Raytracing

Computer Graphics and Imaging
UC Berkeley CS184
Summer 2020

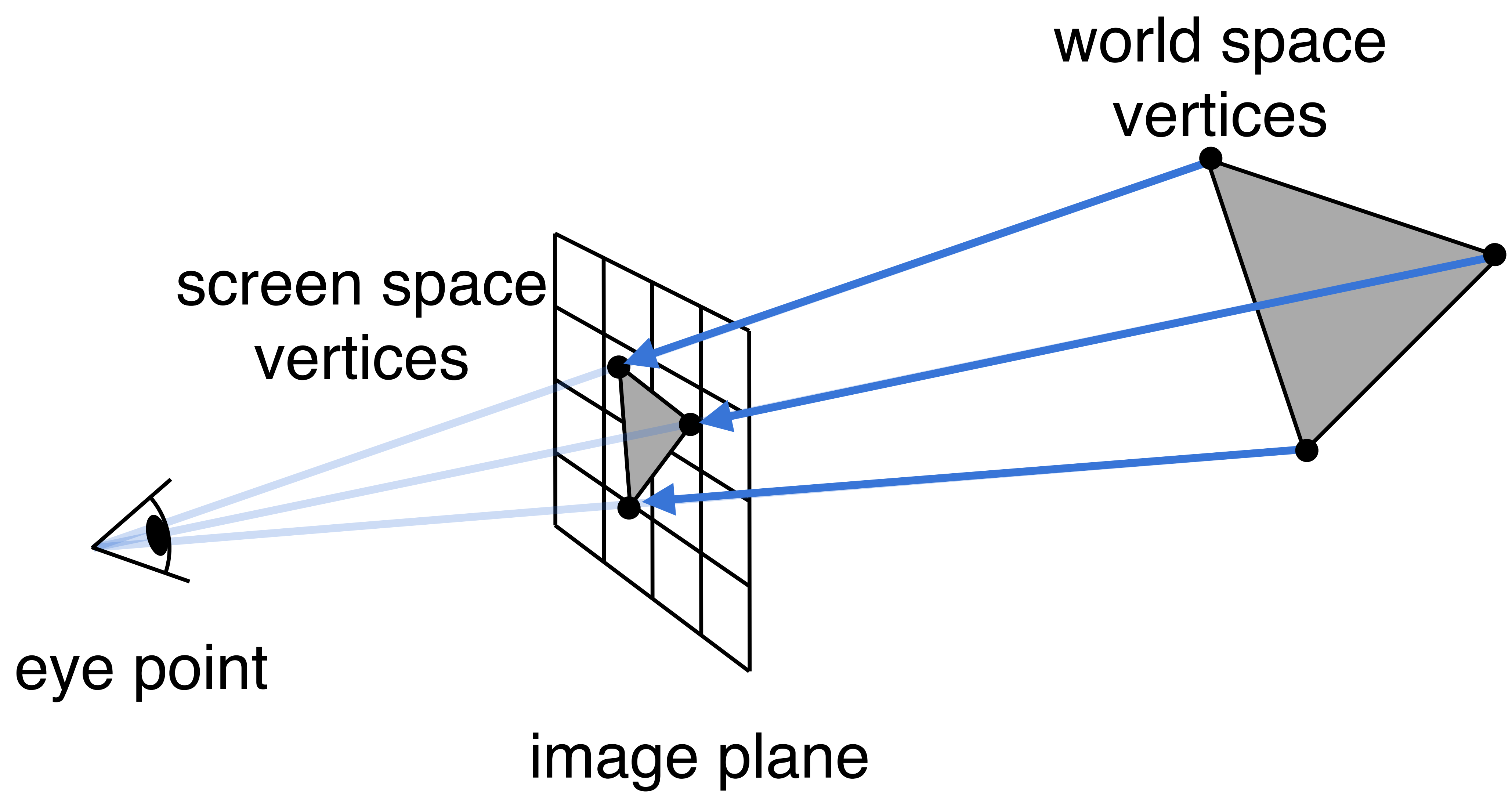
Reminder

- Post questions for live Q&A tomorrow on Piazza

Overview

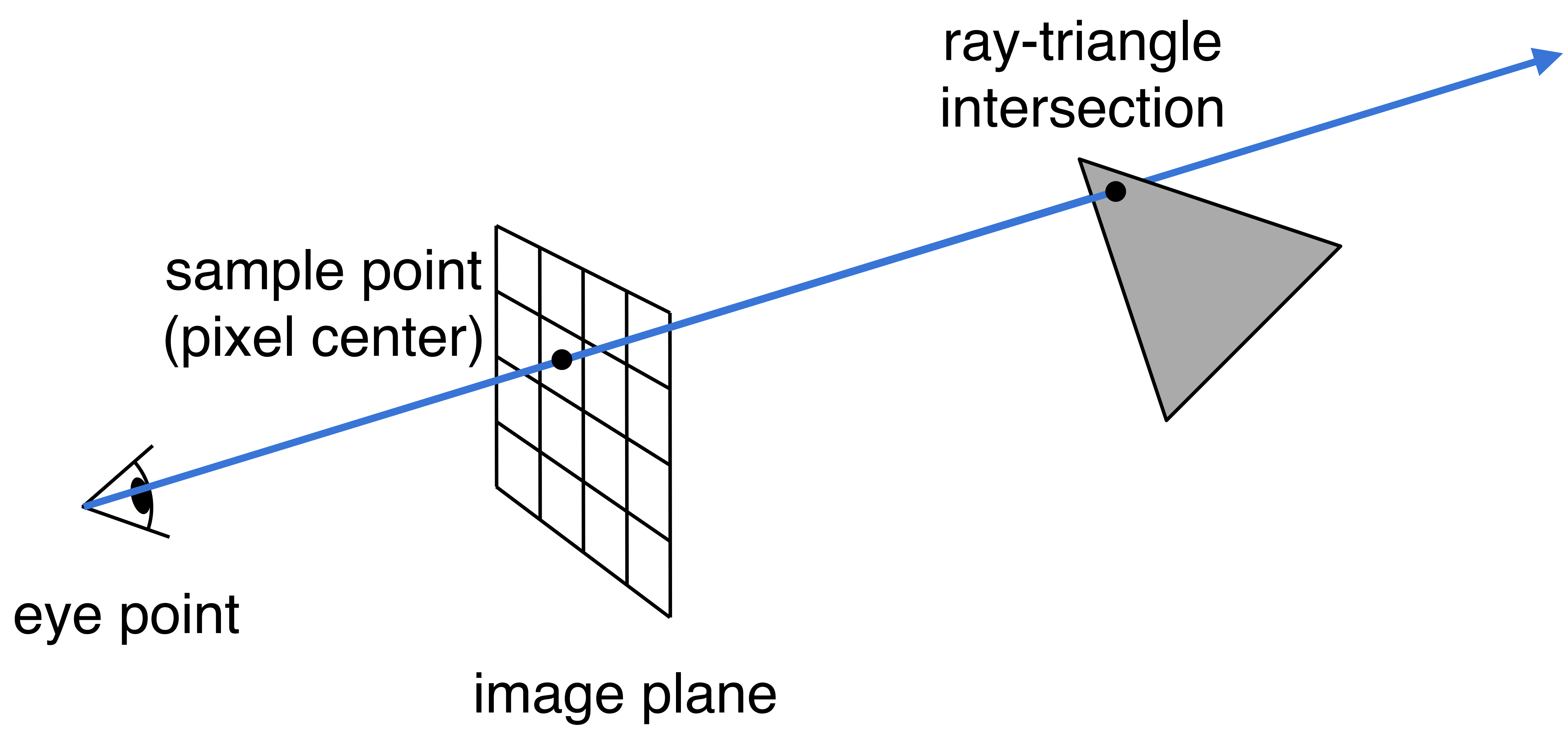
- Raytracing
  - Rasterization vs raytracing
  - Camera rays
  - Basic and recursive raytracing
- Bounding volume hierarchies
  - Ray-box intersection
  - BVH creation, choosing split point
- Quick raytracing demo

Rasterization vs Raytracing



Rasterization: **project vertices down into screen space, then test whether each sample point is inside**

Rasterization vs Raytracing



Raytracing: project sample points out into 3D world using rays, intersect against all scene objects

Rasterization vs Raytracing

Rasterization loop

```
for each tri:
  tri_screen = project(tri)
  for each pixel:
    inside(pixel, tri_screen)
```

Raytracing loop

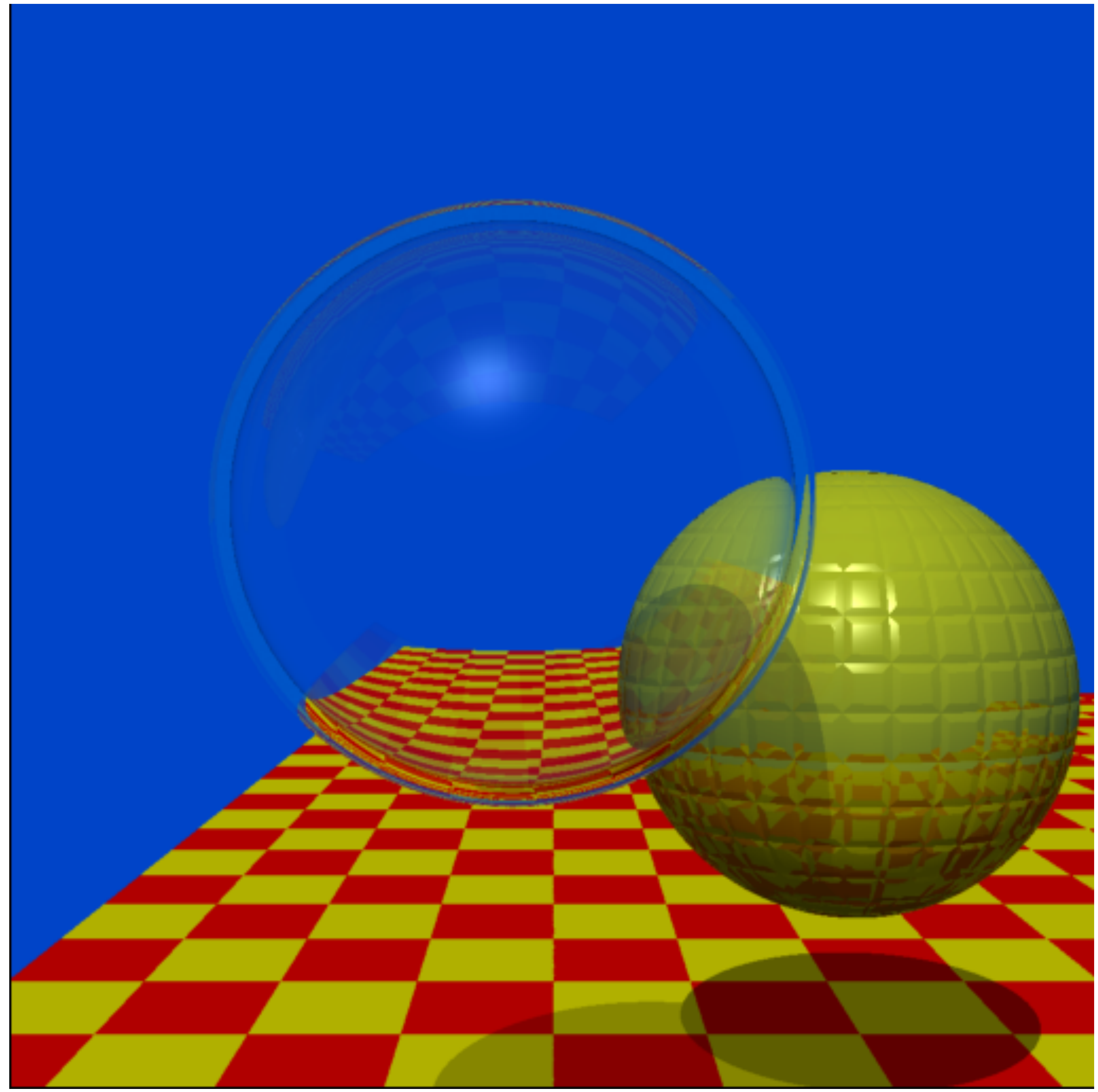
```
for each pixel:
  ray = camera_ray(pixel)
  for each tri:
    intersects(ray, tri)
```

Rasterization vs Raytracing

Rasterization shading:
only use whatever information has been passed into shader

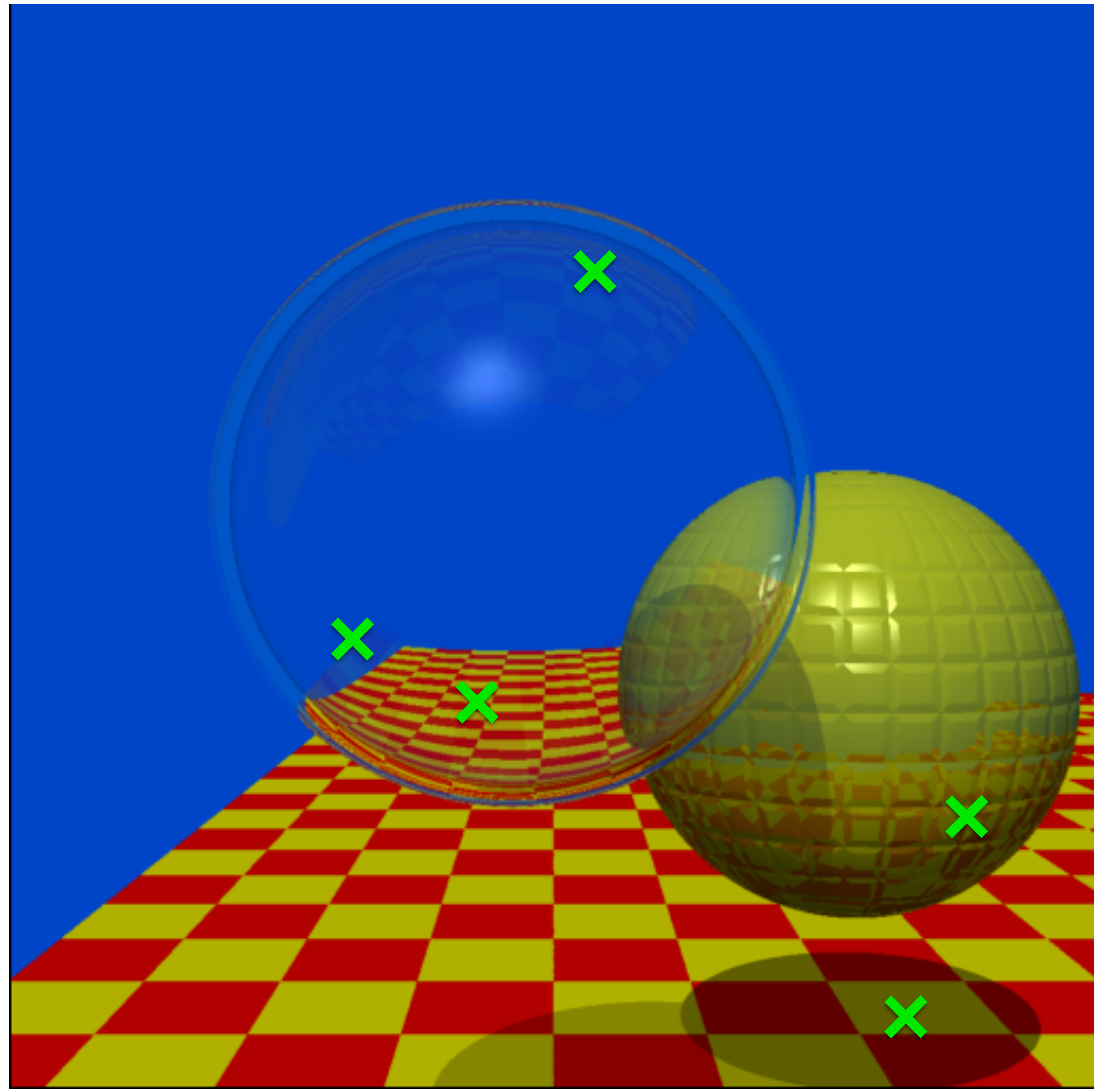
Raytracing shading:
trace rays into rest of scene to calculate shadows, reflections, etc.

