Computer Graphics (CS 543)
Lecture 13b
Ray Tracing (Part 1)

Prof Emmanuel Agu

Computer Science Dept.

Worcester Polytechnic Institute (WPI)

Raytracing

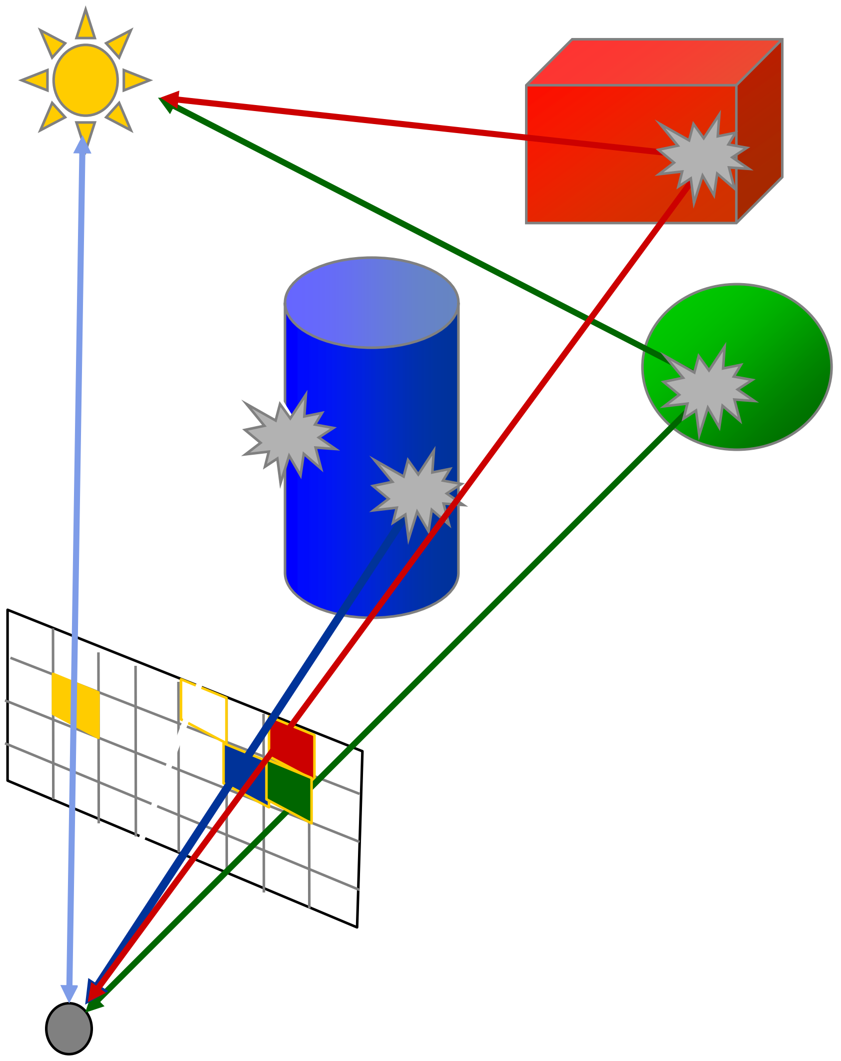
- Global illumination-based rendering method
- Simulates rays of light, natural lighting effects
- Because light path is traced, handles effects tough for openGL:
  - Shadows
  - Multiple inter-reflections
  - Transparency
  - Refraction
  - Texture mapping
- Newer variations… e.g. photon mapping (caustics, participating media, smoke)
- **Note:** raytracing can be semester graduate course
- Today: start with high-level description

Raytracing Uses

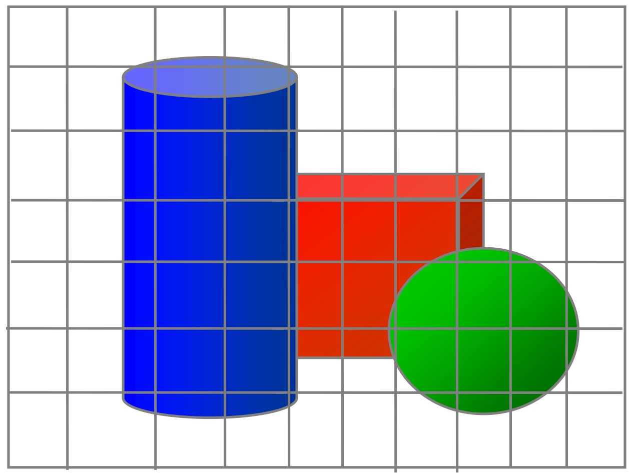
- Entertainment (movies, commercials)
- Games (pre-production)
- Simulation (e.g. military)
- Image: Internet Ray Tracing Contest Winner (April 2003)

[INCUBUS|FINAL]

Ray Casting (Appel, 1968)



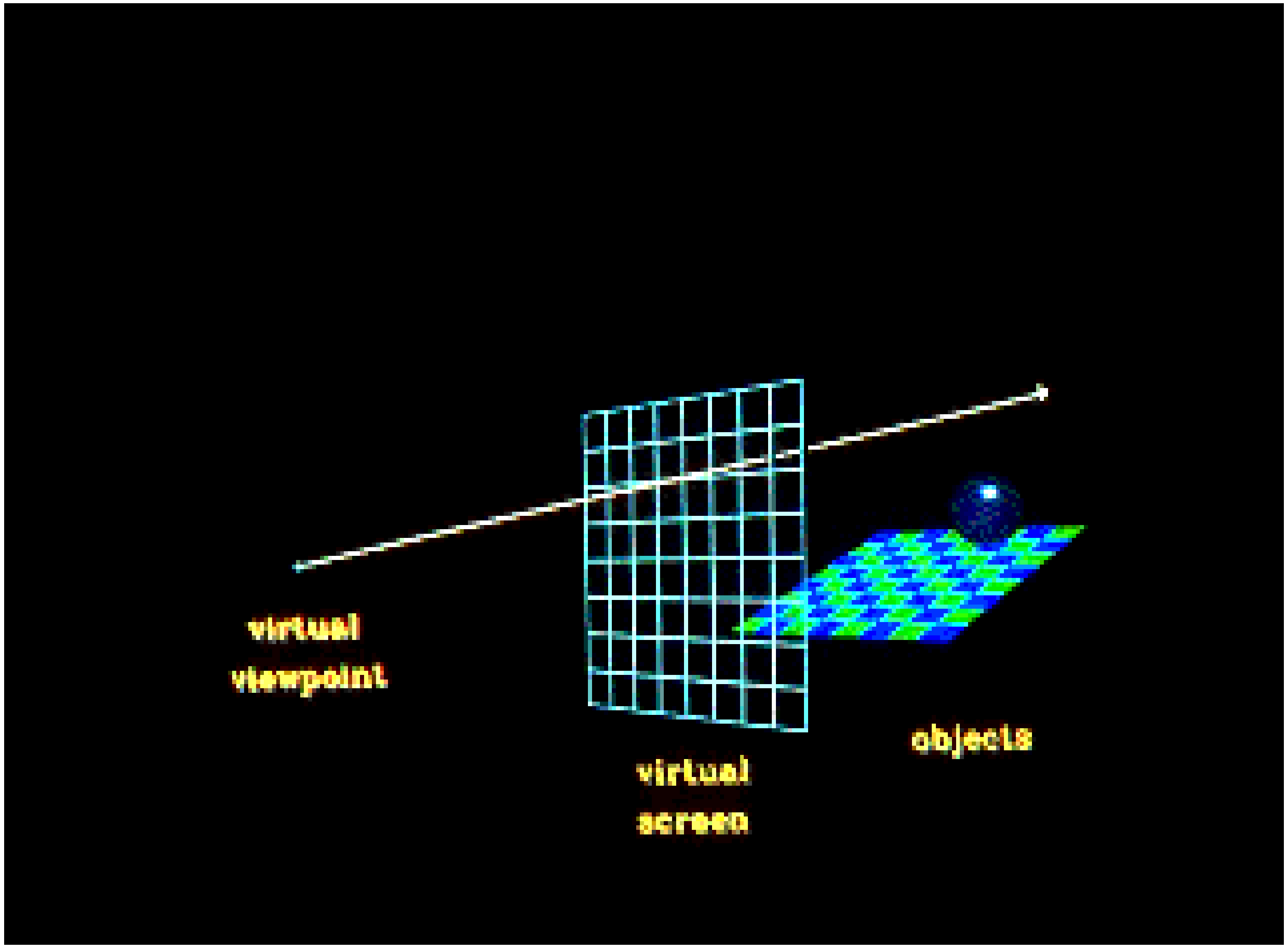
direct illumination
OpenGL does this too



