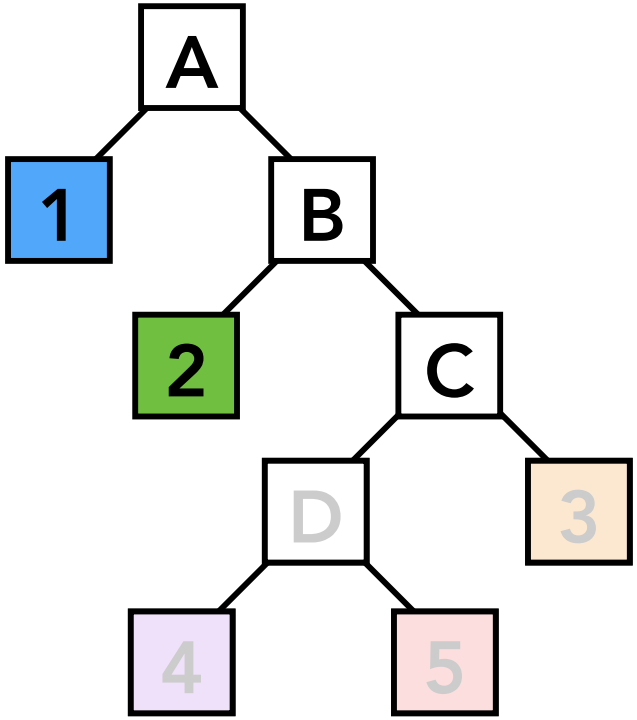




Leaf node: intersect all objects

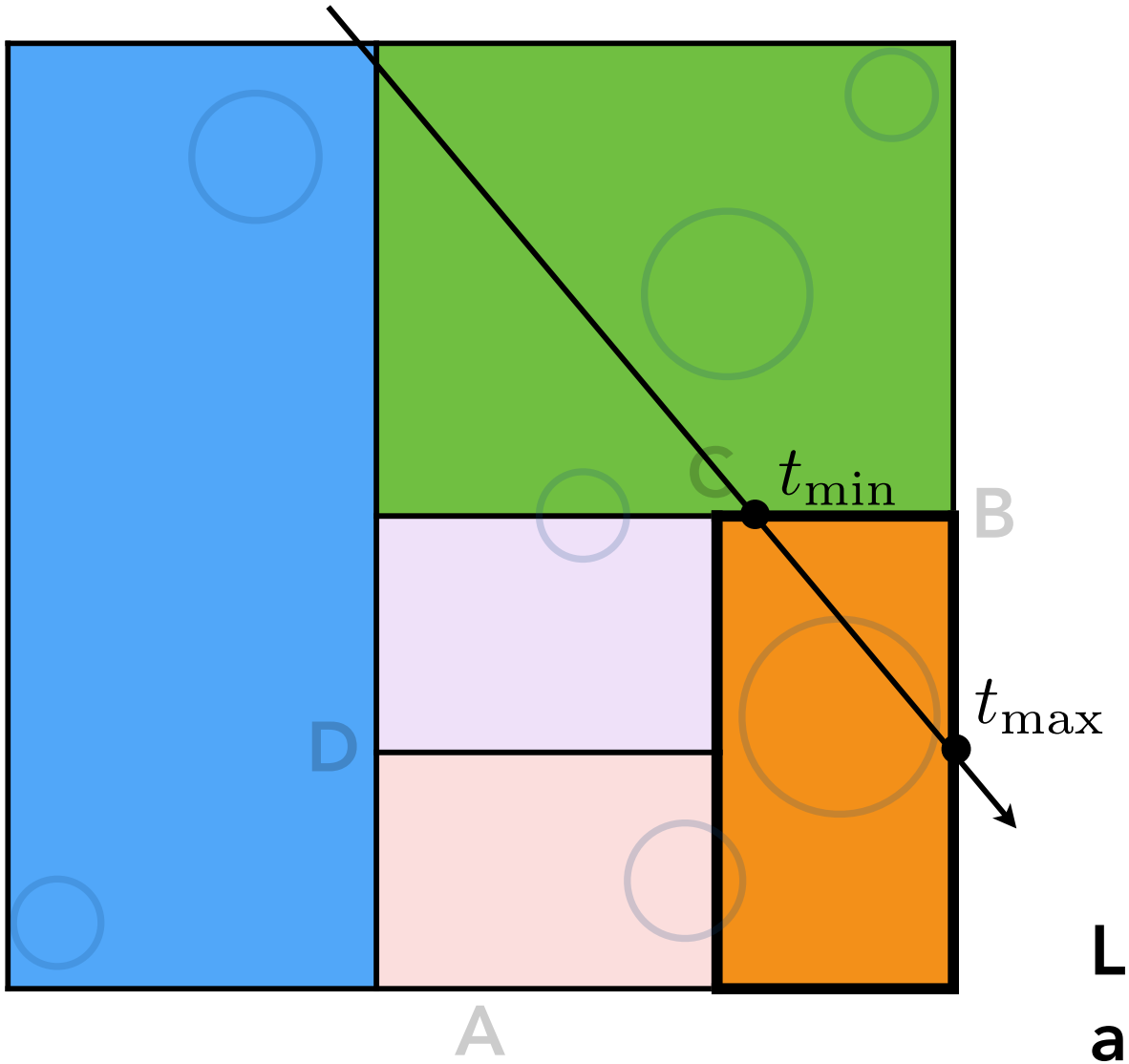
Top-Down Recursive In-Order Traversal

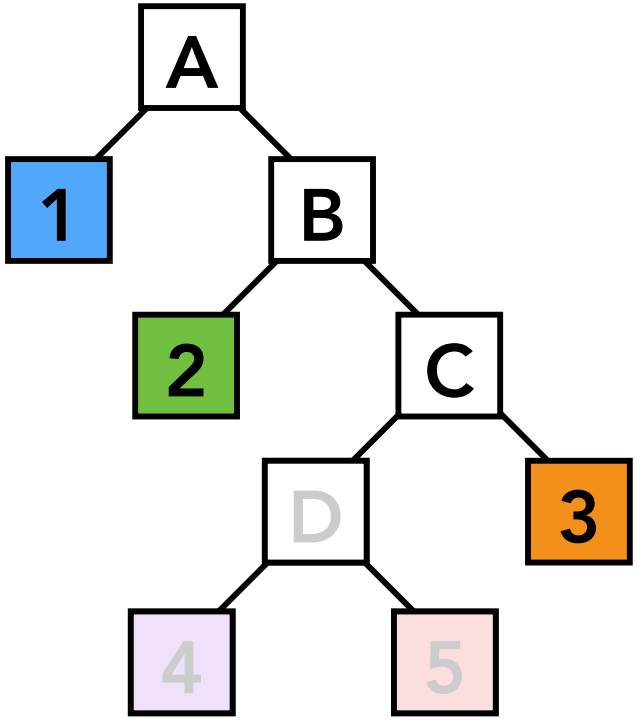