Basic rasterization doesn’t allow for “global” phenomena



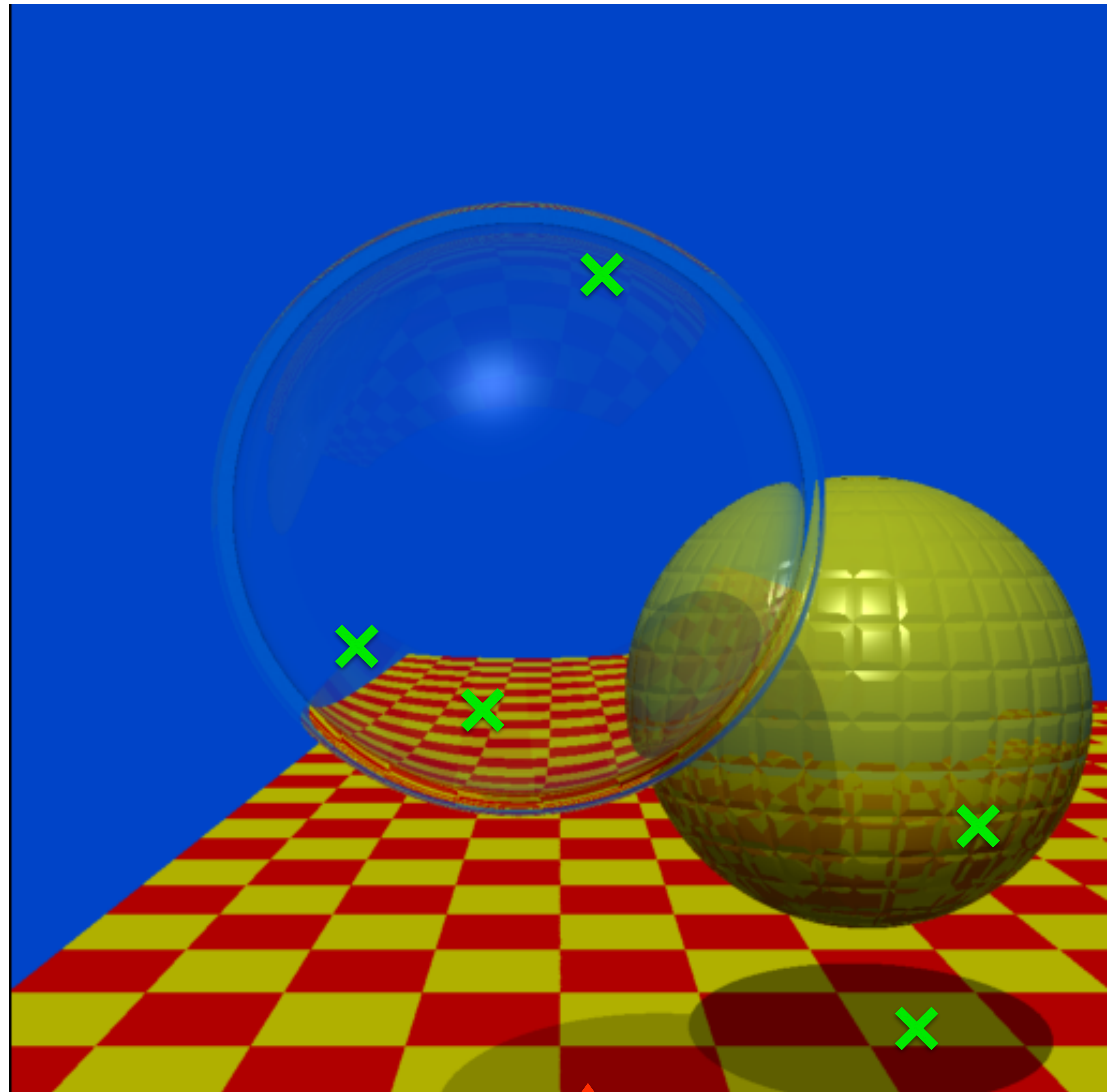
Basic rasterization doesn't allow for "global" phenomena

- Shadows
- Reflection
- Refraction



Basic rasterization doesn't allow for "global" phenomena

- Shadows
- Reflection
- Refraction



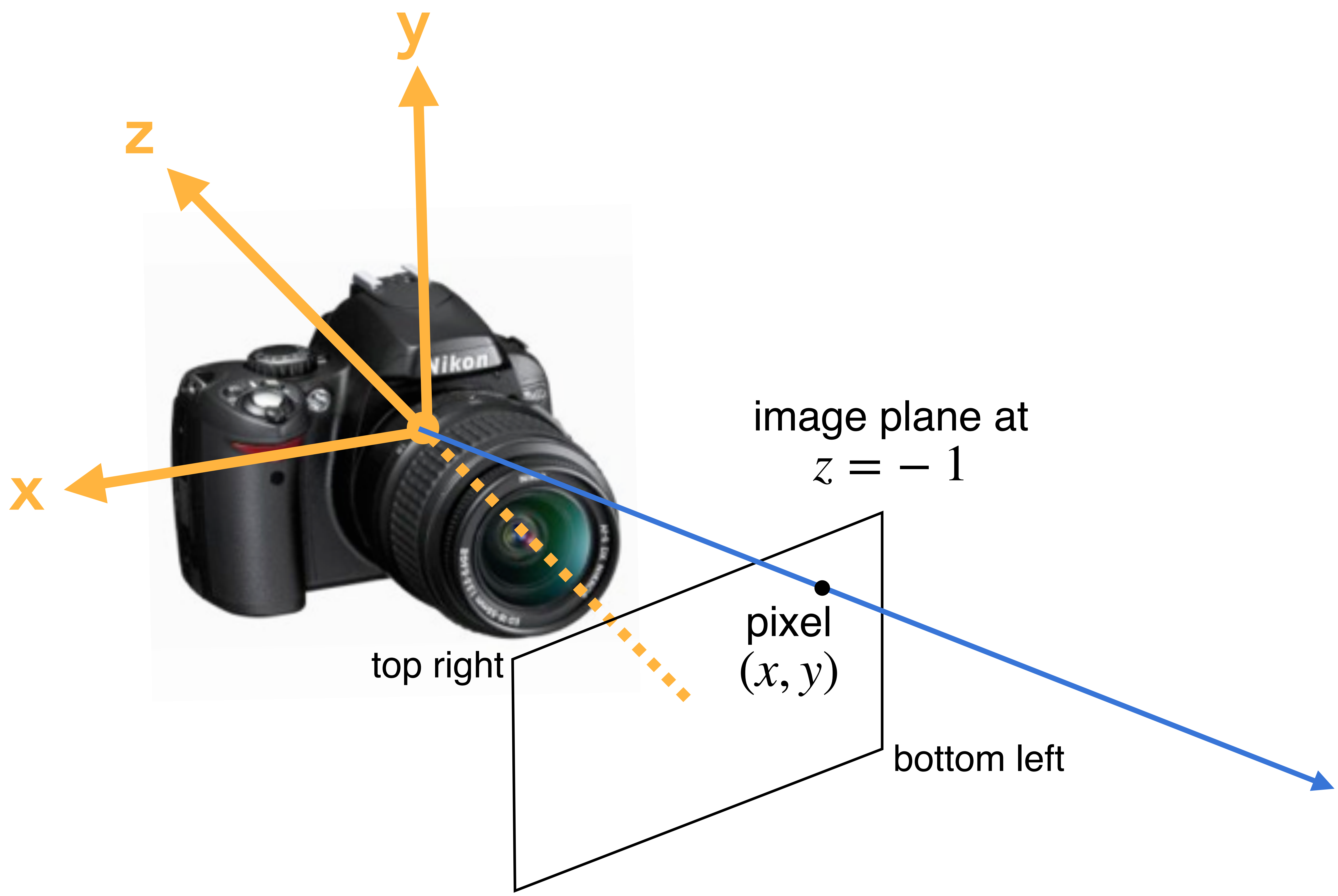
Raytracing giveaway - shadows under transparent objects

“Trace ray” is a very powerful function

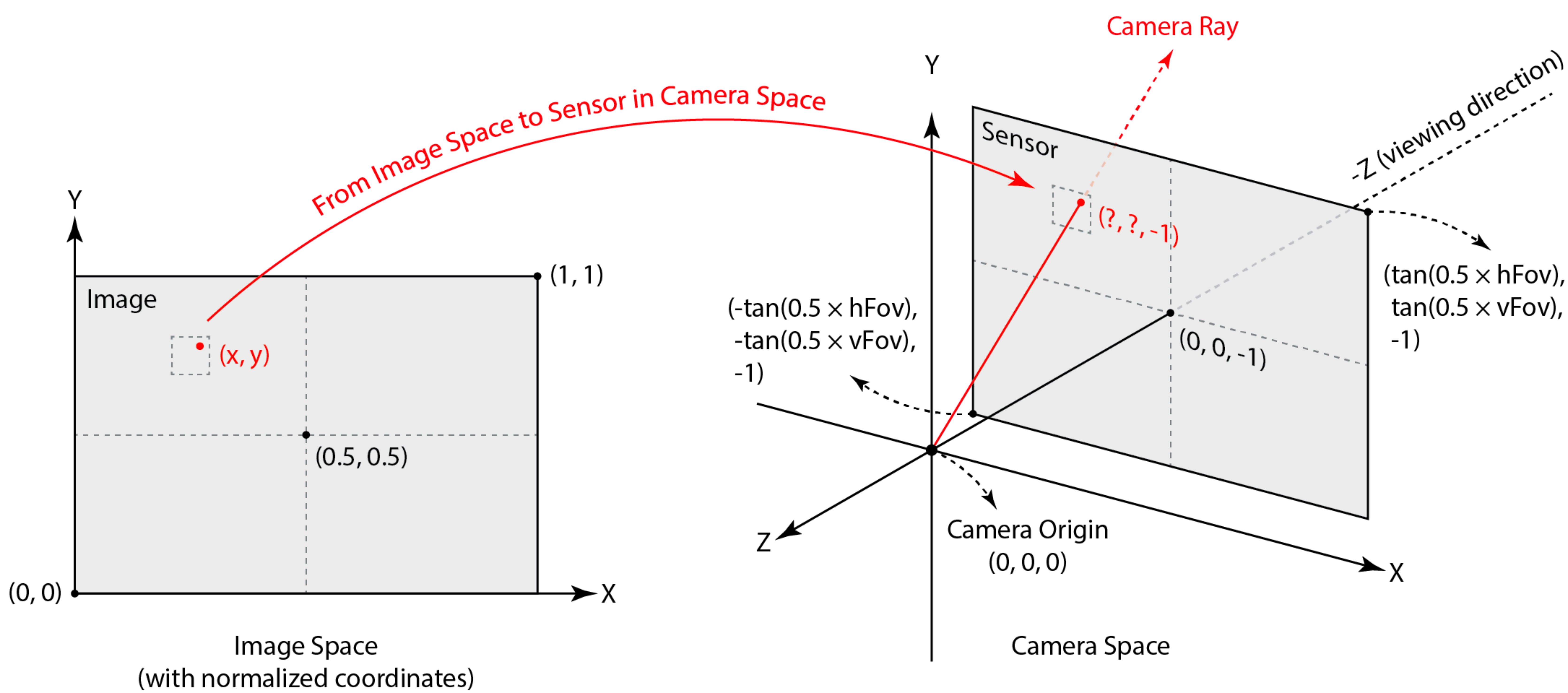
- All calculation done in world space
- Many reasons you want to trace a ray
  - Which shape does this camera pixel see?
  - Which lights are visible from this point in space?
  - If a ray bounces off this mirror, what does it see?
- Matches the way light propagates in reality

Primary “camera” or “eye” rays

Camera space reminder



Camera space reminder

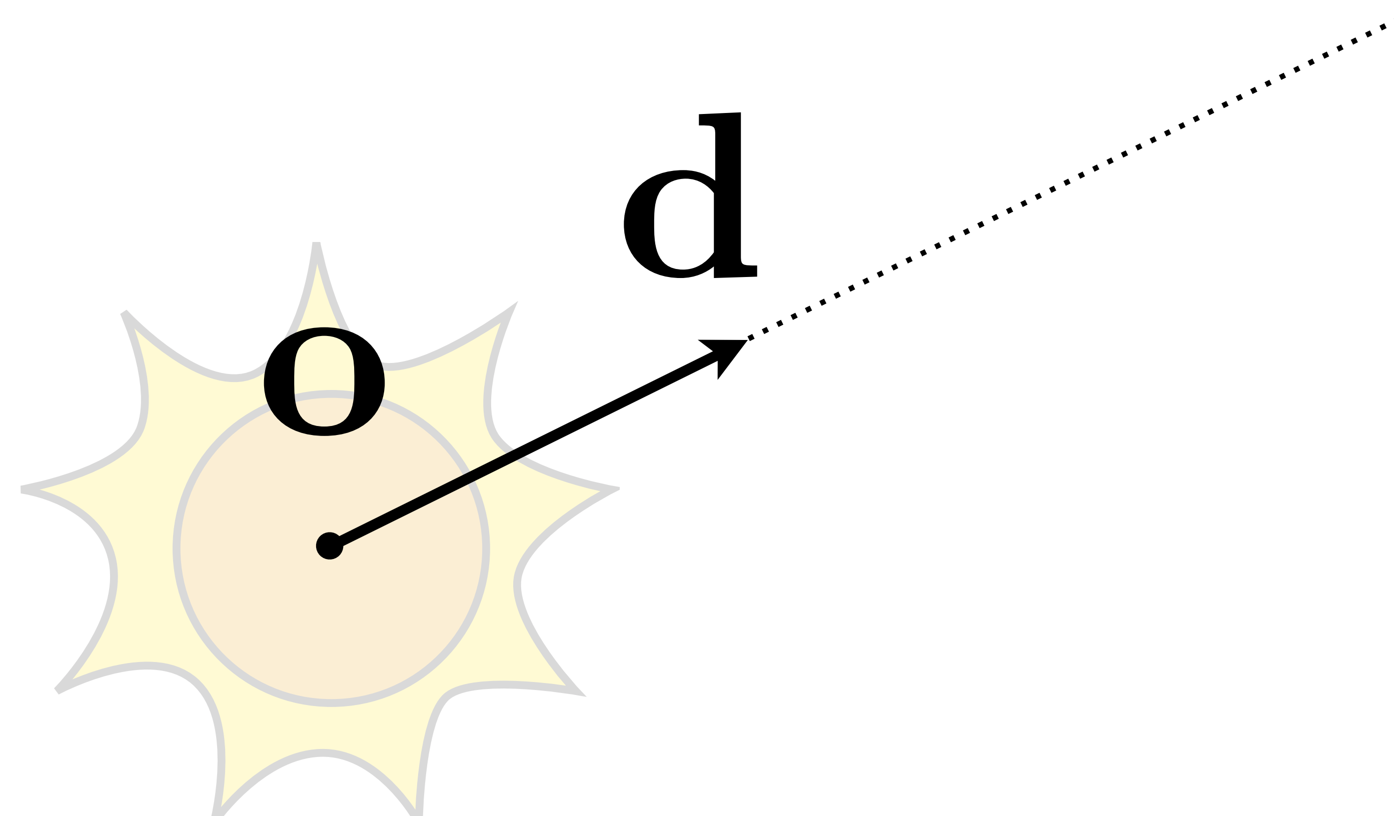


What is a ray?

Ray Equation

Ray is defined by its origin and a direction vector

Example:



Ray equation:

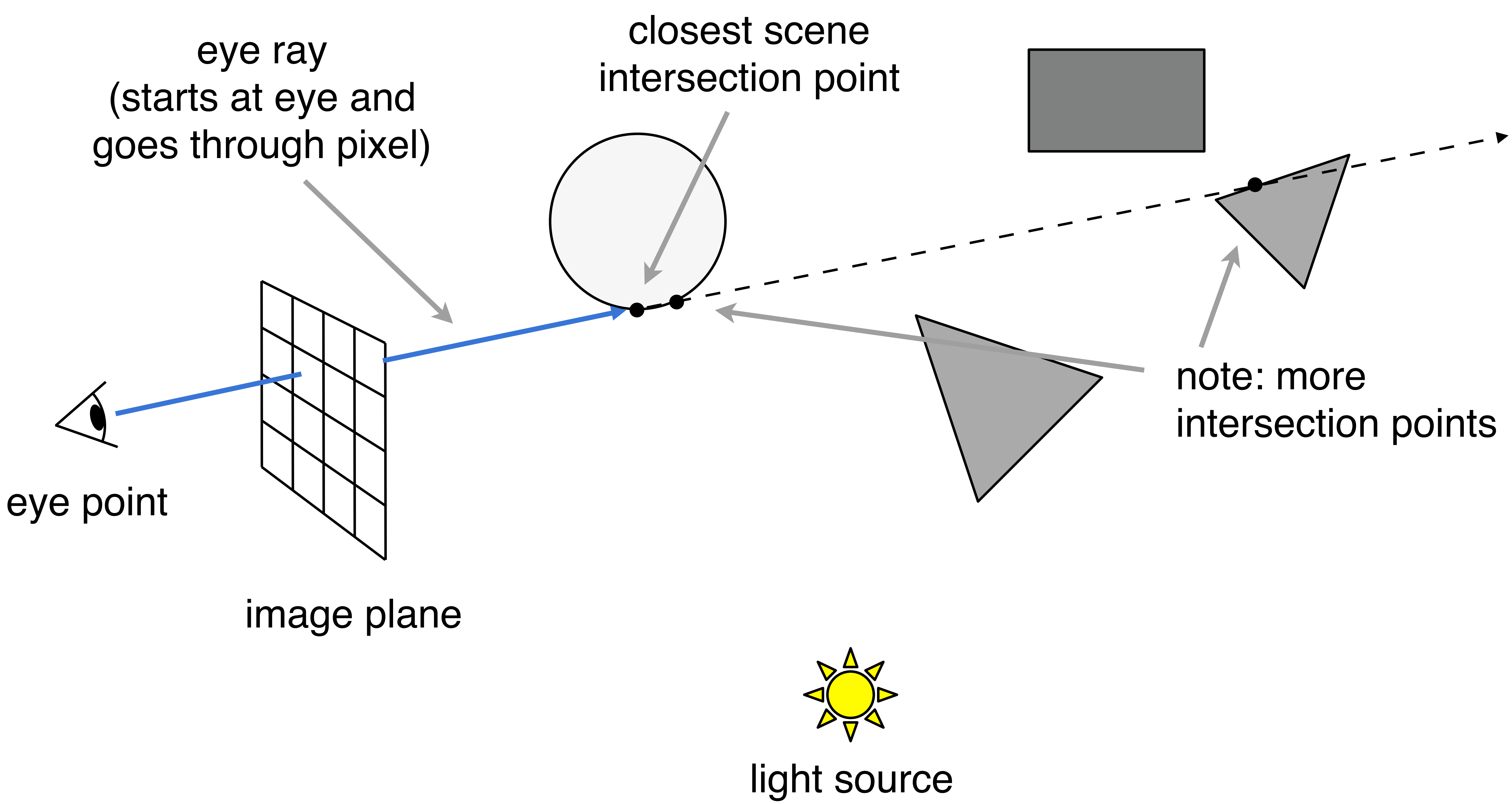
$$\mathbf{r}(t) = \mathbf{o} + t\mathbf{d} \qquad t_{\min} \leq t \leq t_{\max}$$

