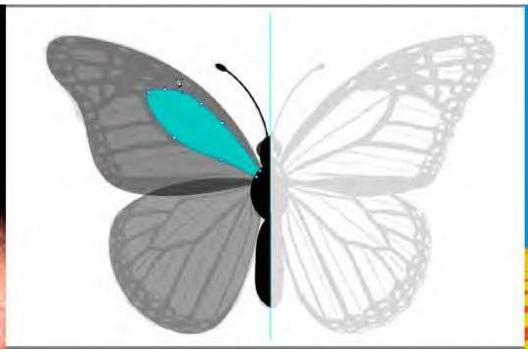
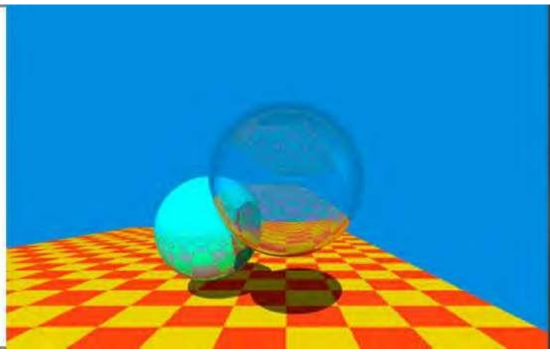
Computer Graphics

Ray Tracing 1
(Whitted-Style Ray Tracing)

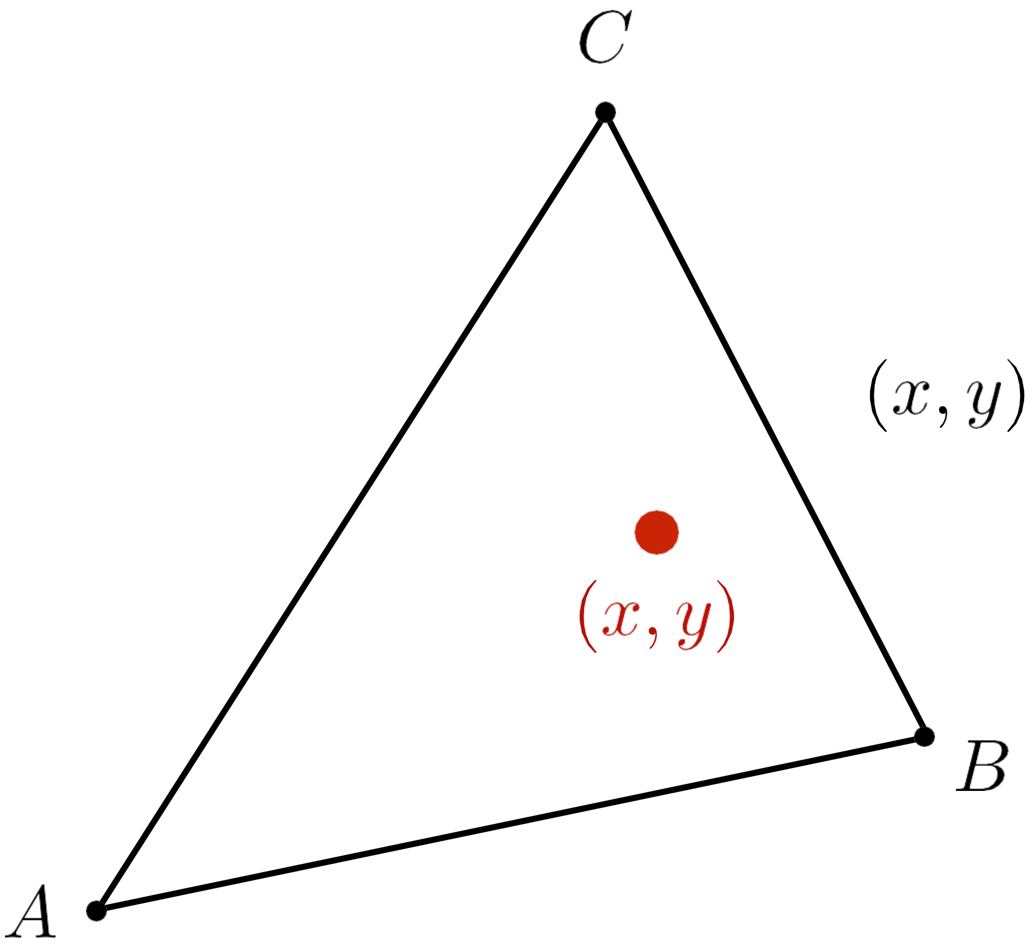





Last Lectures

- Barycentric Coordinates
- Applying Textures
- Texture Magnification
  - Too small?
  - Too larger?
- Mipmap
- Applications of Textures
  - Environment Map
  - Bump / normal mapping
  - Displacement mapping
  - Precomputed Shading

Barycentric Coordinates

A coordinate system for triangles (α, β, γ)



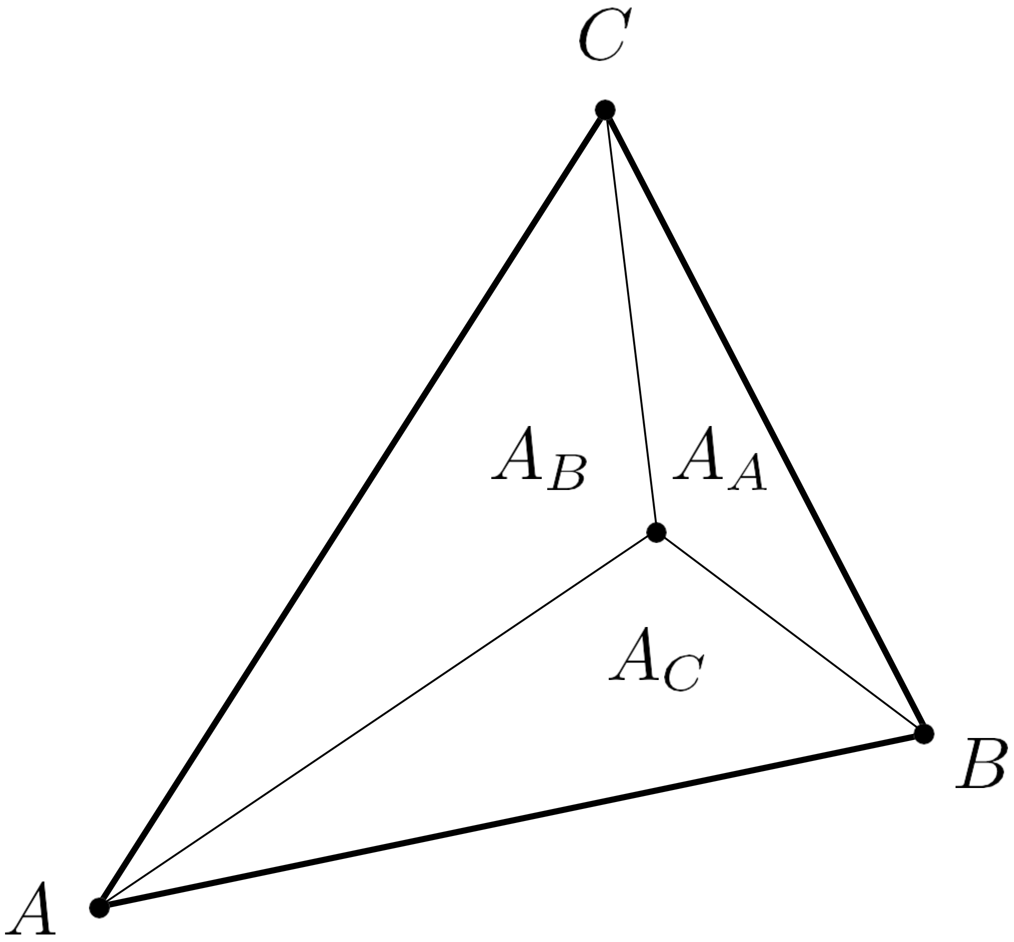
$$(x, y) = \alpha\, A + \beta\, B + \gamma\, C$$
$$\alpha + \beta + \gamma = 1$$

Inside the triangle if all three coordinates are non-negative

https://www.inf.usi.ch/hormann/barycentric/index.html

Barycentric Coordinates

Geometric viewpoint — proportional areas



$$\alpha = \frac{A_A}{A_A + A_B + A_C}$$

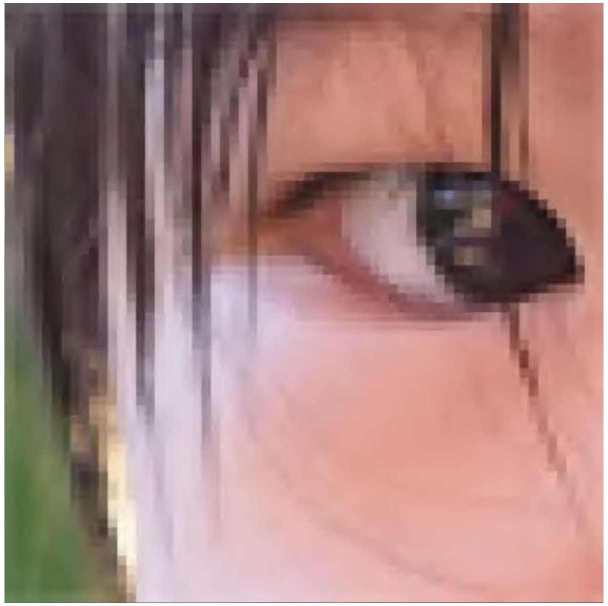
$$\beta = \frac{A_B}{A_A + A_B + A_C}$$

$$\gamma = \frac{A_C}{A_A + A_B + A_C}$$

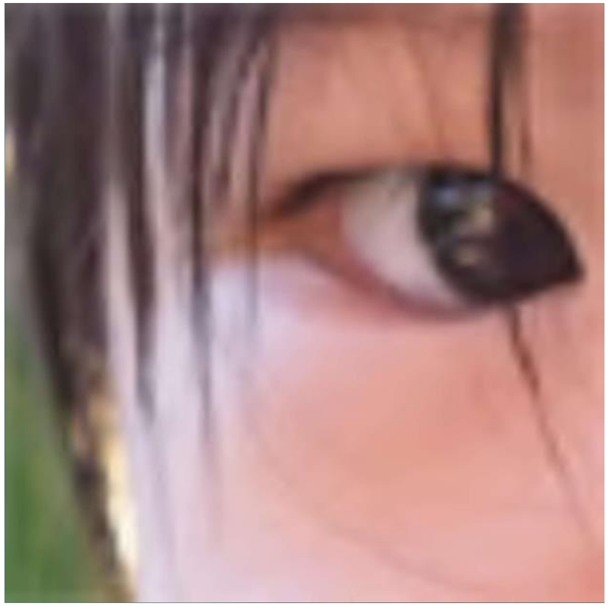
Texture Magnification - Easy Case

Generally don’t want this — insufficient texture resolution

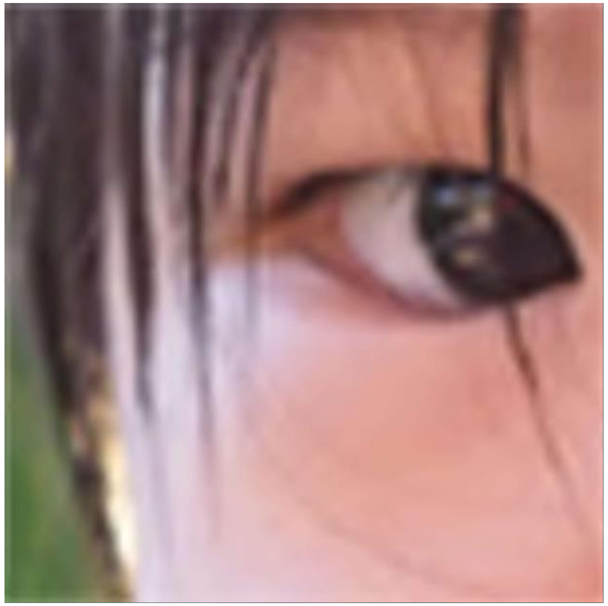
A pixel on a texture — a texel (纹理元素、纹素)



Nearest

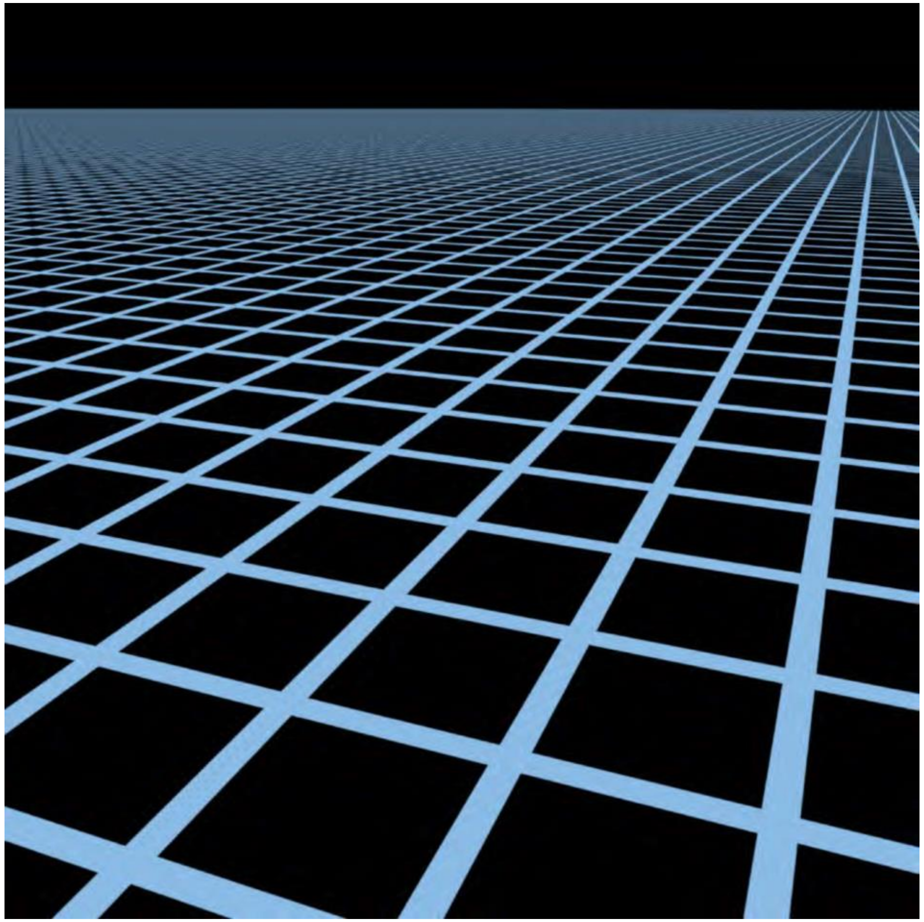


Bilinear

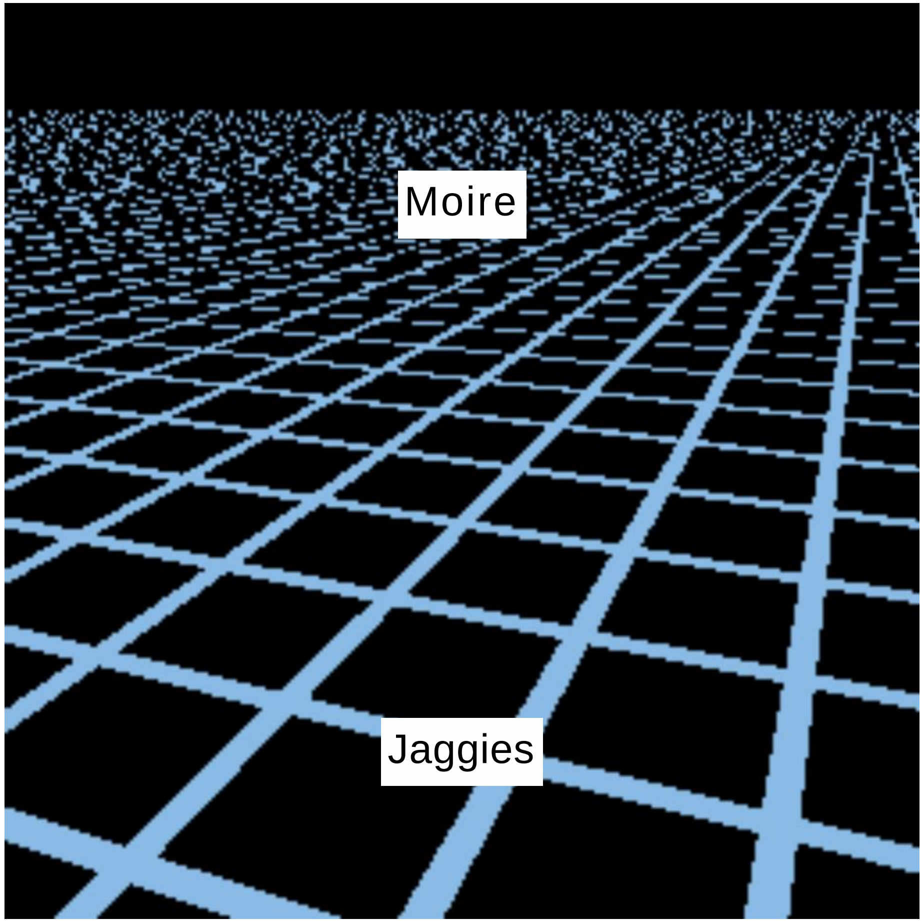


Bicubic

Point Sampling Textures — Problem

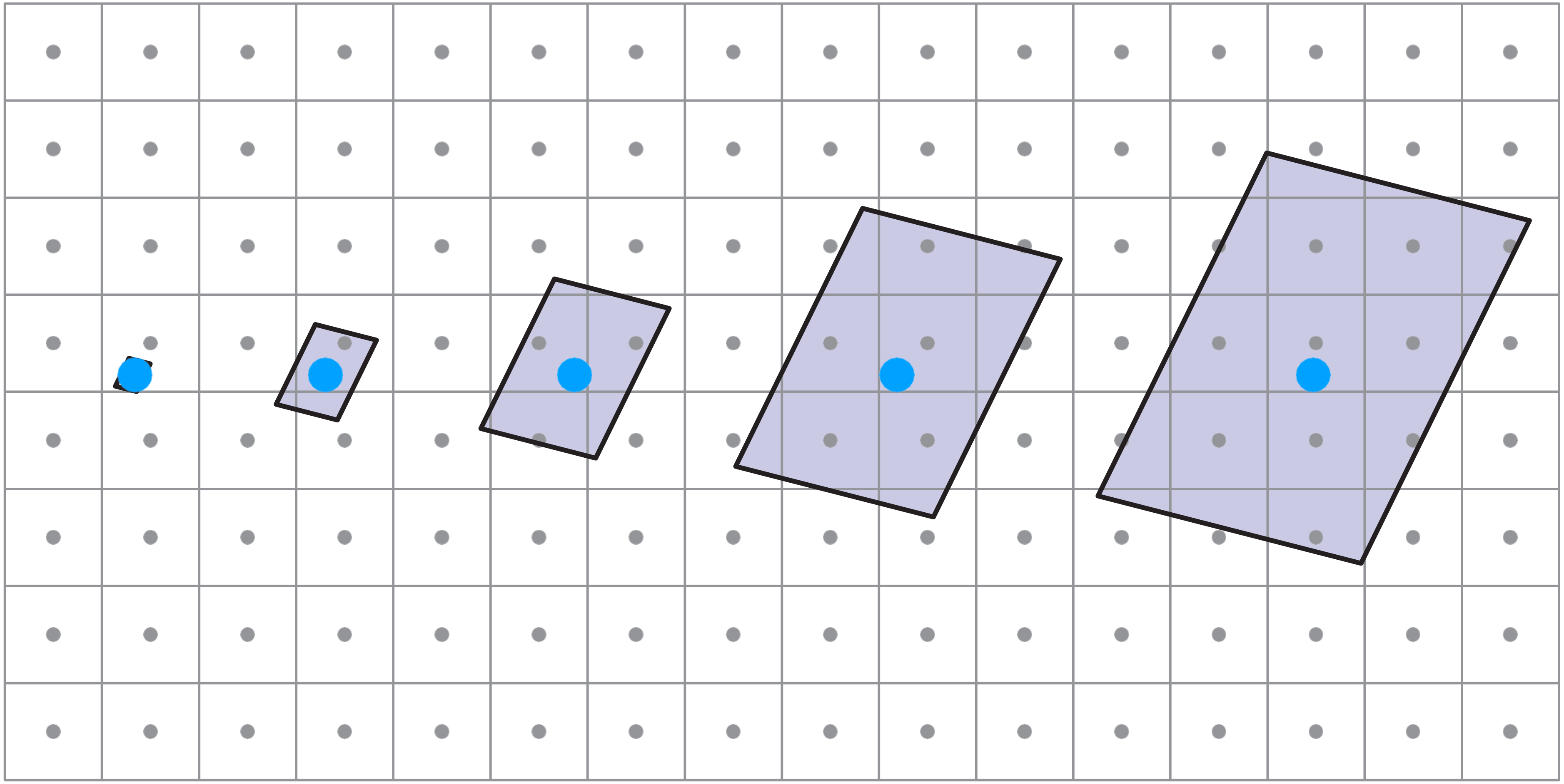


Reference



Point sampled

Screen Pixel “Footprint” in Texture



Upsampling
(Magnification)

Downsampling
(Minification)

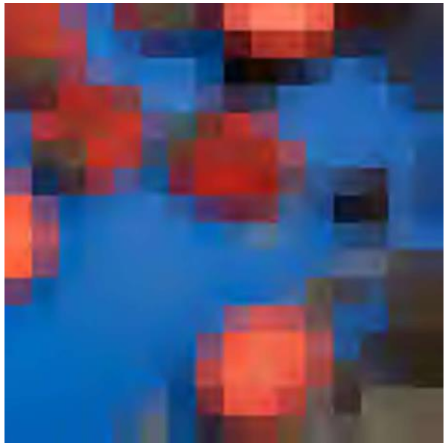
Mipmap (L. Williams 83)

“Mip” comes from the Latin “multum in parvo", meaning a multitude in a small space