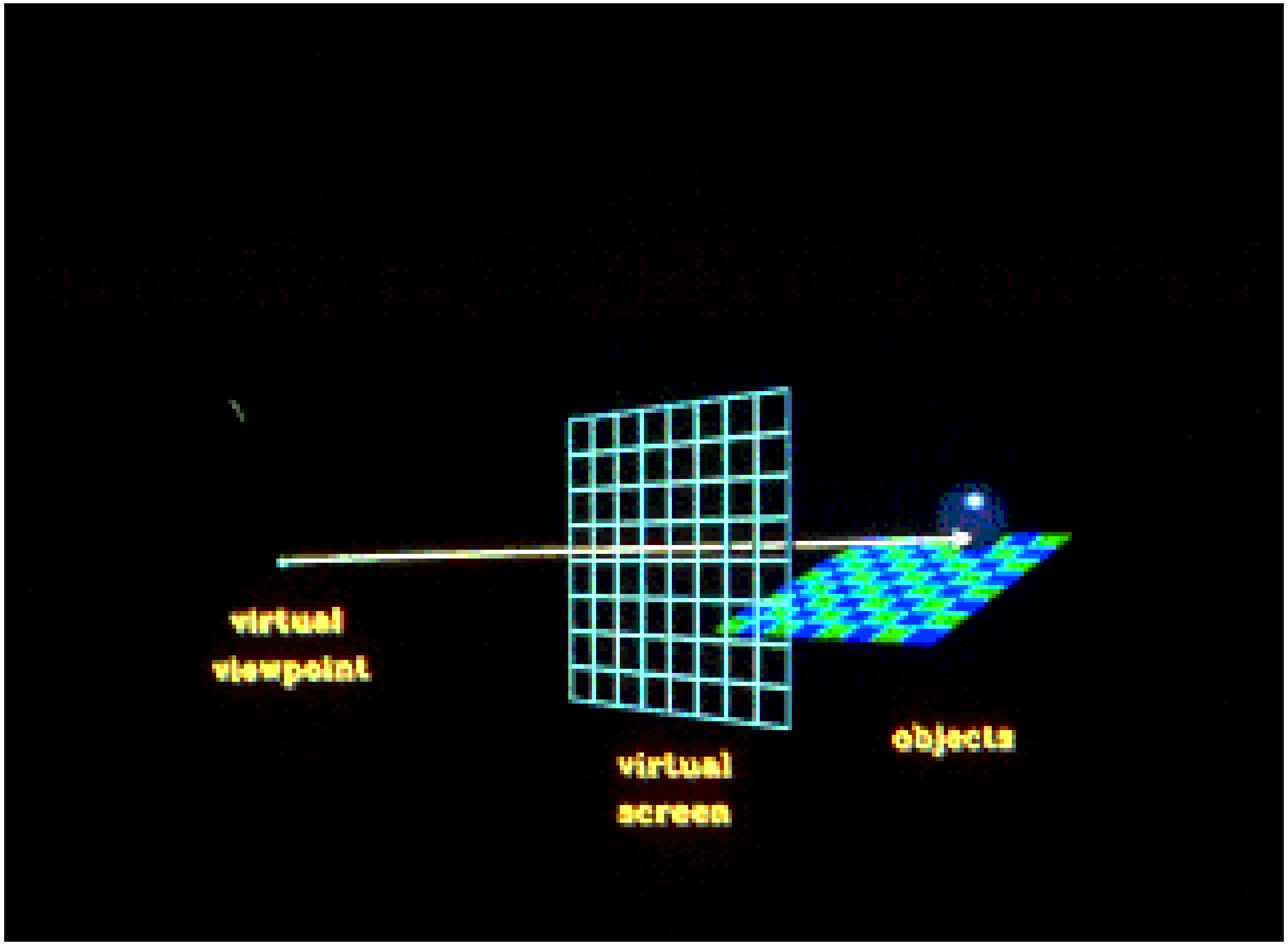
How Raytracing Works

- OpenGL is object space rendering
  - start from world objects, rasterize them
- Ray tracing is image space method
  - Start from pixel, what do you see through this pixel?
- Looks through each pixel (e.g. 640 x 480)
- Determines what eye sees through pixel
- Basic idea:
  - Trace light rays: eye -> pixel (image plane) -> scene
  - If a ray intersect any scene object in this direction
    - Yes? render pixel using object color
    - No? it renders the pixel using the background color
- Automatically solves hidden surface removal problem

Case A: Ray misses all objects

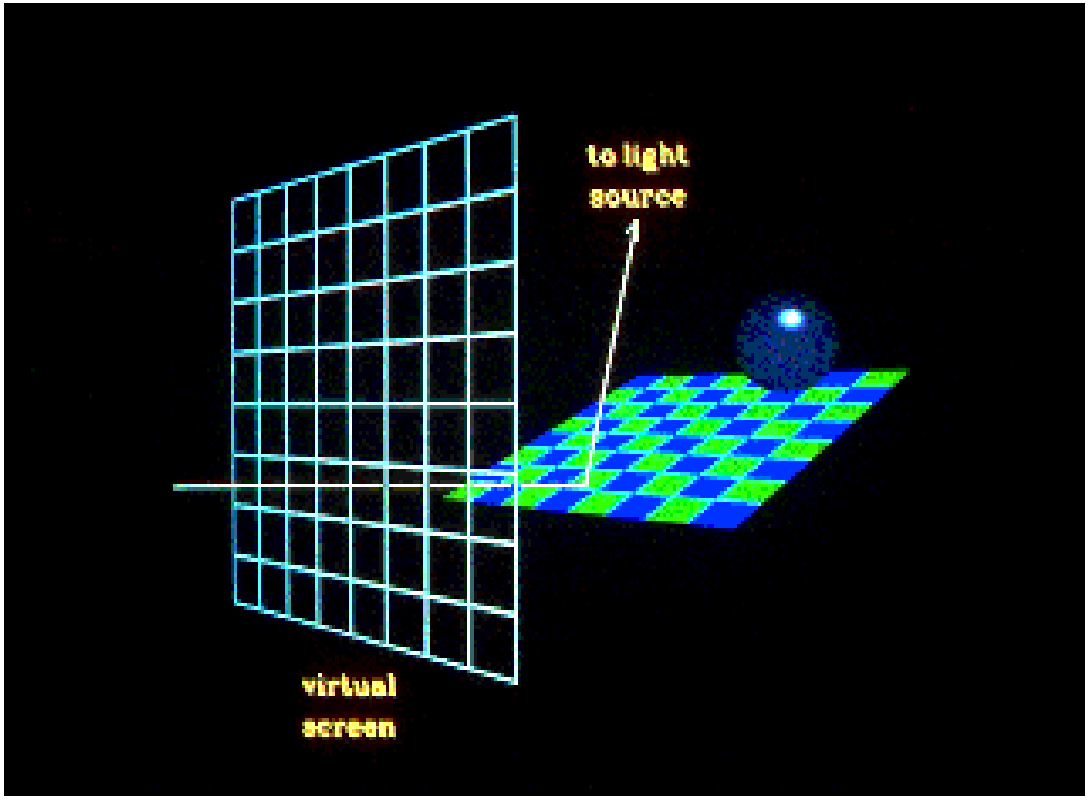


Case B: Ray hits an object



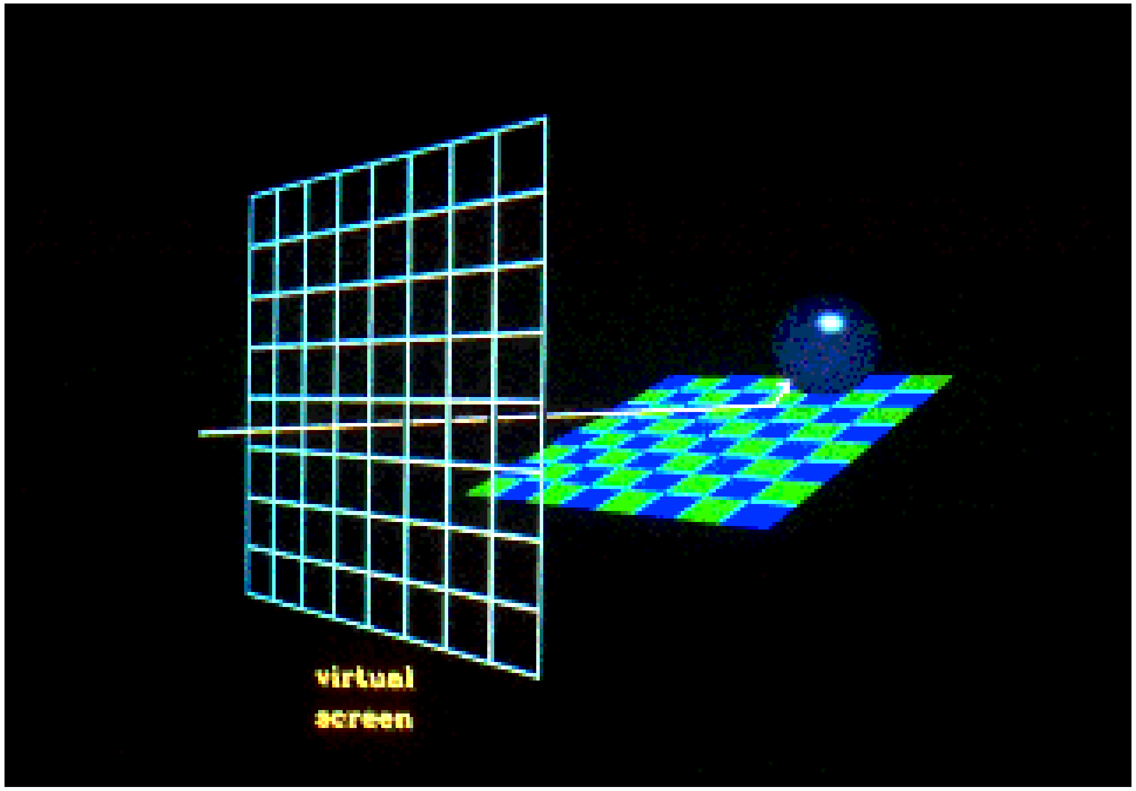
Case B: Ray hits an object

- **Ray hits object:** Check if hit point is in shadow, build secondary ray (shadow ray) towards light sources.