point along ray "time" origin unit direction

Basic Ray-Tracing Algorithm

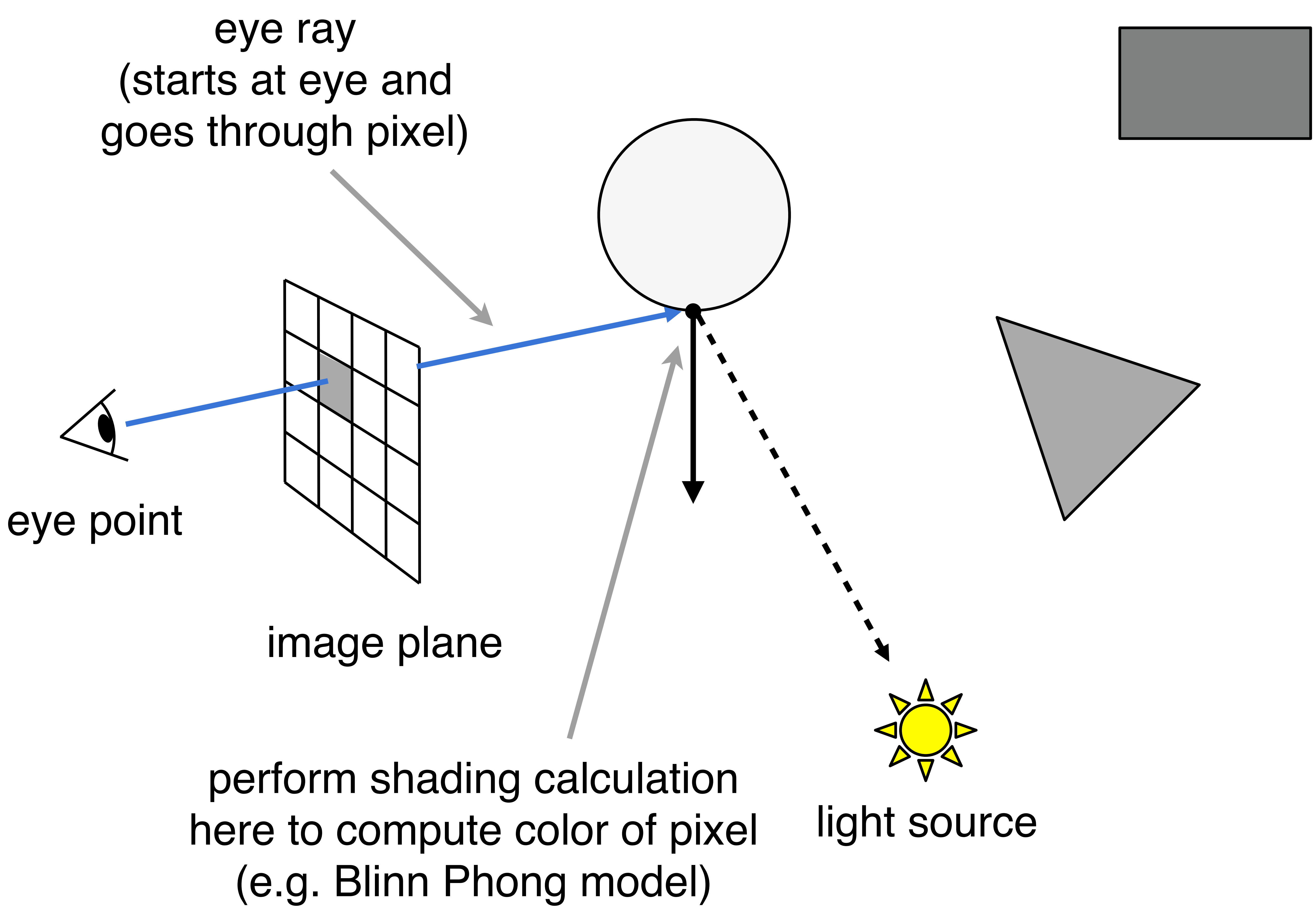
Ray Casting - Generating Eye Rays

Pinhole Camera Model



Ray Casting - Shading Pixels (Local Only)

Pinhole Camera Model

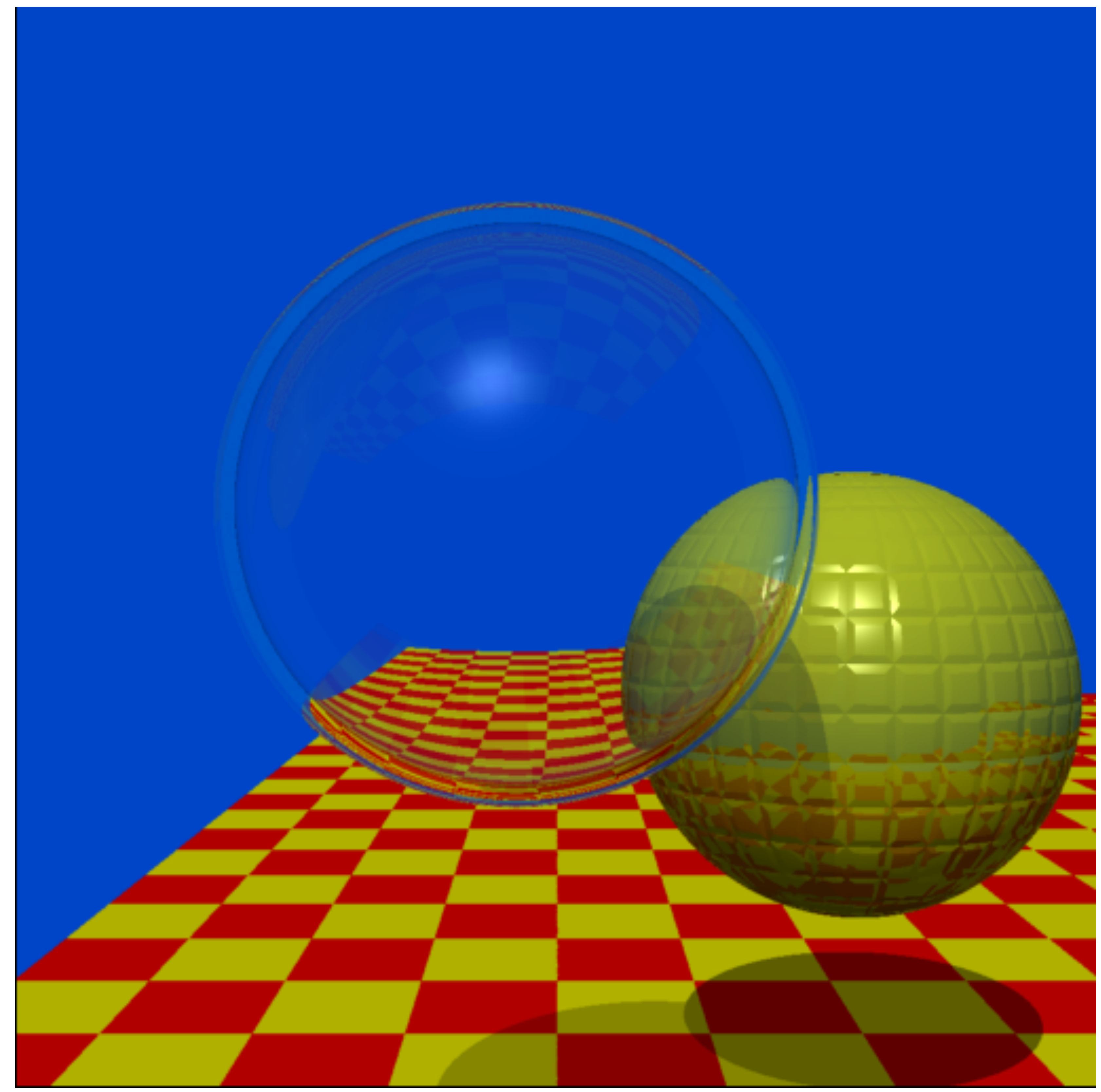


Recursive Ray Tracing

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

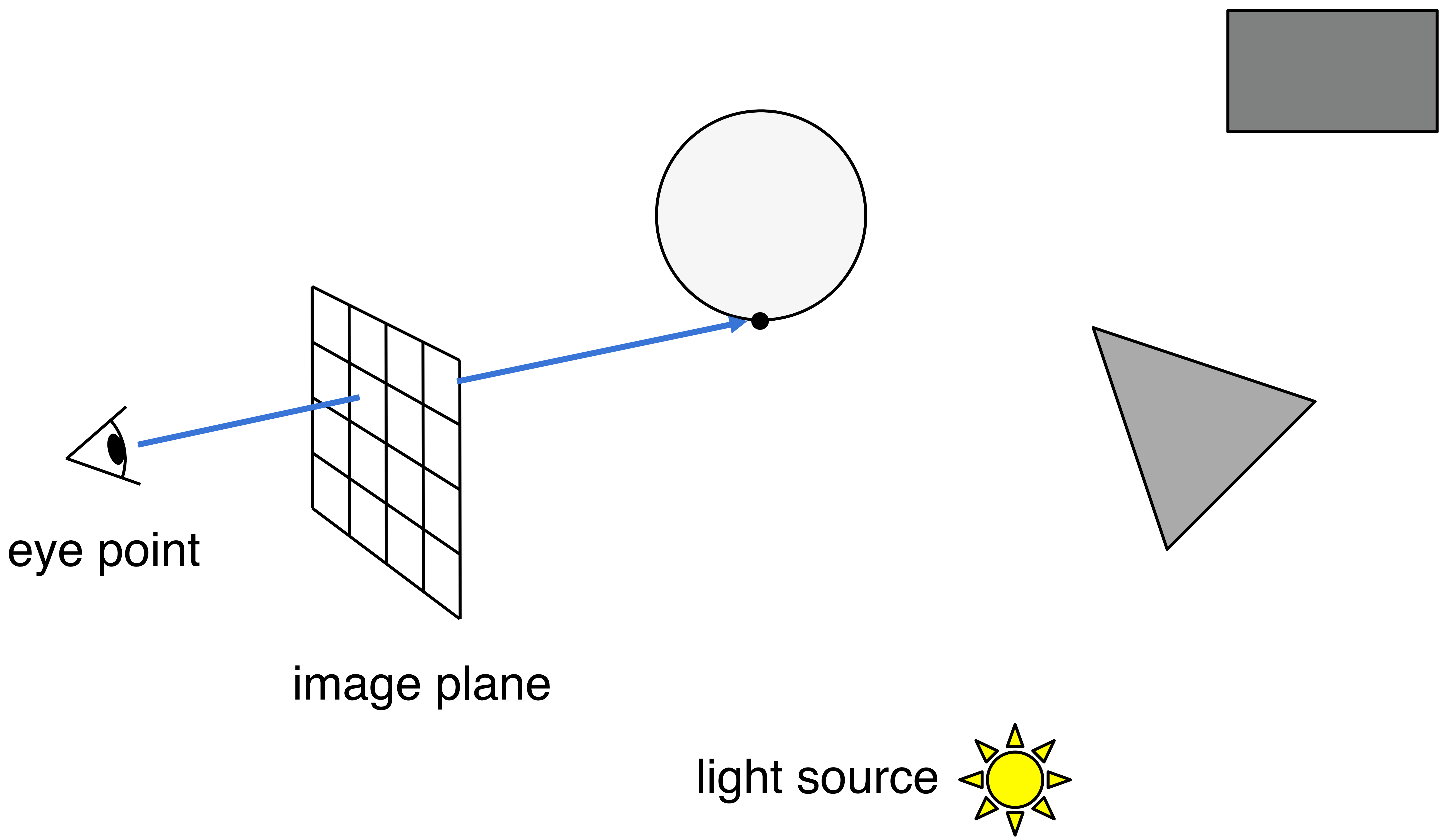
Time:

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

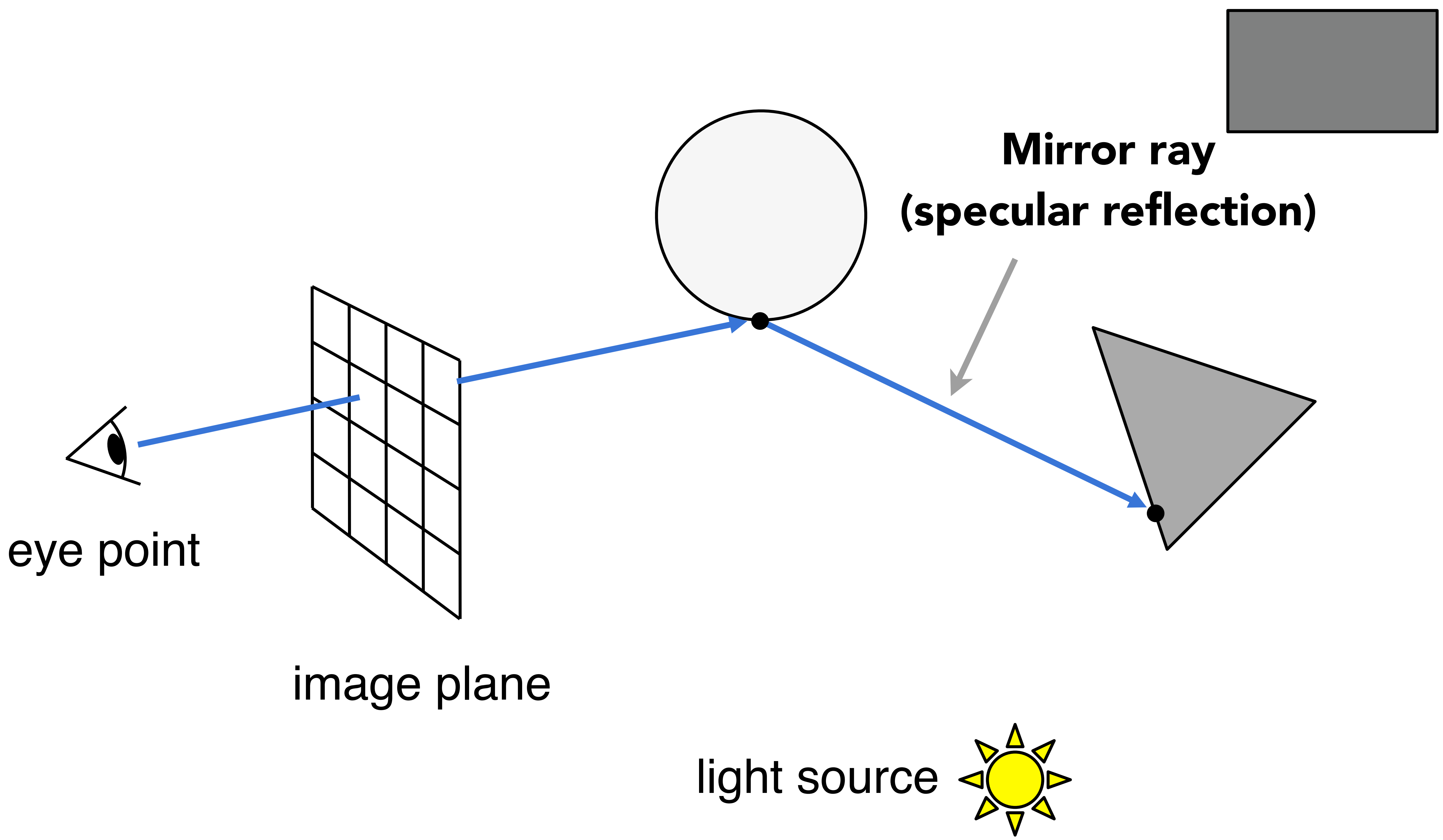


Spheres and Checkerboard, T. Whitted, 1979

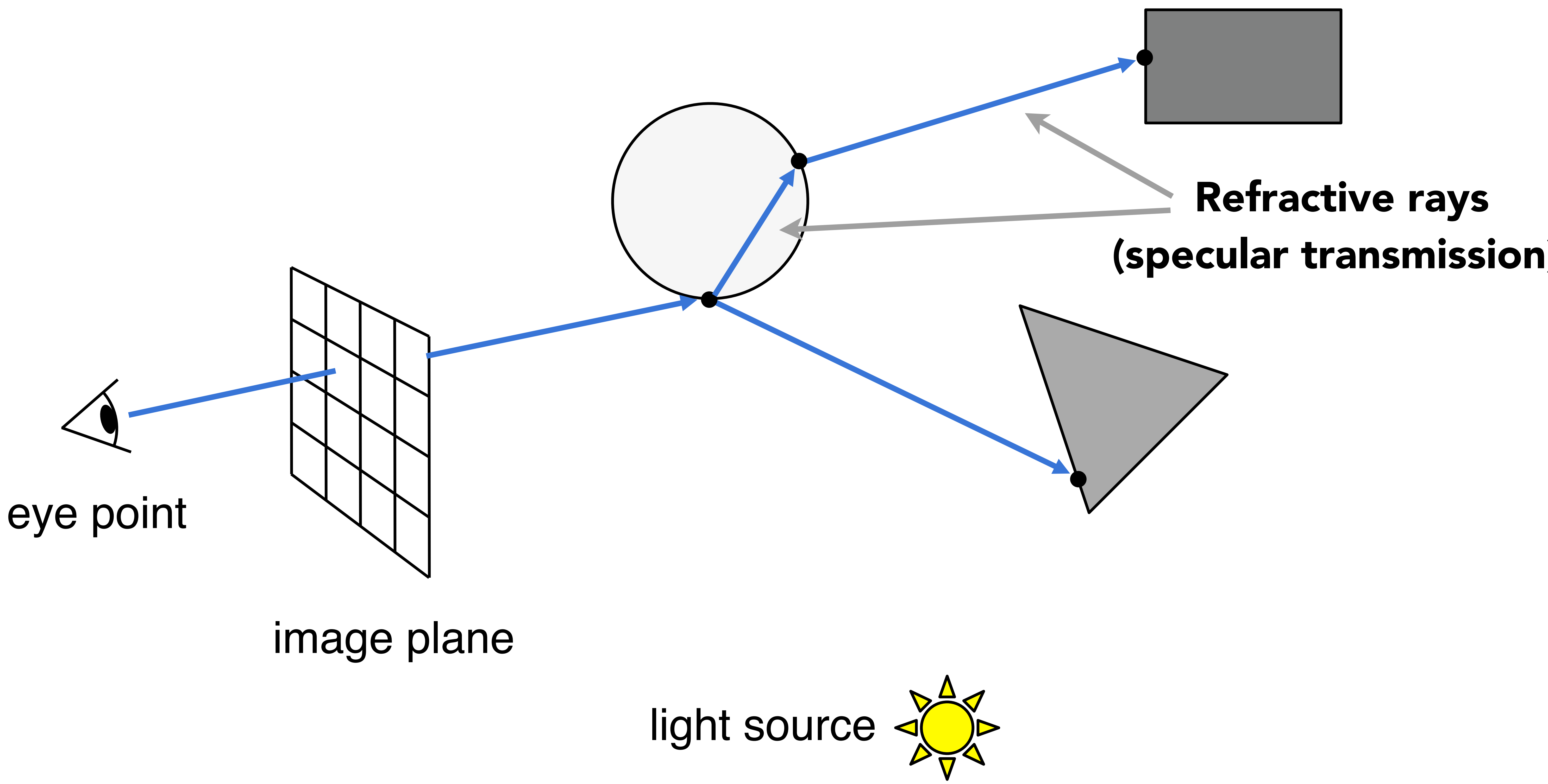
Recursive Ray Tracing



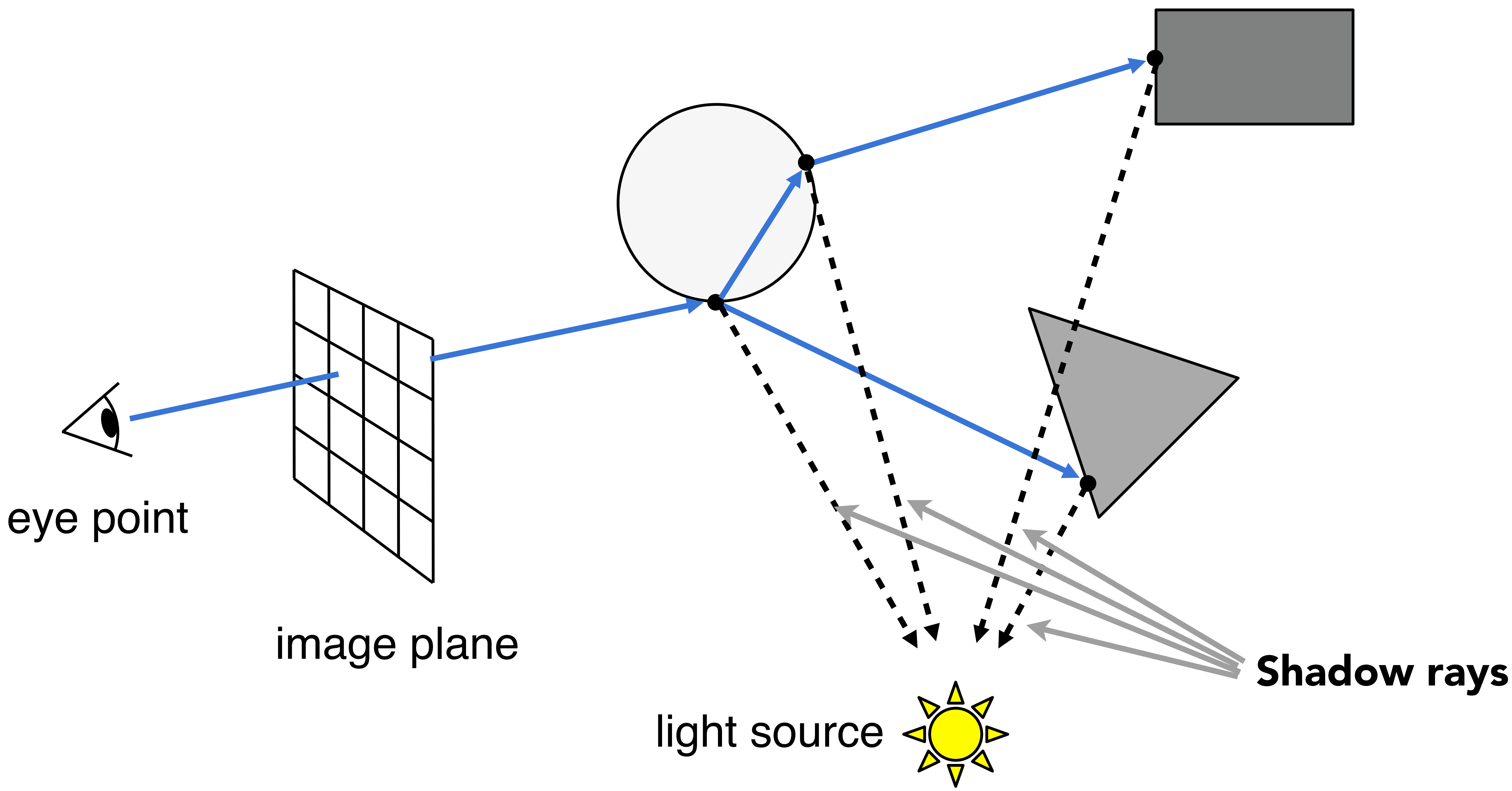
Recursive Ray Tracing



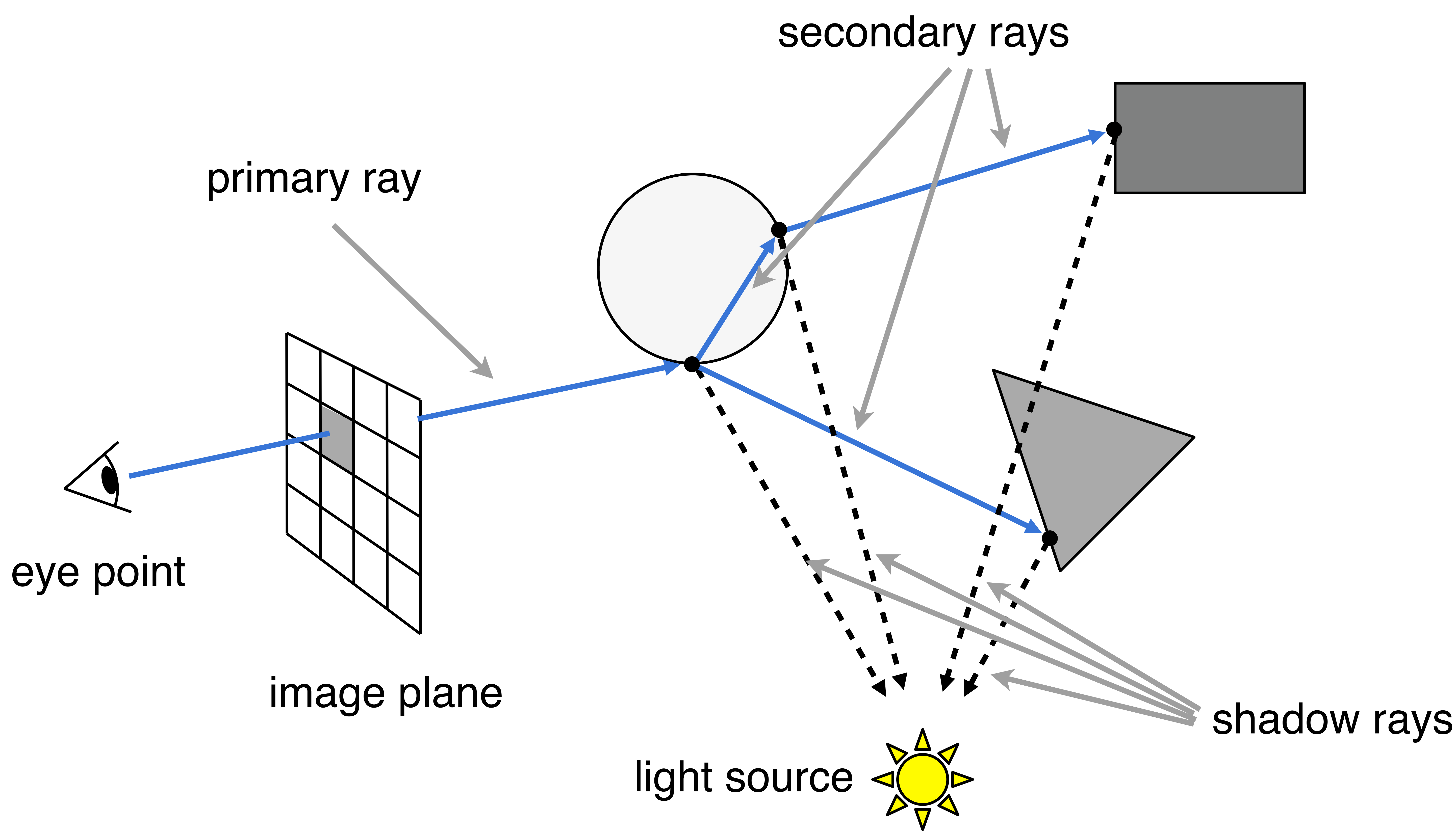
Recursive Ray Tracing



Recursive Ray Tracing



Recursive Ray Tracing

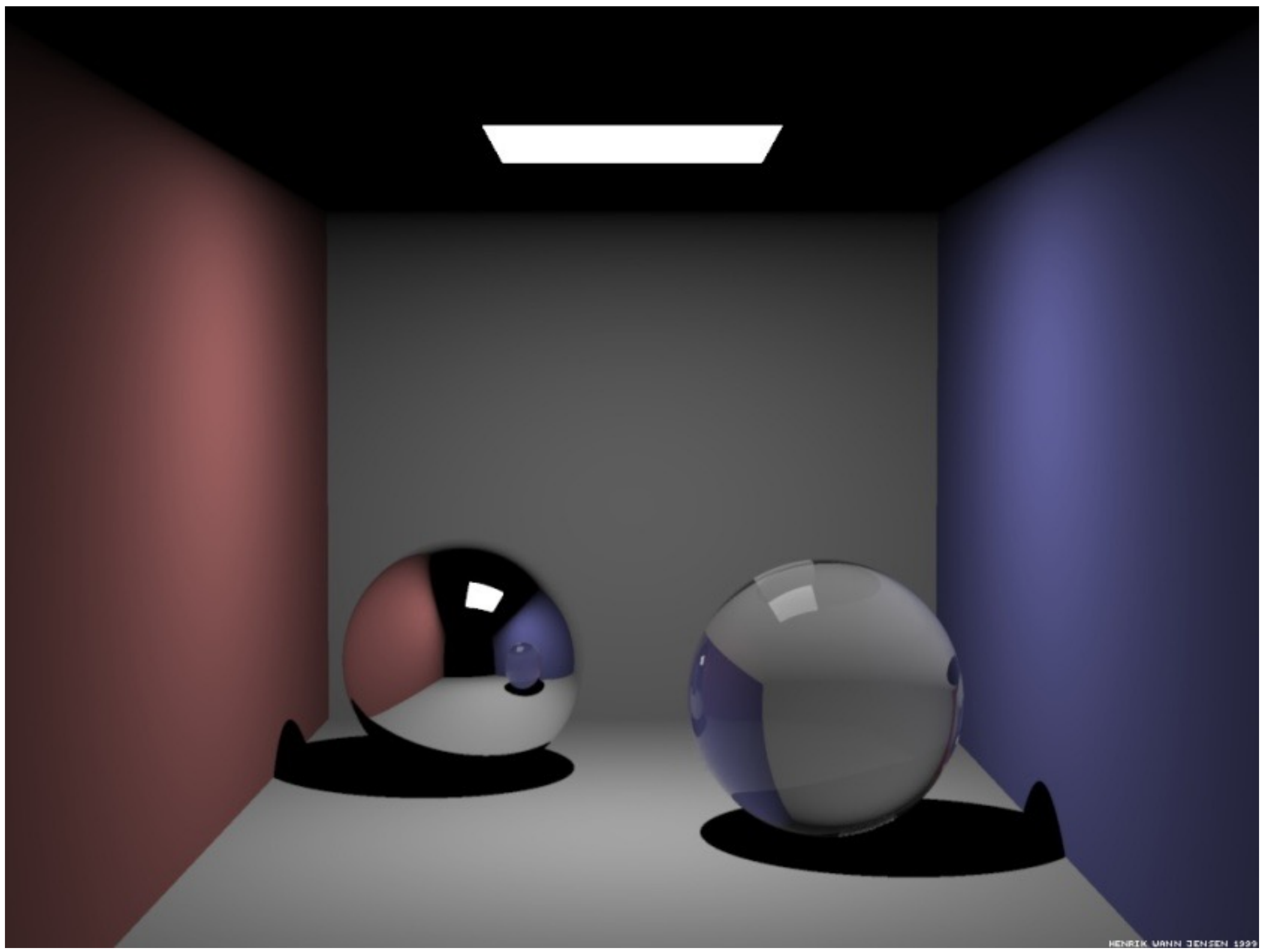


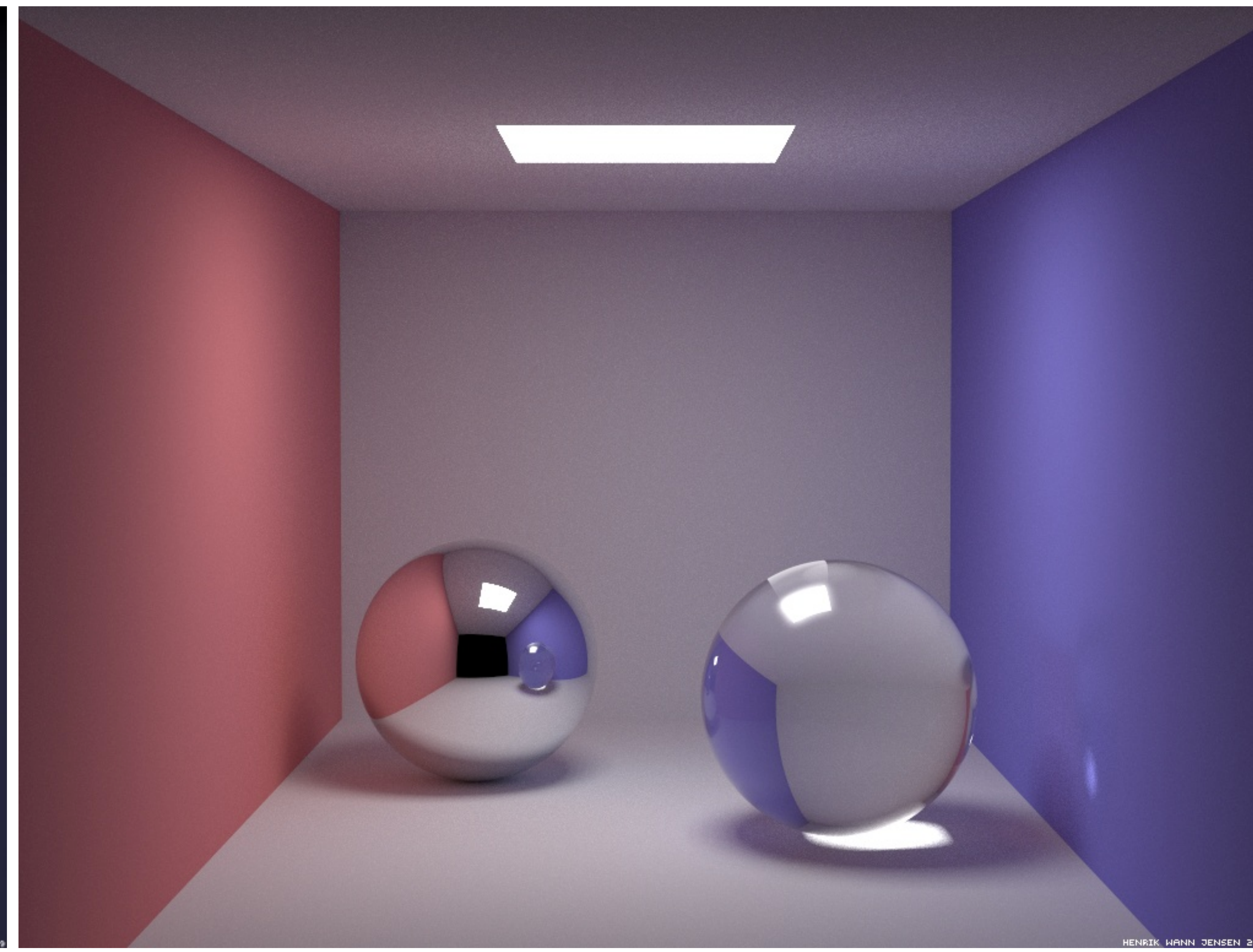
- Trace secondary rays recursively until hit a non-specular surface (or max desired levels of recursion)
- At each hit point, trace shadow rays to test light visibility (no contribution if blocked)
- Final pixel color is weighted sum of contributions along rays, as shown
- Gives more sophisticated effects (e.g. specular reflection, refraction, shadows), but we will go much further to derive a physically-based illumination model