Internal node: split

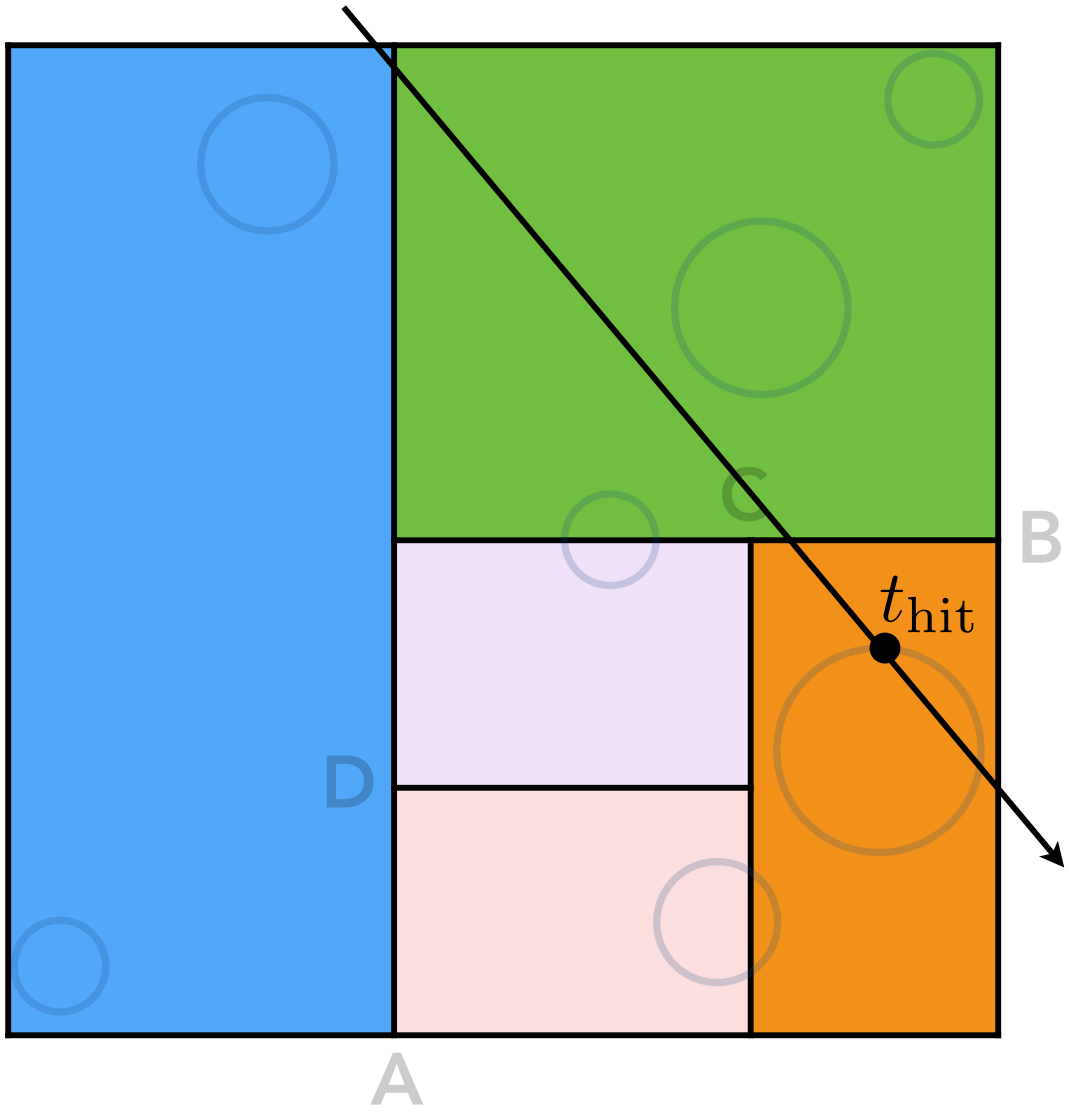
Top-Down Recursive In-Order Traversal

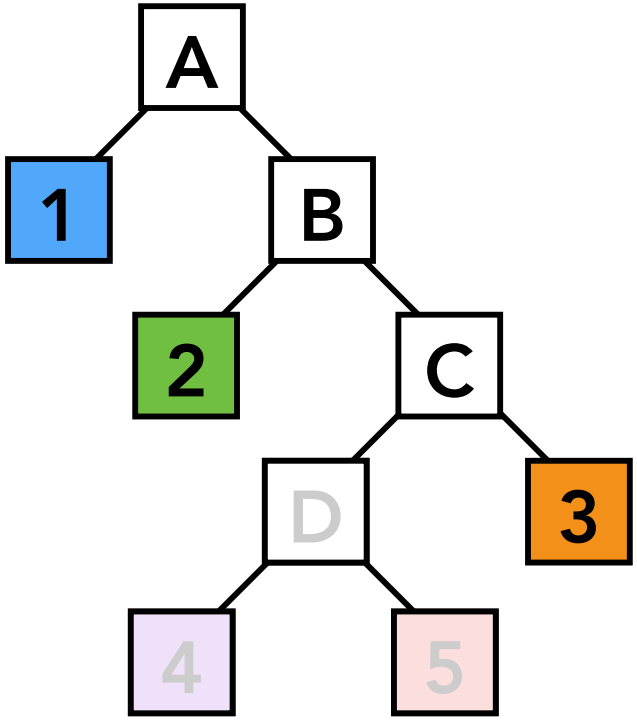




Leaf node: intersect all objects

Top-Down Recursive In-Order Traversal