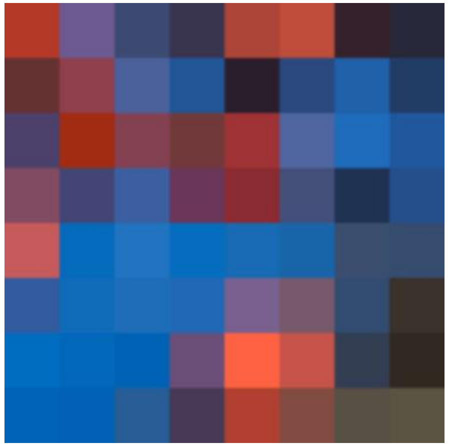
Level 0 = 128x128

Level 1 = 64x64

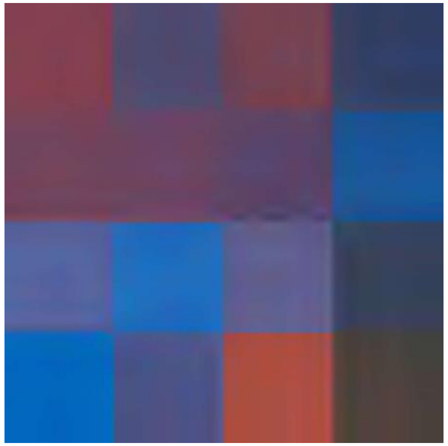
Level 2 = 32x32



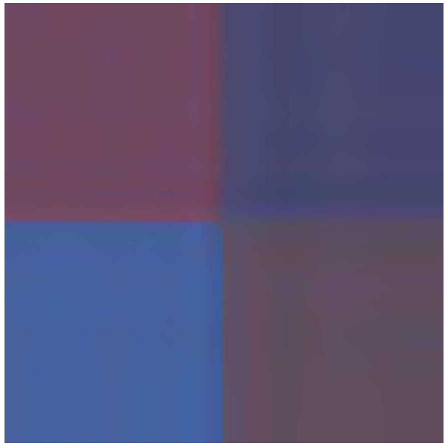
Level 3 = 16x16



Level 4 = 8x8



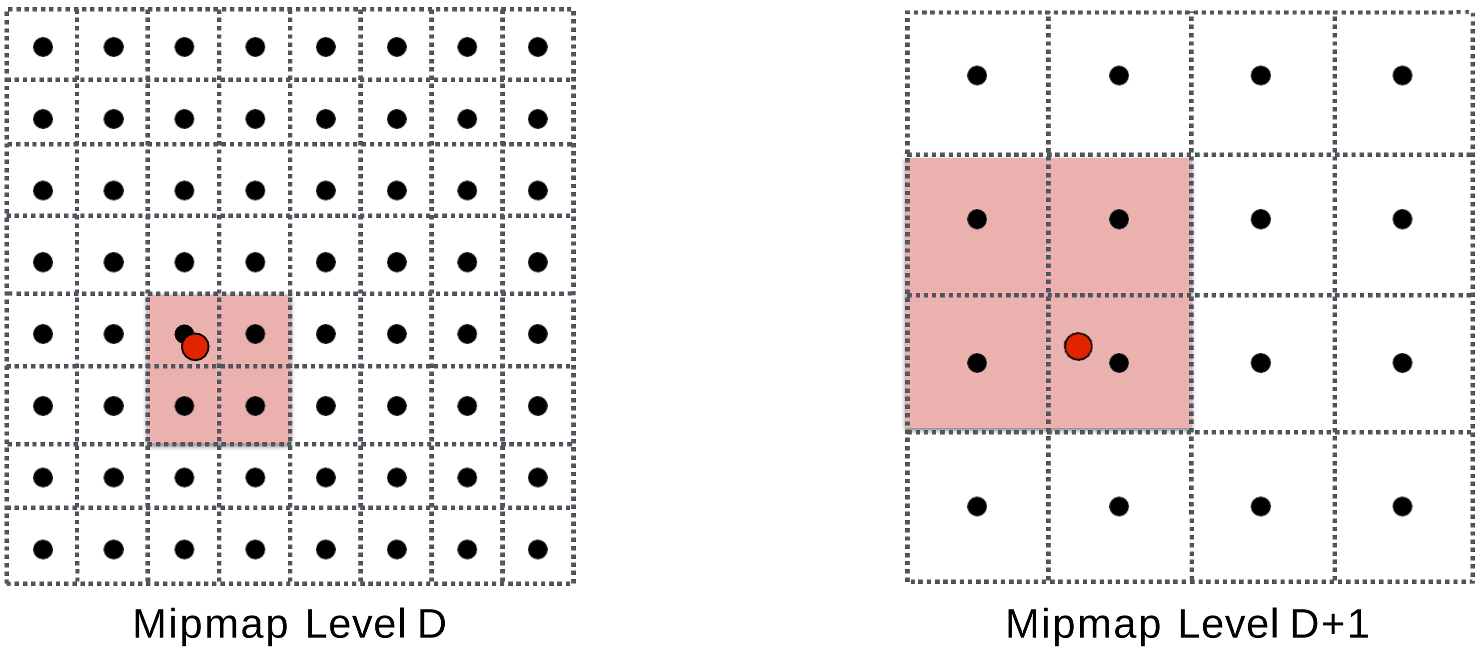
Level 5 = 4x4



Level 6 = 2x2

Level 7 = 1x1

Trilinear Interpolation



Bilinear result Bilinear result

Linear interpolation based on continuous D value

Environment Map

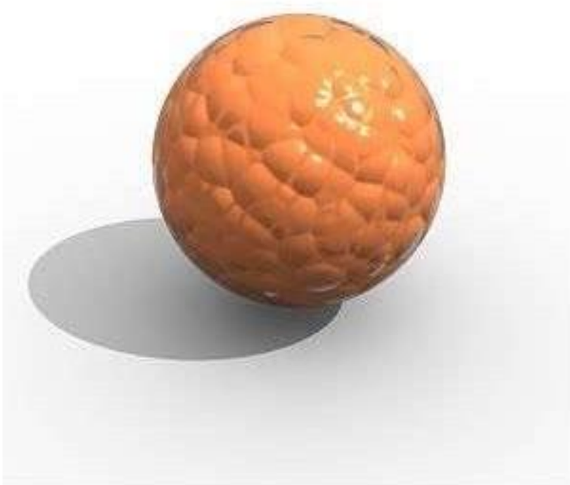
Light from the environment

Rendering with the environment

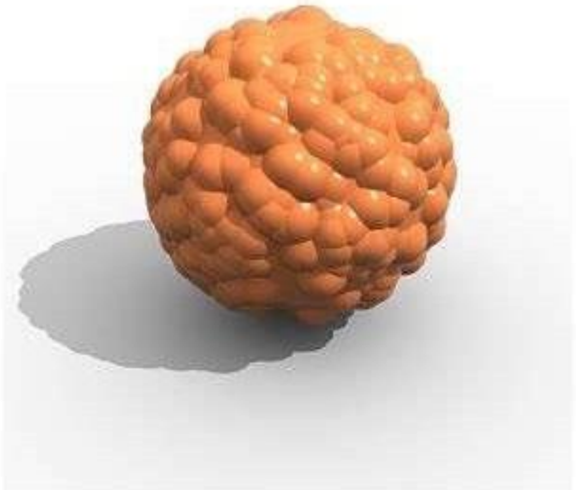
[Blinn & Newell 1976]

Textures can affect shading!

- Displacement mapping — a more advanced approach
  - Uses the same texture as in bumping mapping
  - Actually moves the vertices



Bump / Normal mapping



Displacement mapping

多选题 1分

设置

重心坐标可以用于插值哪些信息?

- A Depth
- B Material attributes
- C RGB from Mipmap
- D Normal
- E Positions
- F UV coordinate

提交

单选题 1分

设置

Mipmap增加了多少存储空间?

A 1/4

B 1/3

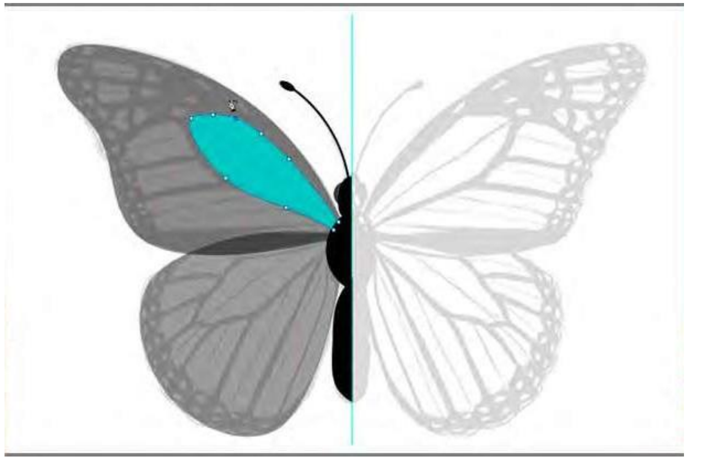
C 1/2

D 1

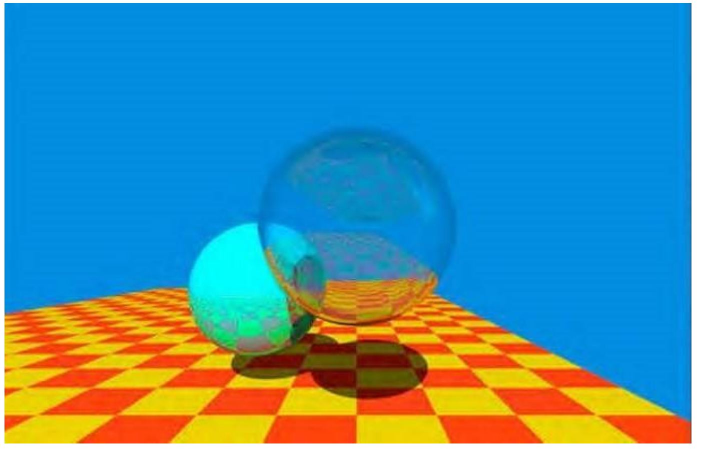
提交

Course Roadmap

Rasterization



Geometry



Ray tracing

Animation /simulation

Why Ray Tracing?

- Rasterization couldn't handle global effects well
  - (Soft) shadows
  - And especially when the light bounces more than once

Soft shadows

Glossy reflection

Indirect illumination

Why Ray Tracing?

- Rasterization is fast, but quality is relatively low

Buggy, from PlayerUnknown's Battlegrounds (PC game)

Why Ray Tracing?

- Ray tracing is accurate, but is very slow
  - Rasterization: real-time, ray tracing: offline
  - ~10K CPU core hours to render one frame in production

Zootopia, Disney Animation

Basic Ray-Tracing Algorithm

Light Rays

Three ideas about light rays

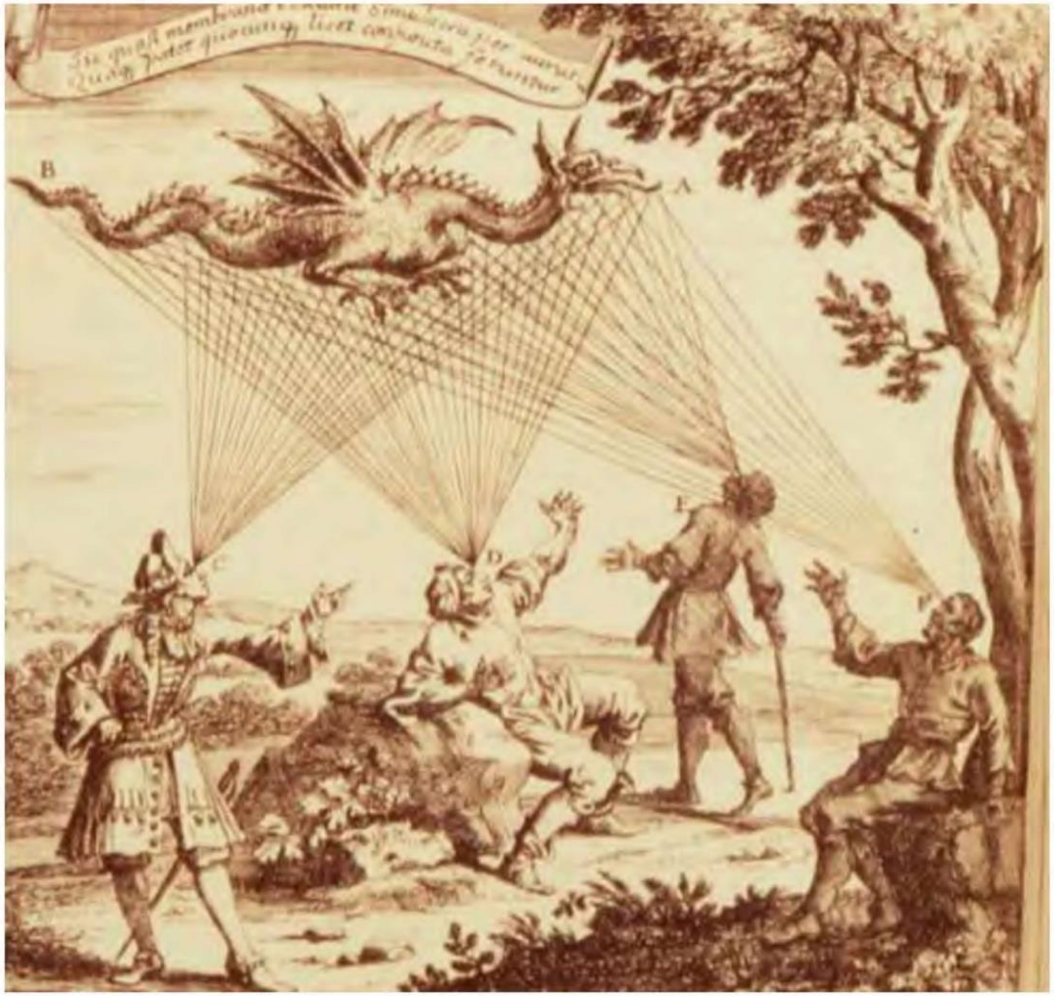
1. Light travels in straight lines (though this is wrong)
2. Light rays do not "collide" with each other if they cross (though this is still wrong)
3. Light rays travel from the light sources to the eye (but the physics is invariant under path reversal - reciprocity).

可逆性

"And if you gaze long into an abyss, the abyss also gazes into you." — Friedrich Wilhelm Nietzsche (translated)

Emission Theory of Vision

"For every complex problem there is an answer that is clear, simple, and wrong."
-- H. L. Mencken



Eyes send out "feeling rays" into the world

Supported by:
- Empedocles 恩培多克勒
- Plato 柏拉图
- Euclid (kinda) 欧几里得
- Ptolemy 托勒密
- …
- 50% of US college students*

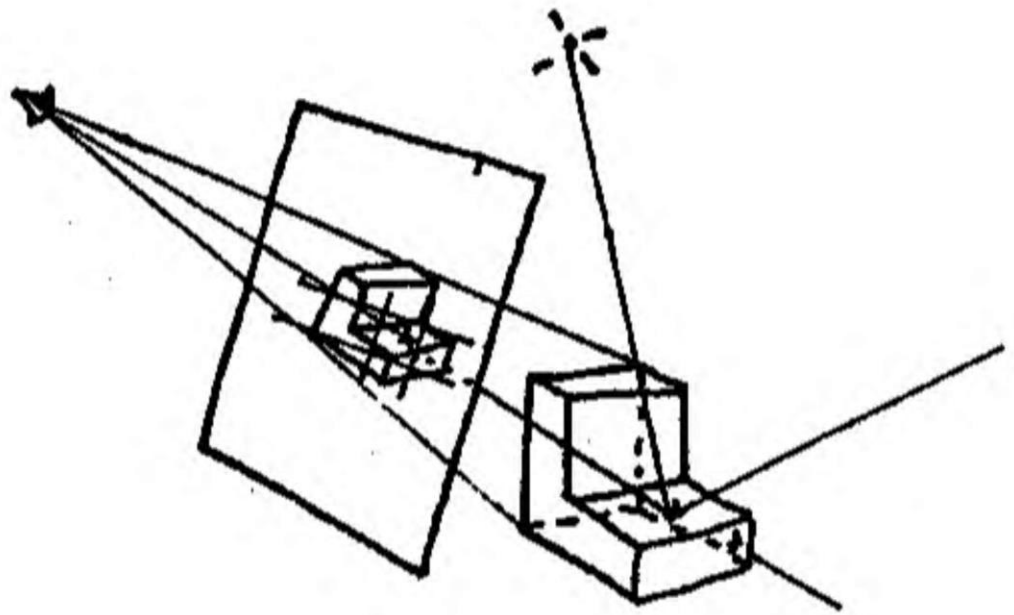
*http://www.ncbi.nlm.nih.gov/pubmed/12094435?dopt=Abstract

Slide courtesy of Prof. Alexei Efros, UC Berkeley

Ray Casting

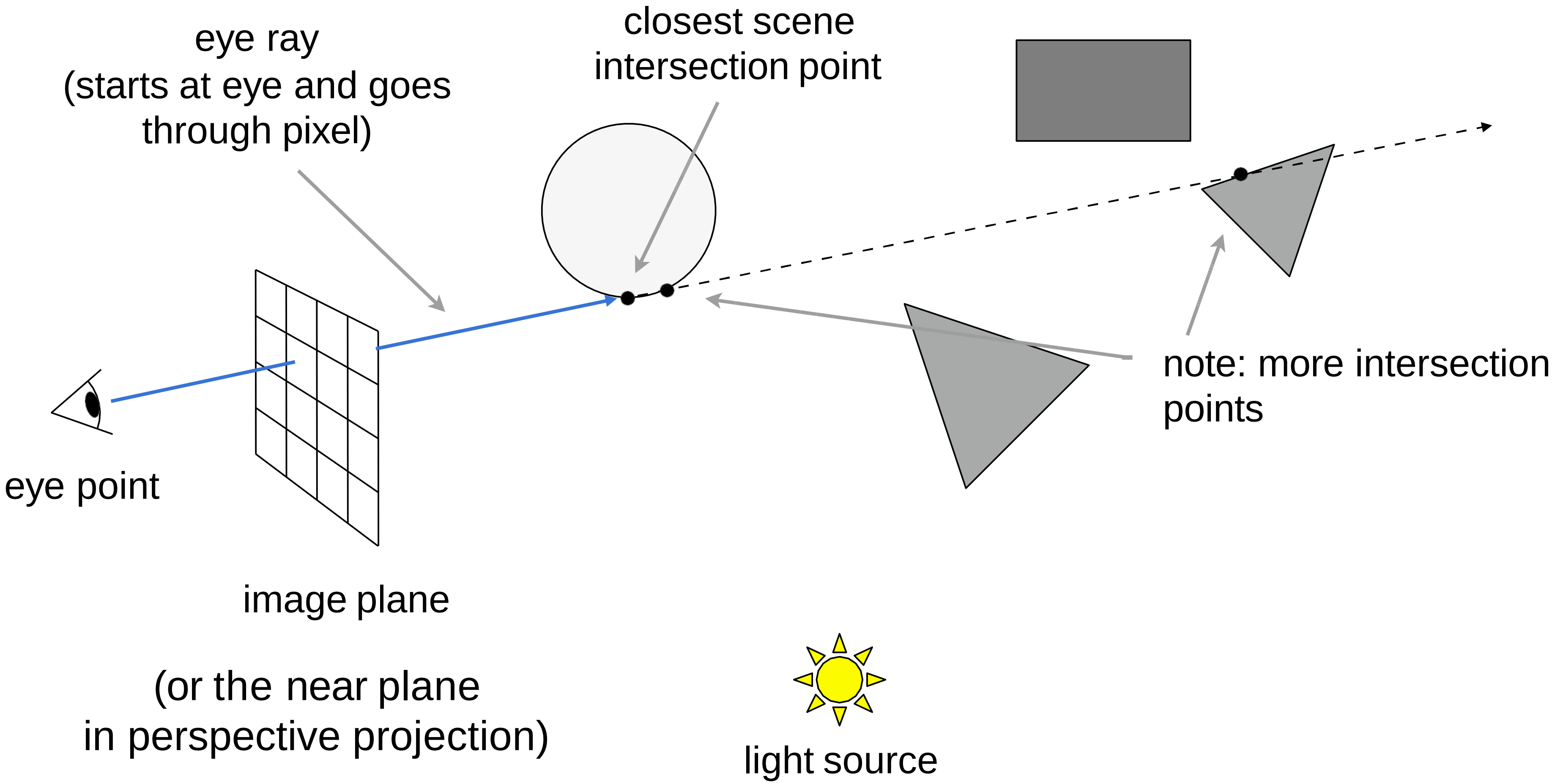
Appel 1968 - Ray casting

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



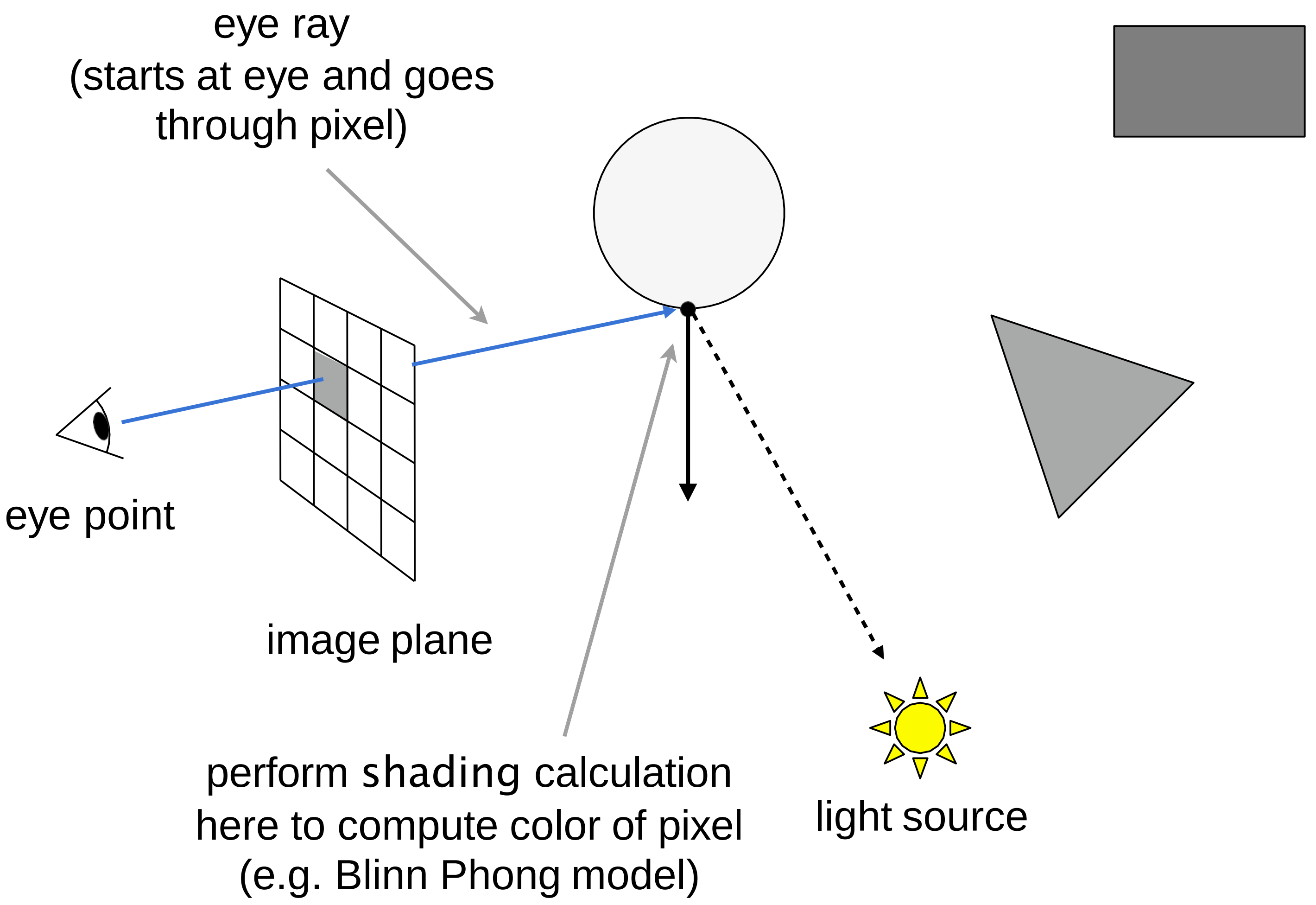
Ray Casting - Generating Eye Rays

Pinhole Camera Model



Ray Casting - Shading Pixels (Local Only)

