



Ray Tracing

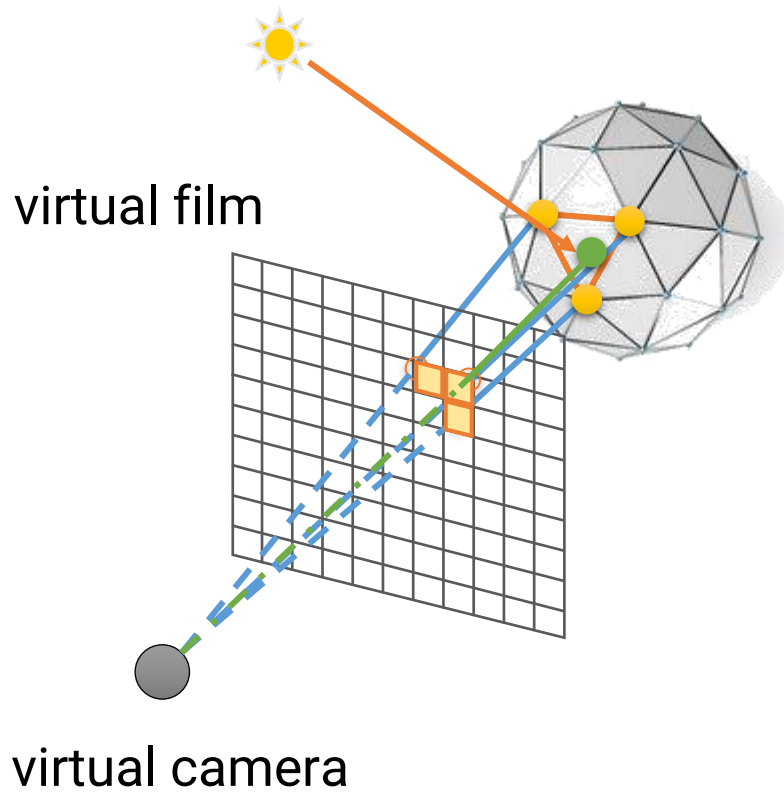
Computer Graphics

Yu-Ting Wu

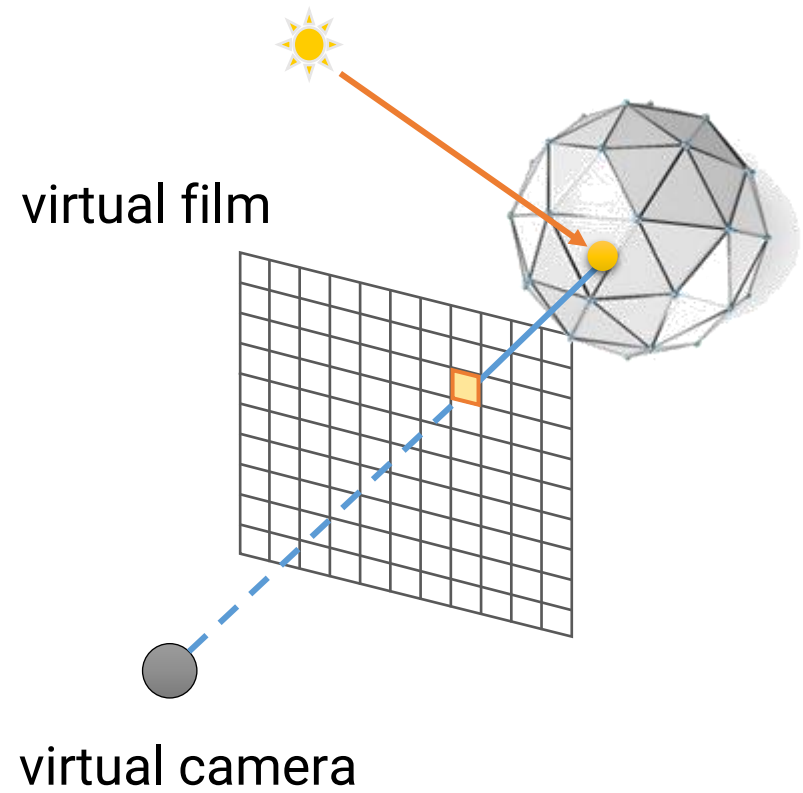
(Some of this slides are borrowed from Prof. Yung-Yu Chuang)