Intersection found

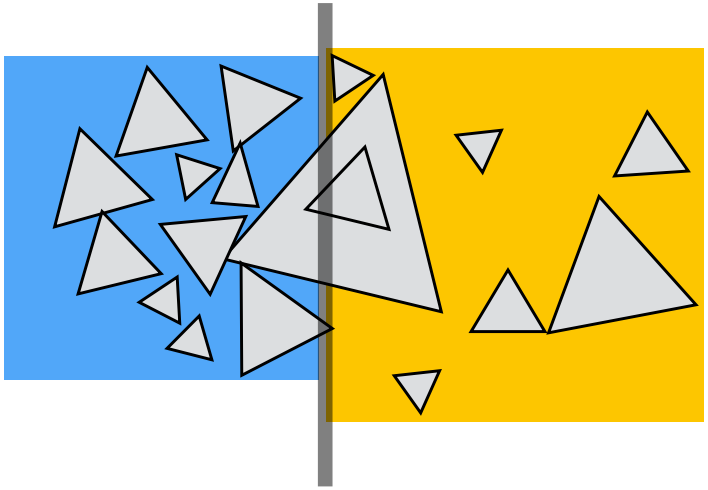
Object Partitions &

Bounding Volume Hierarchy (BVH)

Spatial vs Object Partitions

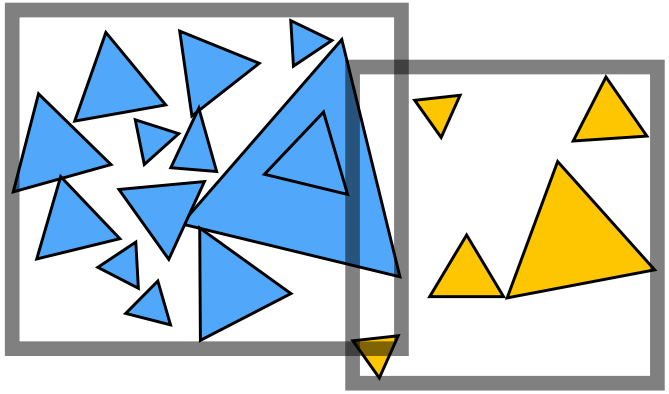
Spatial partition (e.g.KD-tree)