Pinhole Camera Model

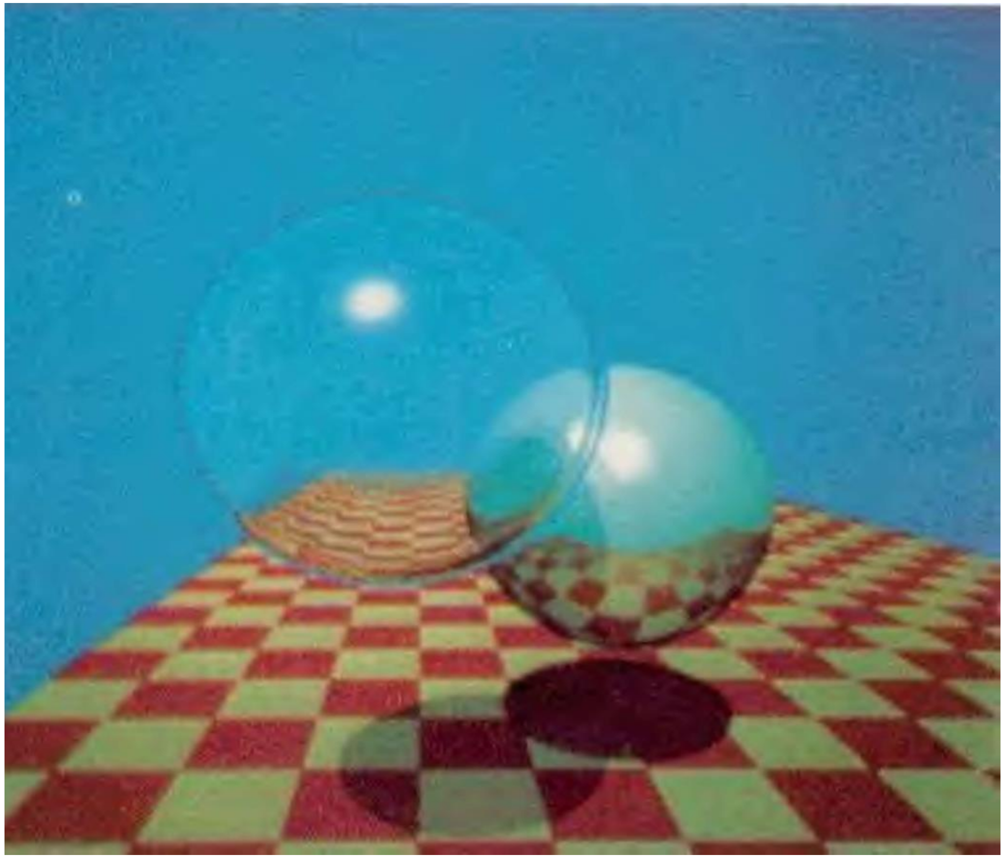


Recursive (Whitted-Style) Ray Tracing

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

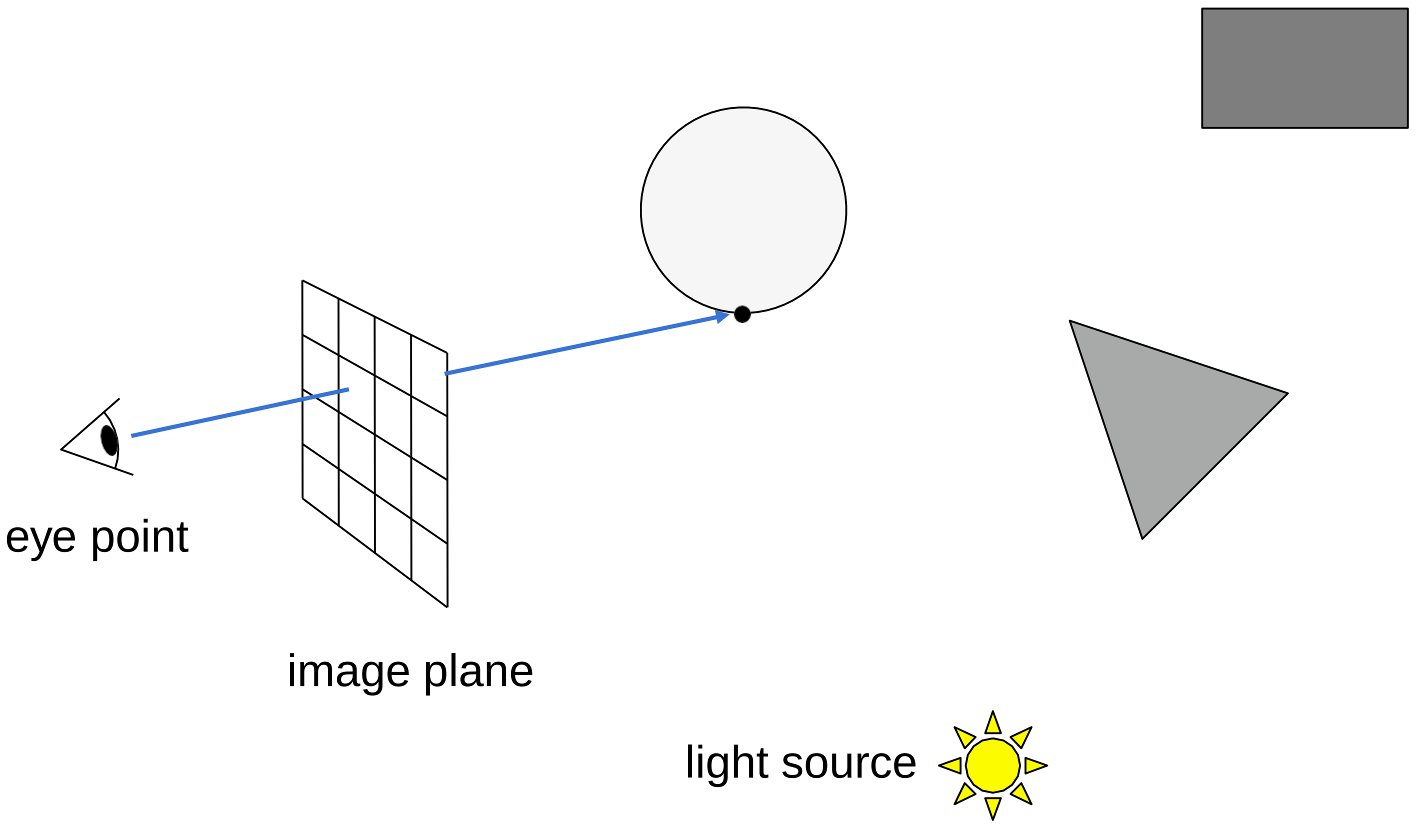
Time:

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

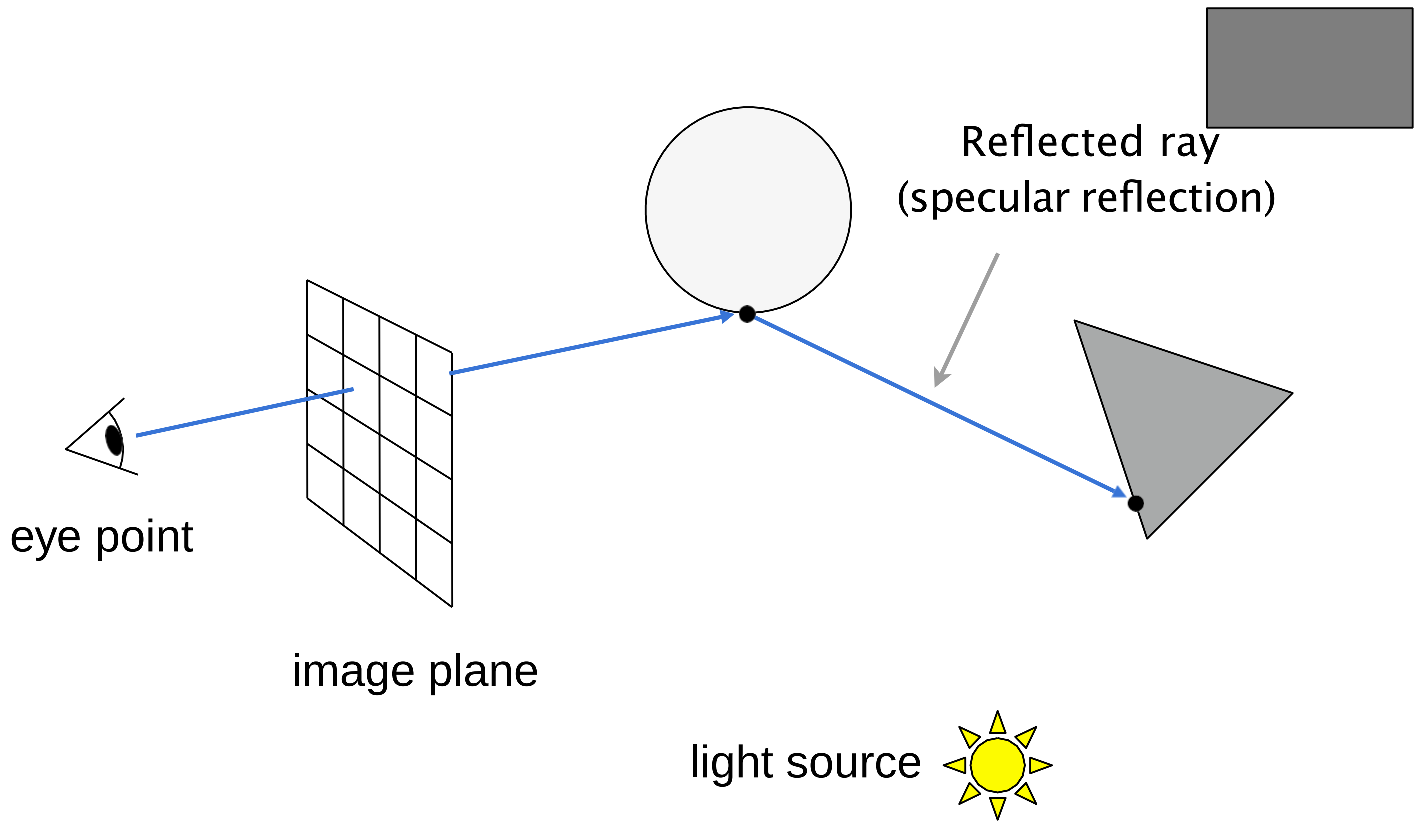


Spheres and Checkerboard, T. Whitted, 1979

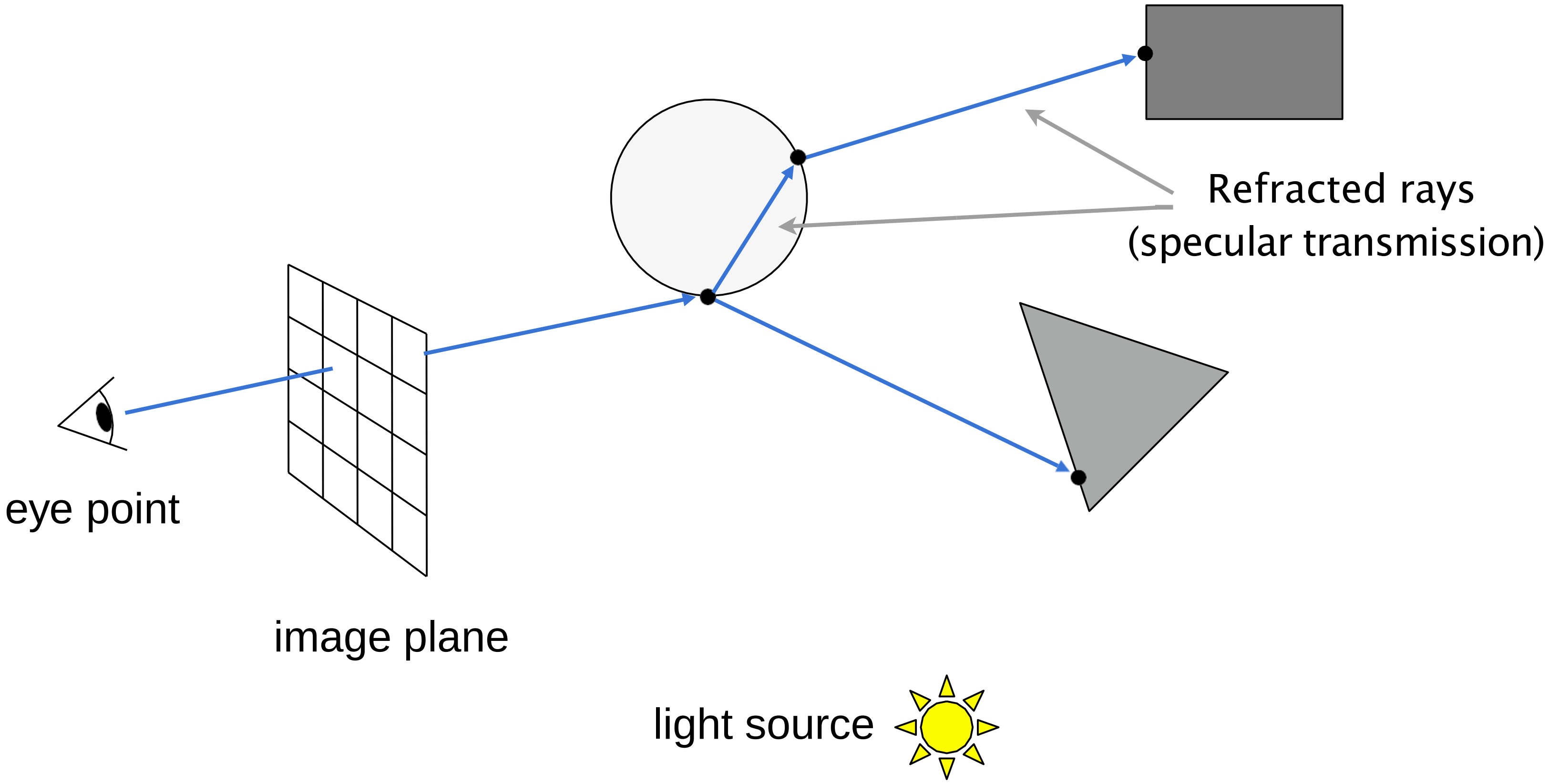
Recursive Ray Tracing



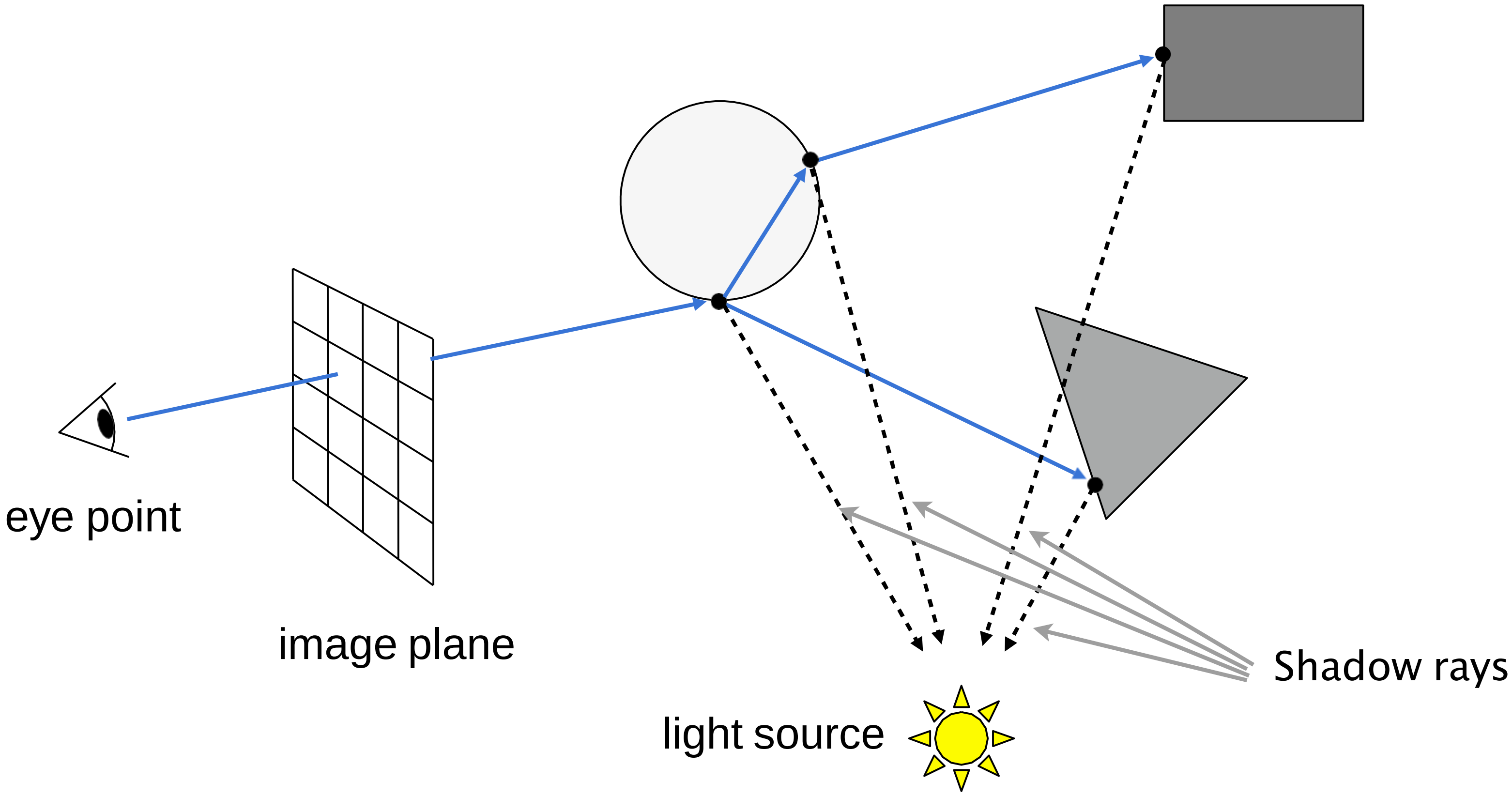
Recursive Ray Tracing



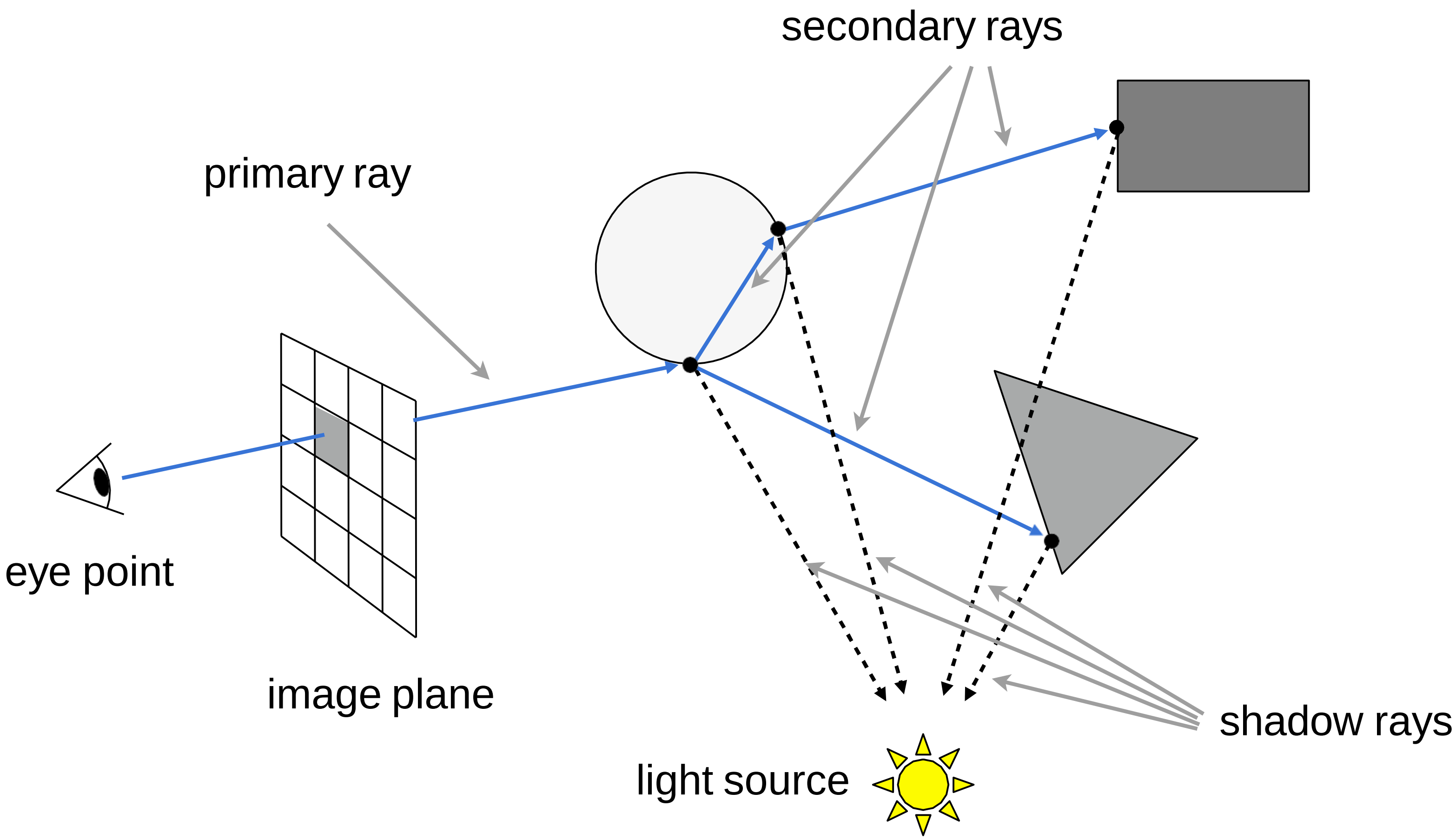
Recursive Ray Tracing



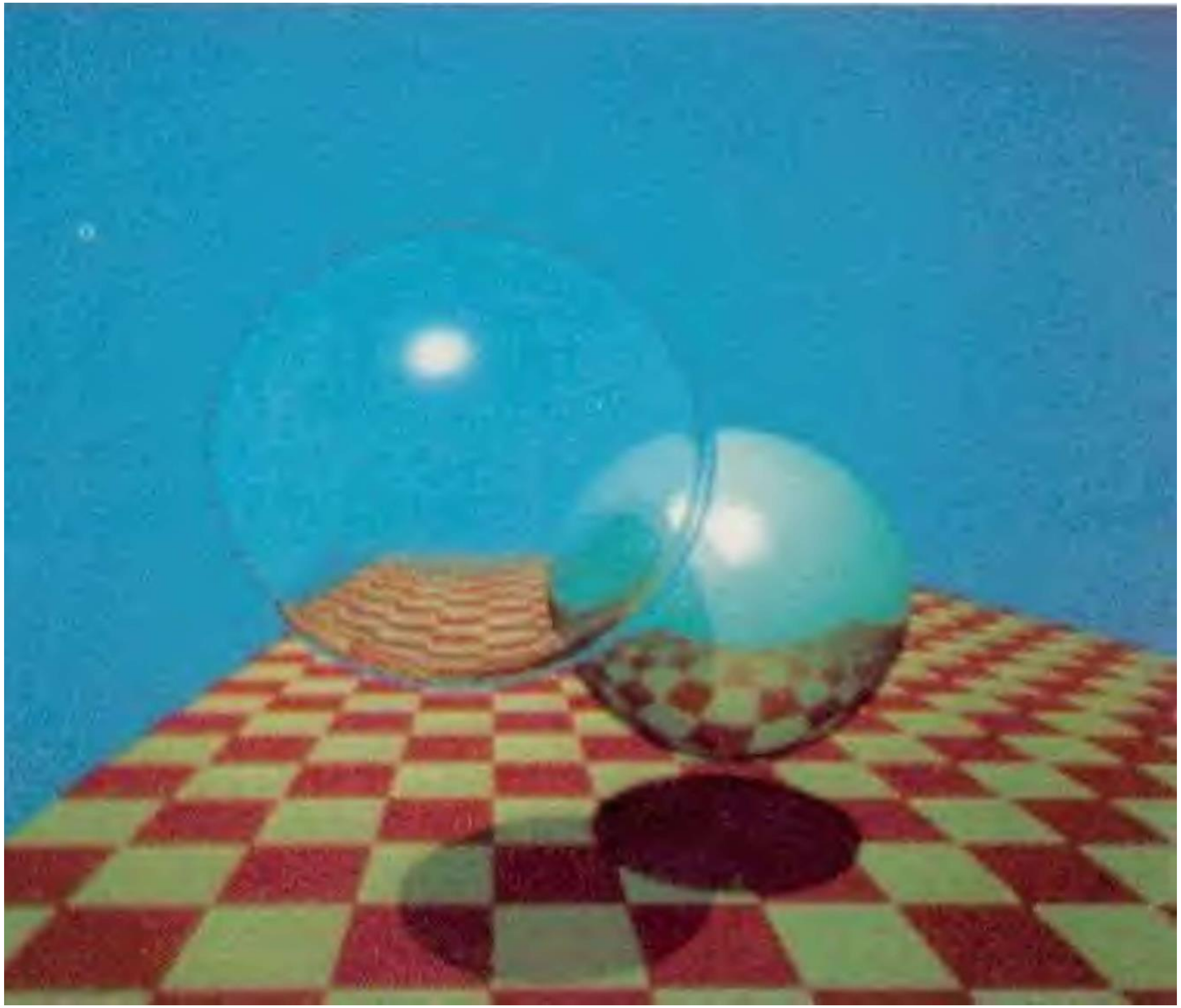
Recursive Ray Tracing



Recursive Ray Tracing



Recursive Ray Tracing

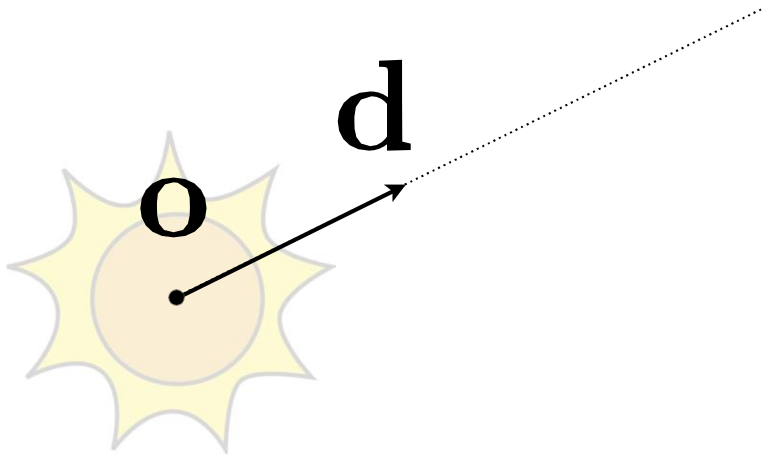


Ray-Surface Intersection

Ray Equation

Ray is defined by its origin and a direction vector

Example:



Ray equation:

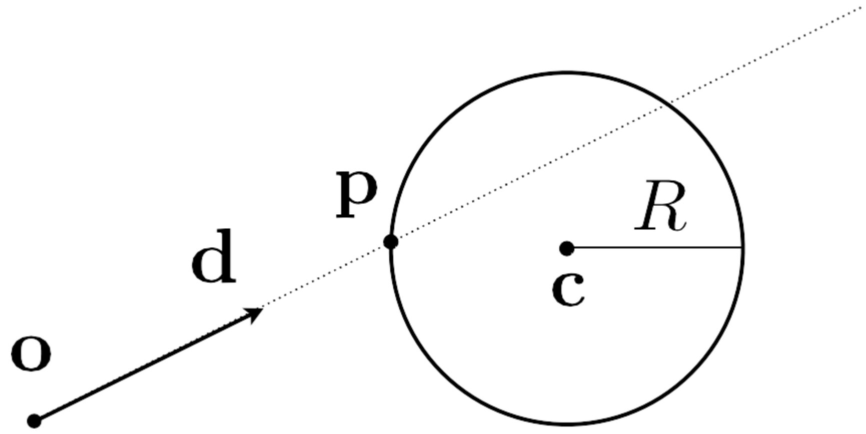
$$\mathbf{r}(t) = \mathbf{o} + t\mathbf{d} \qquad 0 \leq t < \infty$$

point along ray — "time" — origin — (normalized) direction

Ray Intersection With Sphere

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

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



What is an intersection?