Note: Raytracing vs. pathtracing

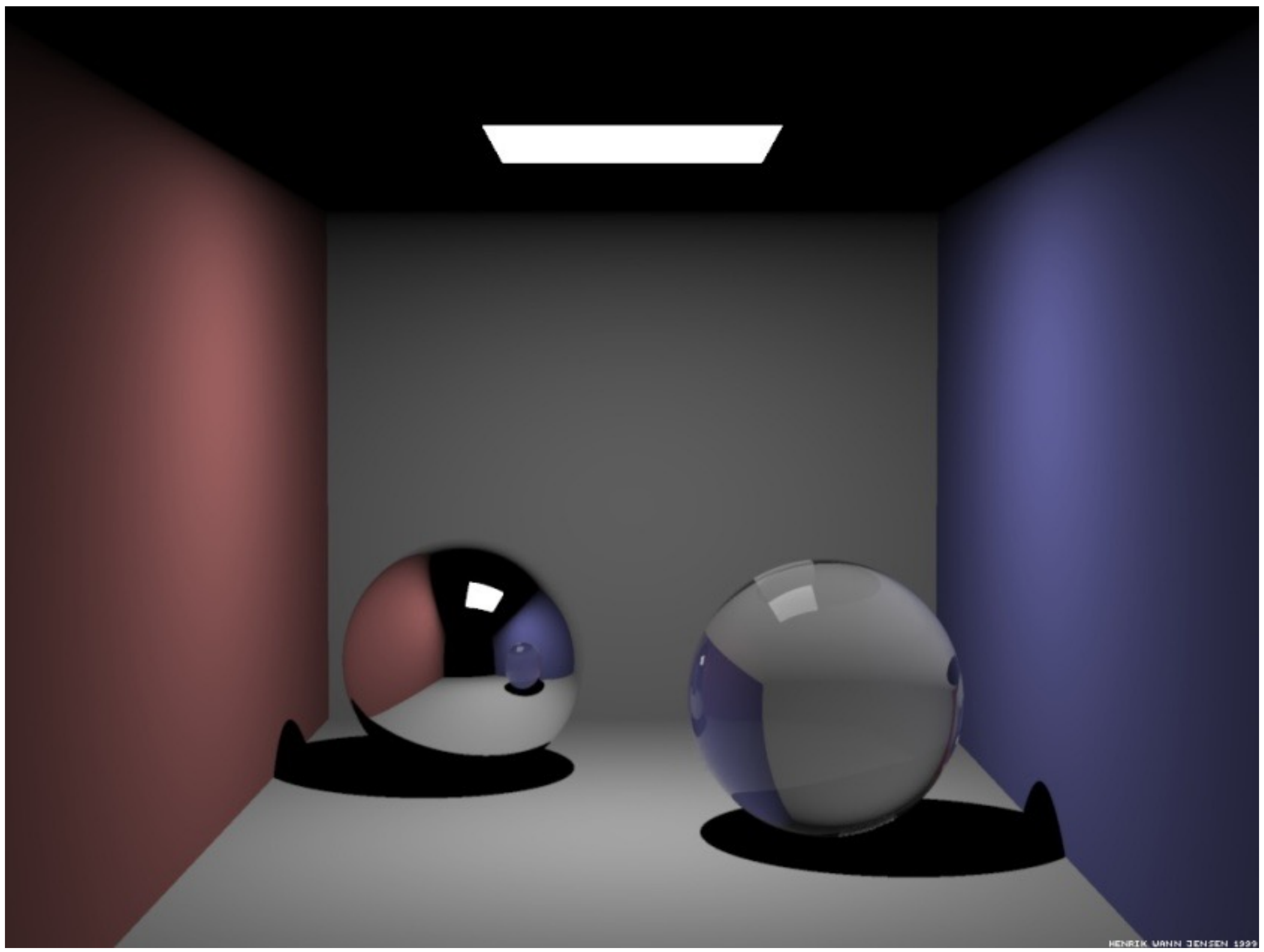
- Not very well defined in graphics community, but…
- One common interpretation is that “raytracing” is when you *stop* at the first non-specular surface you hit, only tracing shadow rays from that point
  - This means no “indirect” illumination: no “diffuse to diffuse” bounces
- Random sampling allowed for simple effects like area lights (soft shadows) or depth of field blur
  - Often called “distributed” ray tracing

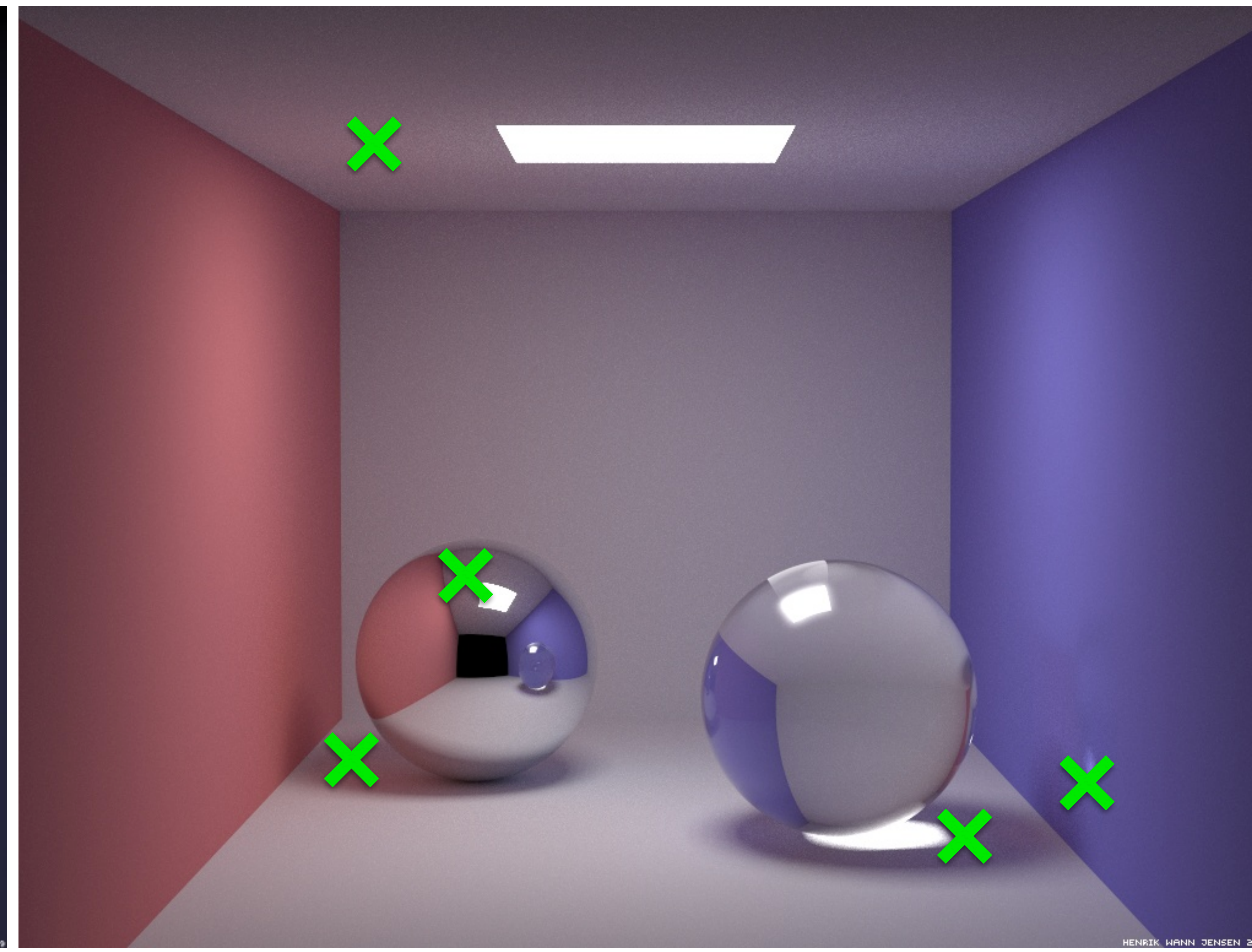
What are the differences between these images?





What are the differences between these images?





Accelerating Ray Tracing: Bounding Volumes

Bounding Volumes

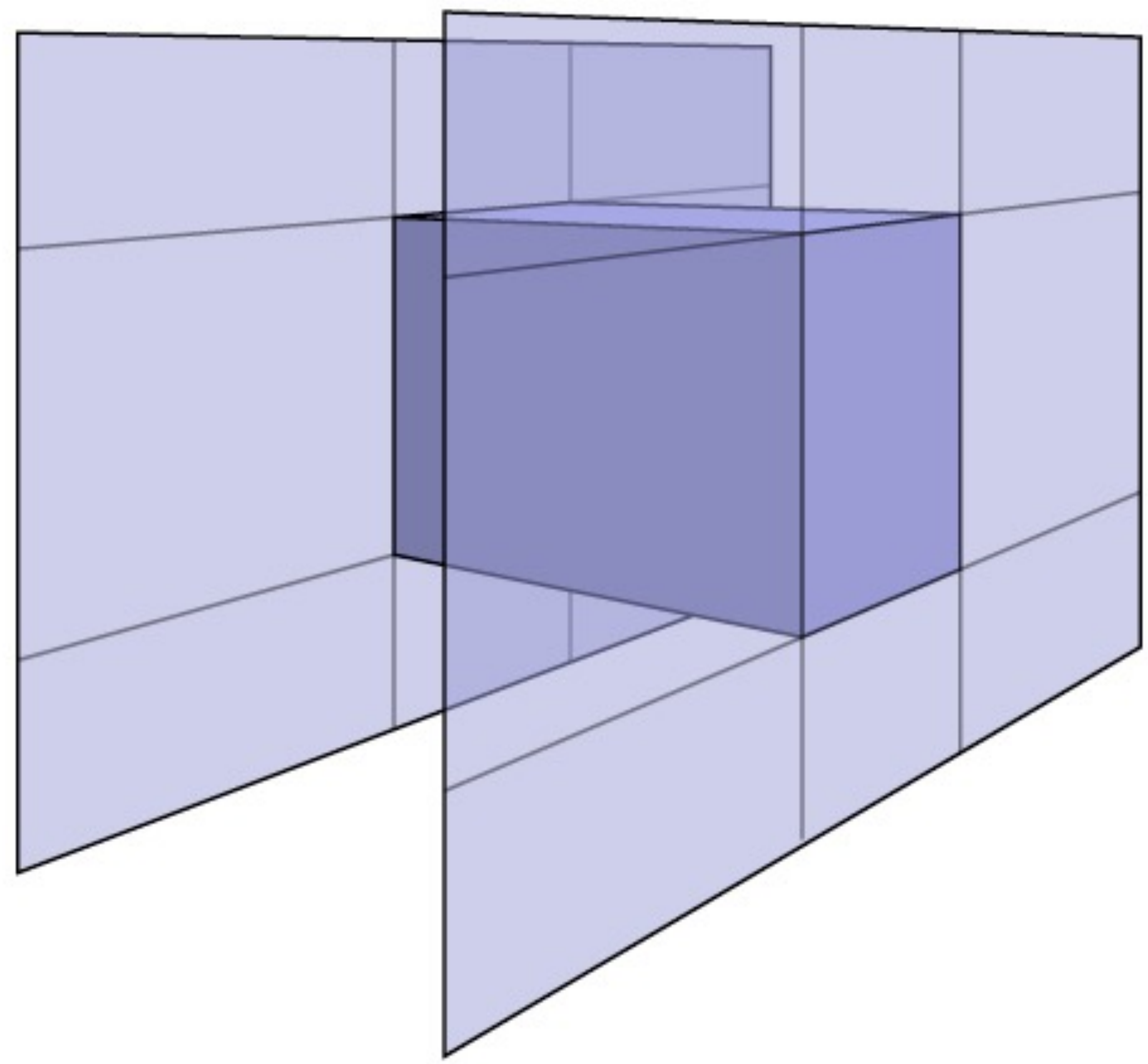
Quick way to avoid intersections: bound complex object with a simple volume

- 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
- Reminiscent of using triangle's screen space bounding box during rasterization

Ray-Intersection With Box

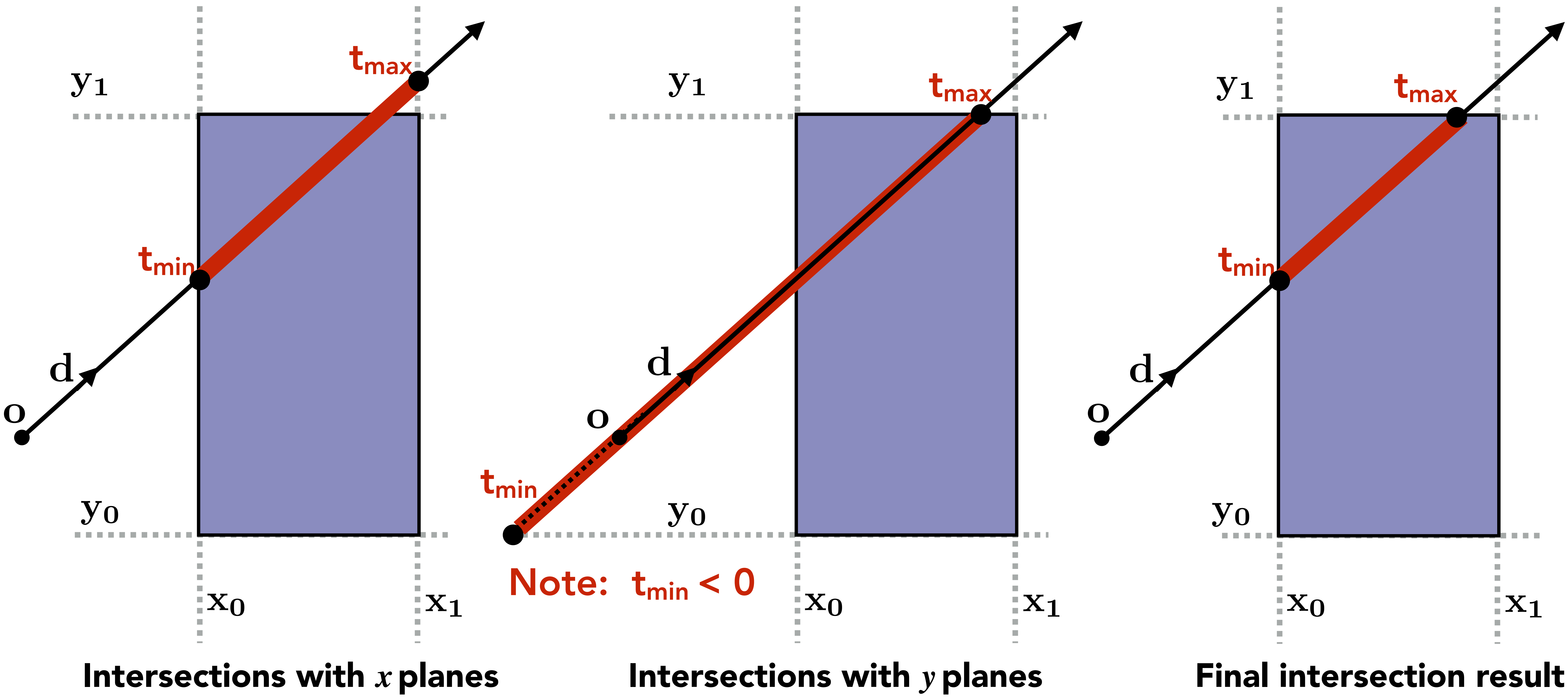
Could intersect with 6 faces individually

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

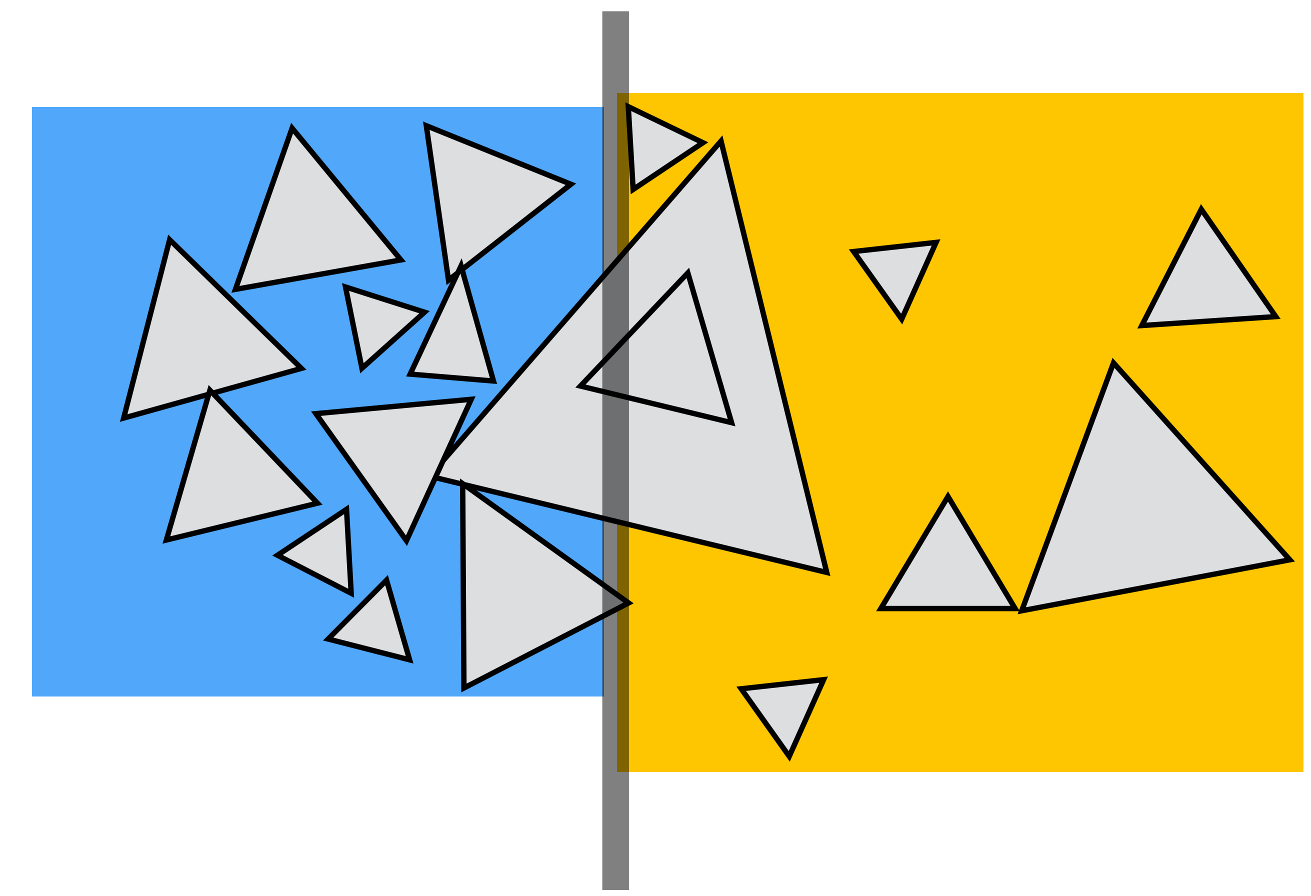


How do we know when the ray misses the box?