- Partition space into non-overlapping regions
- Objects can be contained in multiple regions

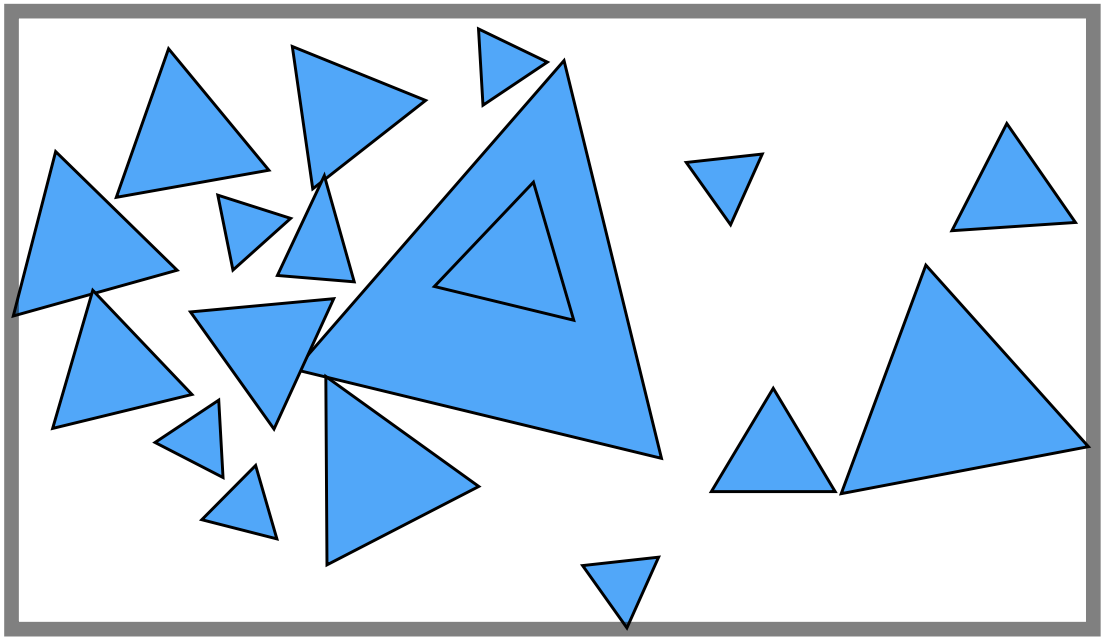


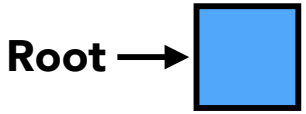
Object partition (e.g. BVH)