The intersection p must satisfy both ray equation and sphere equation

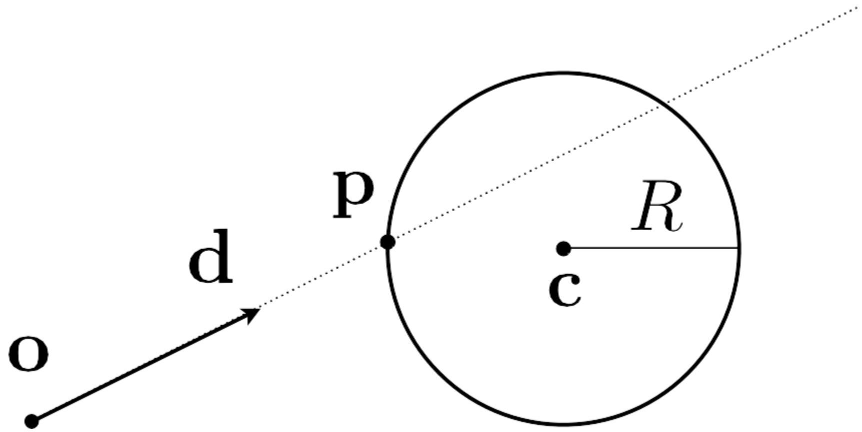
Solve for intersection:

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

Ray Intersection With Sphere

Solve for intersection:

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



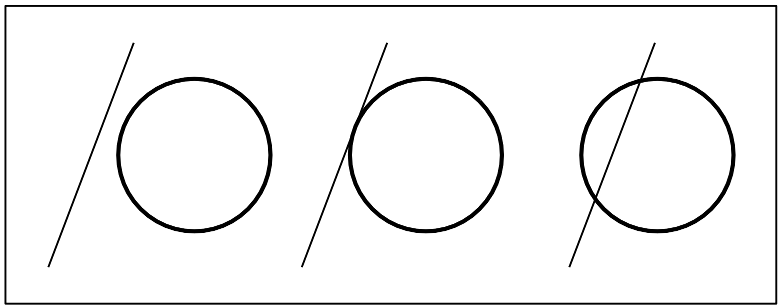
$$a\,t^2 + b\,t + c = 0, \text{ where}$$

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

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

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

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



Ray Intersection With Implicit Surface

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

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

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

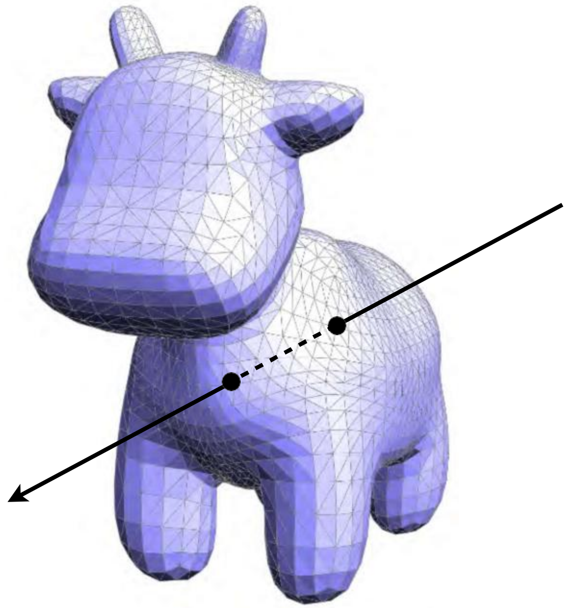
Solve for real, positive roots

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

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

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

Ray Intersection With Triangle Mesh



Why?

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

How to compute?

Let's break this down:

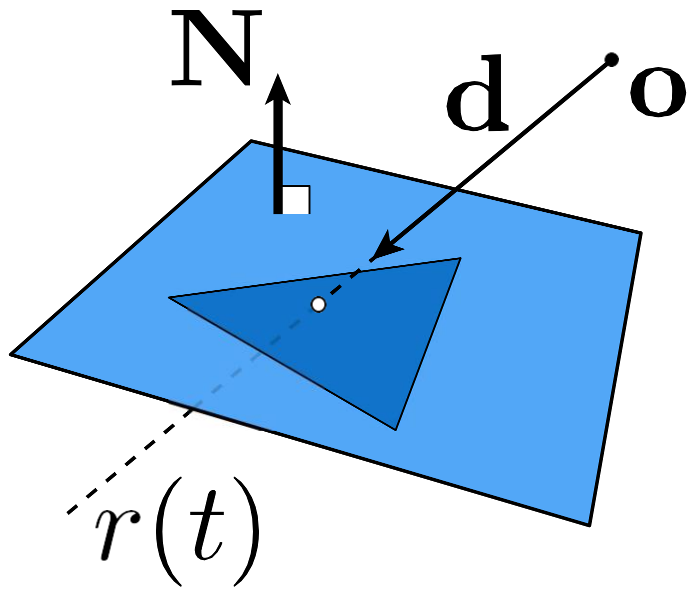
- Simple idea: just intersect ray with each triangle
- Simple, but slow (acceleration?)
- Note: can have 0, 1 intersections (ignoring multiple intersections)

Ray Intersection With Triangle

Triangle is in a plane

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

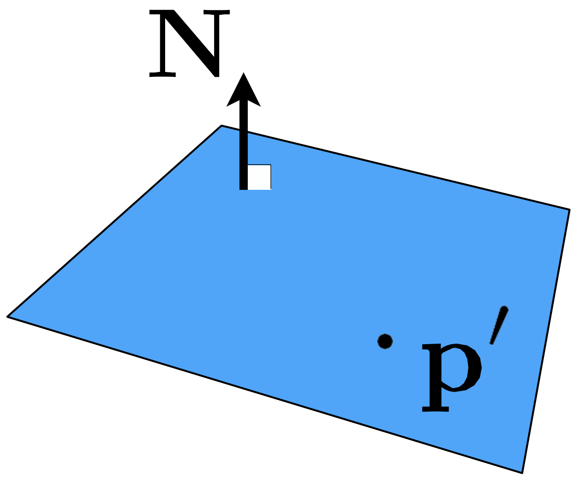
Many ways to optimize…



Plane Equation

Plane is defined by normal vector and a point on plane

Example:



Plane Equation (if p satisfies it, then p is on the plane):

$$\mathbf{p} : (\mathbf{p} - \mathbf{p}') \cdot \mathbf{N} = 0 \qquad ax + by + cz + d = 0$$

all points on plane

one point on plane

normal vector

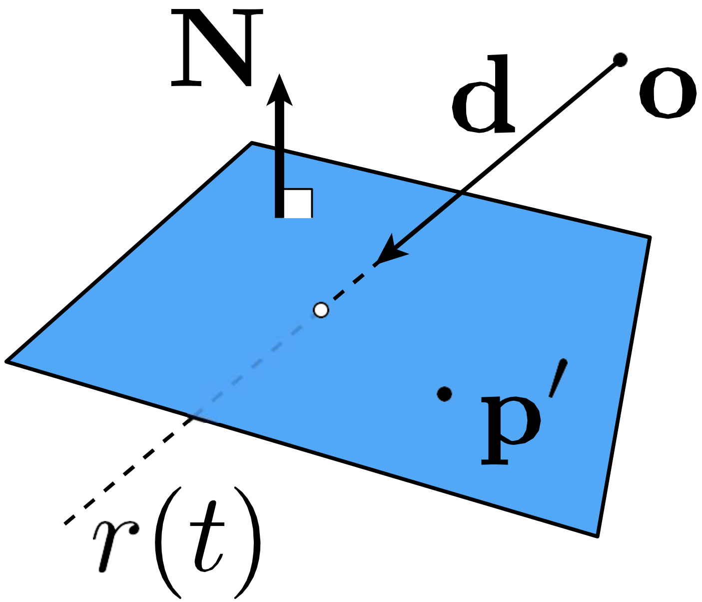
Ray Intersection With Plane

Ray equation:

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

Plane equation:

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



Solve for intersection

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

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

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

Check: $0 \leq t < \infty$

Möller Trumbore Algorithm

A faster approach, giving barycentric coordinate directly

Derivation in the discussion section!

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

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

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

Where:

$$\vec{\mathbf{E}}_1 = \vec{\mathbf{P}}_1 - \vec{\mathbf{P}}_0$$
$$\vec{\mathbf{E}}_2 = \vec{\mathbf{P}}_2 - \vec{\mathbf{P}}_0$$
$$\vec{\mathbf{S}} = \vec{\mathbf{O}} - \vec{\mathbf{P}}_0$$
$$\vec{\mathbf{S}}_1 = \vec{\mathbf{D}} \times \vec{\mathbf{E}}_2$$
$$\vec{\mathbf{S}}_2 = \vec{\mathbf{S}} \times \vec{\mathbf{E}}_1$$

Recall: How to determine if the “intersection” is inside the triangle?

Hint:
(1-b1-b2), b1, b2 are barycentric coordinates!

Accelerating Ray-Surface Intersection

Ray Tracing – Performance Challenges

Simple ray-scene intersection

- Exhaustively test ray-intersection with every triangle
- Find the closest hit (i.e. minimum t)

Problem:

- Naive algorithm = #pixels $\times$# traingles ($\times$#bounces)
- Very slow!

For generality, we use the term objects instead of triangles later (but doesn't necessarily mean entire objects)

Ray Tracing – Performance Challenges

San Miguel Scene, 10.7M triangles

Ray Tracing – Performance Challenges

Plant Ecosystem, 20M triangles

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

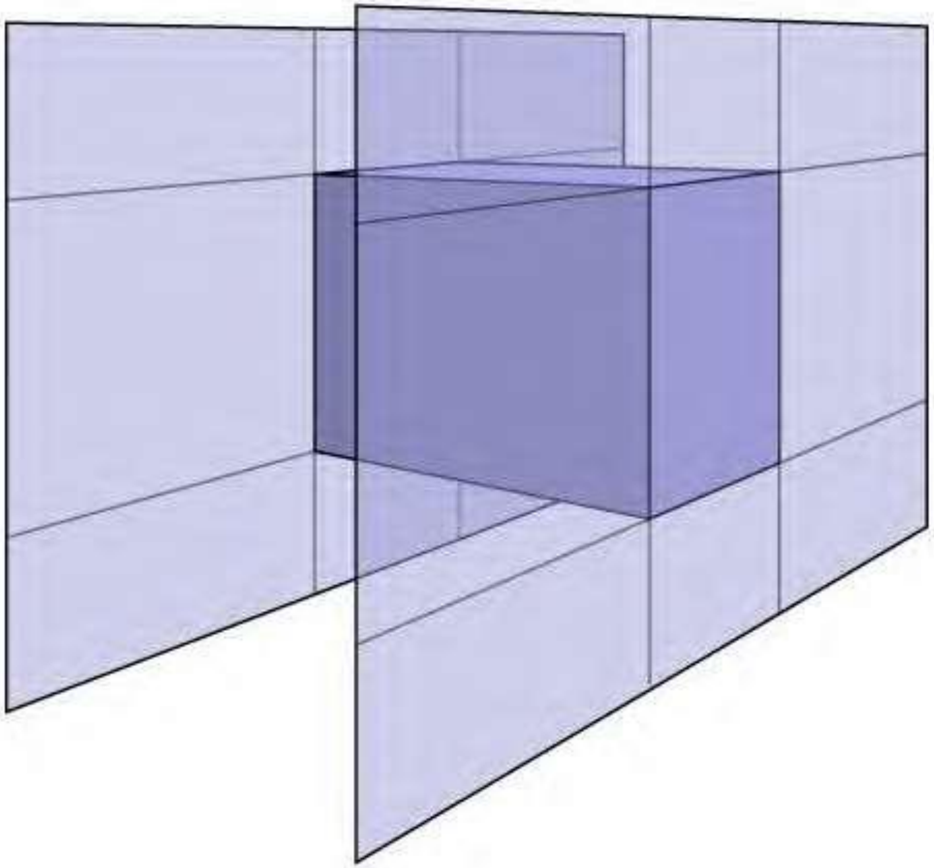
Ray-Intersection With Box

Understanding: box is the intersection of 3 pairs of slabs

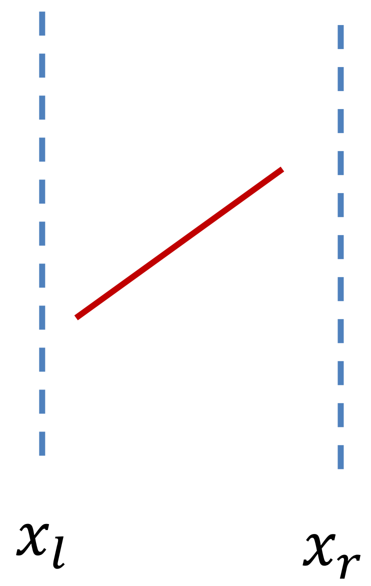
Specifically:

We often use an Axis-Aligned Bounding Box (AABB)
(轴对齐包围盒)

i.e. any side of the BB is along either x, y, or z axis



Liang-baskey algorithm



Which part of the line segment is visible?

$$x_l \overset{①}{\leq} x_1 + u\Delta x \overset{②}{\leq} x_r$$

$$\Delta x = x_2 - x_1,\ \Delta y = y_2 - y_1$$

Rewrite as

$$up_k \leq q_k$$

① $-u\Delta x \leq x_1 - x_l$ ① $\begin{cases} p_1 = -\Delta x \\ q_1 = x_1 - x_l \end{cases}$

② $u\Delta x \leq x_r - x_1$ ② $\begin{cases} p_2 = \Delta x \\ q_2 = x_r - x_1 \end{cases}$

如果$p_k < 0$： 入边

如果$p_k > 0$： 出边

1. 如果Δx=0，线段平行于X
   1. $q_k < 0$, **outside**
   2. $q_k \geq 0$, *inside*
2. 如果$\Delta x \neq 0$, 线段不平行
   - **Compute all $r_k=q_k/p_k$**
   - **u_1 is max(0, r_{in})**
   - **u_2 is min(1, r_{out})**
   - **If $u_1>u_2$, outside the window**
   - **else, the part is from u_1 -> u_2**

u_1 and u_2 defines the part of segment that is inside the window

Liang-baskey algorithm

$$x_l \le x_1 + u\Delta x \le x_r$$
$$y_b \le y_1 + u\Delta y \le y_t$$
$$\Delta x = x_2 - x_1,\ \Delta y = y_2 - y_1$$

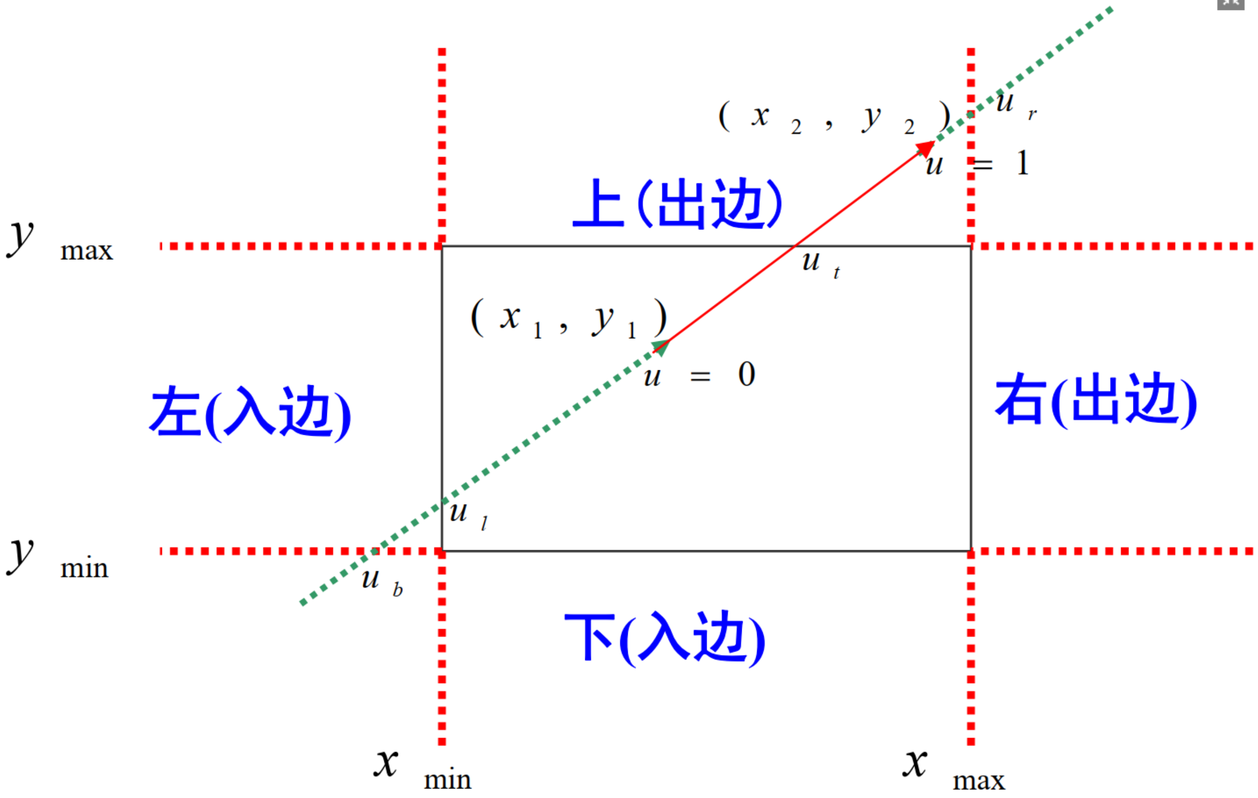
$$up_k \le q_k$$

$$\begin{cases} p_1 = -\Delta x \\ q_1 = x_1 - x_l \end{cases} \quad \begin{cases} p_2 = \Delta x \\ q_2 = x_r - x_1 \end{cases} \quad \begin{cases} p_3 = -\Delta y \\ q_3 = y_1 - y_b \end{cases} \quad \begin{cases} p_4 = \Delta y \\ q_4 = y_t - y_1 \end{cases}$$

- **u_1 and u_2 defines the part of segment that is inside the window**
  - **Compute all r_k=q_k/p_k**
  - **u_1 is max(0, r_{in1}, r_{in2})**
  - **u_2 is min(1, r_{out1}, r_{out2})**
  - **If u_1>u_2, the segment is completely outside the window**
  - **else, the part is from u_1 -> u_2**

Liang-baskey algorithm

如果用u_1，u_2分别表示线段（$u_1 \leqslant u_2$）可见部分的开始和结束



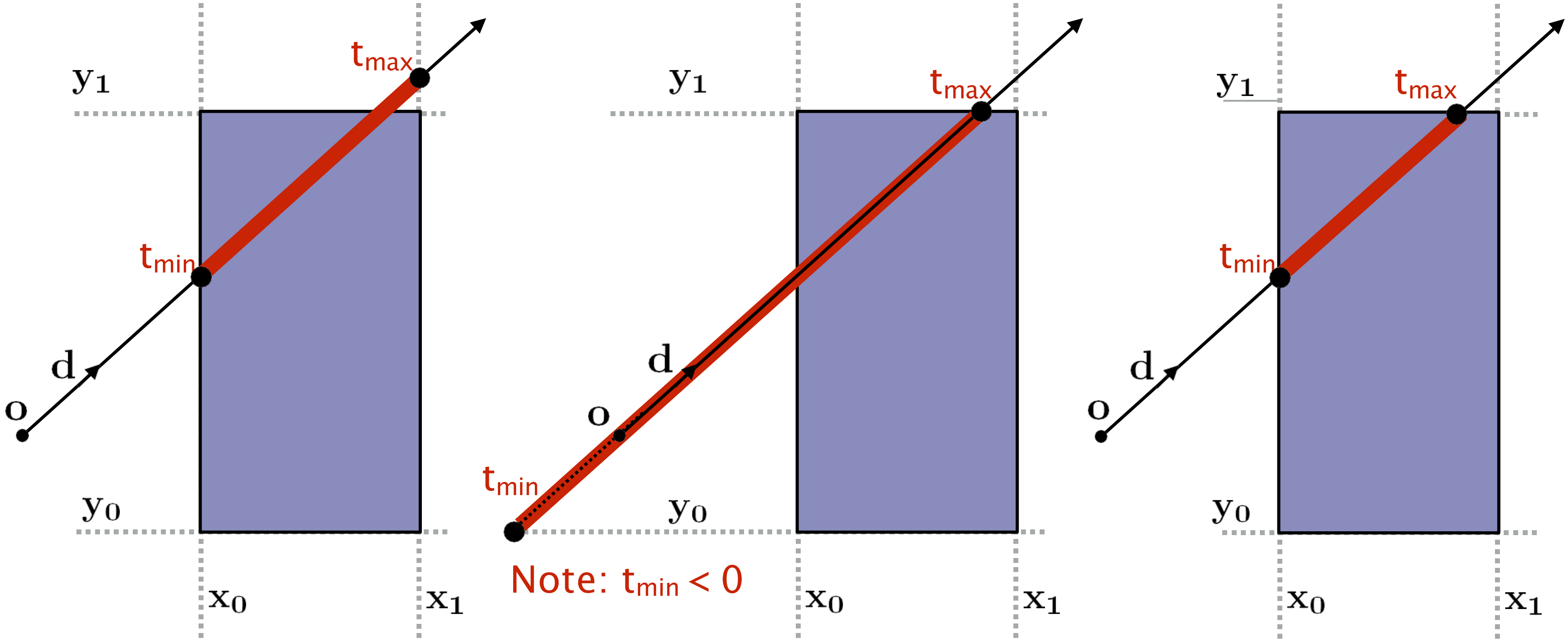
$$u_1 = \max(0, u_l, u_b)$$

$$u_2 = \min(1, u_t, u_r)$$

这就是梁先生的重大发现！

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

How do we know when the ray intersects the box?