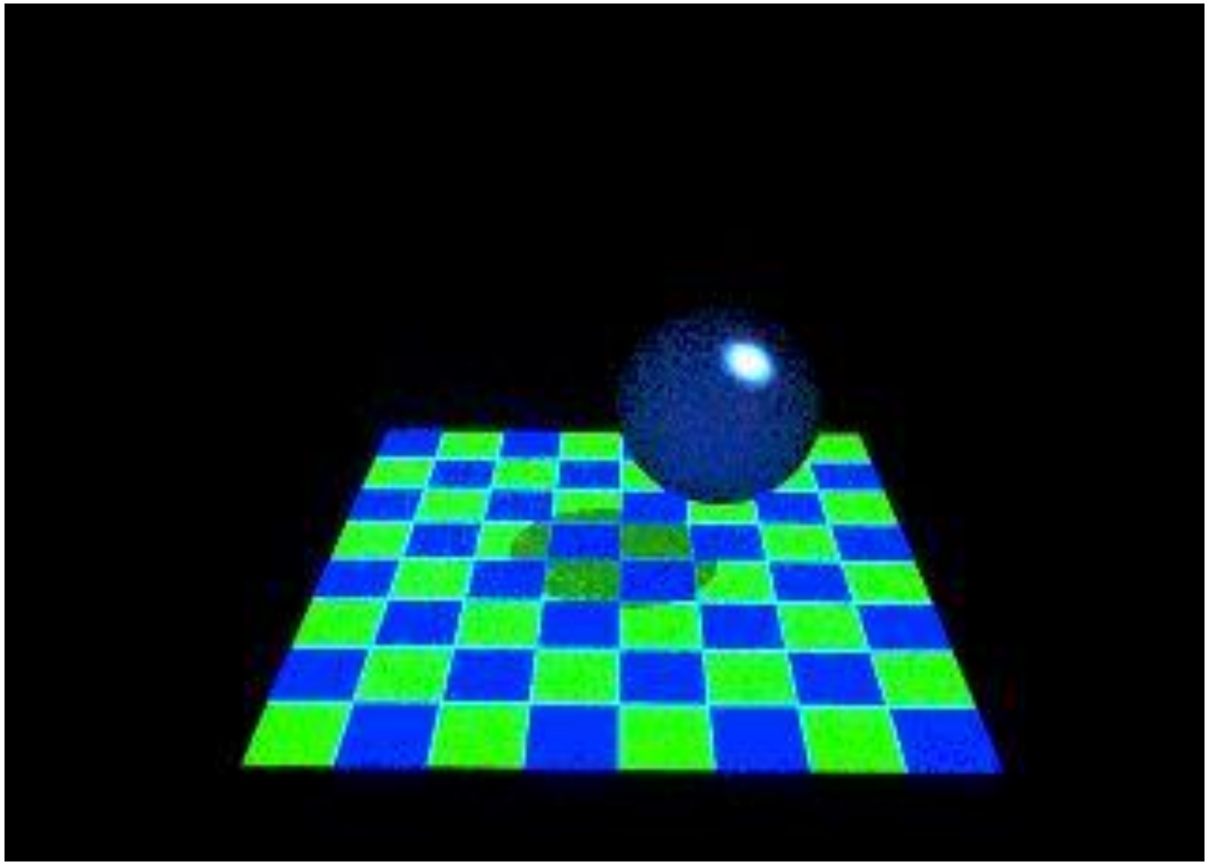
Case B: Ray hits an object

- If shadow ray hits another object before light source: first intersection point is in shadow of the second object. Otherwise, collect light contributions



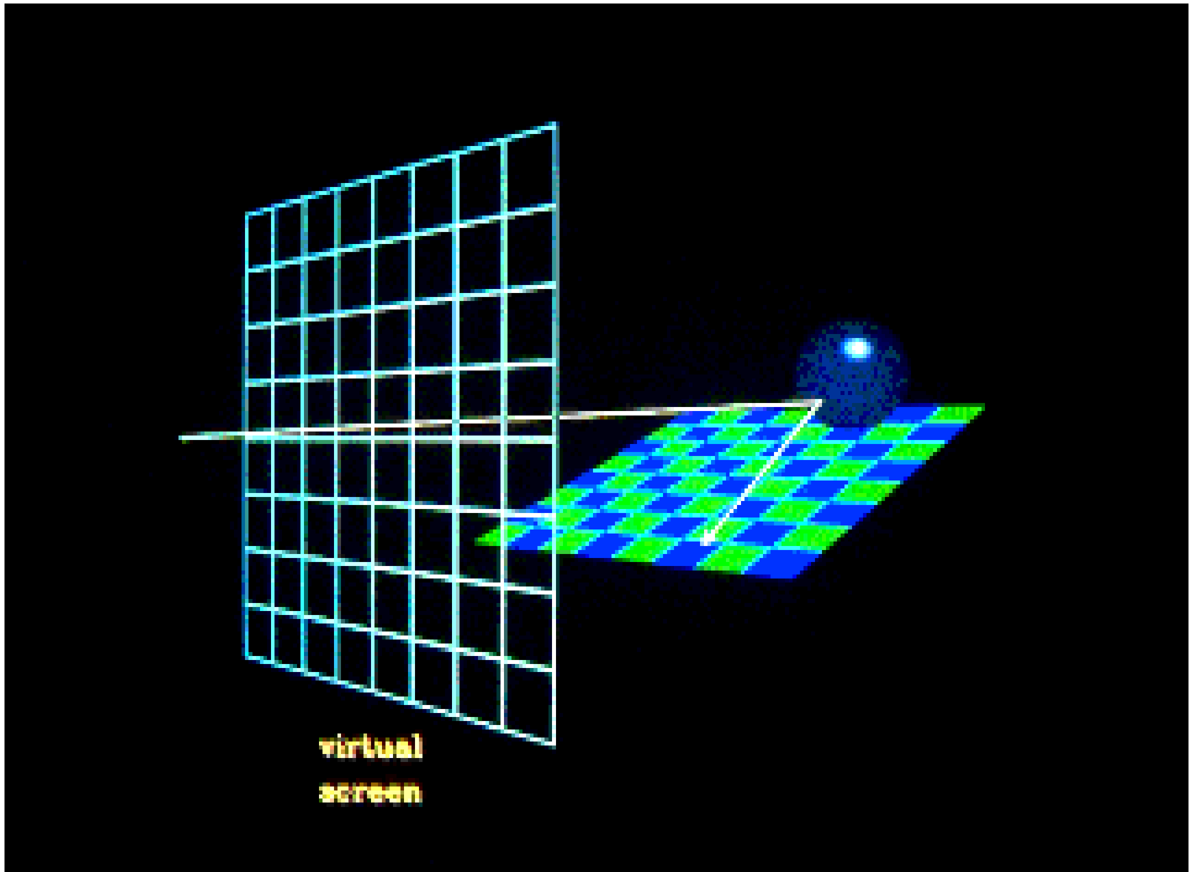
Case B: Ray hits an object

- First Intersection point in the shadow of the second object is the shadow area.

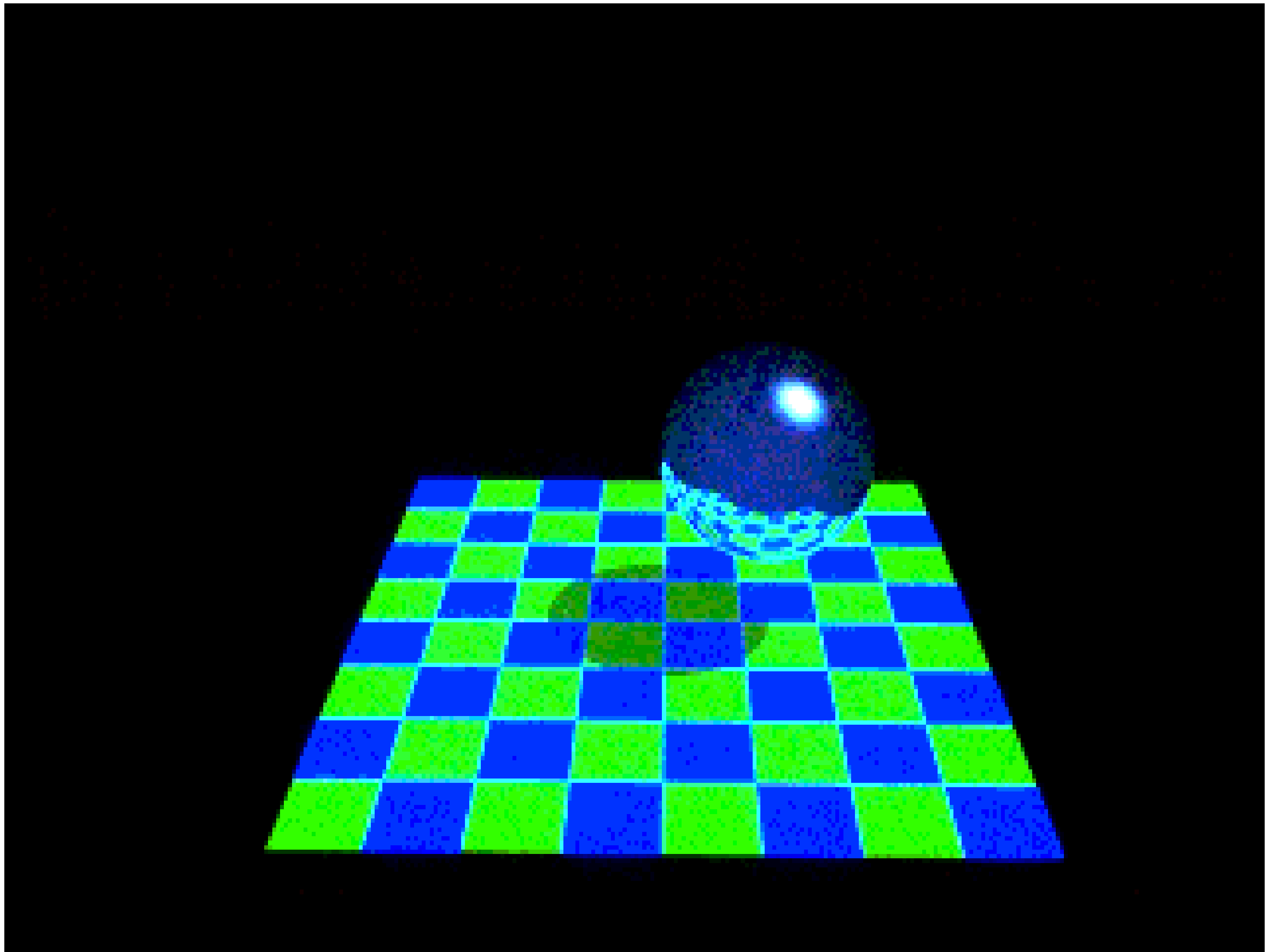


Reflected Ray

- When a ray hits an object, a reflected ray is generated which is tested against all of the objects in the scene.