- Partition set of objects into disjoint subsets
- Bounding boxes for each set may overlap in space

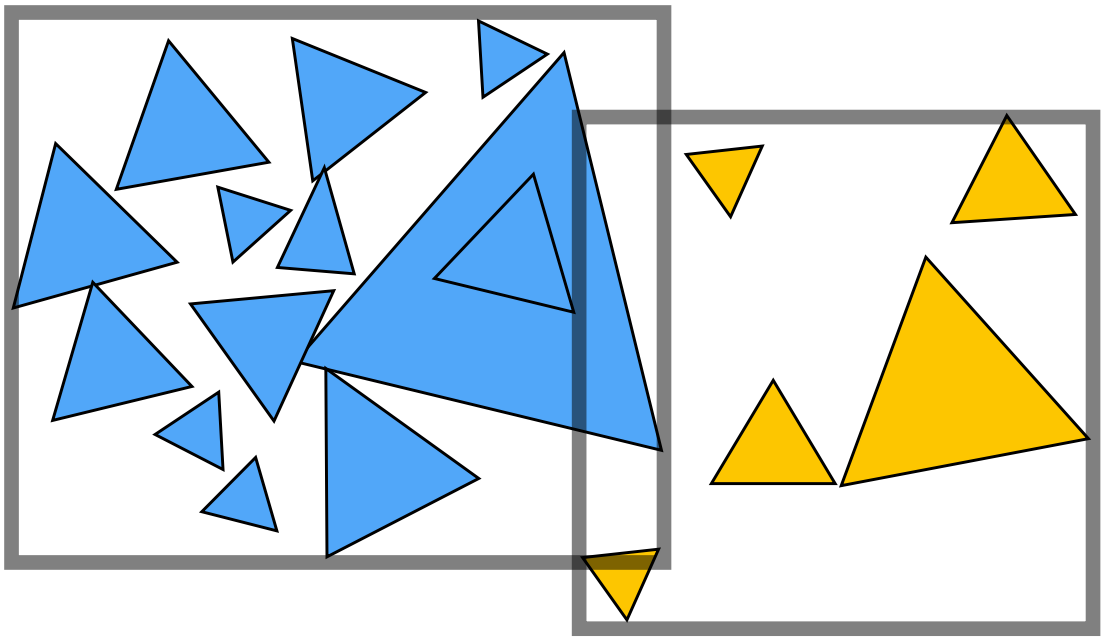


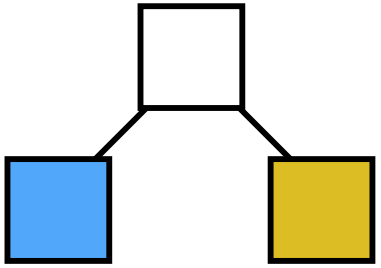
Bounding Volume Hierarchy (BVH)