Ray Intersection with Axis-Aligned Box

- Recall: a box (3D) = three pairs of infinitely large slabs
- Key ideas
  - The ray enters the box only when it enters all pairs of slabs
  - The ray exits the box as long as it exits any pair of slabs
- For each pair, calculate the t_{min} and t_{max} (negative is fine)
- For the 3D box, $t_{enter} = \max\{t_{min}\}$, $t_{exit} = \min\{t_{max}\}$
- If $t_{enter} < t_{exit}$, we know the ray stays awhile in the box (so they must intersect!) (not done yet, see the next slide)

Ray Intersection with Axis-Aligned Box

- However, ray is not a line
  - Should check whether t is negative for physical correctness!

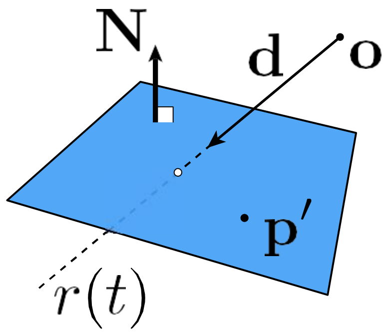
- What if $t_{exit} < 0$?
  - The box is "behind" the ray — no intersection!

- What if $t_{exit} >= 0$ and $t_{enter} < 0$?
  - The ray's origin is inside the box — have intersection!

- In summary, ray and AABB intersect iff
  - $t_{enter} < t_{exit}$ && $t_{exit} >= 0$

Why Axis-Aligned?

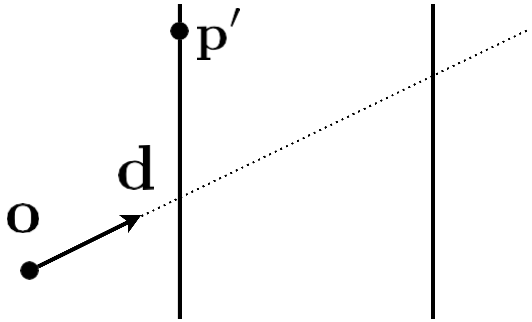
General



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

3 subtractions, 6 multiplies, 1 division

Slabs perpendicular to x-axis

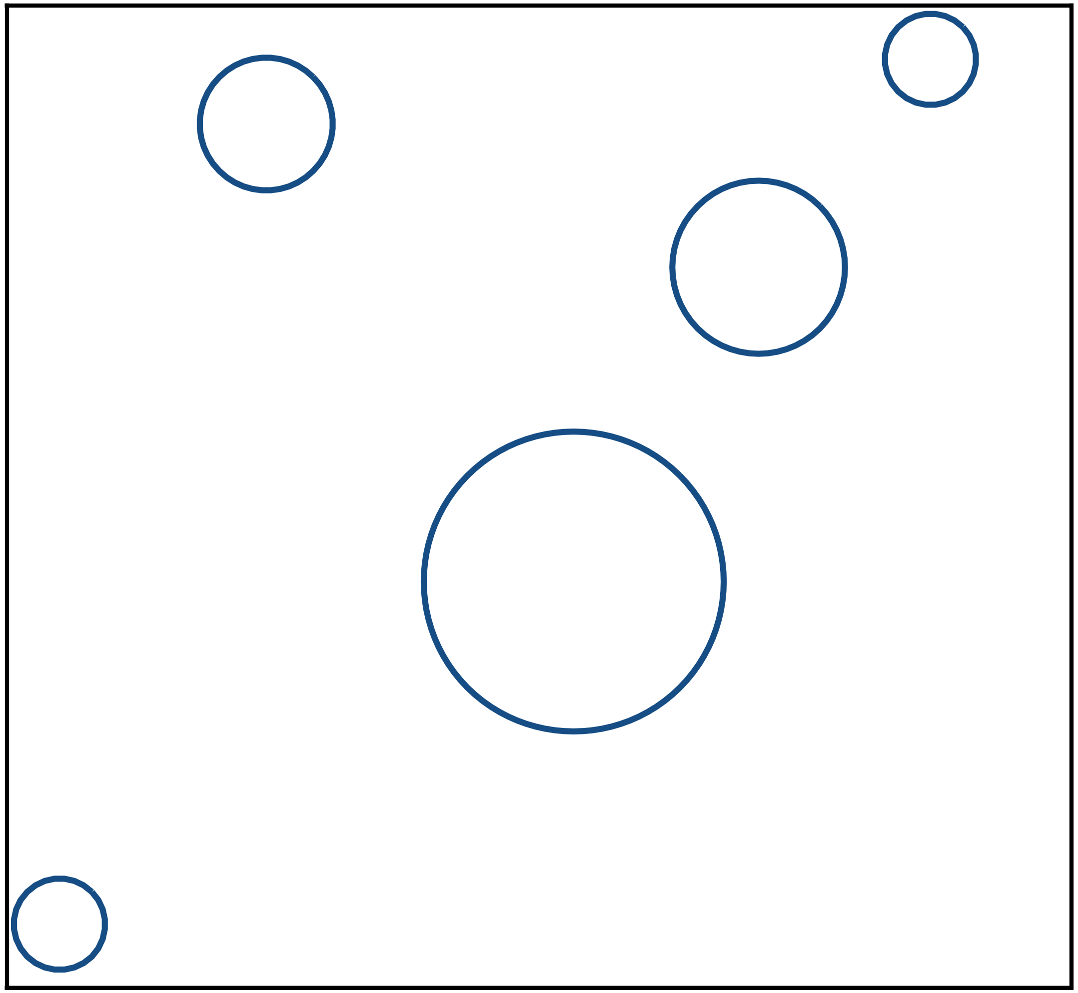


$$t = \frac{\mathbf{p}'_x - \mathbf{o}_x}{\mathbf{d}_x}$$

1 subtraction, 1 division

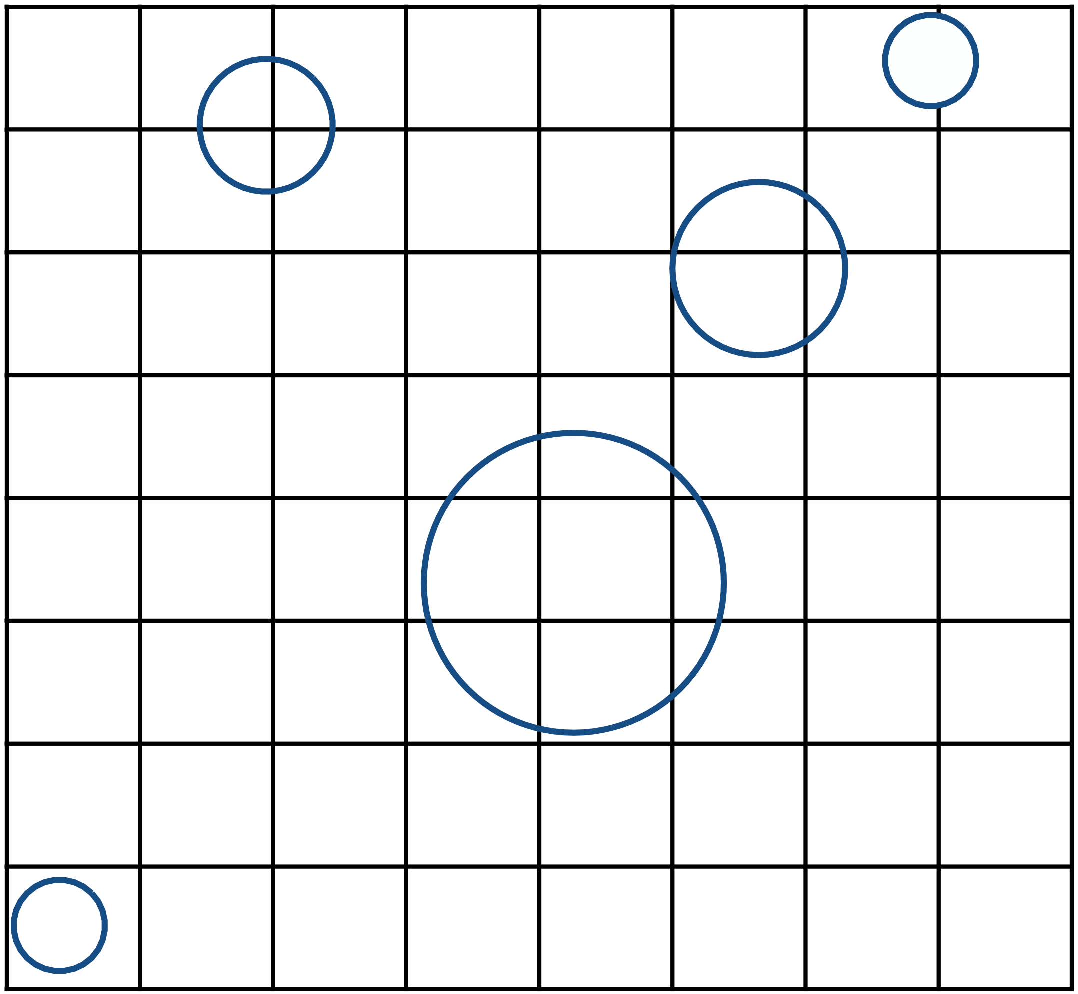
Uniform Spatial Partitions (Grids)

Preprocess – Build Acceleration Grid



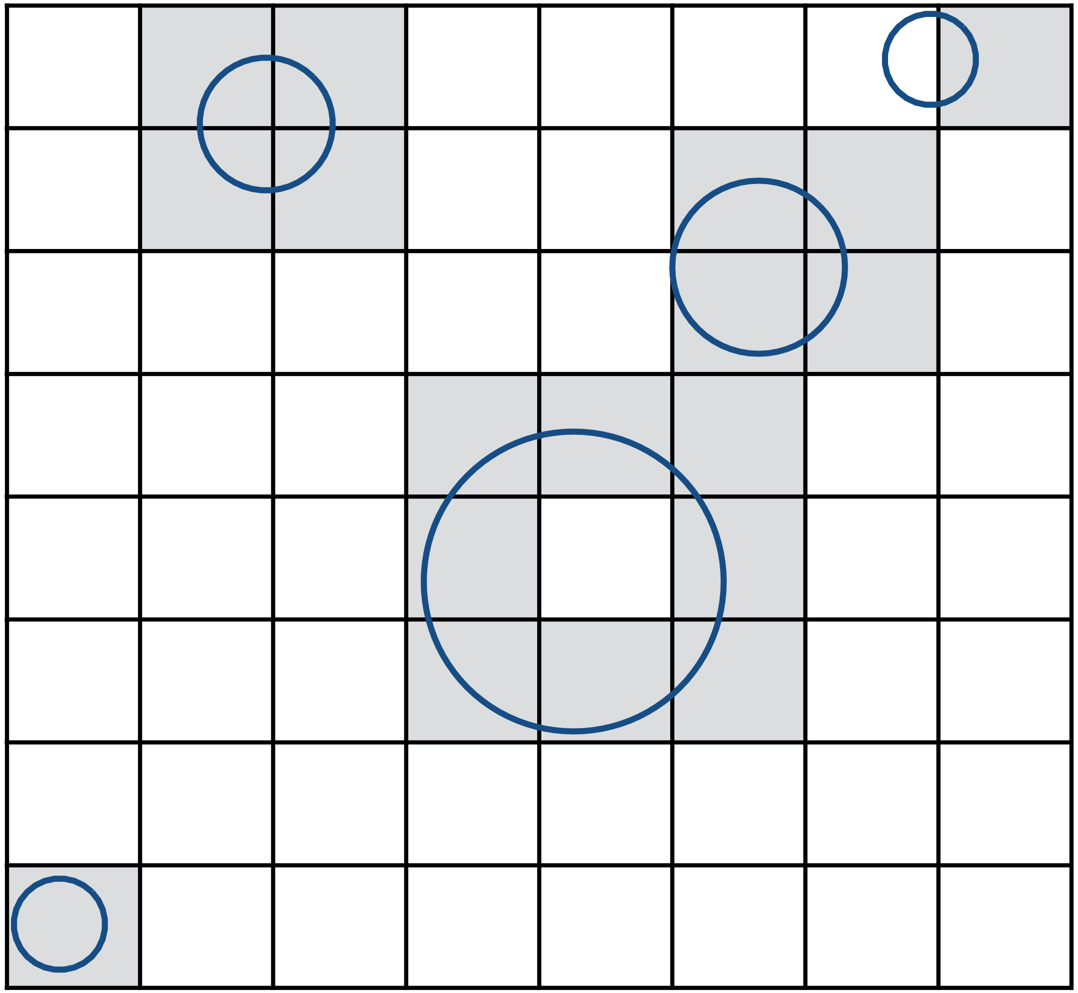
1. Find bounding box

Preprocess – Build Acceleration Grid



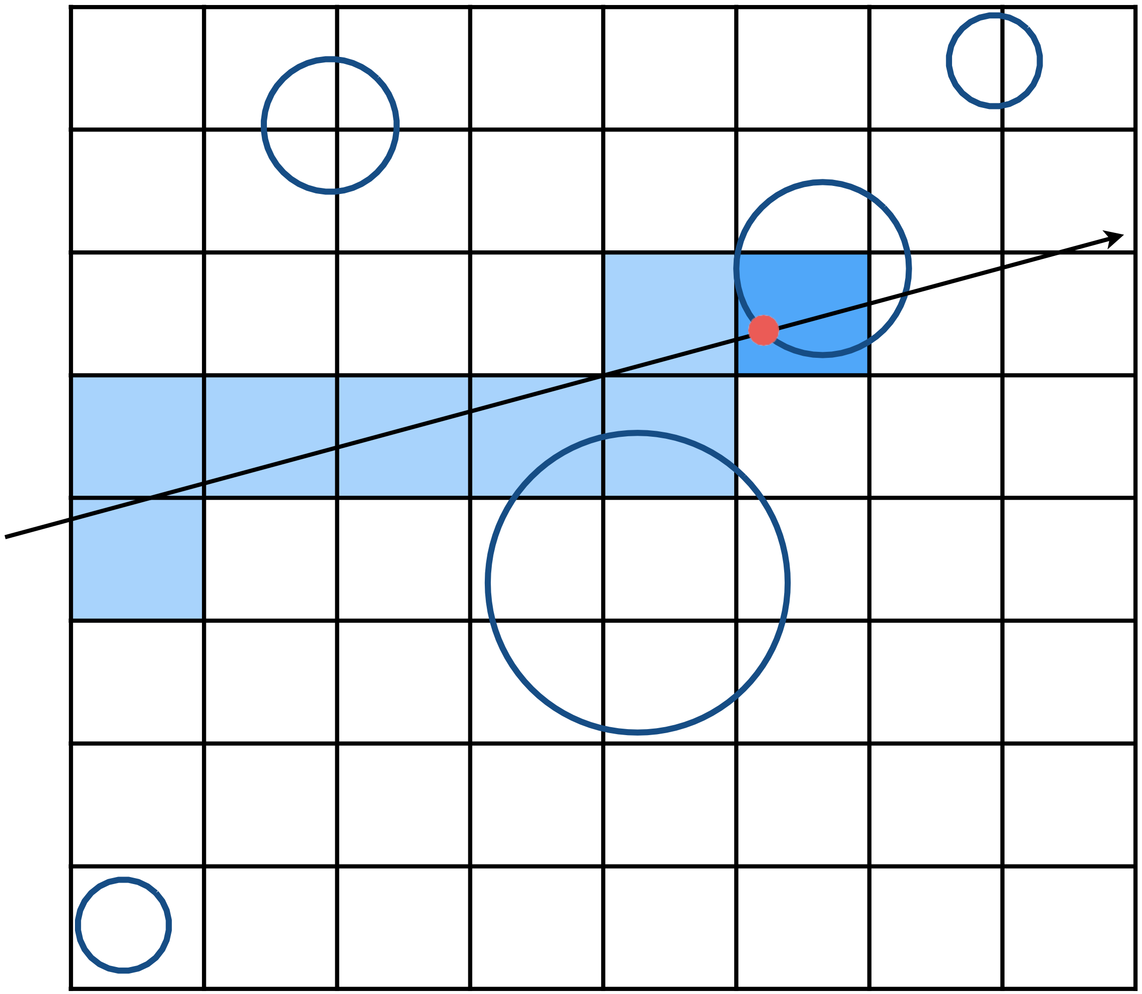
1. Find bounding box
2. Create grid

Preprocess – Build Acceleration Grid



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

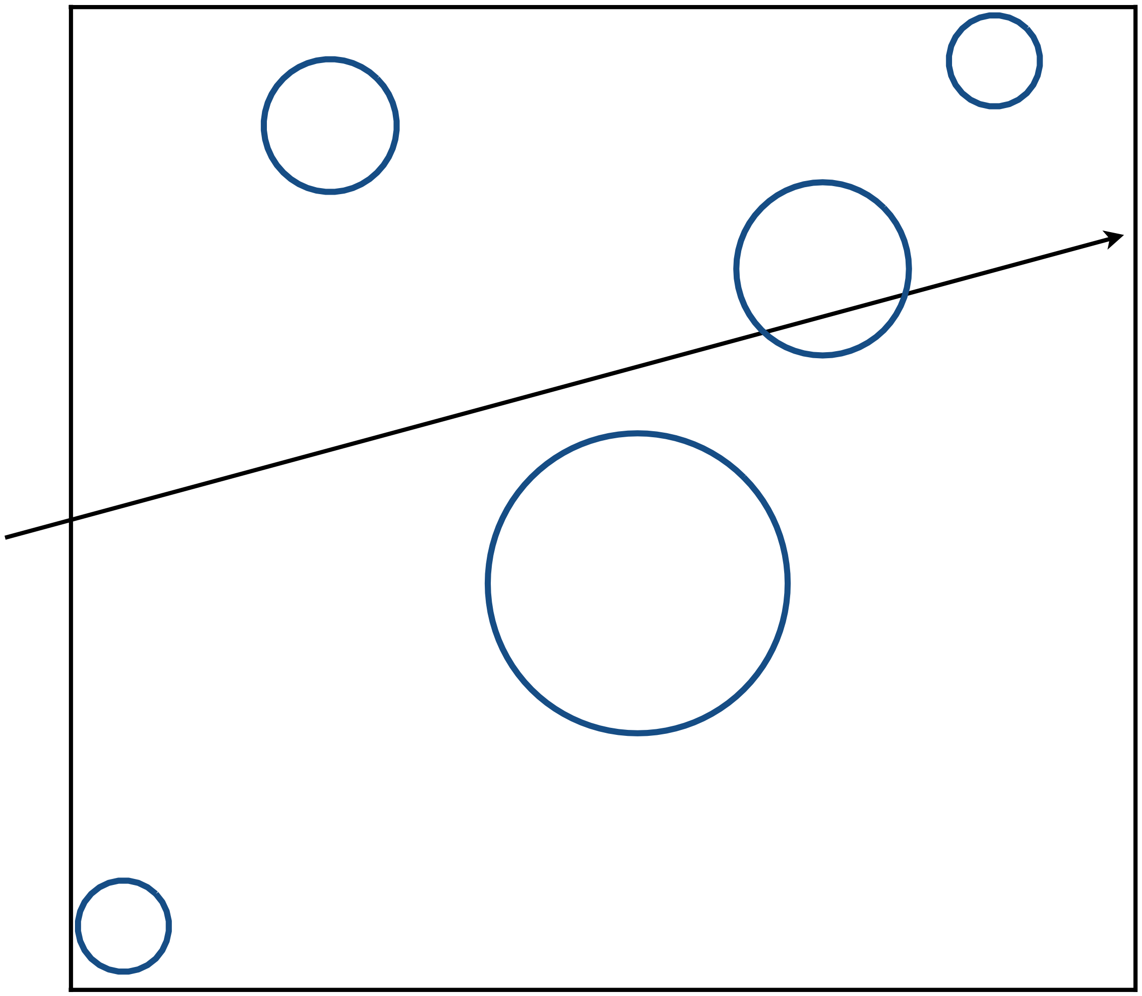
Ray-Scene Intersection



Step through grid in ray traversal order

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

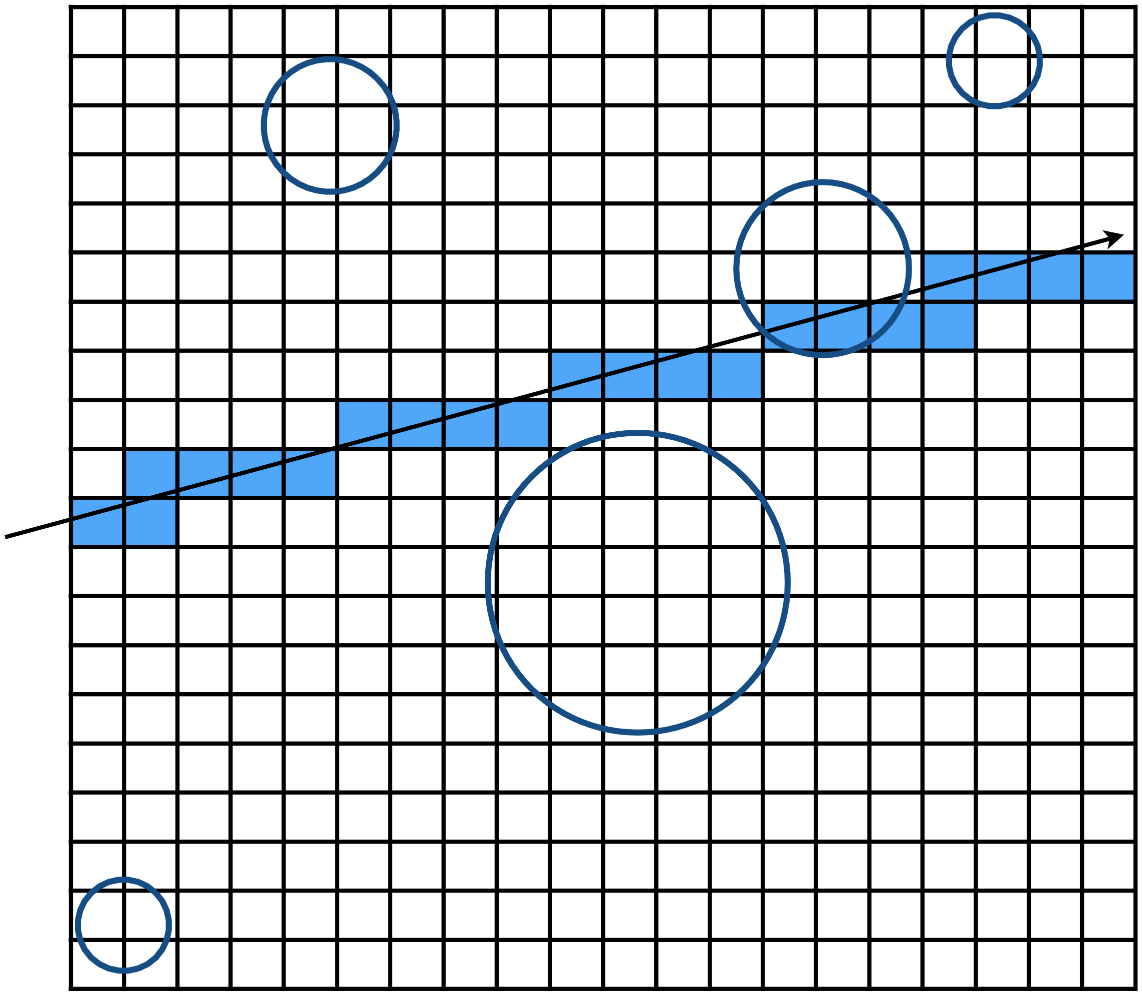
Grid Resolution?