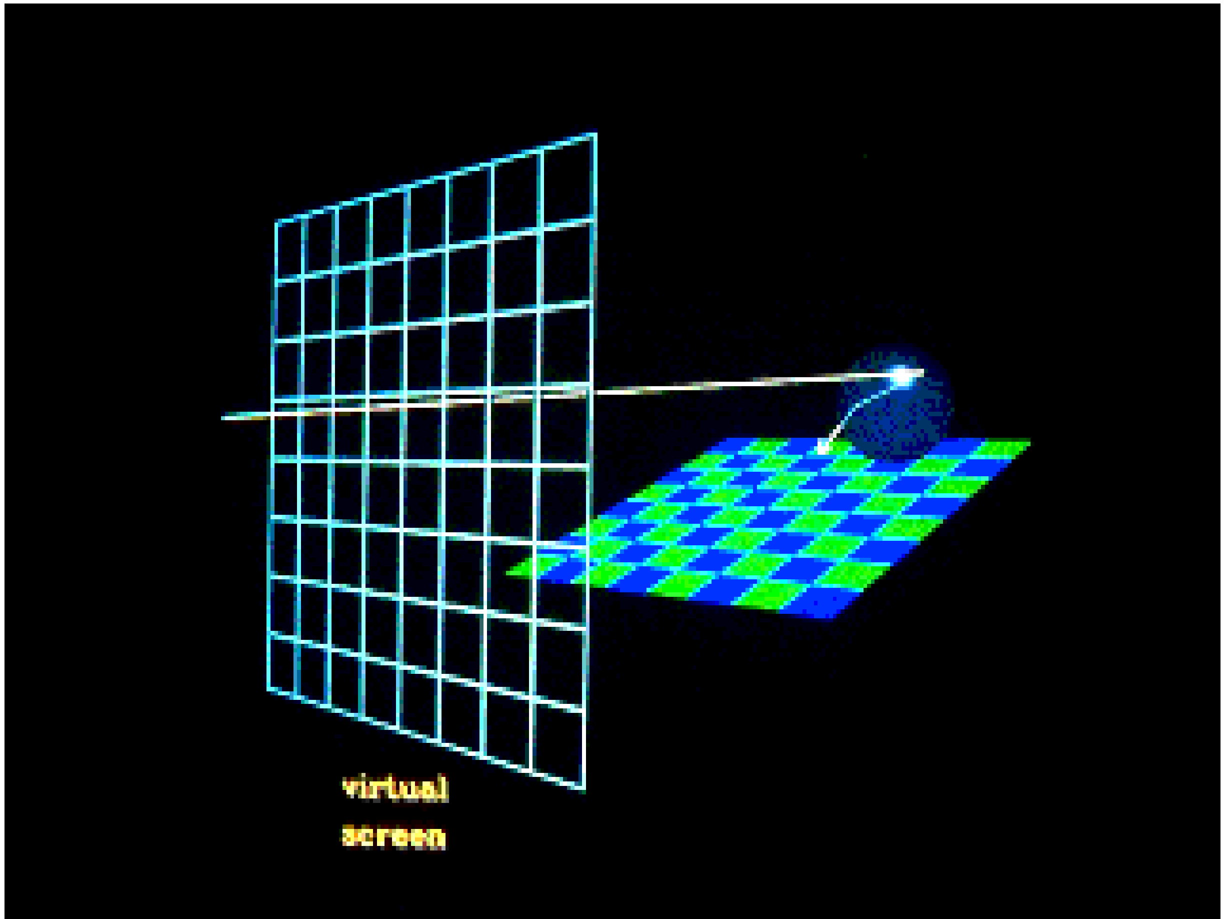


Reflection: Contribution from the reflected ray

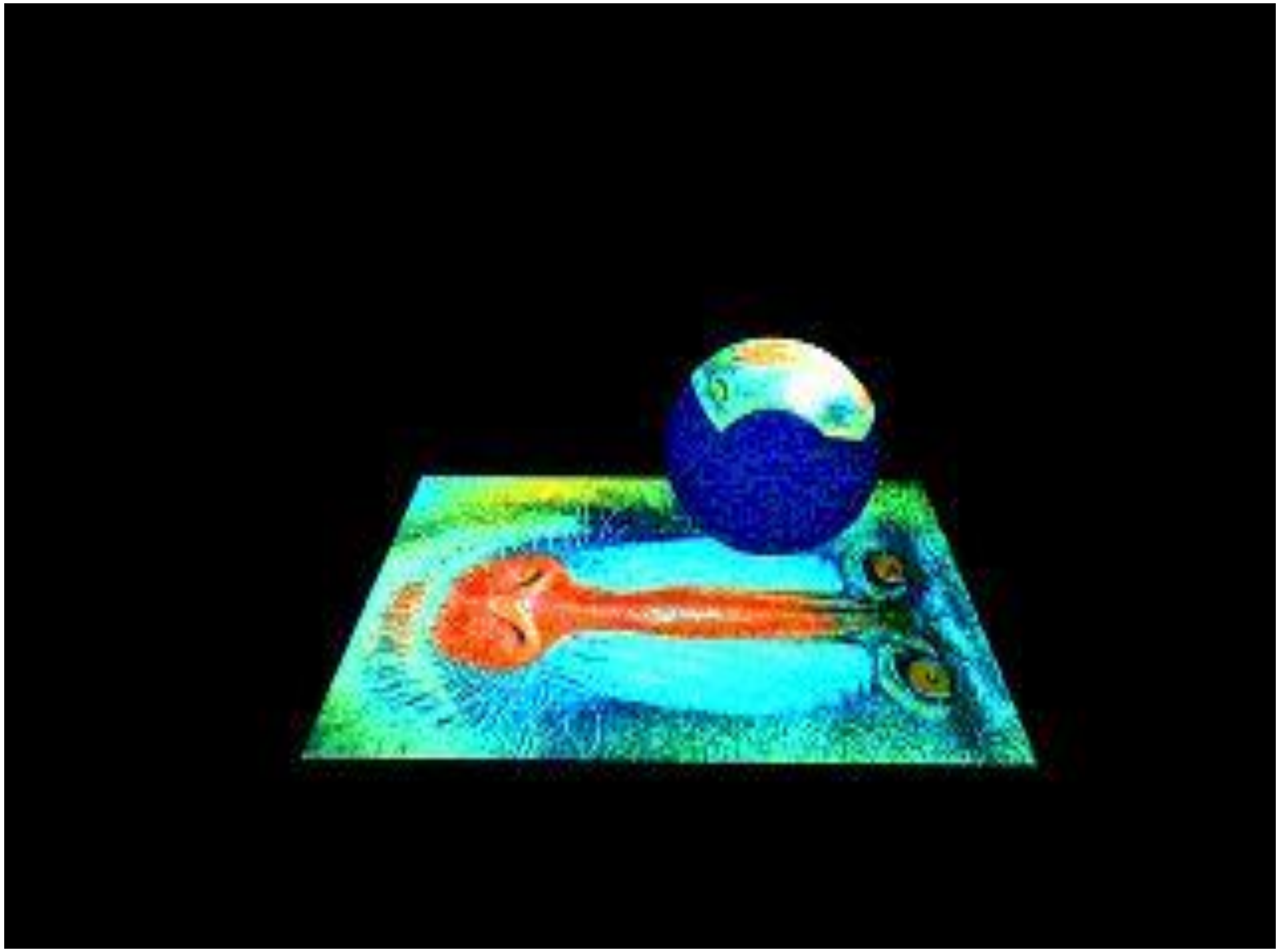


Transparency

- If intersected object is transparent, transmitted ray is generated and tested against all the objects in the scene.