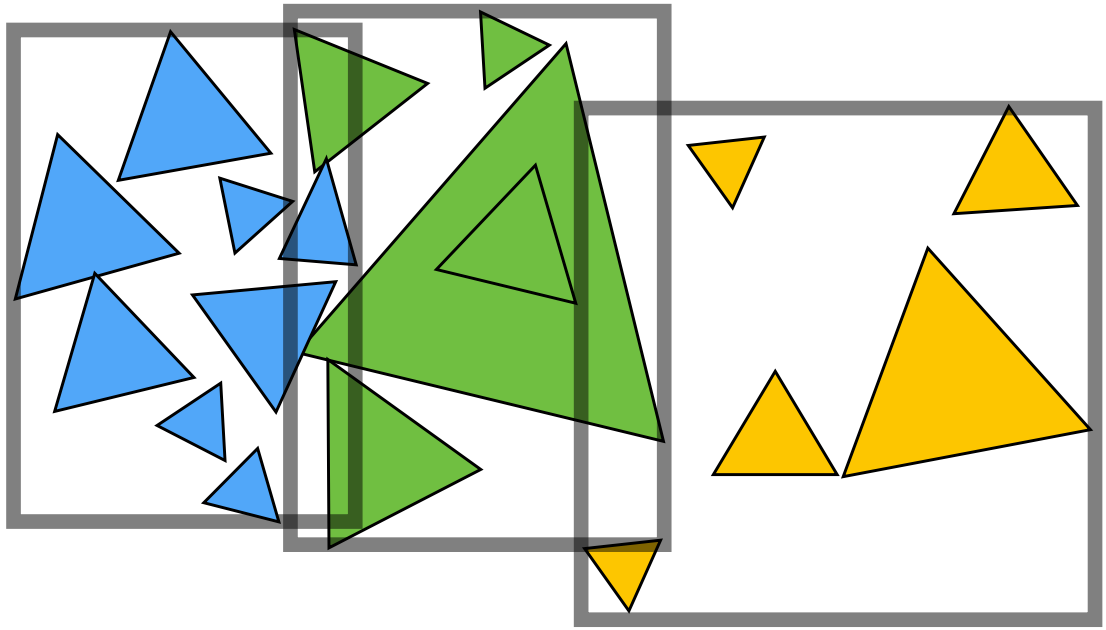


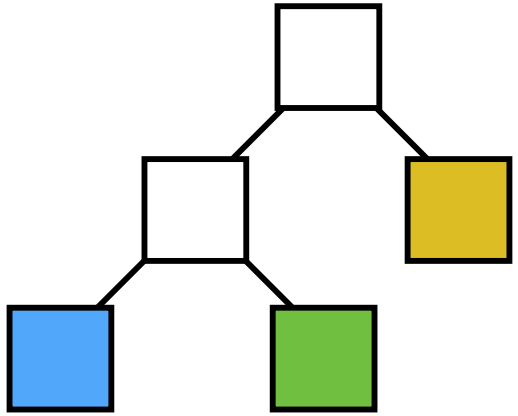
Bounding Volume Hierarchy (BVH)