One cell

- No speedup

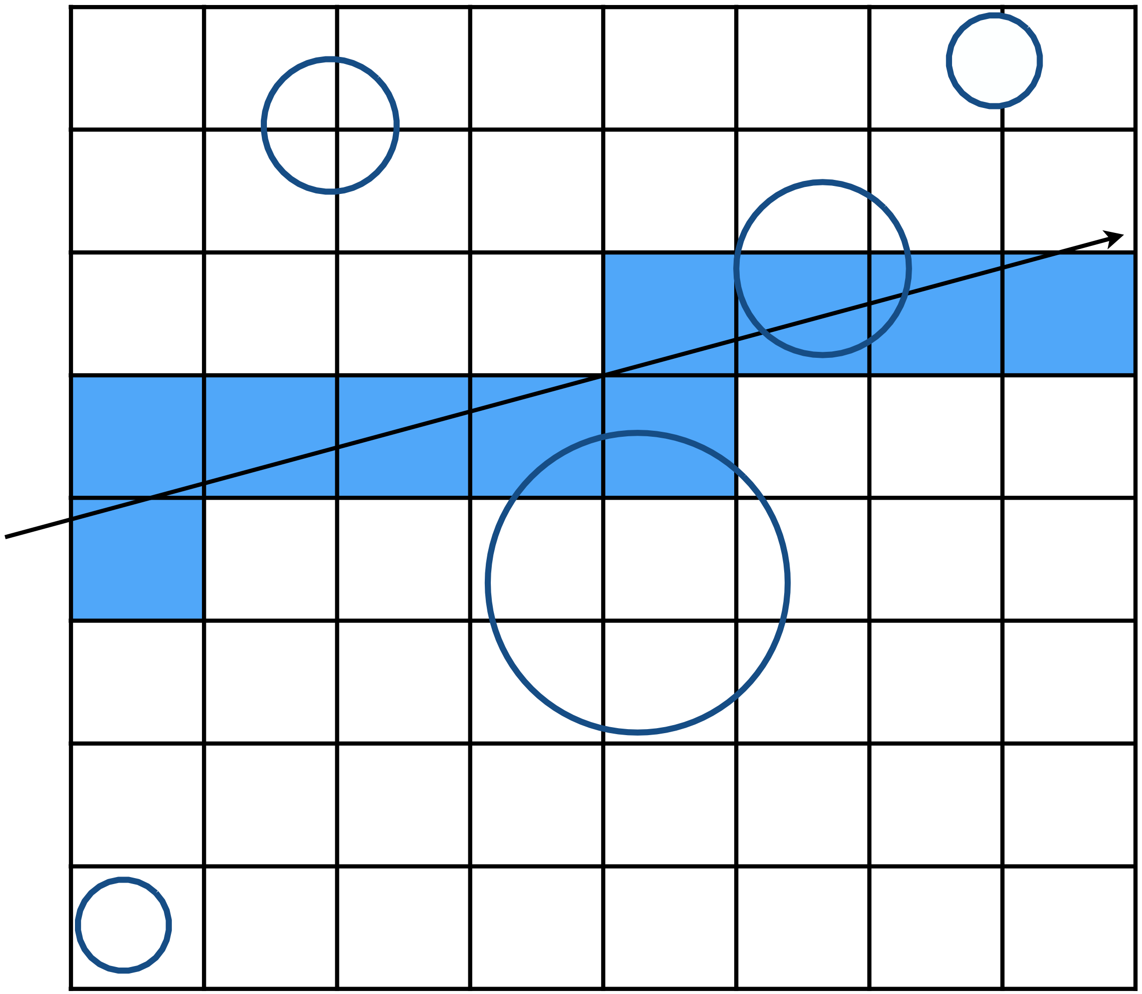
Grid Resolution?



Too many cells

- Inefficiency due to extraneous grid traversal

Grid Resolution?



Heuristic:

- #cells = C * #objs
- C ≈ 27 in 3D

Uniform Grids – When They Work Well

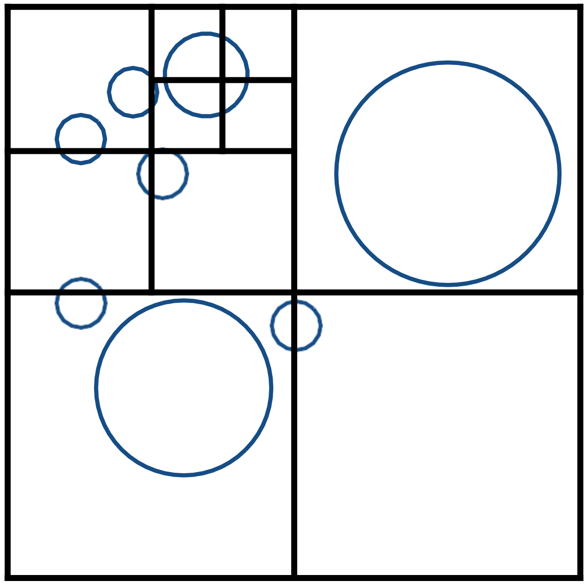
Grids work well on large collections of objects
that are distributed evenly in size and space

Uniform Grids – When They Fail

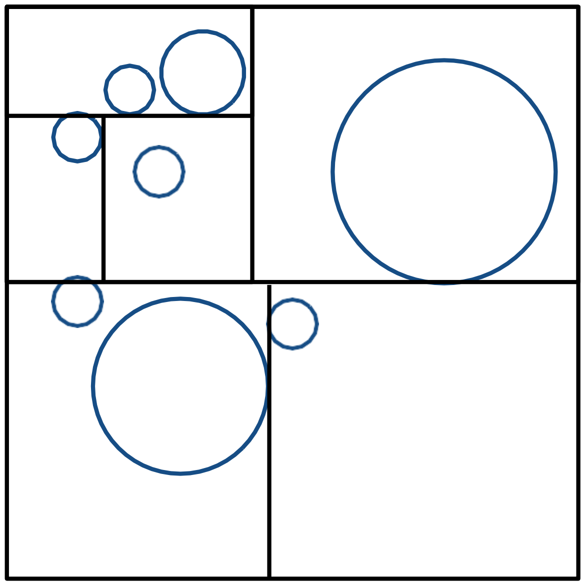
"Teapot in a stadium" problem

Spatial Partitions

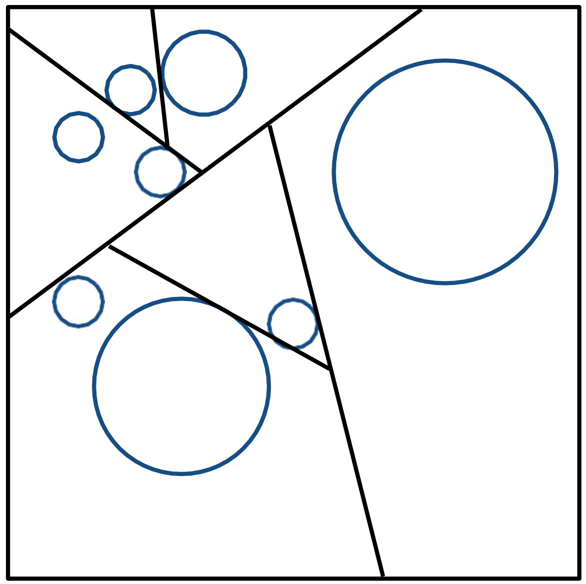
Spatial Partitioning Examples



Oct-Tree



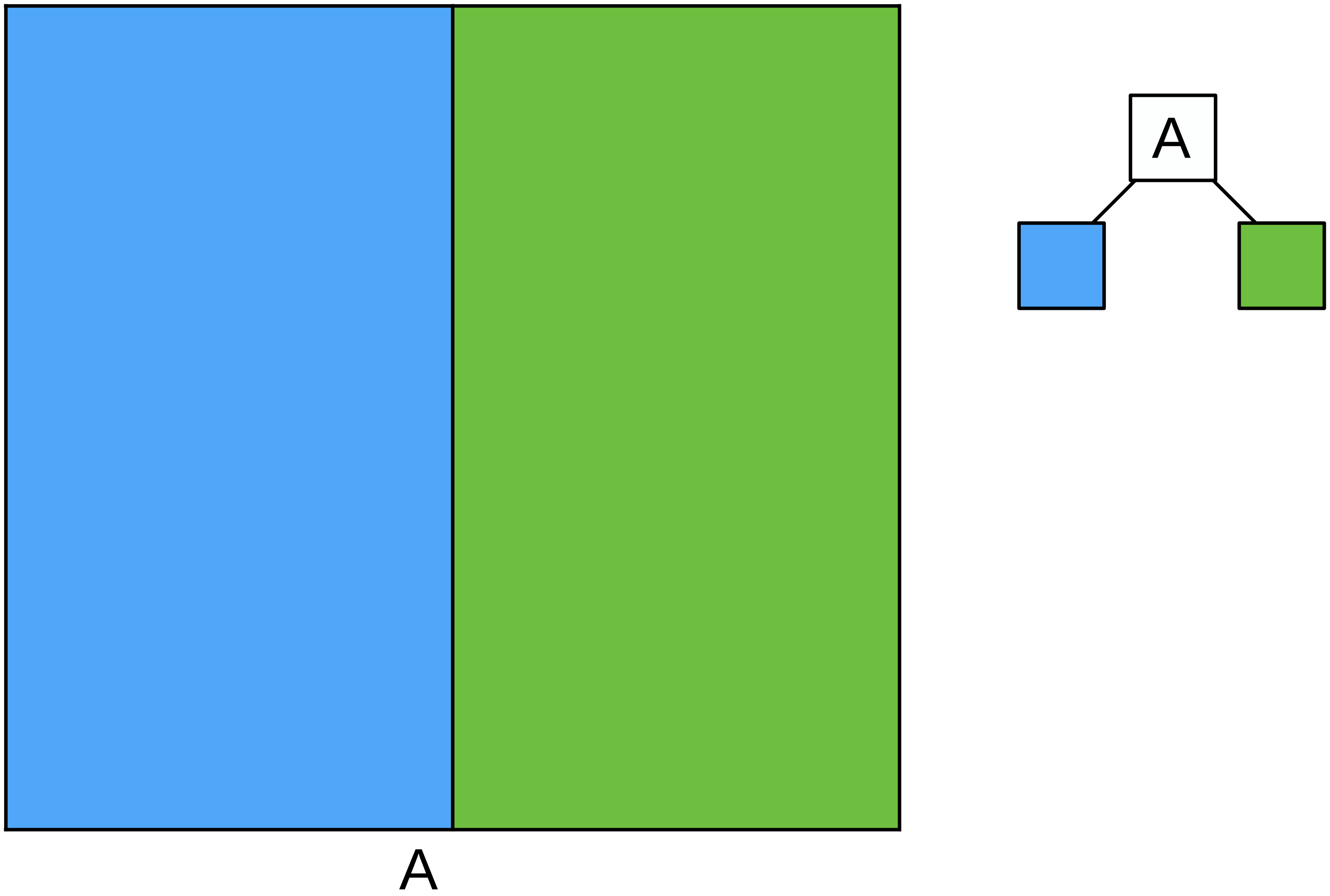
KD-Tree



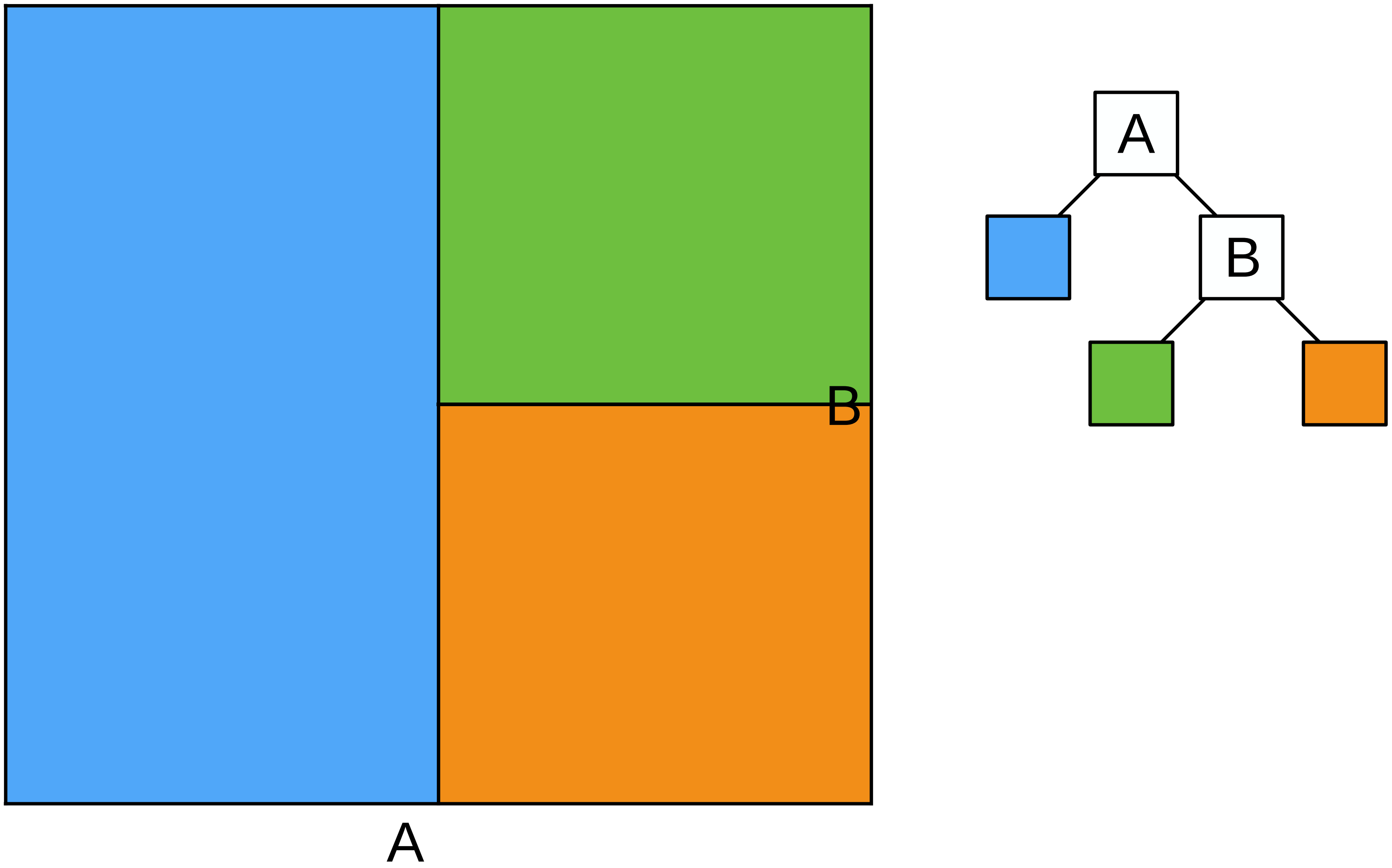
BSP-Tree

Note: you could have these in both 2D and 3D. In lecture we will illustrate principles in 2D.

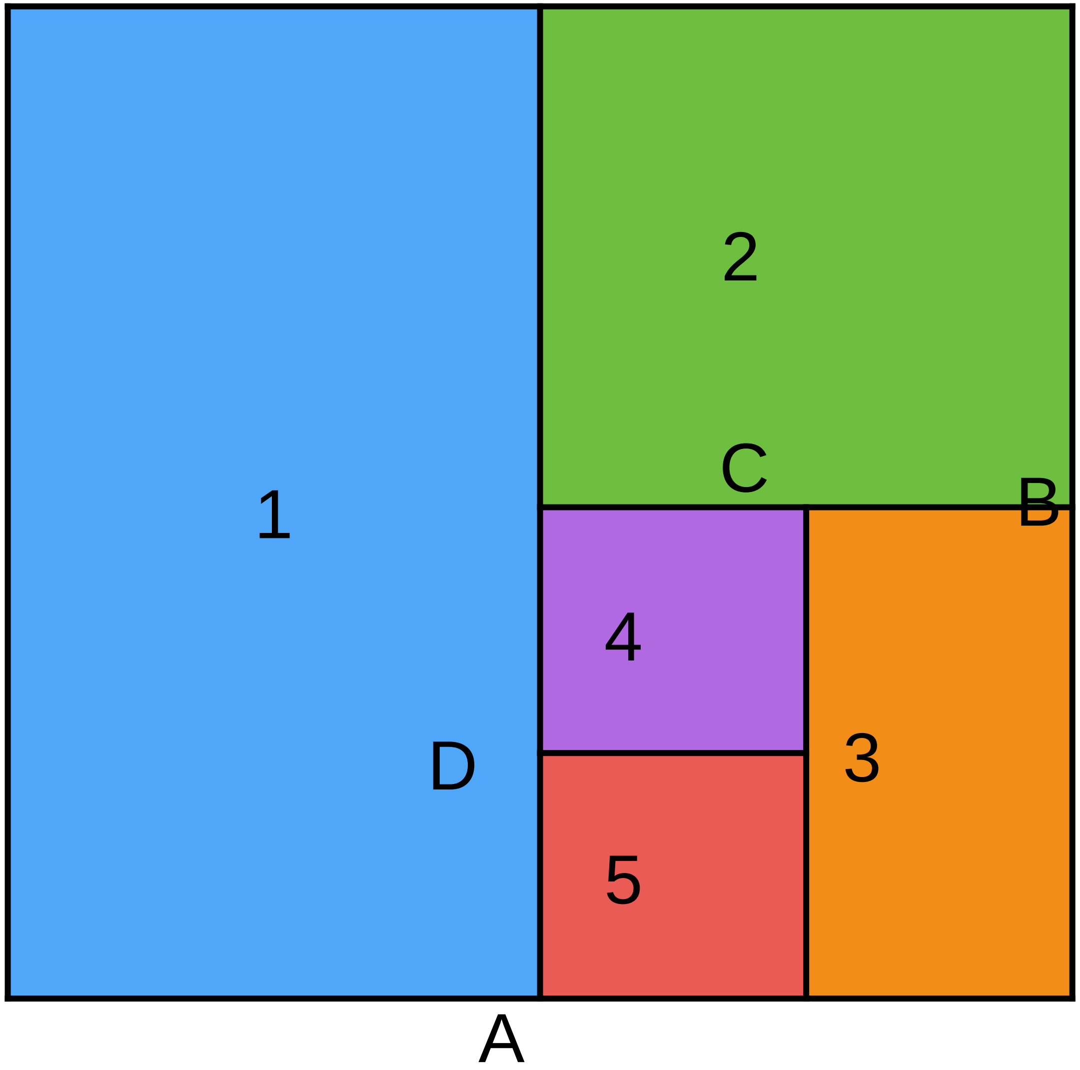
KD-Tree Pre-Processing

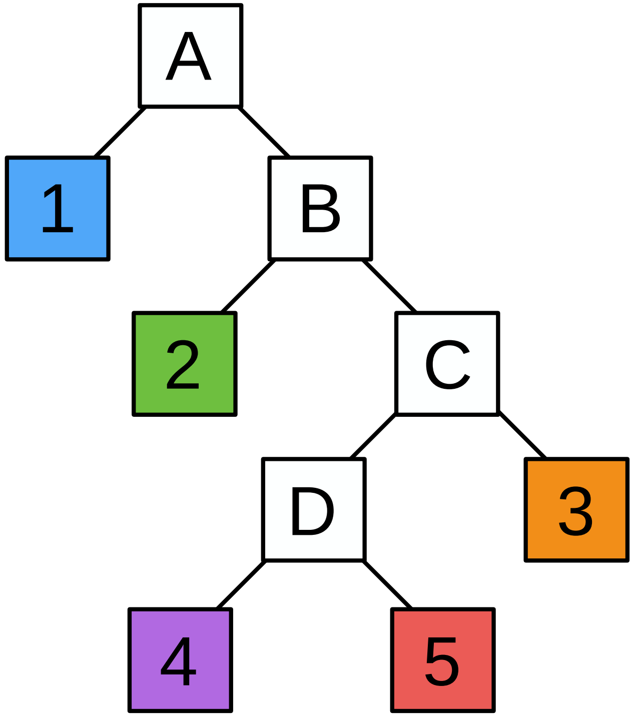


KD-Tree Pre-Processing



KD-Tree Pre-Processing





Note: also subdivide
nodes 1 and 2, etc.