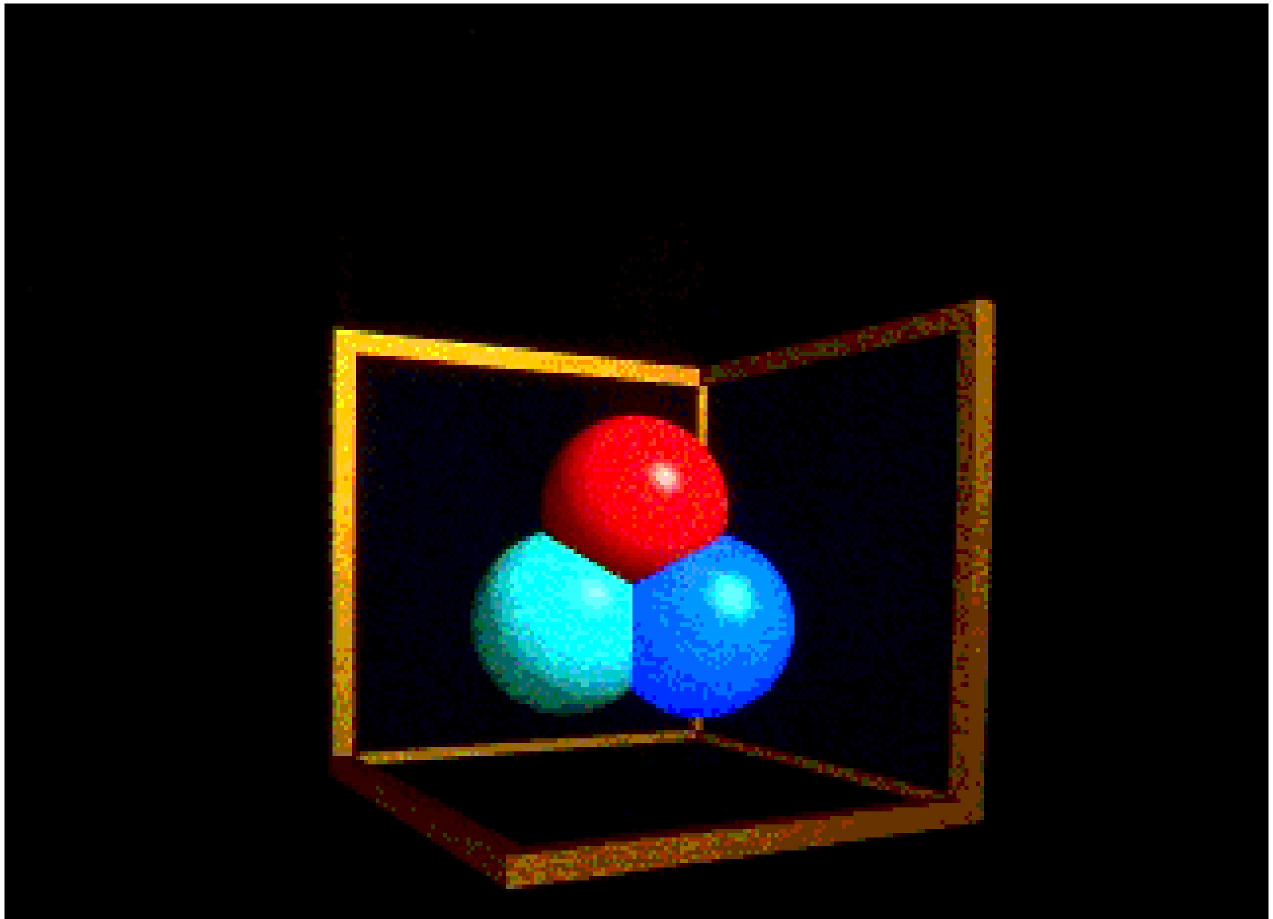


Transparency: Contribution from transmitted ray



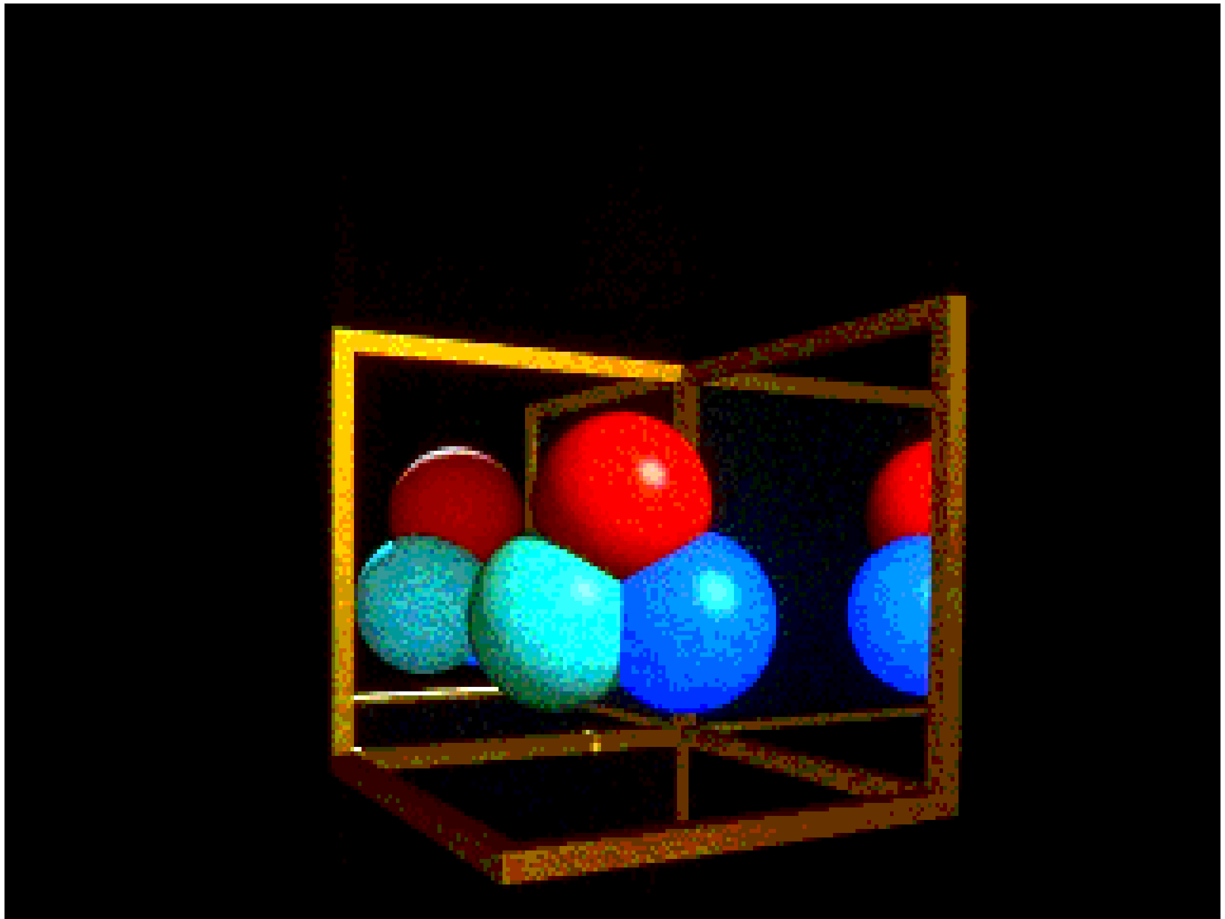
Reflected Ray: Recursion

Reflected rays can generate other reflected rays that can generate other reflected rays, etc. Case A: *Scene with no reflection rays*