Recap: Digital Image Synthesis

Rasterization

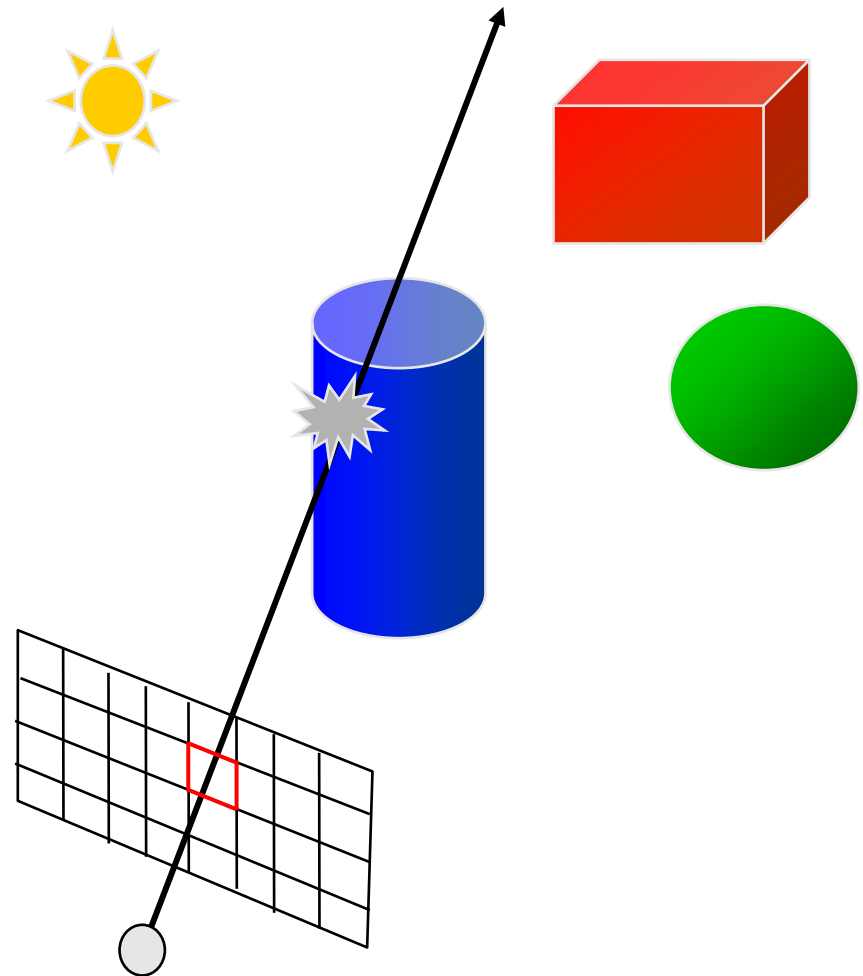


Ray tracing



Ray Casting

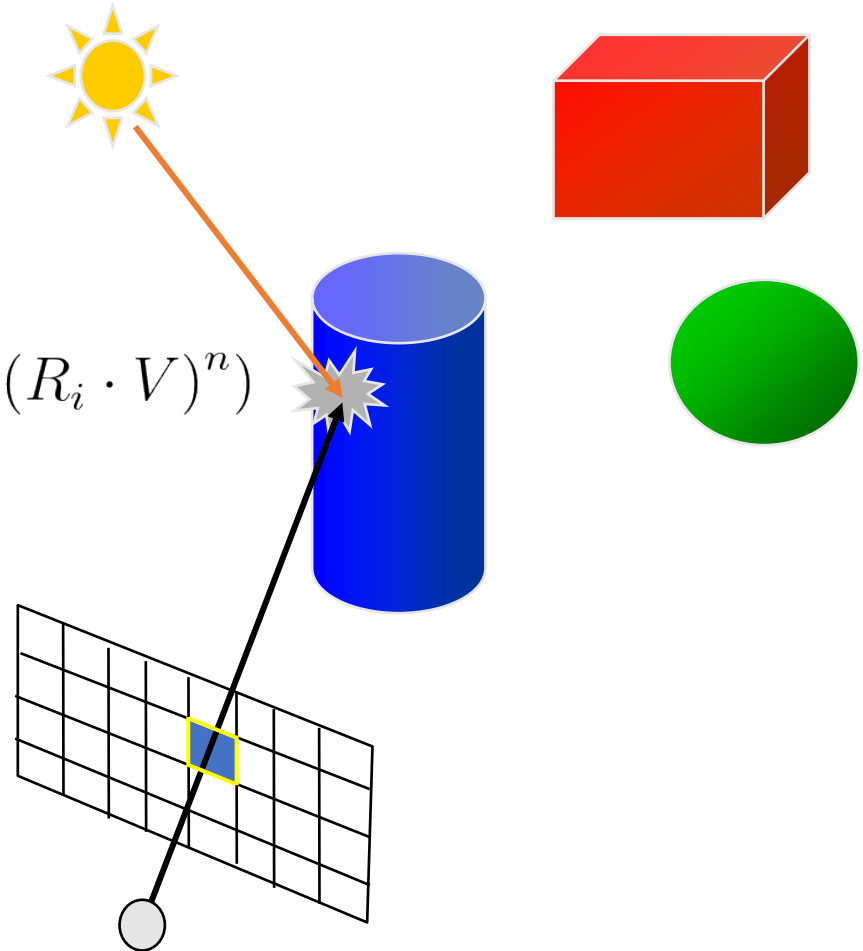
- Proposed by Appel [1968]



Ray Casting (cont.)

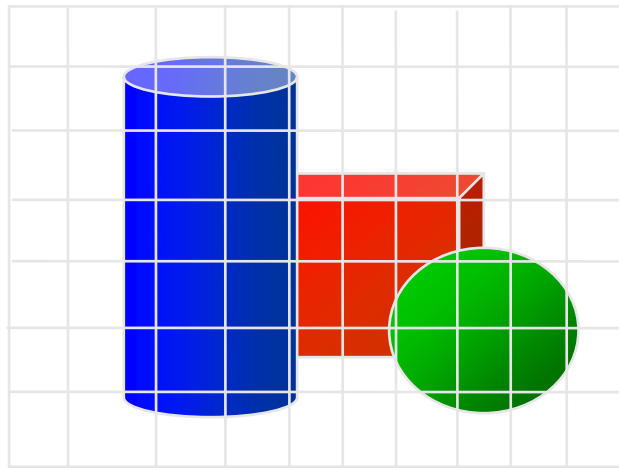
- Proposed by Appel [1968]