Spatial vs Object Partitions

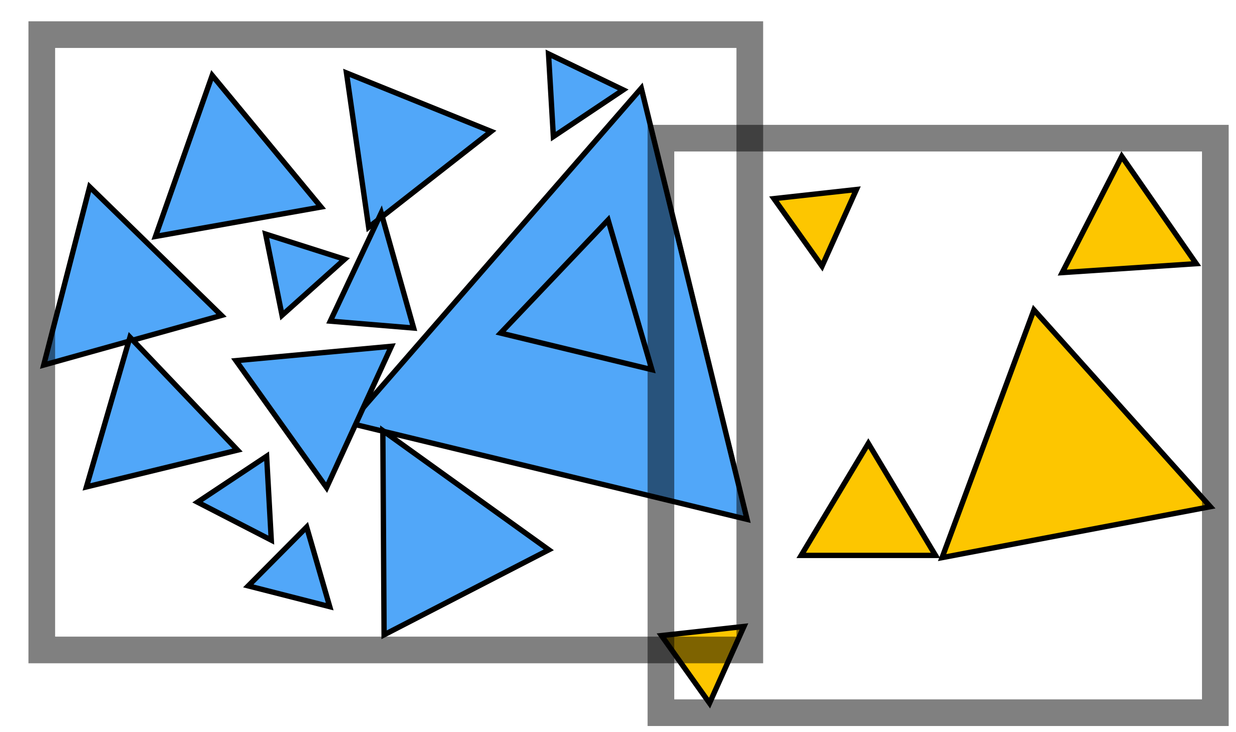
Spatial partition (e.g.KD-tree)

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

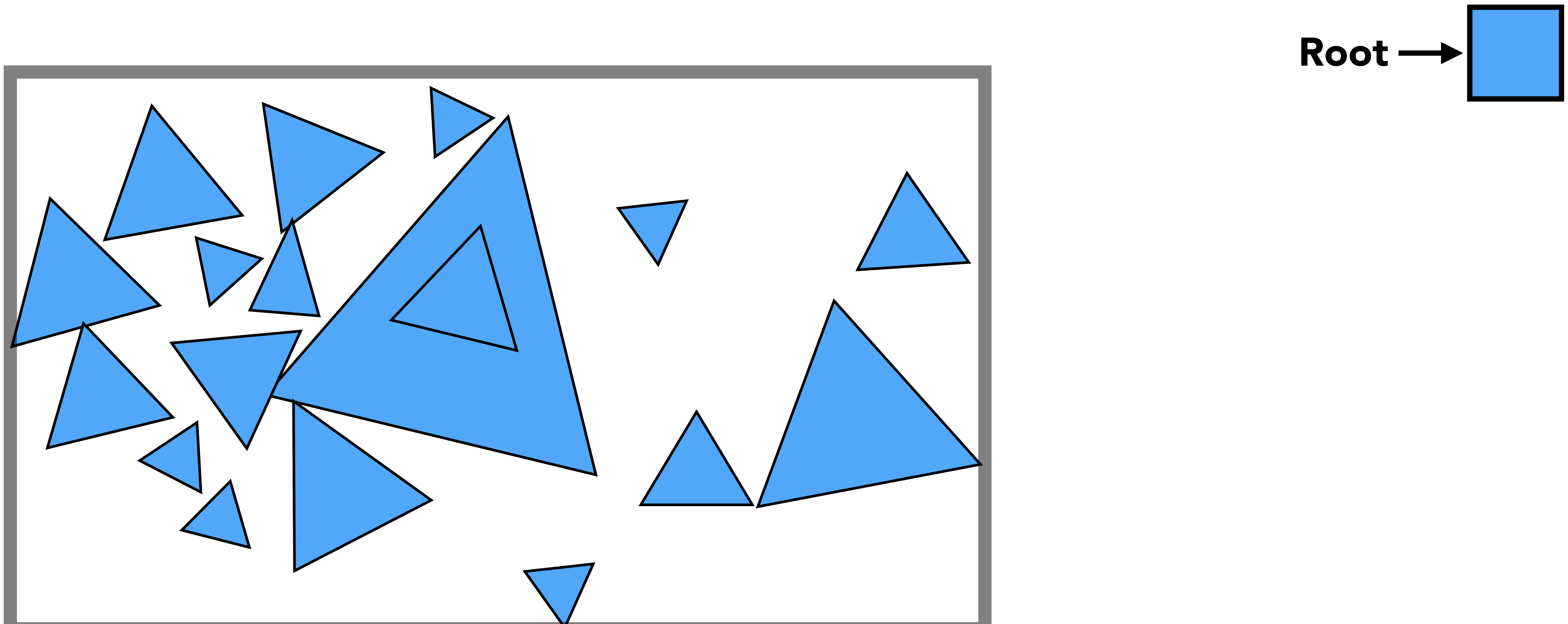


Object partition (e.g. BVH)

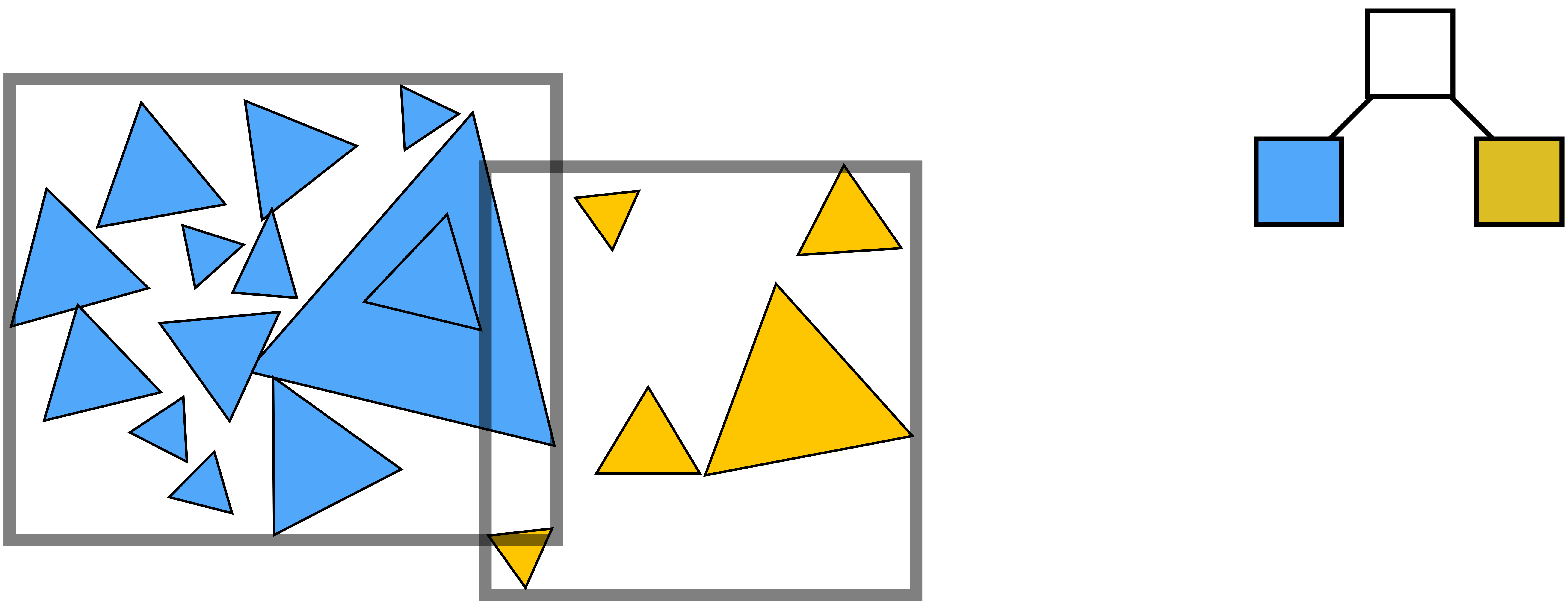
- Partition set of objects into disjoint subsets
- Bounding boxes for each set may overlap in space



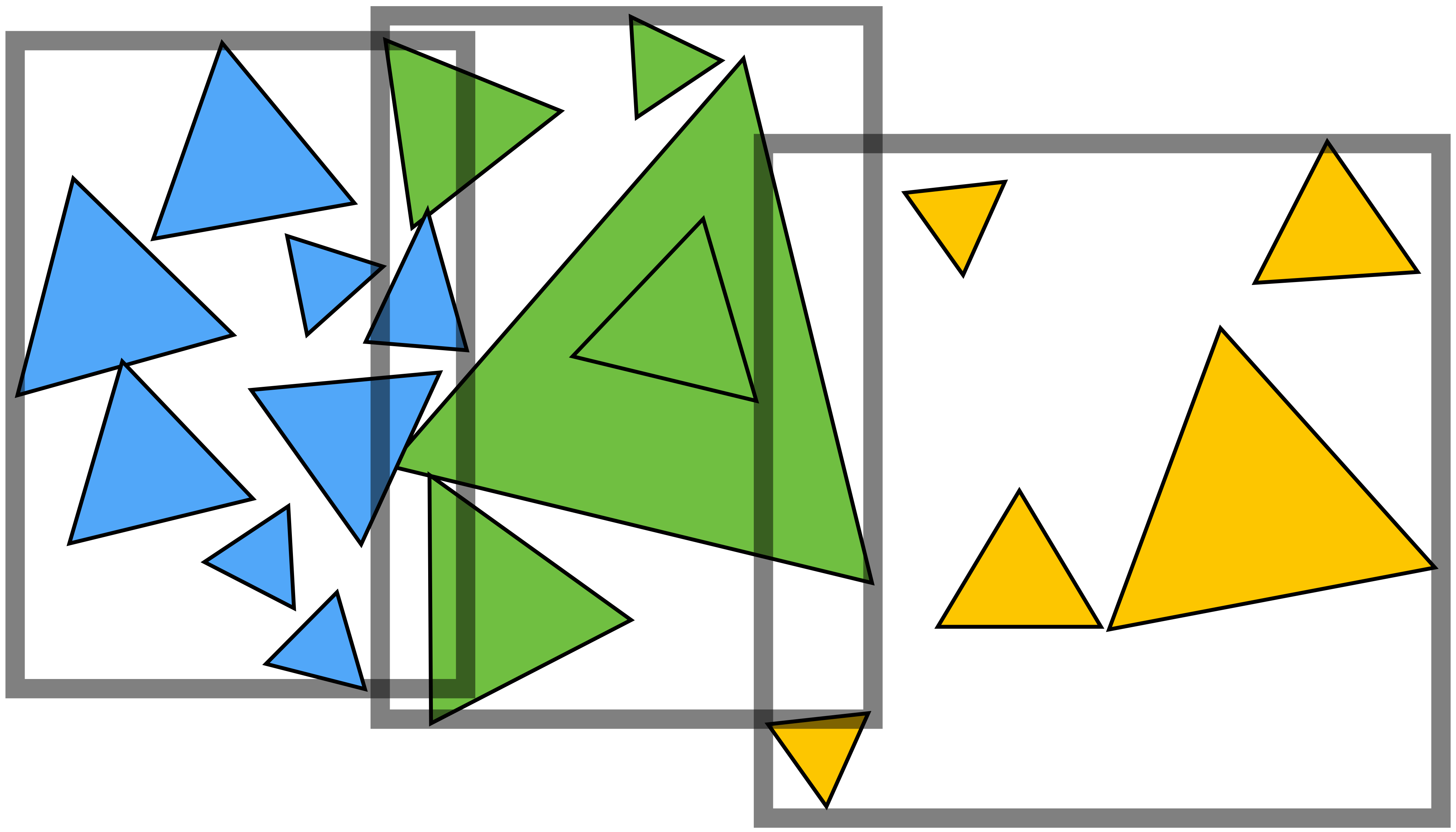
Bounding Volume Hierarchy (BVH)

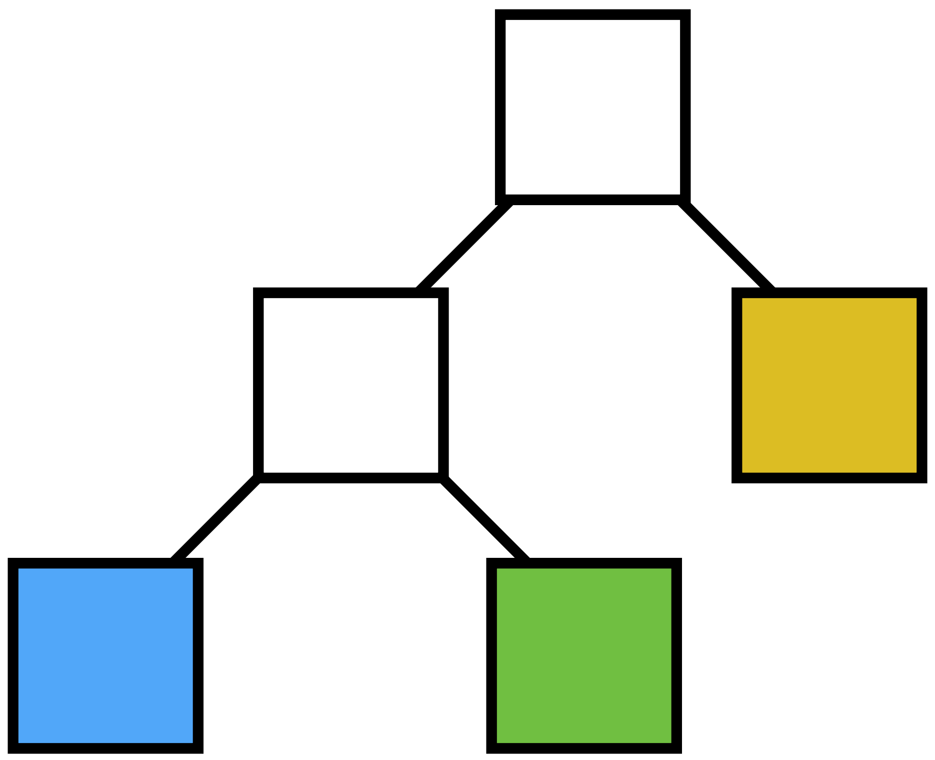


Bounding Volume Hierarchy (BVH)

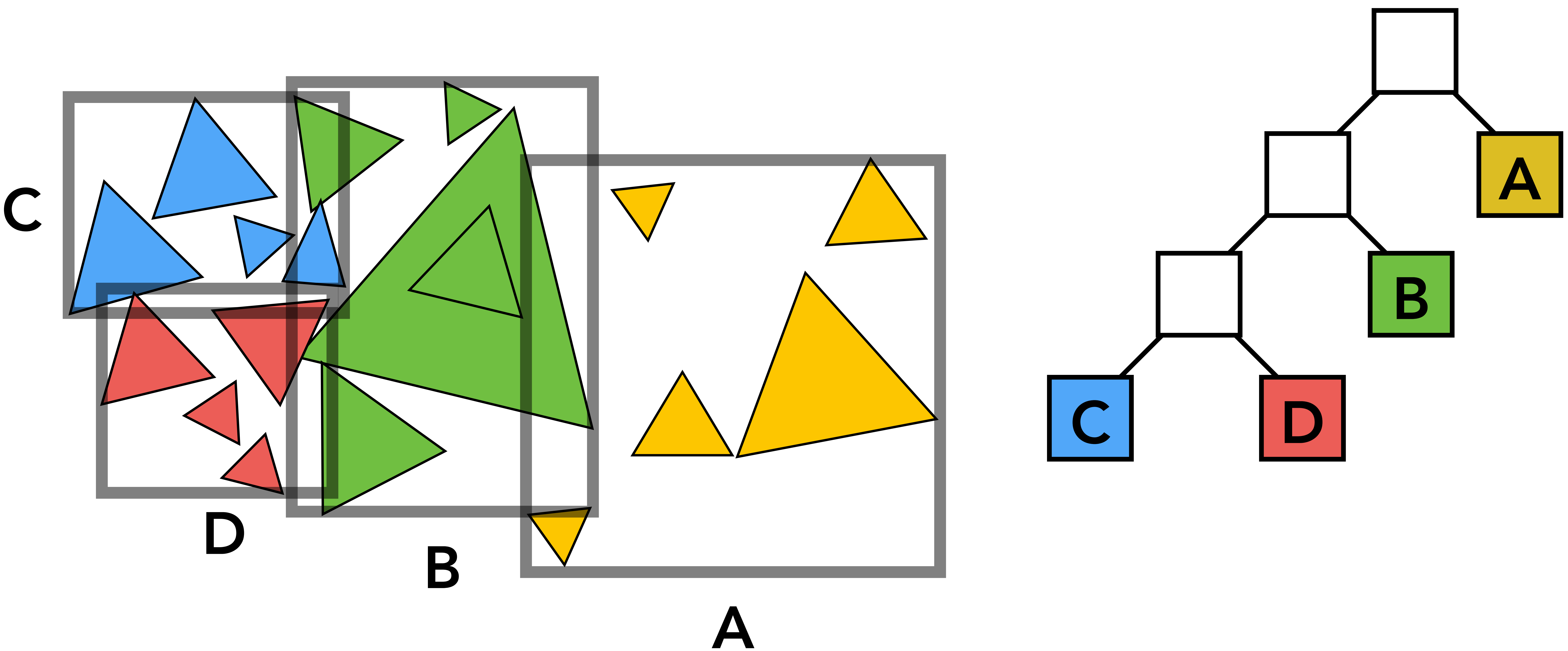


Bounding Volume Hierarchy (BVH)