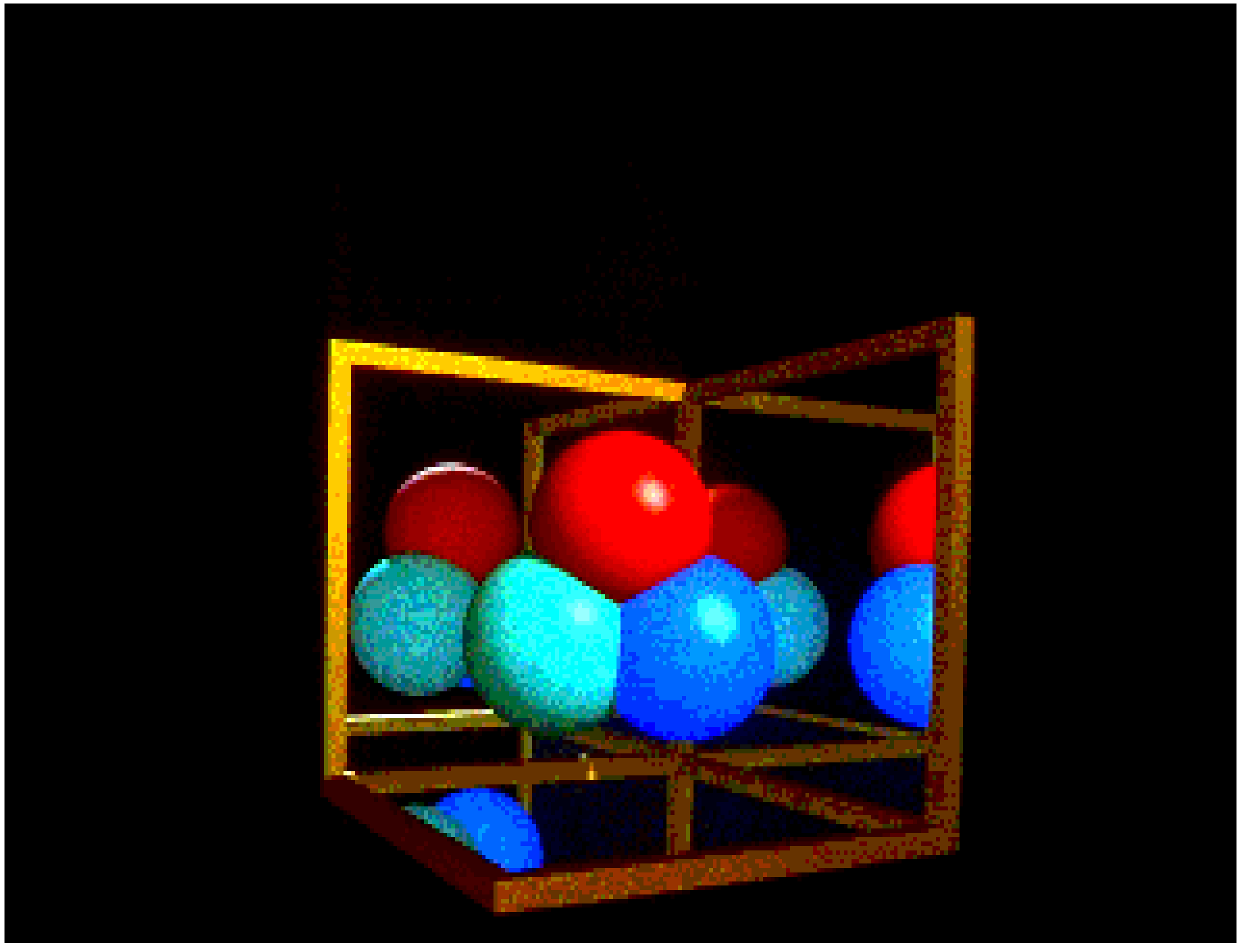
Reflected Ray: Recursion

Case B: Scene with one layer of reflection



Reflected Ray: Recursion

Case C: Scene with two layers of reflection



Ray Tree

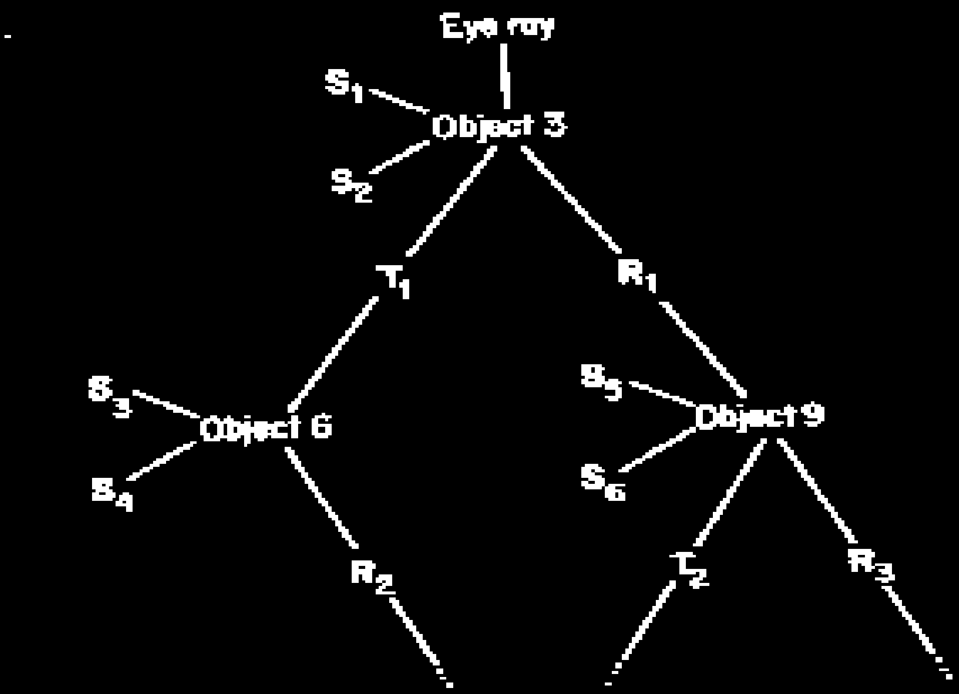


Fig. 12. The ray tree in schematic form.

- Reflective and/or transmitted rays are continually generated until ray leaves the scene without hitting any object or a preset recursion level has been reached.

Ray-Object Intersections

- So, express ray as equation (origin is eye, pixel determines direction)
- Define a ray as:

  `R0 = [x0, y0, z0]` - origin of ray

  `Rd = [xd, yd, zd]` - direction of ray
- then define parametric equation of ray:

  `R(t) = R0 + Rd * t` with $t > 0.0$
- Express all objects (sphere, cube, etc) mathematically
- Ray tracing idea:
  - put ray mathematical equation into object equation
  - determine if real solution exists.
  - Object with smallest hit time is object seen

Ray-Object Intersections

- Dependent on parametric equations of object
    - Ray-Sphere Intersections
    - Ray-Plane Intersections
    - Ray-Polygon Intersections
    - Ray-Box Intersections
    - Ray-Quadric Intersections

    (cylinders, cones, ellipsoids, paraboloids)

Accelerating Ray Tracing

- Ray Tracing is time-consuming because of intersection calculations
- Each intersection requires from a few (5-7) to many (15-20) floating point (fp) operations
- Example: for a scene with 100 objects and computed with a spatial resolution of 512 x 512, assuming 10 fp operations per object test there are about 250,000 X 100 X10 = 250,000,000 fps.
- Solutions:
  - Use faster machines
  - Use specialized hardware, especially parallel processors or graphics card
  - Speed up computations by using more efficient algorithms
  - Reduce the number of ray - object computations

Reducing Ray-Object Intersections

- Adaptive Depth Control: Stop generating reflected/transmitted rays when computed intensity becomes less than certain threshold.
- Bounding Volumes:
  - Enclose groups of objects in sets of hierarchical bounding volumes
  - First test for intersection with the bounding volume
  - Then only if there is an intersection, against the objects enclosed by the volume.
- First Hit Speed-Up: use modified Z-buffer algorithm to determine the first hit.

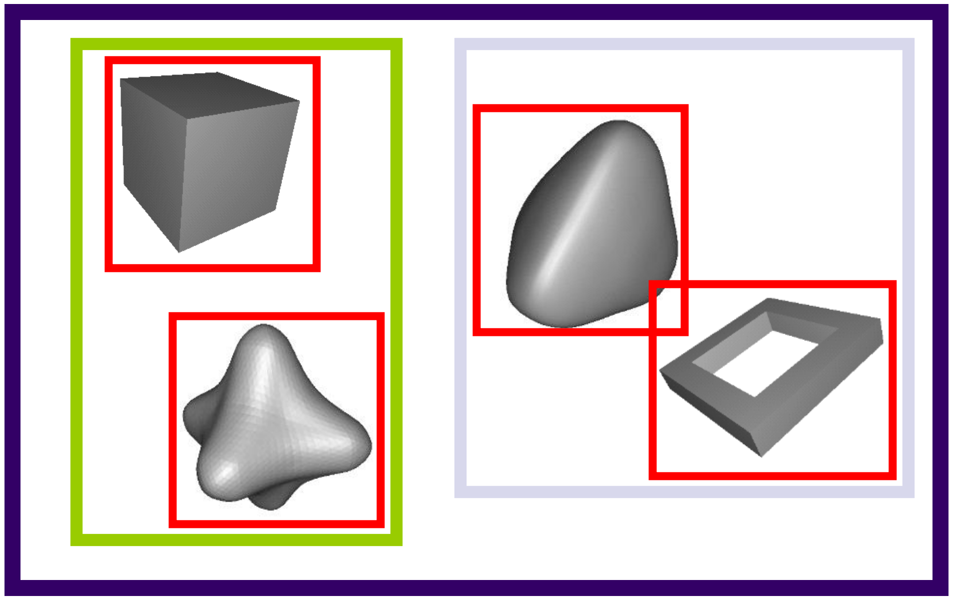
Popular Spatial Acceleration Structures

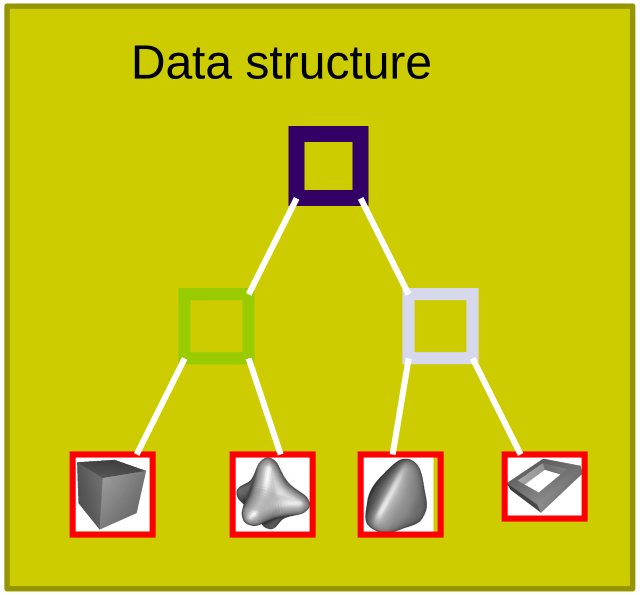
- **Spatial Data Structures:** manage scene geometry
  - Bounding Volume Hierarchies
  - BSP Trees
  - Octrees
  - Scene Graphs

How?

- Organizes geometry in some hierarchy

In 2D space

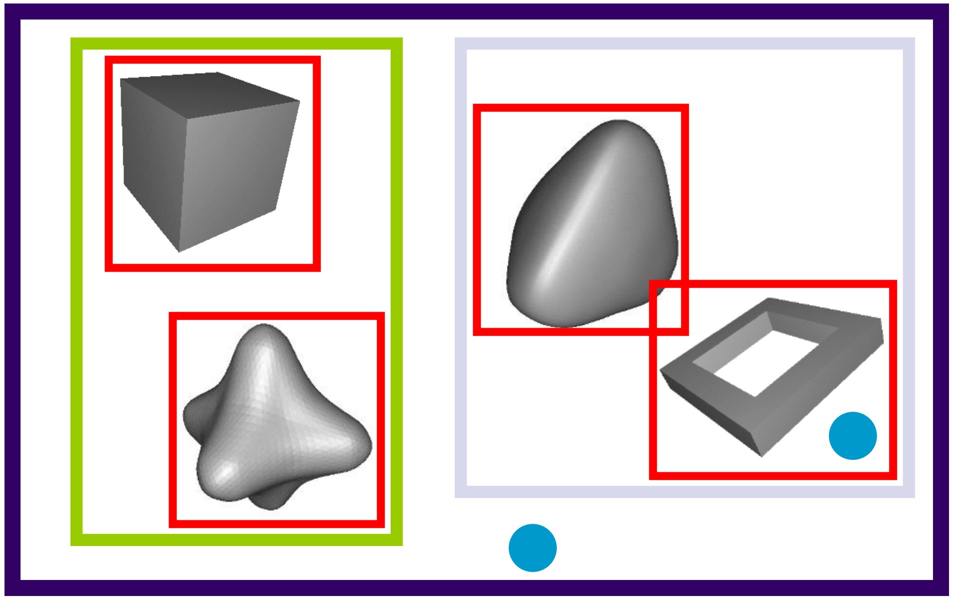


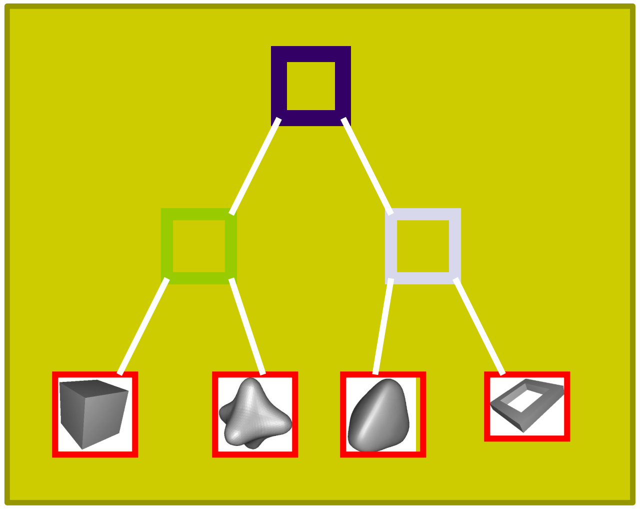


Bounding Volume Hierachy

What's the point? An example

- Assume we click on screen, and want to find which object we clicked on





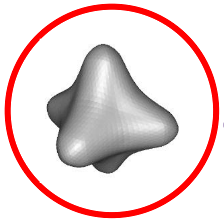
click!