Data Structure for KD-Trees

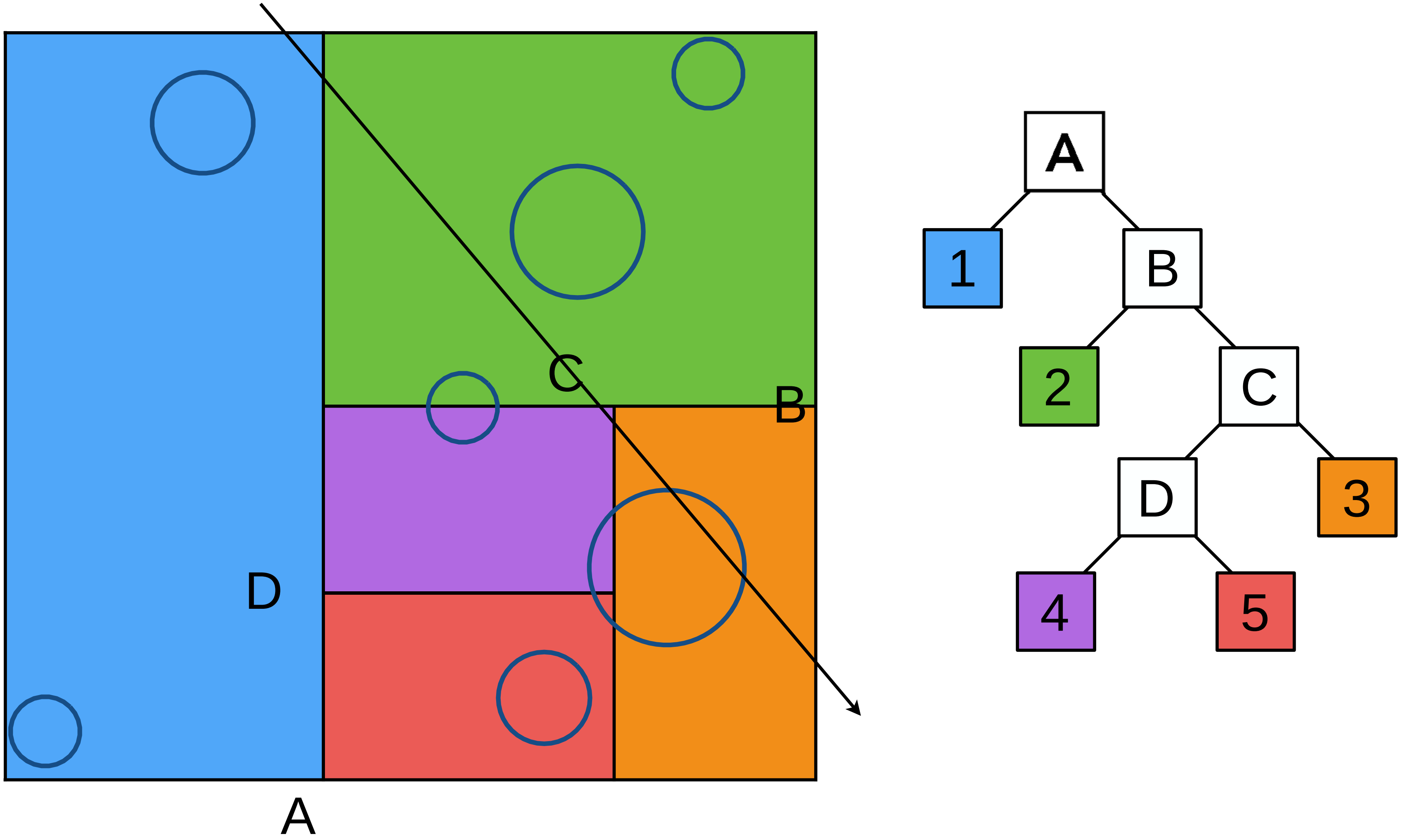
Internal nodes store

- split axis: x-, y-, or z-axis
- split position: coordinate of split plane along axis
- children: pointers to child nodes
- No objects are stored in internal nodes

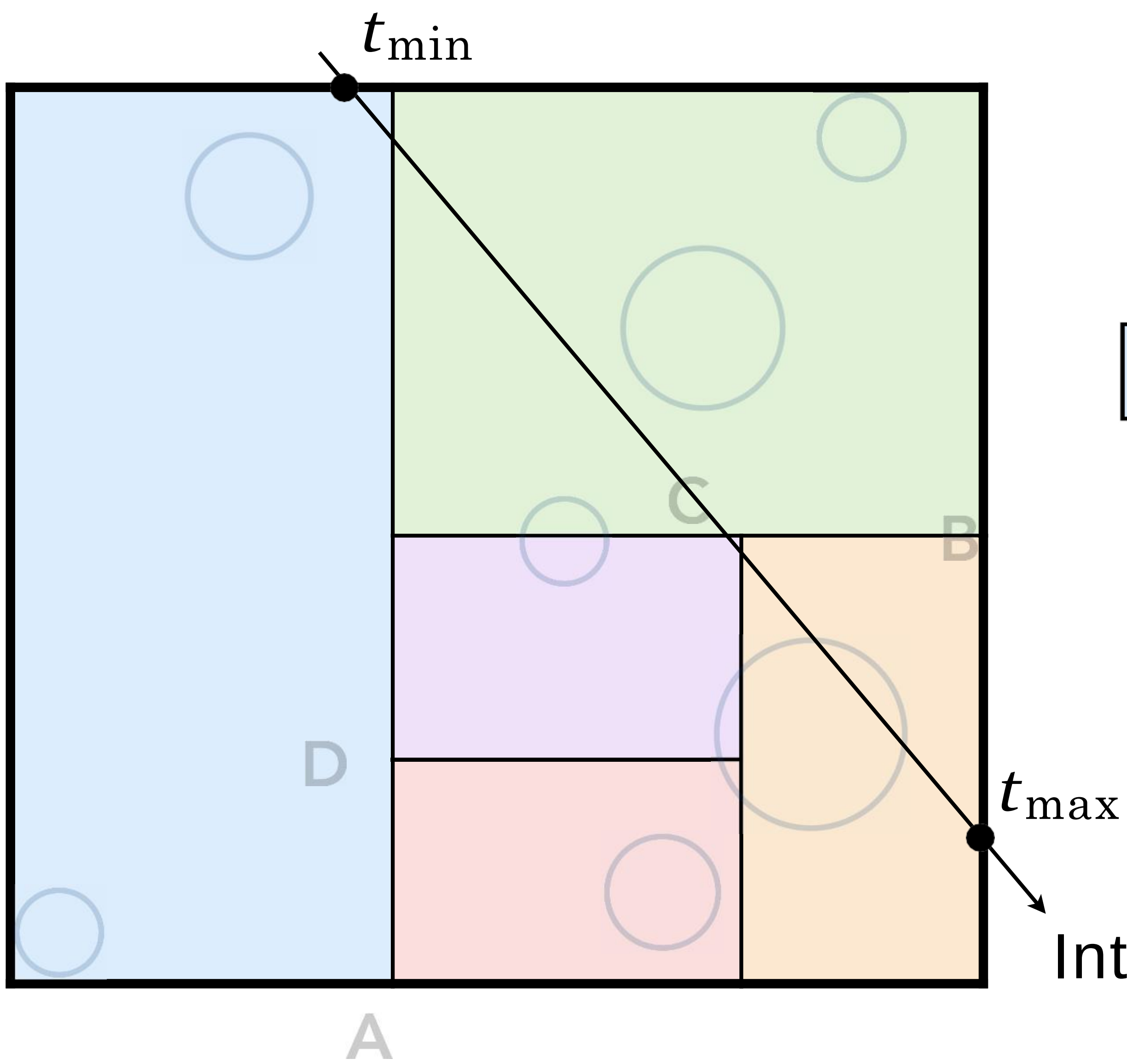
Leaf nodes store

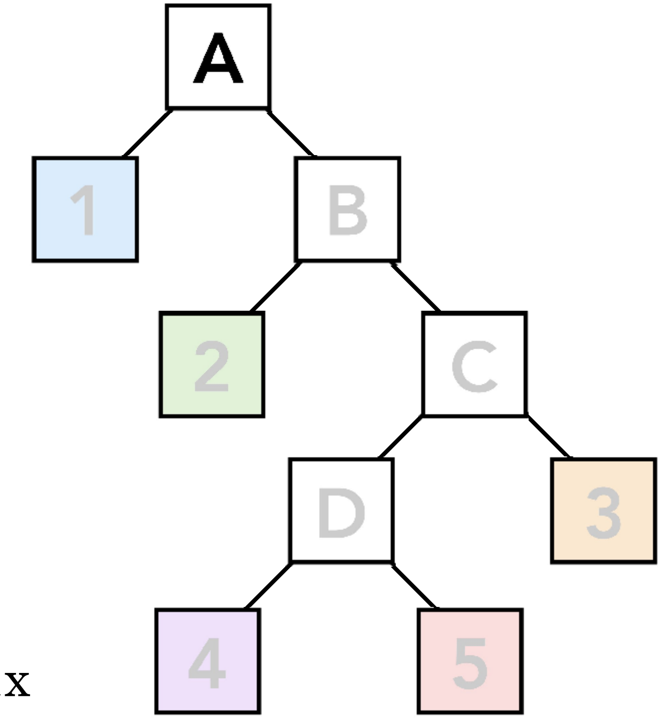
- list of objects

Traversing a KD-Tree



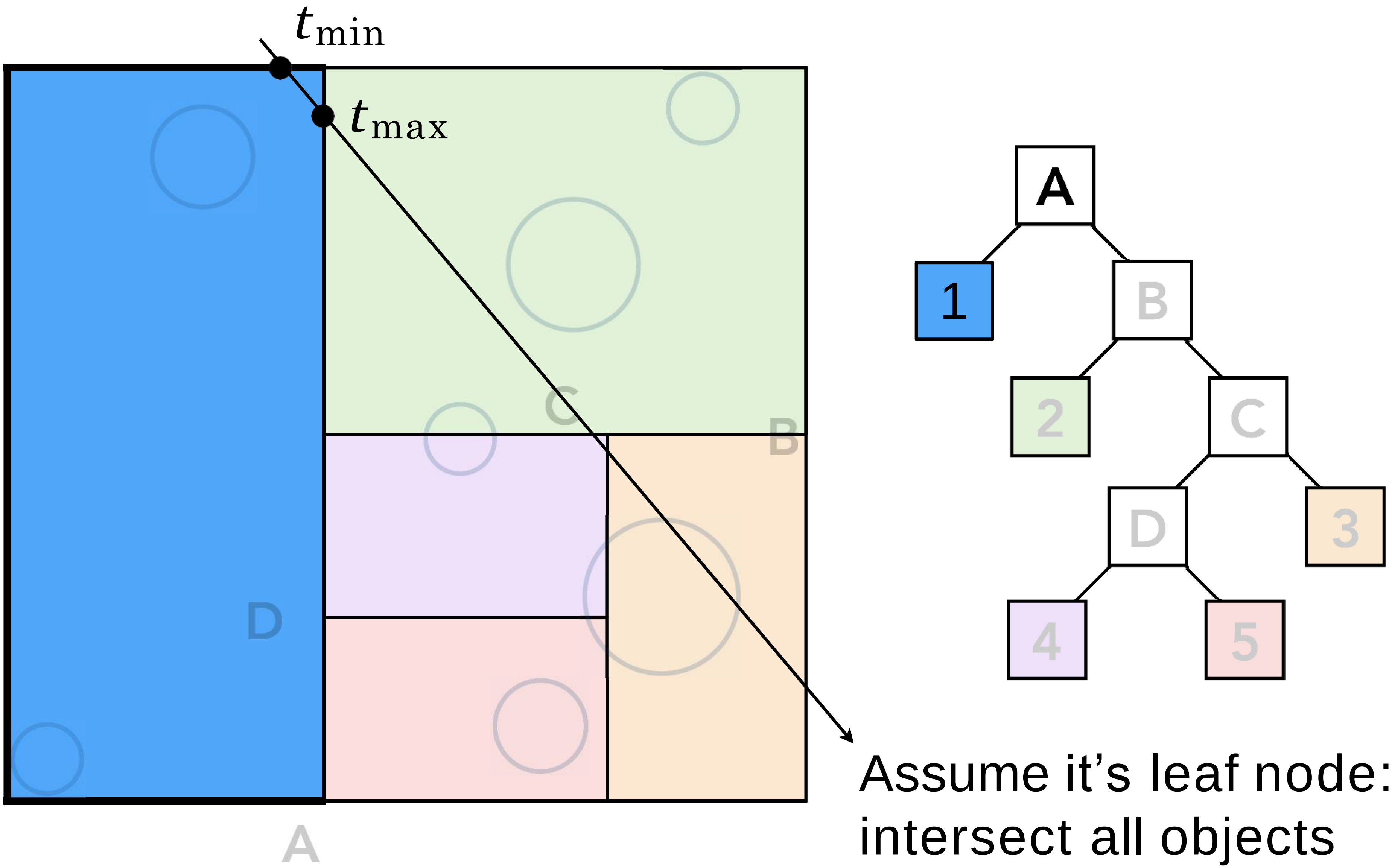
Traversing a KD-Tree





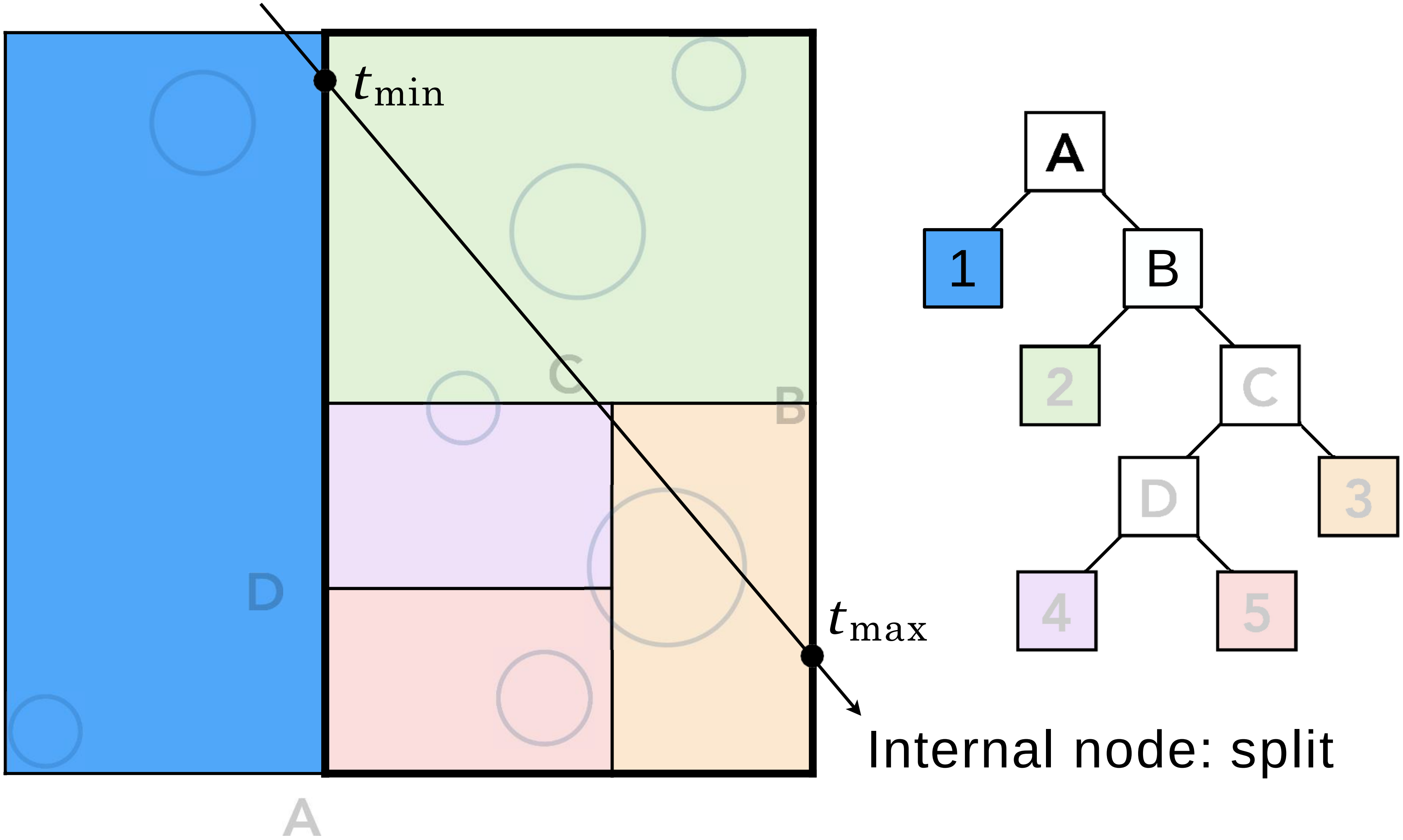
Internal node: split

Traversing a KD-Tree



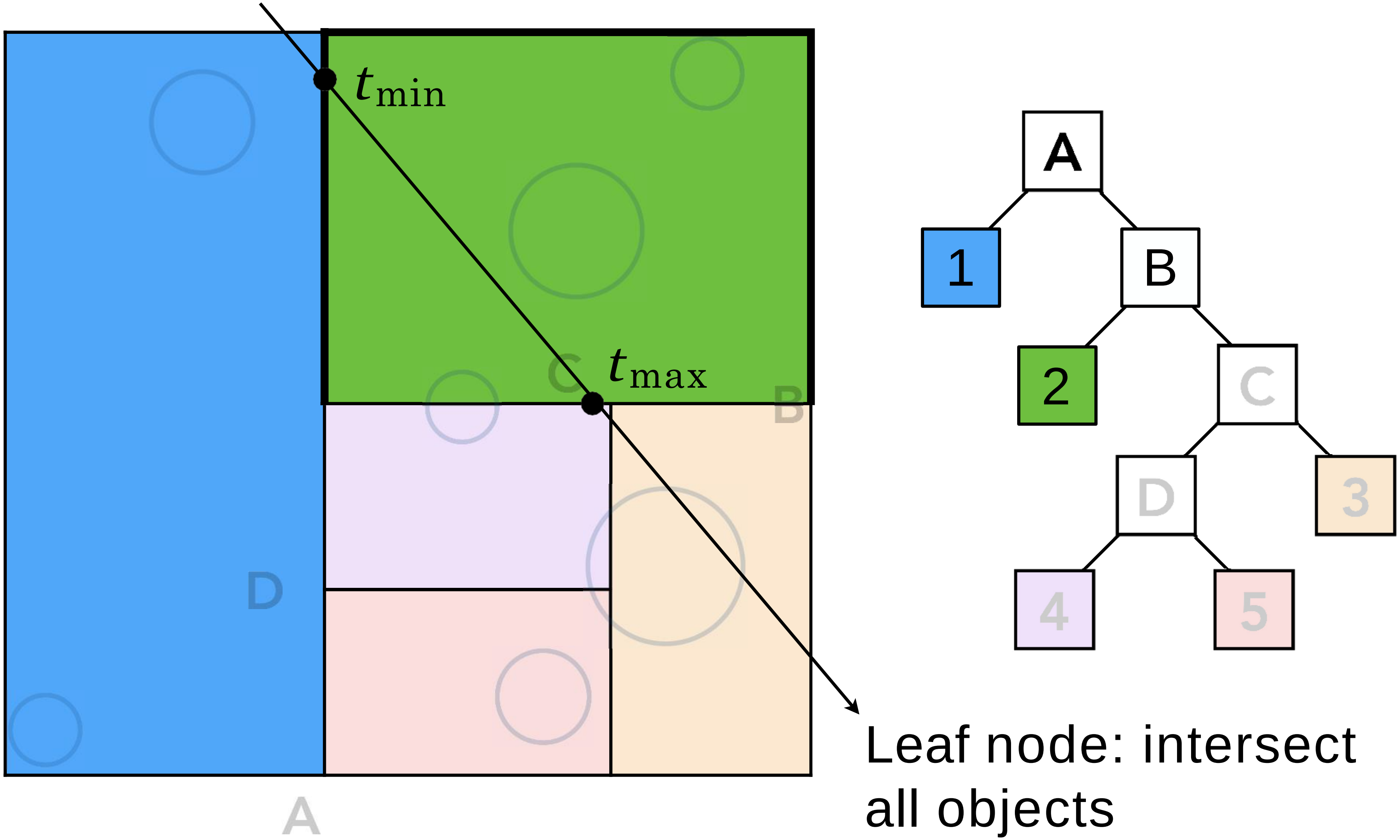
Assume it's leaf node:
intersect all objects