Bounding Volume Hierarchy (BVH)



Bounding Volume Hierarchy (BVH)

Internal nodes store

- Bounding box
- Children: reference to child nodes

Leaf nodes store

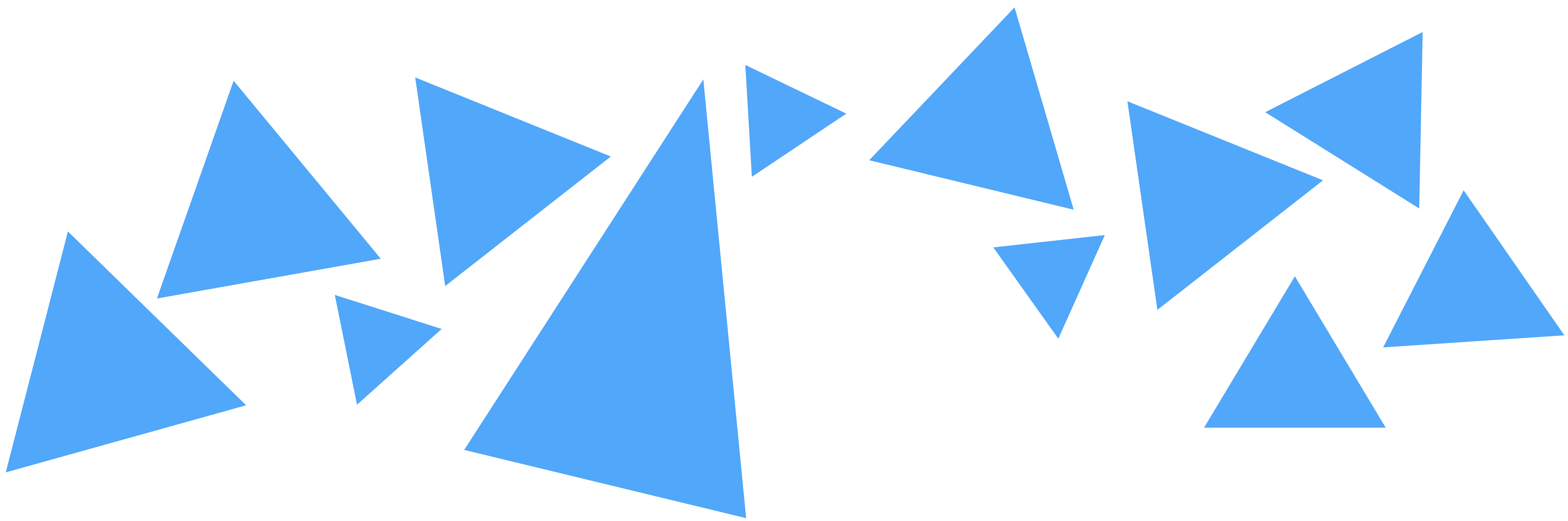
- Bounding box
- List of objects

Nodes represent subset of primitives in scene

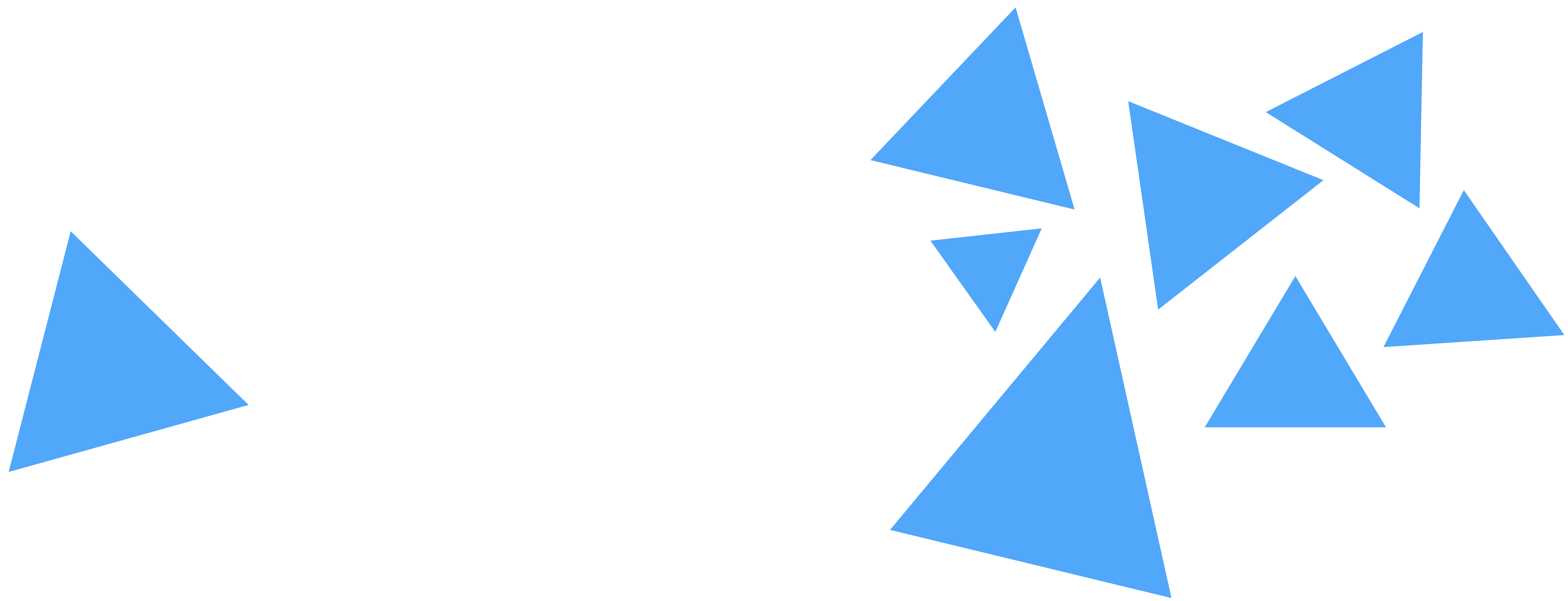
- All objects in subtree

Optimizing Hierarchical Partitions
(How to Split?)

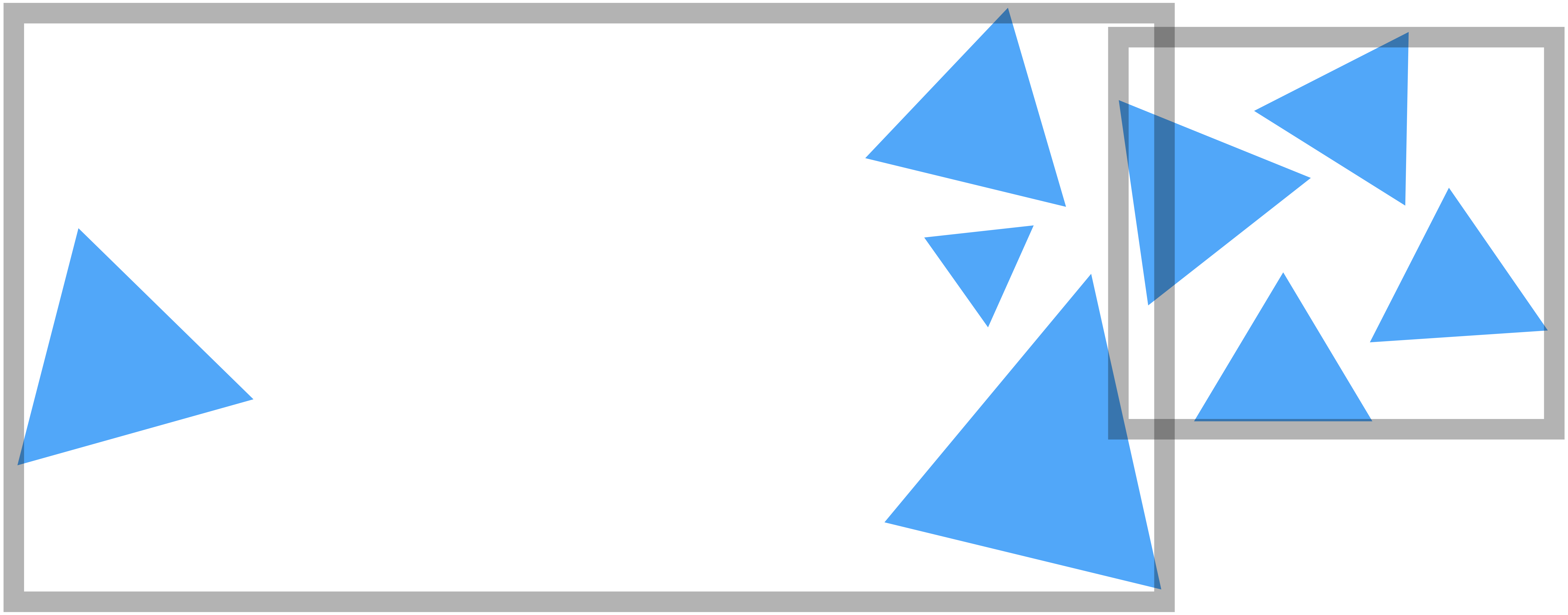
How to Split into Two Sets? (BVH)



How to Split into Two Sets? (BVH)



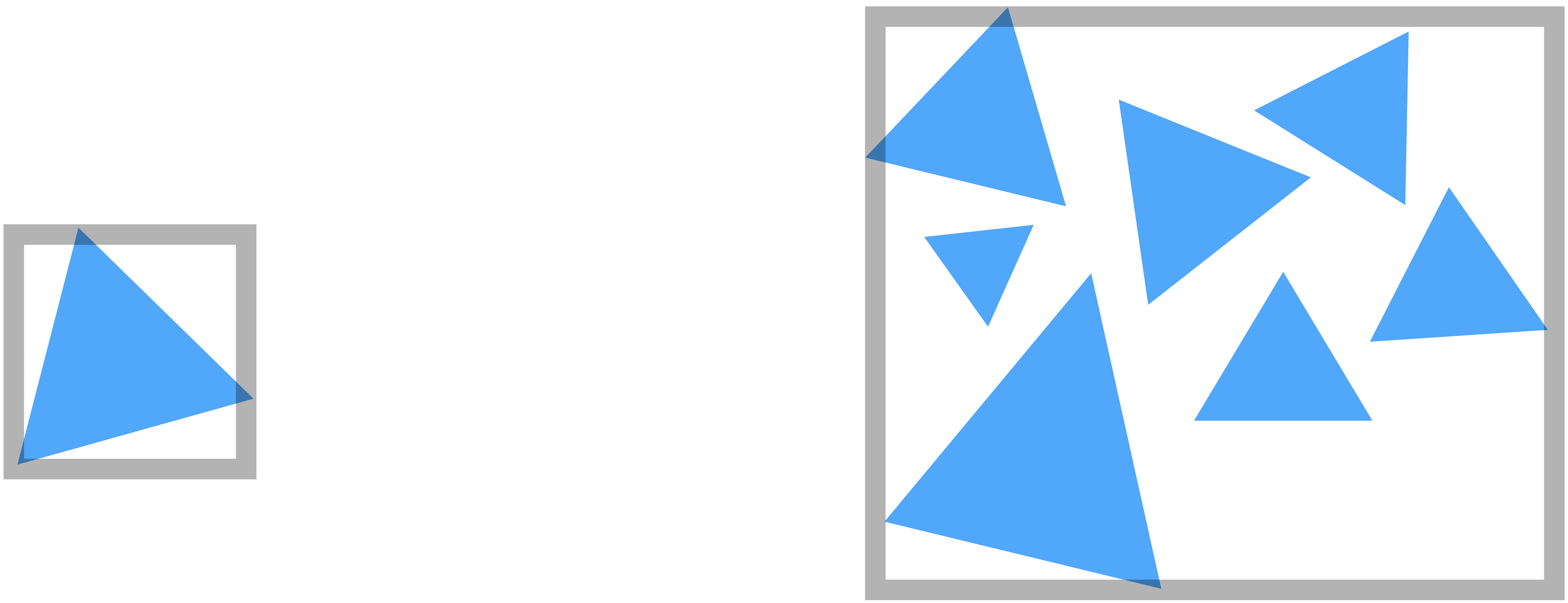
How to Split into Two Sets? (BVH)



Split at median element?

Child nodes have equal numbers of elements

How to Split into Two Sets? (BVH)



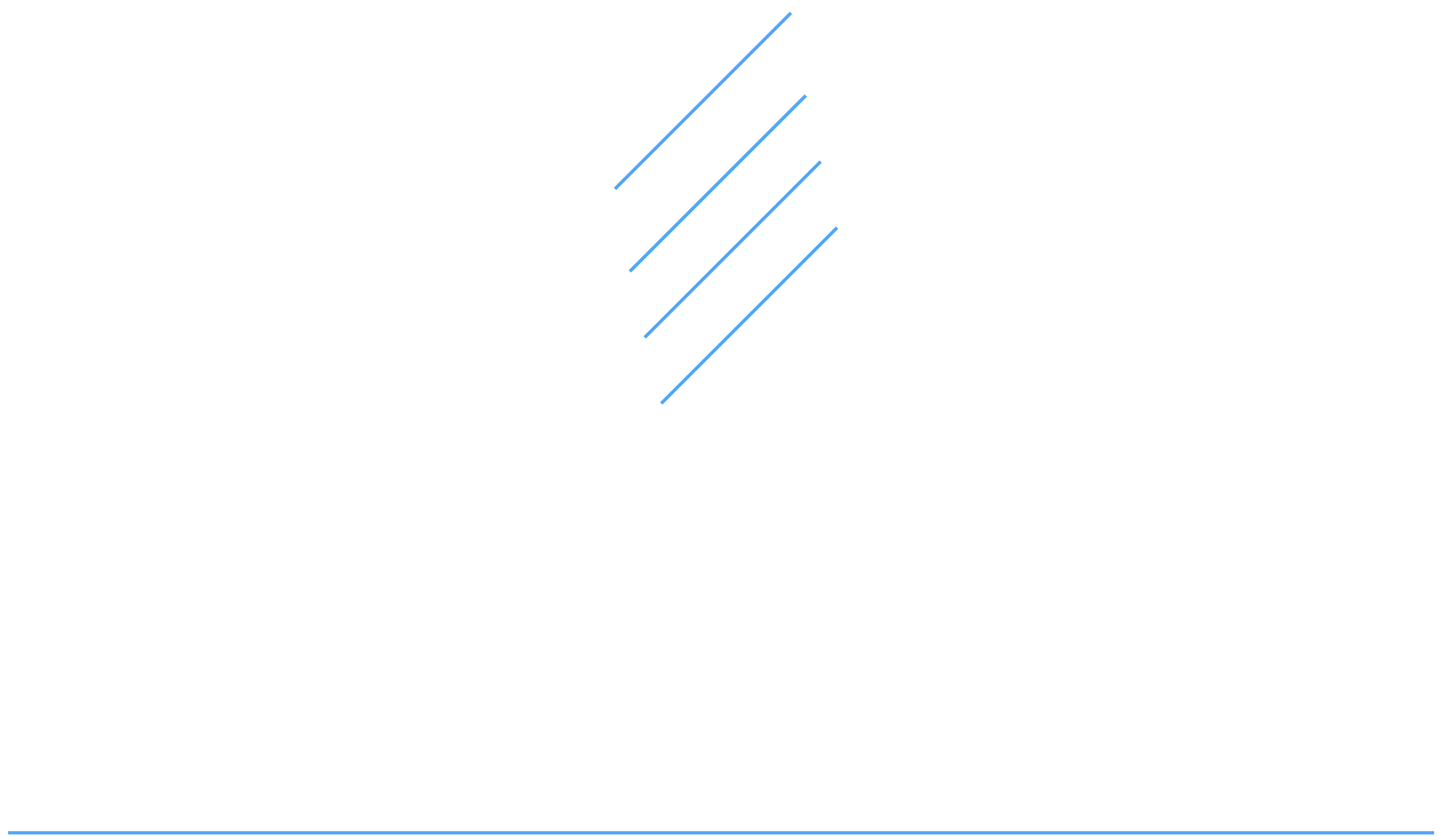
A better split?