1) Test the root first
2) Descend recursively as needed
3) Terminate traversal as soon as possible

In general: get O(log n) instead of O(n)

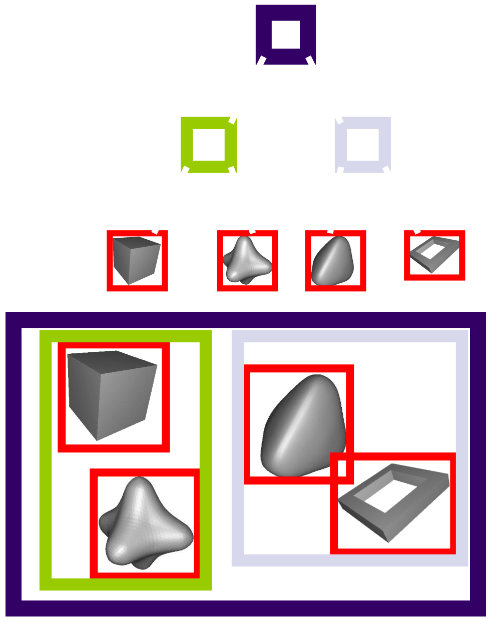
Bounding Volume Hierarchy (BVH)

- Use simple shapes to enclose complex geometry
- Most common bounding volumes (BVs):
  - Spheres, boxes (AABB and OBB)



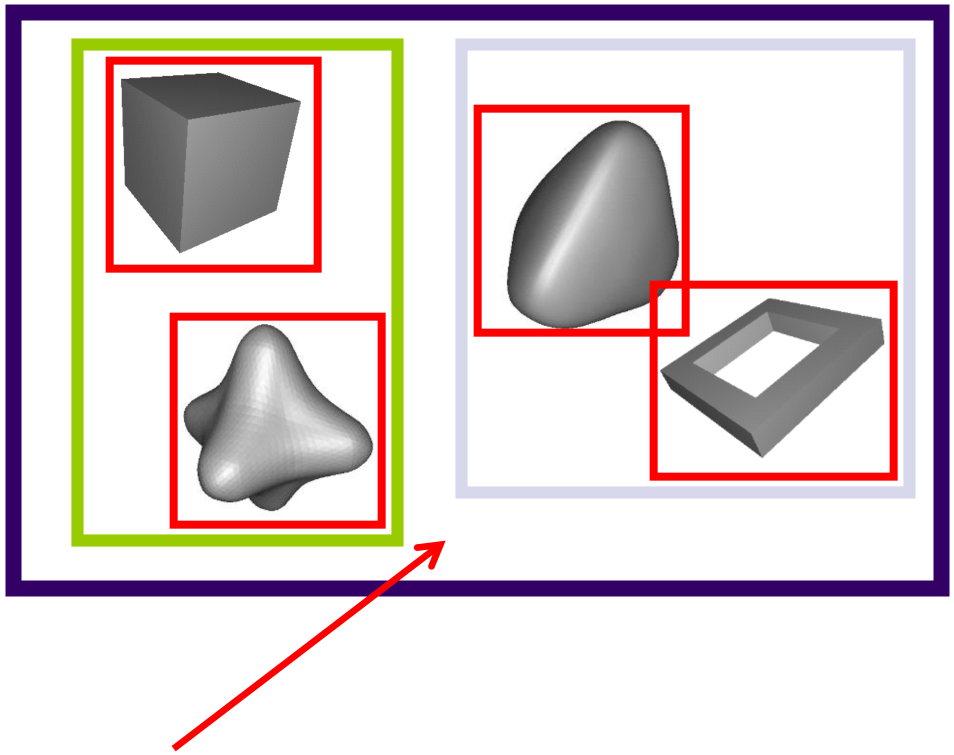
- The BV does not contibute to the rendered image -- rather, encloses an object

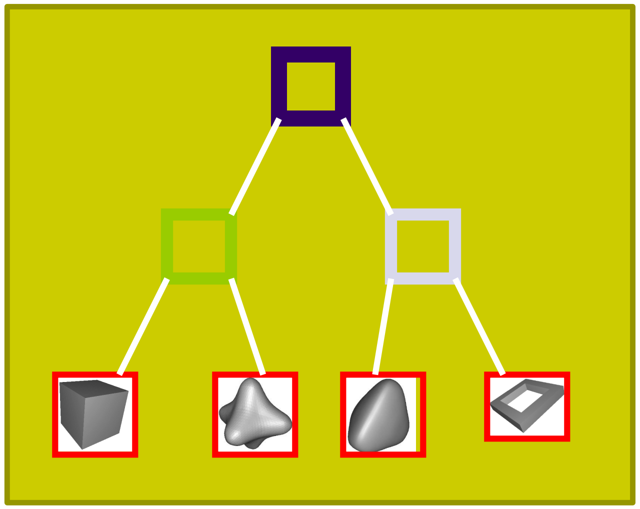
- The data structure is a *k*-ary tree
  - Leaves hold geometry
  - Internal nodes have at most *k* children
  - Internal nodes hold BVs that enclose all geometry in its subtree



Example Application of BVH: Intersection Testing in RT

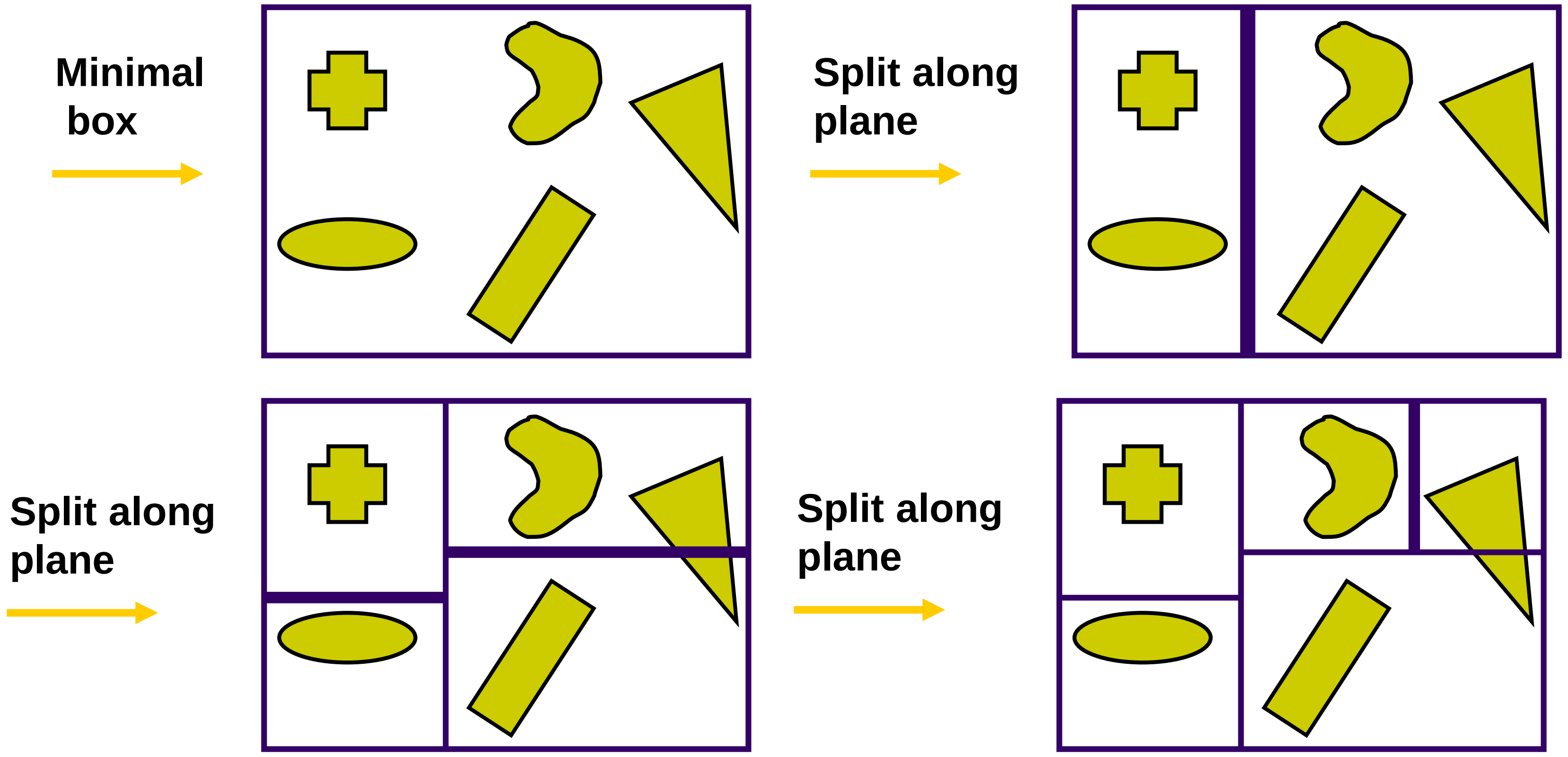
- Enclose scene geometry in BVH
- Cube/box much easier to test for intersections
- Large time savings if ray misses portions of scene