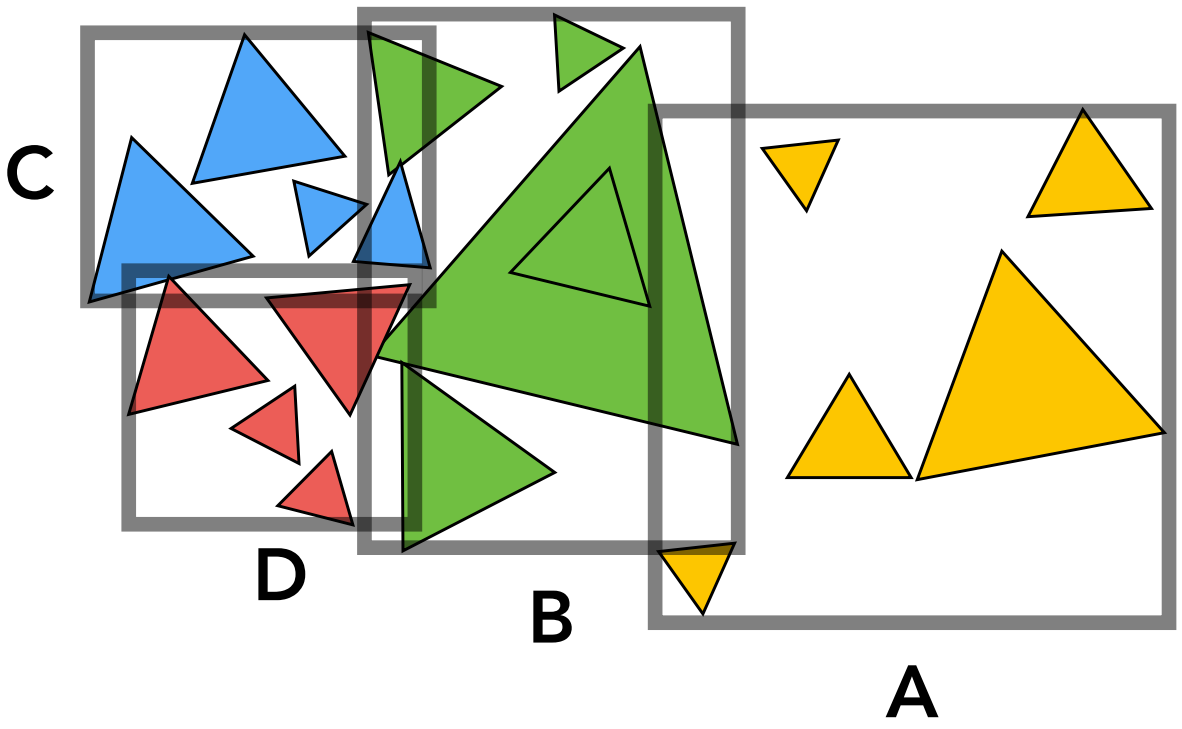


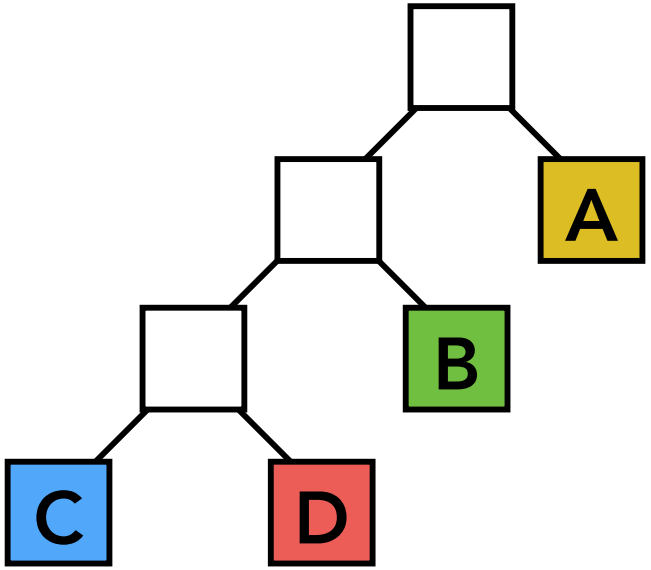
Bounding Volume Hierarchy (BVH)





Bounding Volume Hierarchy (BVH)





Bounding Volume Hierarchy (BVH)