Smaller bounding boxes, avoid overlap and empty space

BVH in your assignment

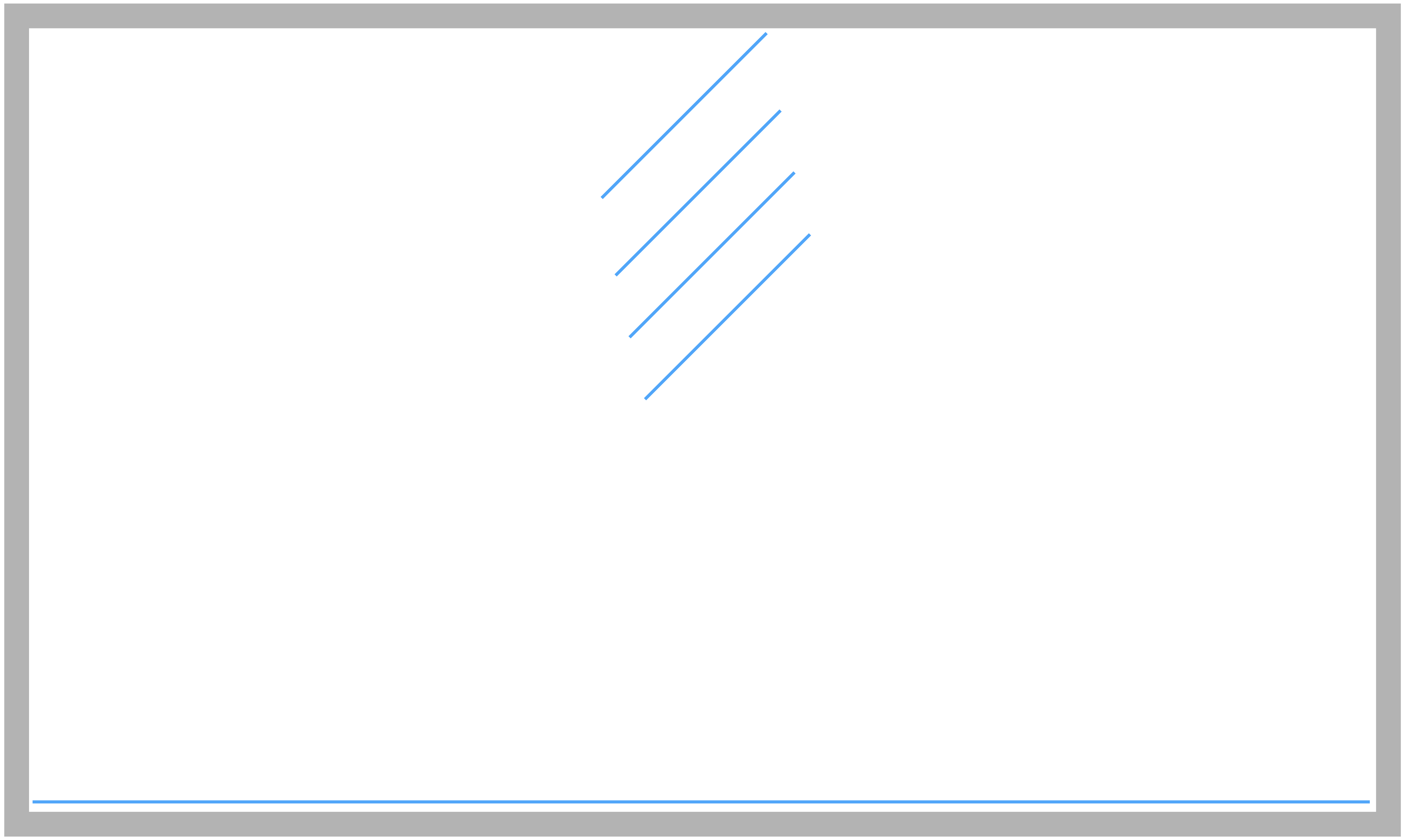
- Bounding box centroid split heuristic (directly computed from bounding boxes of primitives)
- Surface area split heuristic is more efficient but requires more complicated code to initialize (for each split, have to optimize over many choices)

Small choices have a big effect on BVH structure



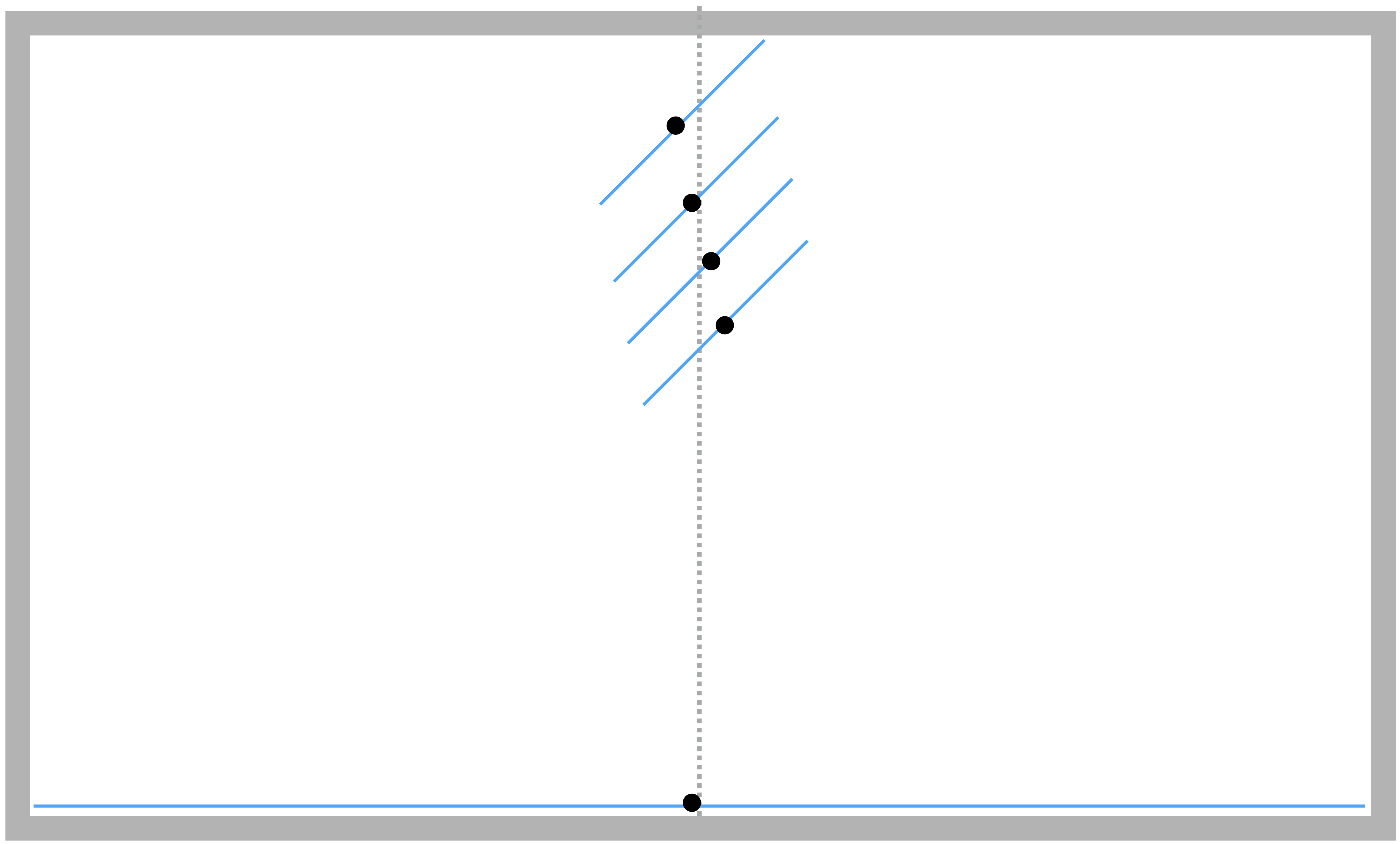
Reduce problem to 2D BVH for a set of line primitives

Object above a base plane



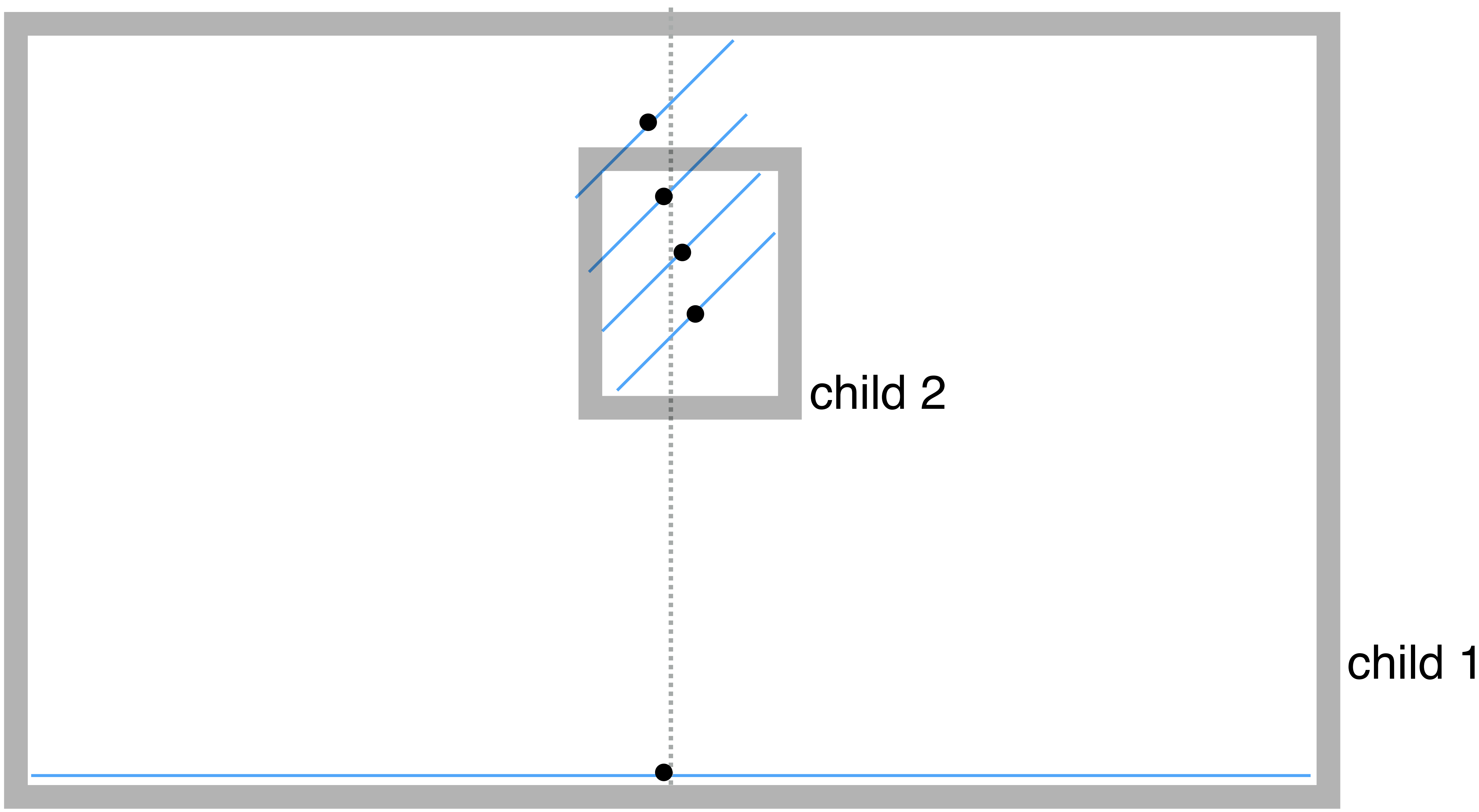
Bounding box for this scene is wider than it is tall

Object above a base plane



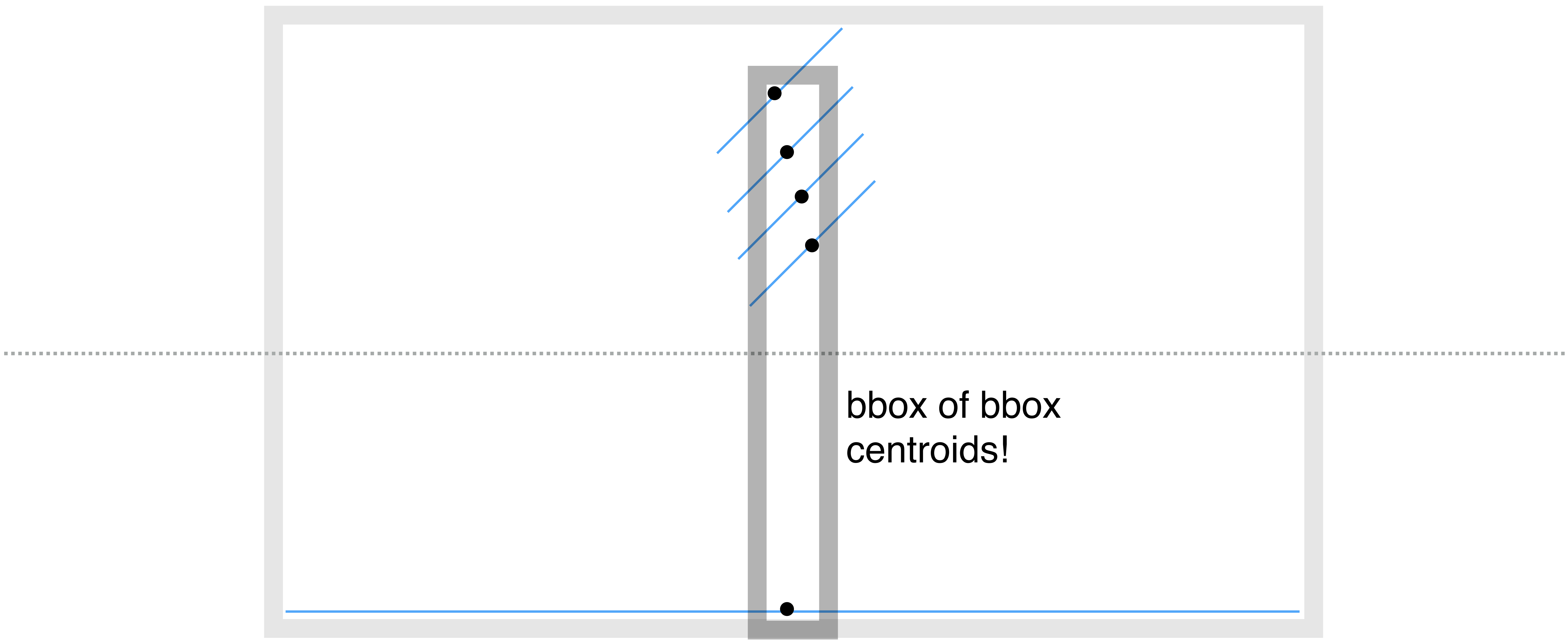
But splitting on that axis is a bad idea

Object above a base plane



One of the child nodes isn't even smaller!

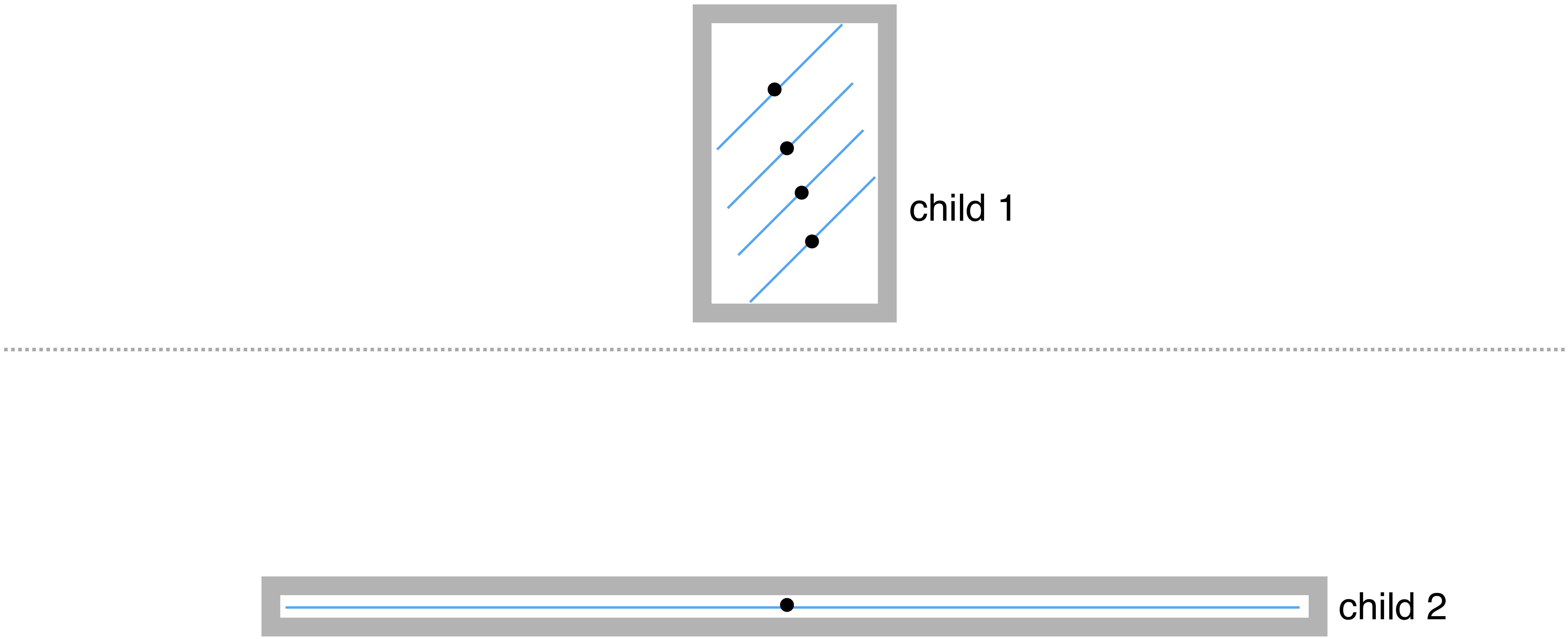
Object above a base plane



Instead, split based on the axis with most *centroid* variation

This box is much taller than it is wide

Object above a base plane



Resulting children are much more compact

Demo