Traversing a KD-Tree



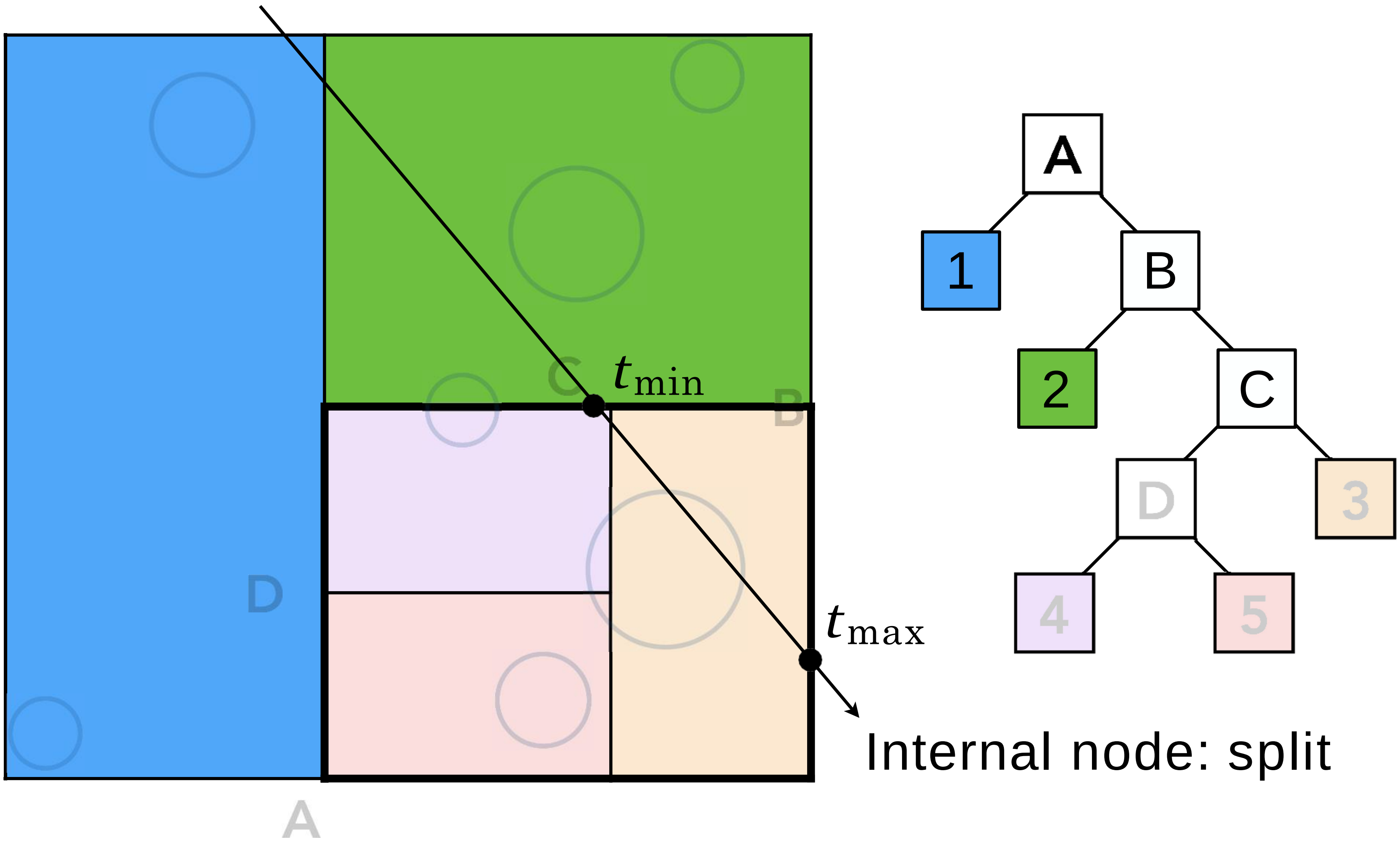
Internal node: split

Traversing a KD-Tree



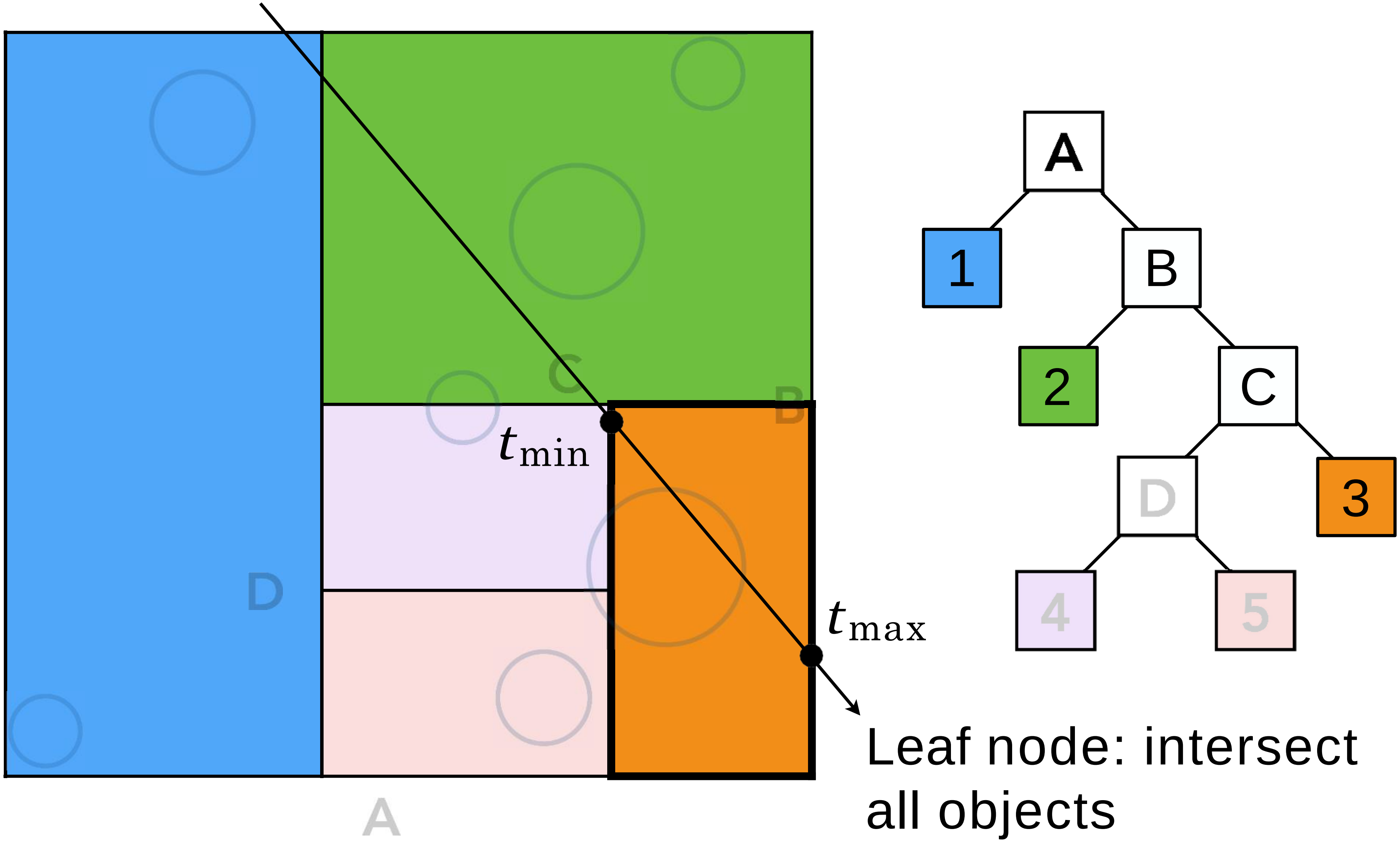
Leaf node: intersect all objects

Traversing a KD-Tree



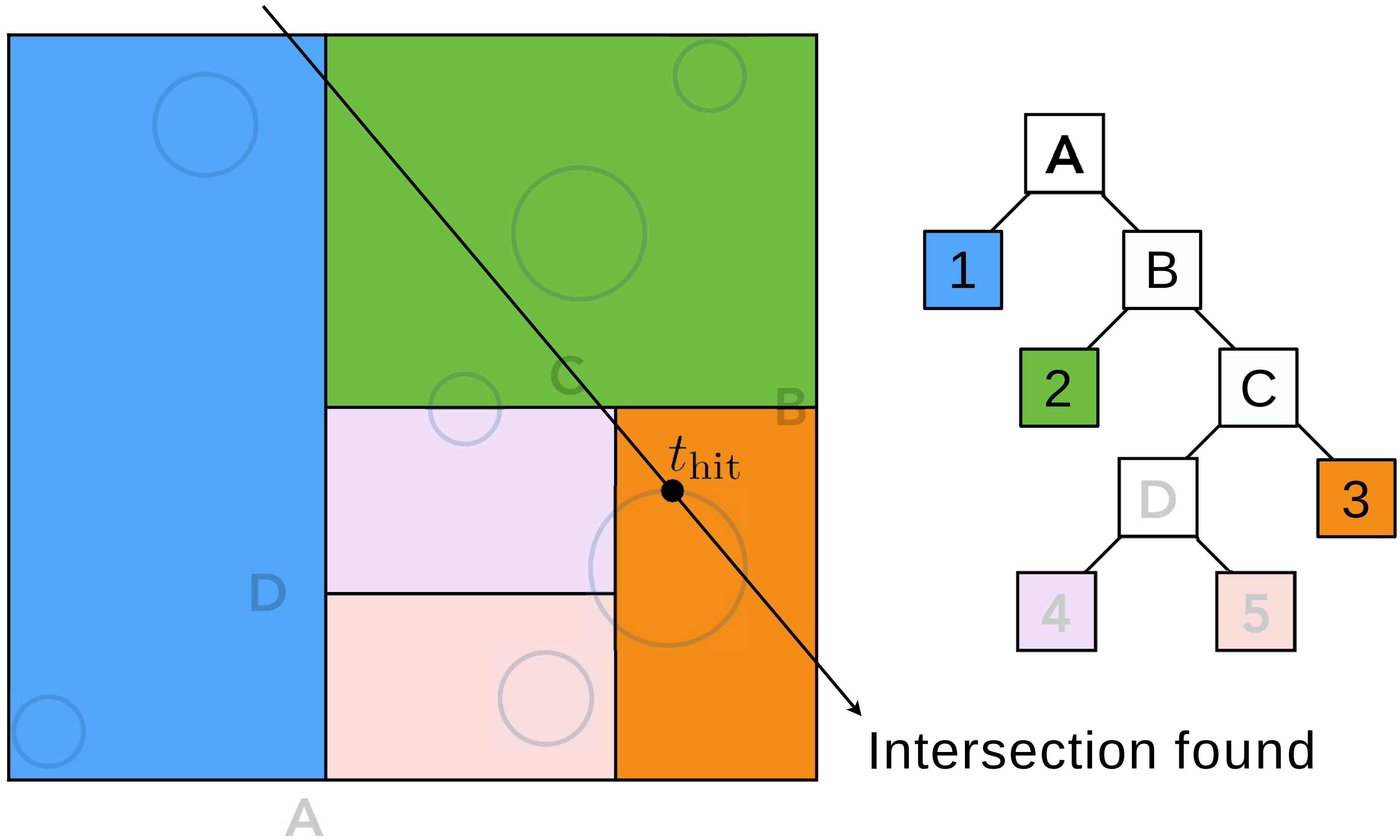
Internal node: split

Traversing a KD-Tree



Leaf node: intersect all objects

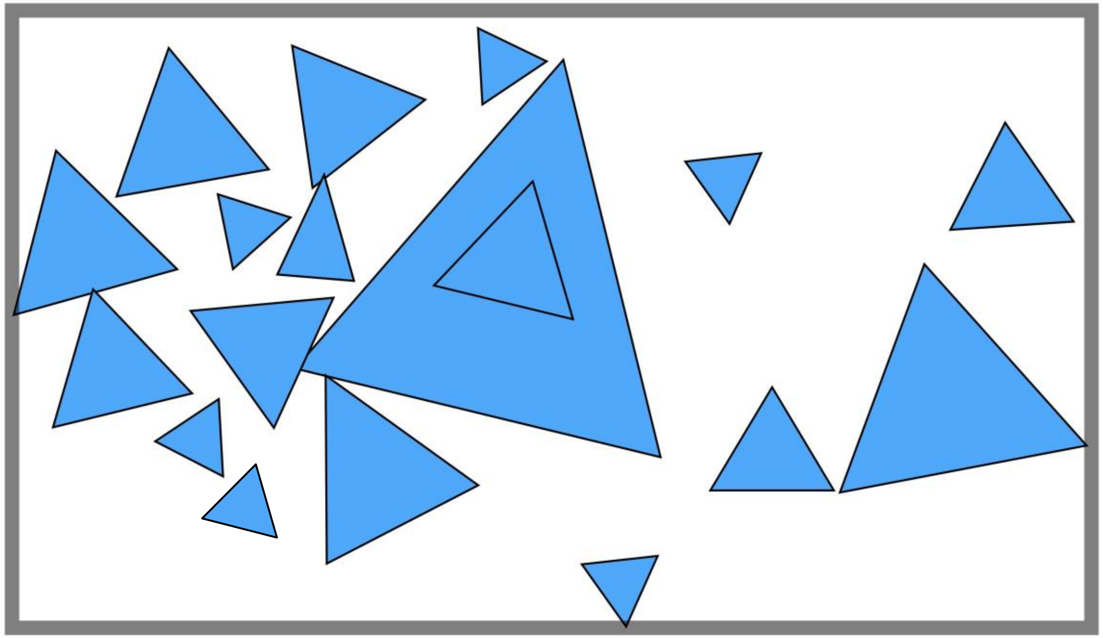
Traversing a KD-Tree

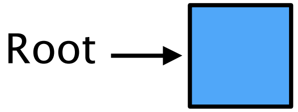


Object Partitions &

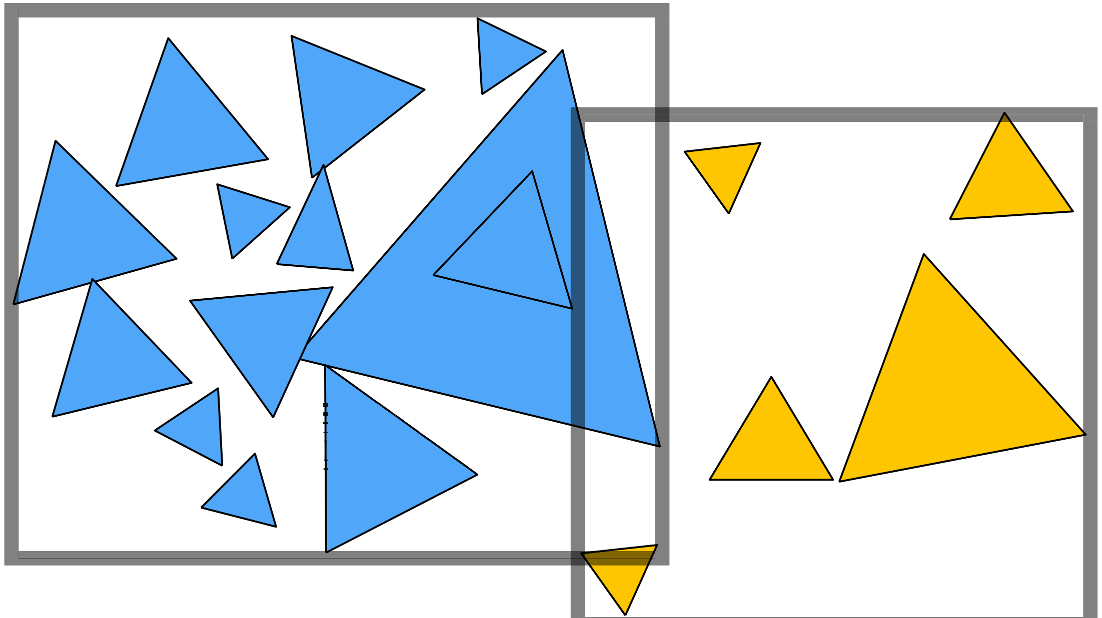
Bounding Volume Hierarchy (BVH)

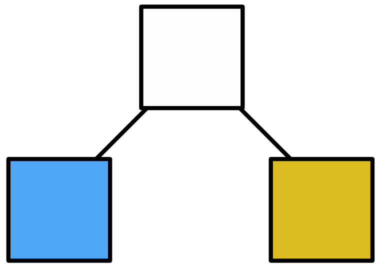
Bounding Volume Hierarchy (BVH)



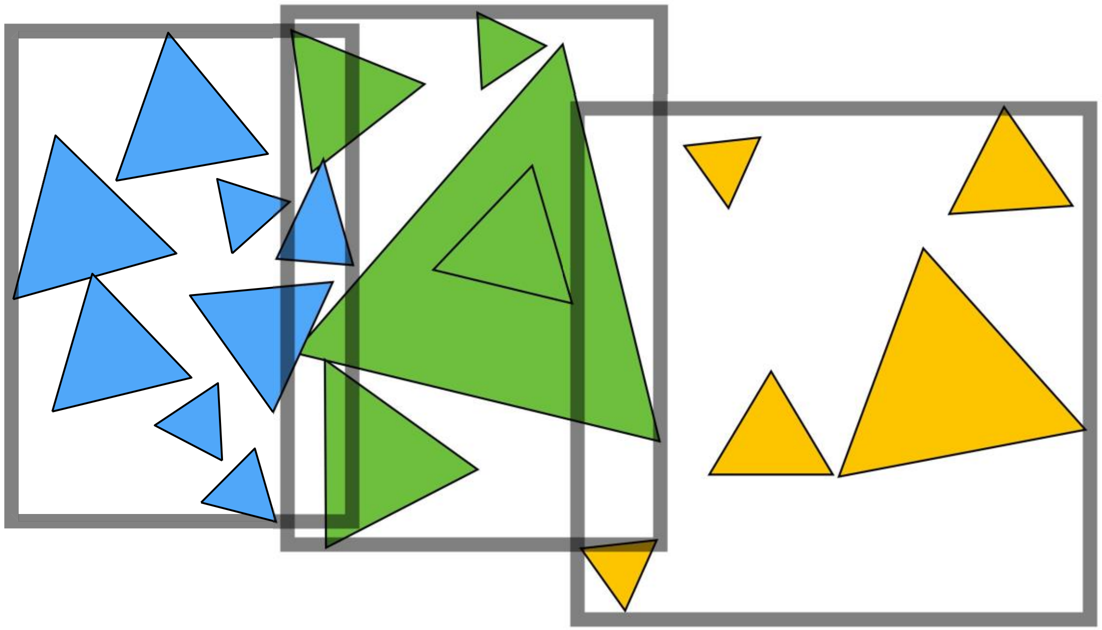


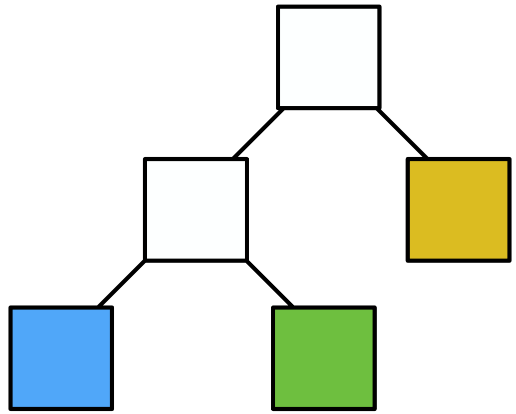
Bounding Volume Hierarchy (BVH)



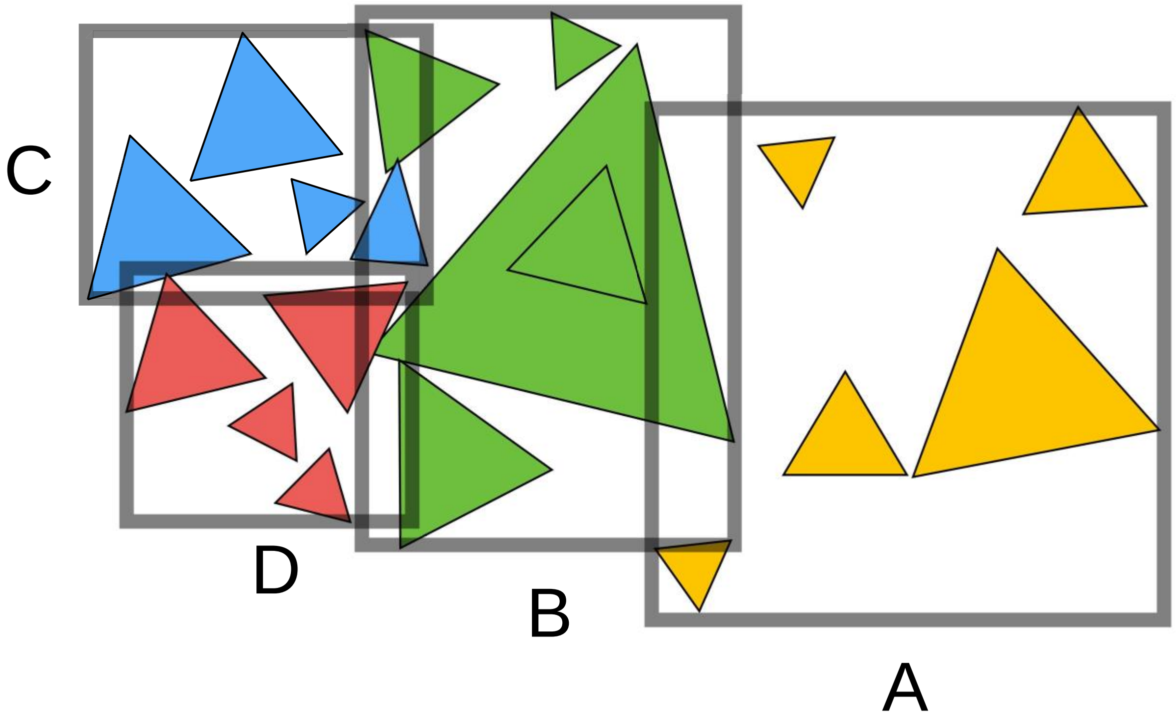


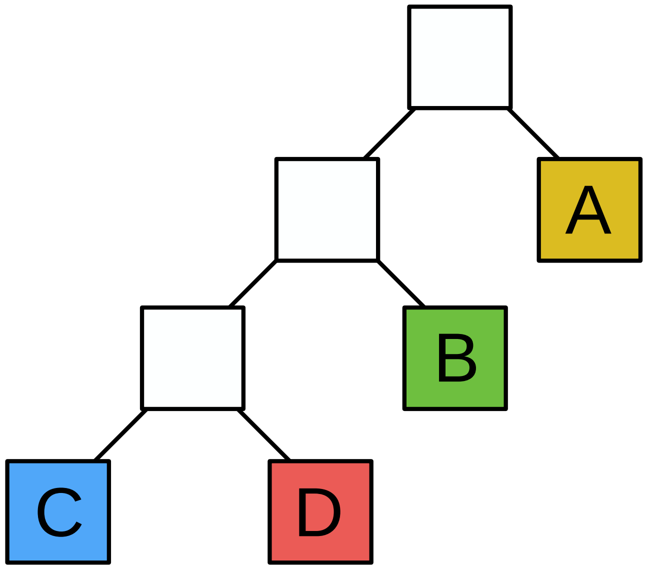
Bounding Volume Hierarchy (BVH)



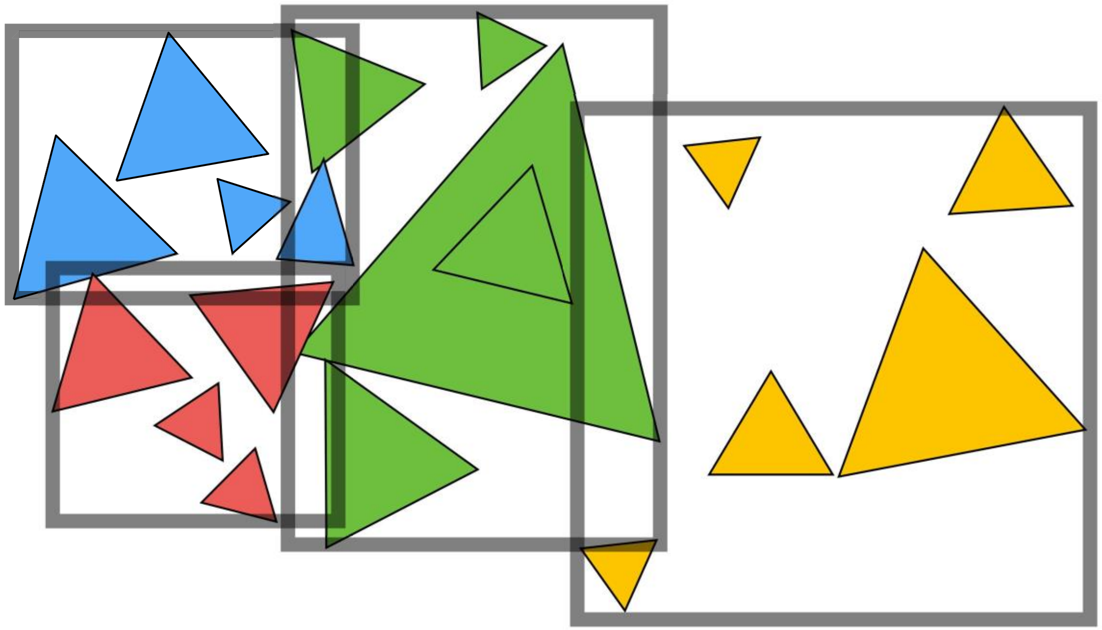


Bounding Volume Hierarchy (BVH)





Summary: Building BVHs



- Find bounding box
- Recursively split set of objects in two subsets
- Recompute the bounding box of the subsets
- Stop when necessary
- Store objects in each leaf node

Building BVHs

How to subdivide a node?

- Choose a dimension to split
- Heuristic #1: Always choose the longest axis in node
- Heuristic #2: Split node at location of median object

Termination criteria?

- Heuristic: stop when node contains few elements (e.g. 5)

Data Structure for BVHs

Internal nodes store

- Bounding box
- Children: pointers to child nodes

Leaf nodes store

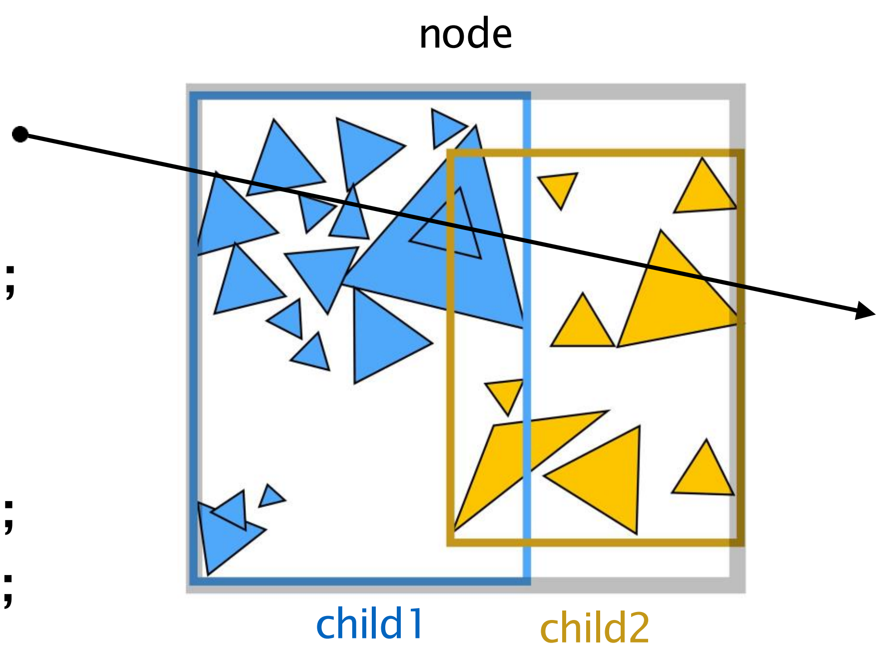
- Bounding box
- List of objects

Nodes represent subset of primitives in scene

- All objects in subtree

BVH Traversal

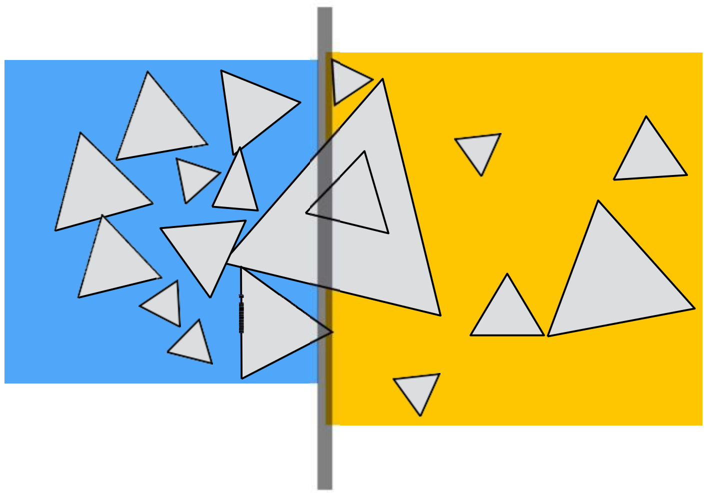
```
Intersect(Ray ray, BVH node) {
  if (ray misses node.bbox) return;

  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 the closer of hit1, hit2;
}
```



Spatial vs Object Partitions

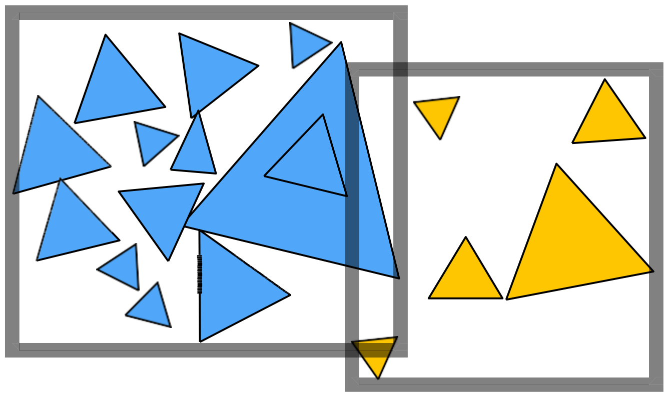
Spatial partition (e.g.KD-tree)

- Partition space into non-overlapping regions
- An object 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



Thank you!

(And thank Prof. Ravi Ramamoorthi and Prof. Ren Ng for many of the slides!)