Internal nodes store

- Bounding box
- Children: reference to child nodes

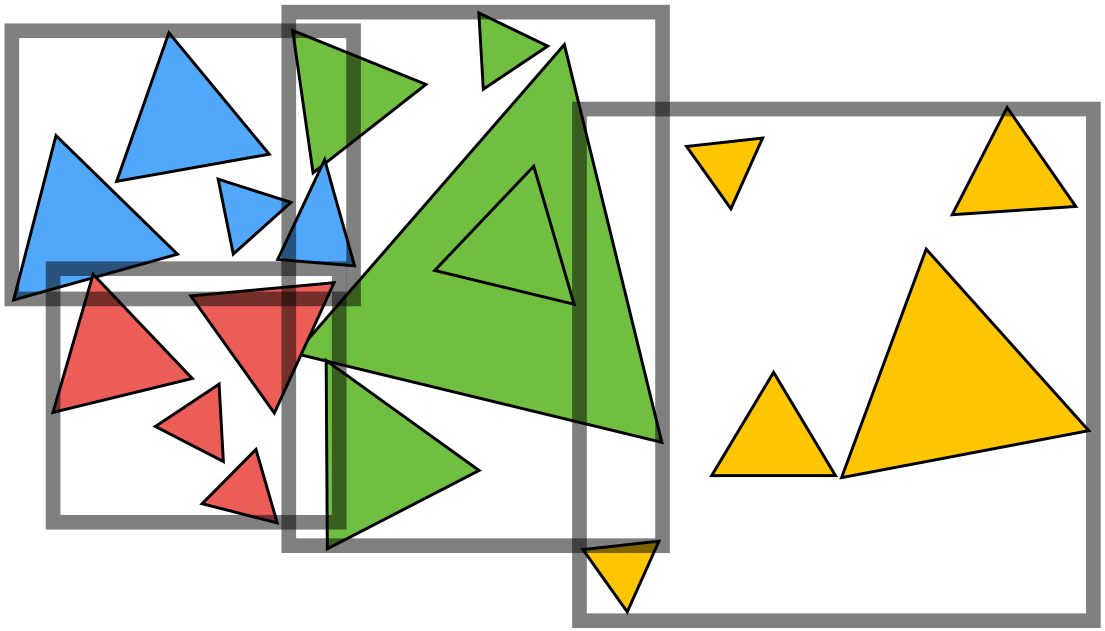
Leaf nodes store

- Bounding box
- List of objects

Nodes represent subset of primitives in scene

- All objects in subtree

BVH Pre-Processing



- Find bounding box
- Recursively split set of objects in two subsets
- Stop when there are just a few objects in each set
- Store obj reference(s) in each leaf node

BVH Pre-Processing

Choosing the set partition

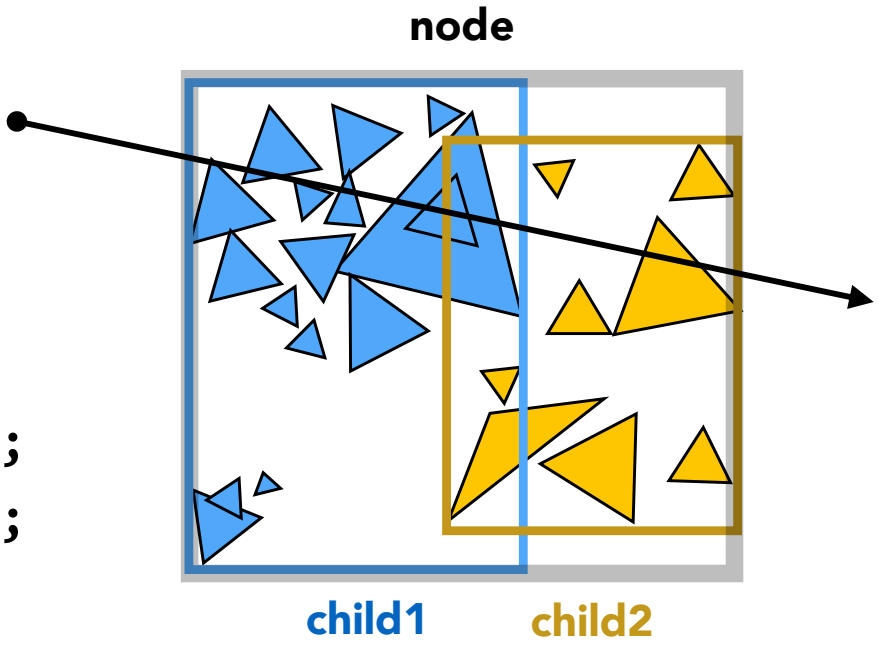
- Choose a spatial dimension to partition over (e.g. x,y,z)
- Simple #1: Split objects around spatial midpoint
- Simple #2: Split at location of median object
- Ideal: split to minimize expected cost of ray intersection

Termination criteria?

- Simple: stop when node contains few elements (e.g. 5)
- Ideal: stop when splitting does not reduce expected cost of ray intersection

BVH Recursive Traversal

```
Intersect (Ray ray, BVH node)
  if (ray misses node.bbox) return;
  if (node is a leaf node)
     test intersection with all objs;
     return closest intersection;
  hit1 = Intersect (ray, node.child1);
  hit2 = Intersect (ray, node.child2);
  return closer of hit1, hit2;
```



Summary of Spatial Acceleration Structures

- Primitive vs. spatial partitioning:
  - Primitive partitioning: partition sets of objects
    - Bounded number of BVH nodes, simpler to update if primitives in scene change position
  - Spatial partitioning: partition space
    - Traverse space in order (first intersection is closest intersection), may intersect primitive multiple times
- Adaptive structures (BVH, K-D tree)
  - More costly to construct (must be able to amortize cost over many geometric queries)
  - Better intersection performance under non-uniform distribution of primitives
- Non-adaptive accelerations structures (uniform grids)
  - Simple, cheap to construct
  - Good intersection performance if scene primitives are uniformly distributed
- Many, many combinations thereof...

Next Time: Monte Carlo Ray Tracing