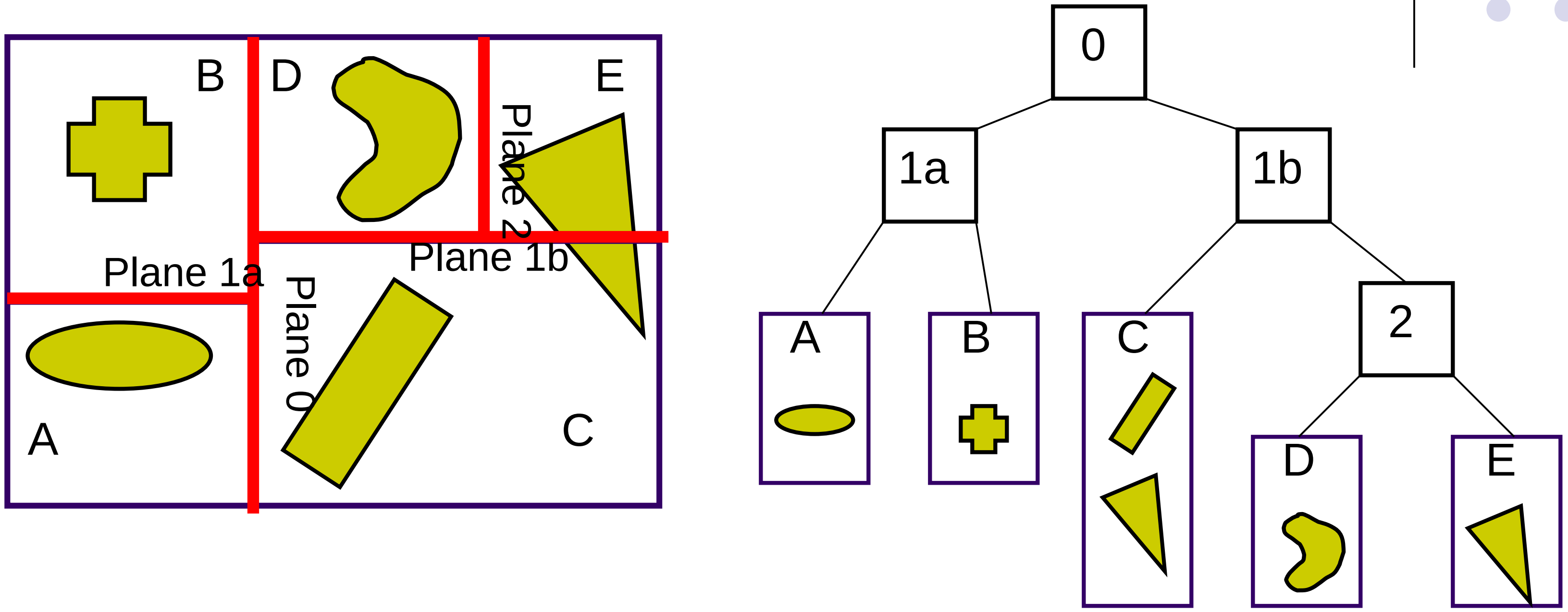


Axis-Aligned BSP tree

- General idea:
  - Divide space with a plane
  - Sort geometry into the space it belongs
  - Can only make a splitting plane along x,y, or z



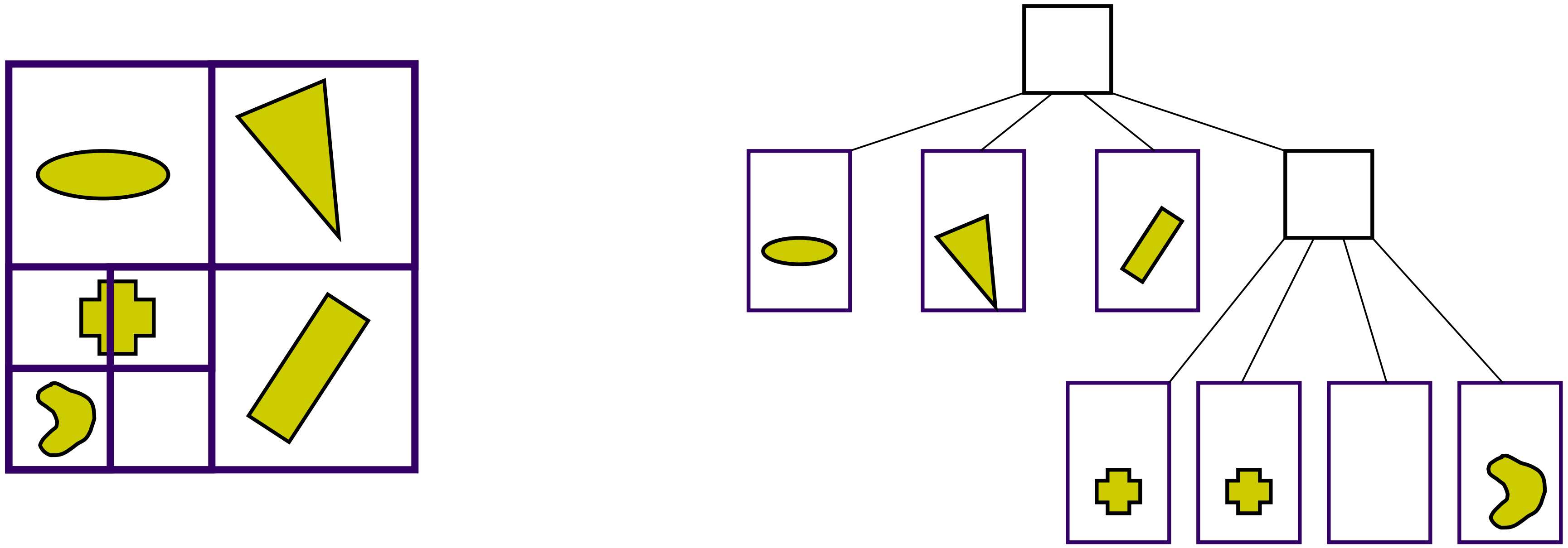
Axis-Aligned BSP tree



- Each internal node holds a divider plane
- Leaves hold geometry
- Differences compared to BVH
  - Encloses **entire space**
  - BVHs can use any desirable type of BV

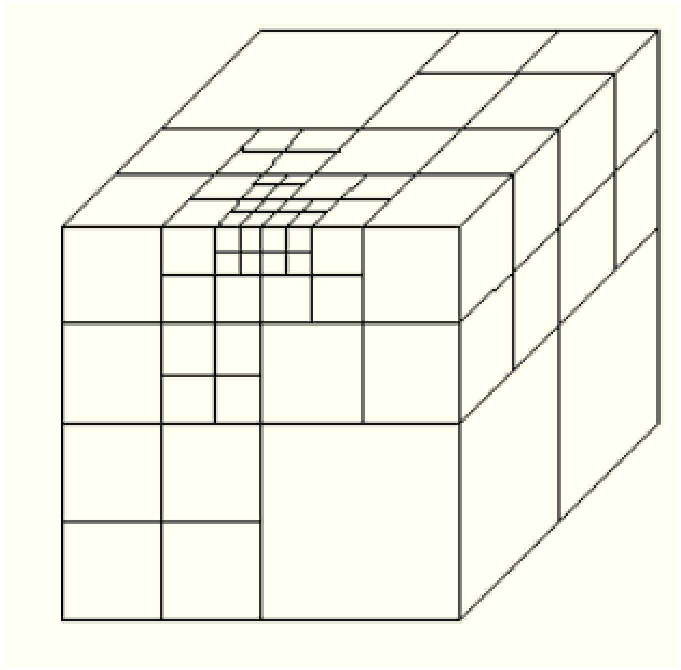
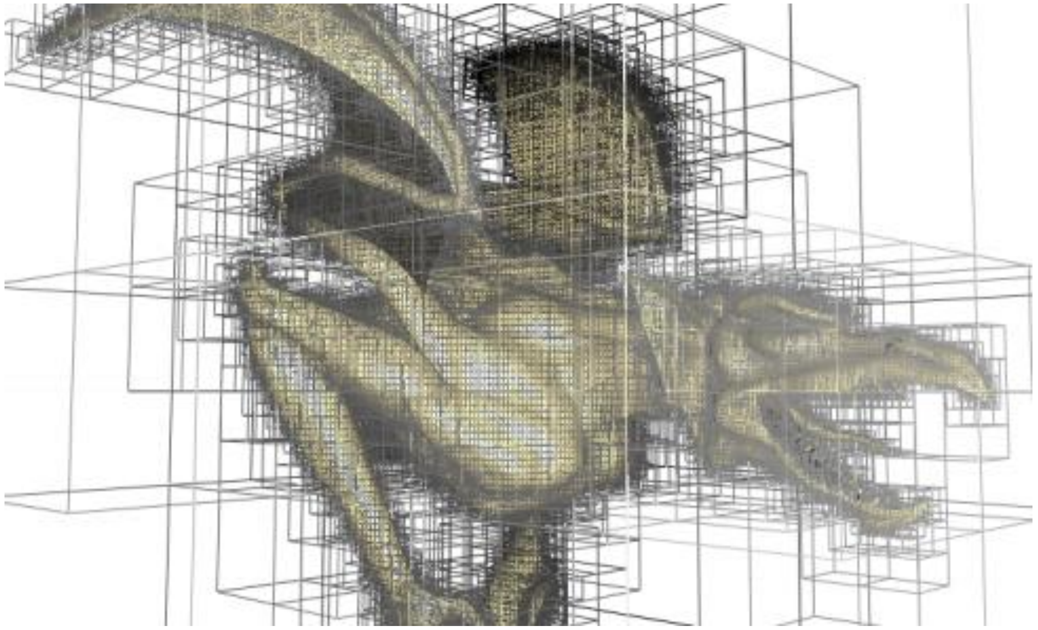
Octrees

- Similar to axis-aligned BSP trees but **regular**
- Will explain the quadtree, which is the 2D variant of an octree



- In 3D each square (or rectangle) becomes a box, and 8 children

Example of Octrees

References

- Hill and Kelley, Computer Graphics using OpenGL, 3rd edition, Chapter 12
- Akenine-Moller, Eric Haines and Naty Hoffman, Real Time Rendering (3rd edition)