$$K_a I_a + \sum_{i=1}^{nls} I_i (K_d (L_i \cdot N) + K_s (R_i \cdot V)^n)$$

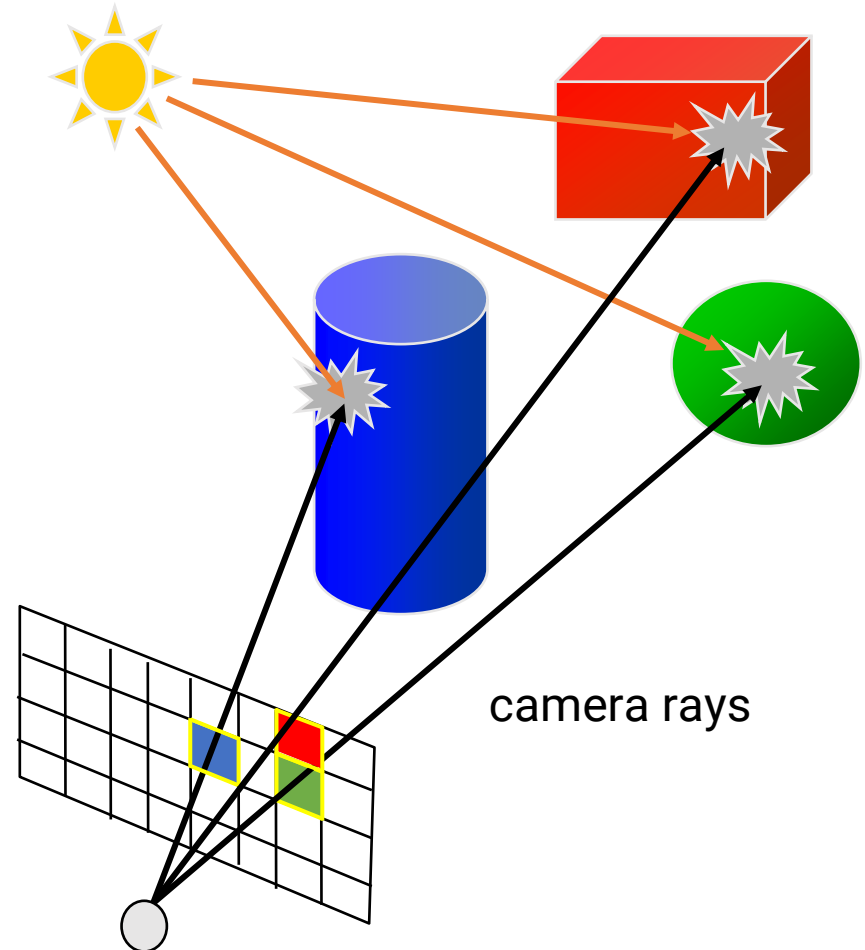


Ray Casting (cont.)

- Proposed by Appel [1968]



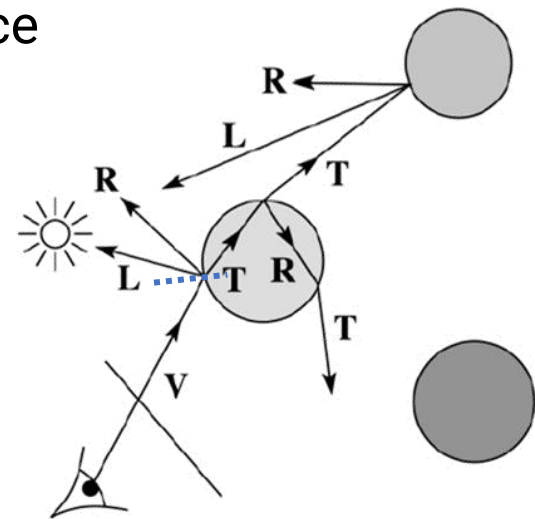
local illumination



camera rays

Whitted Ray Tracing

- Proposed by Whitted, 1980
- **Recursive** trace rays for **shadows**, perfect **specular** (e.g., mirror), and perfect **transparent** (e.g., glass) objects
 - For each pixel, trace a primary ray in the direction \mathbf{V} to the first visible surface
 - For each intersection, trace secondary rays including
 - Shadow rays (\mathbf{L}) to each light source
 - Reflected ray (\mathbf{R})
 - Refracted ray (\mathbf{T})



Whitted Ray Tracing (cont.)

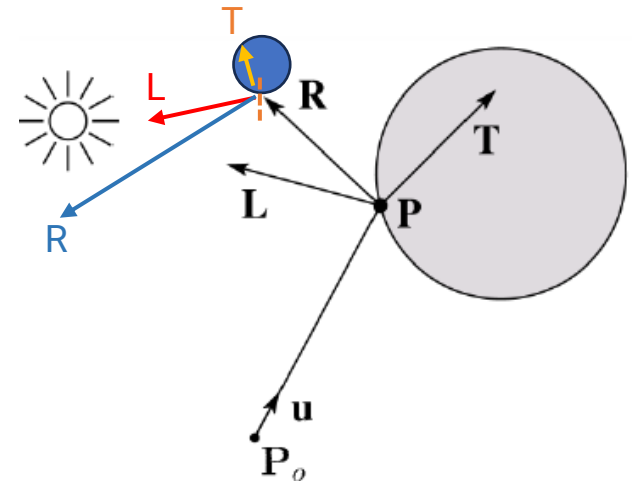
- Recursive shading
 - If $I(P_0, u)$ is the intensity seen from the point P along direction u

$$I(P_0, u) = I_{\text{direct}} + I_{\text{reflected}} + I_{\text{refracted}}$$

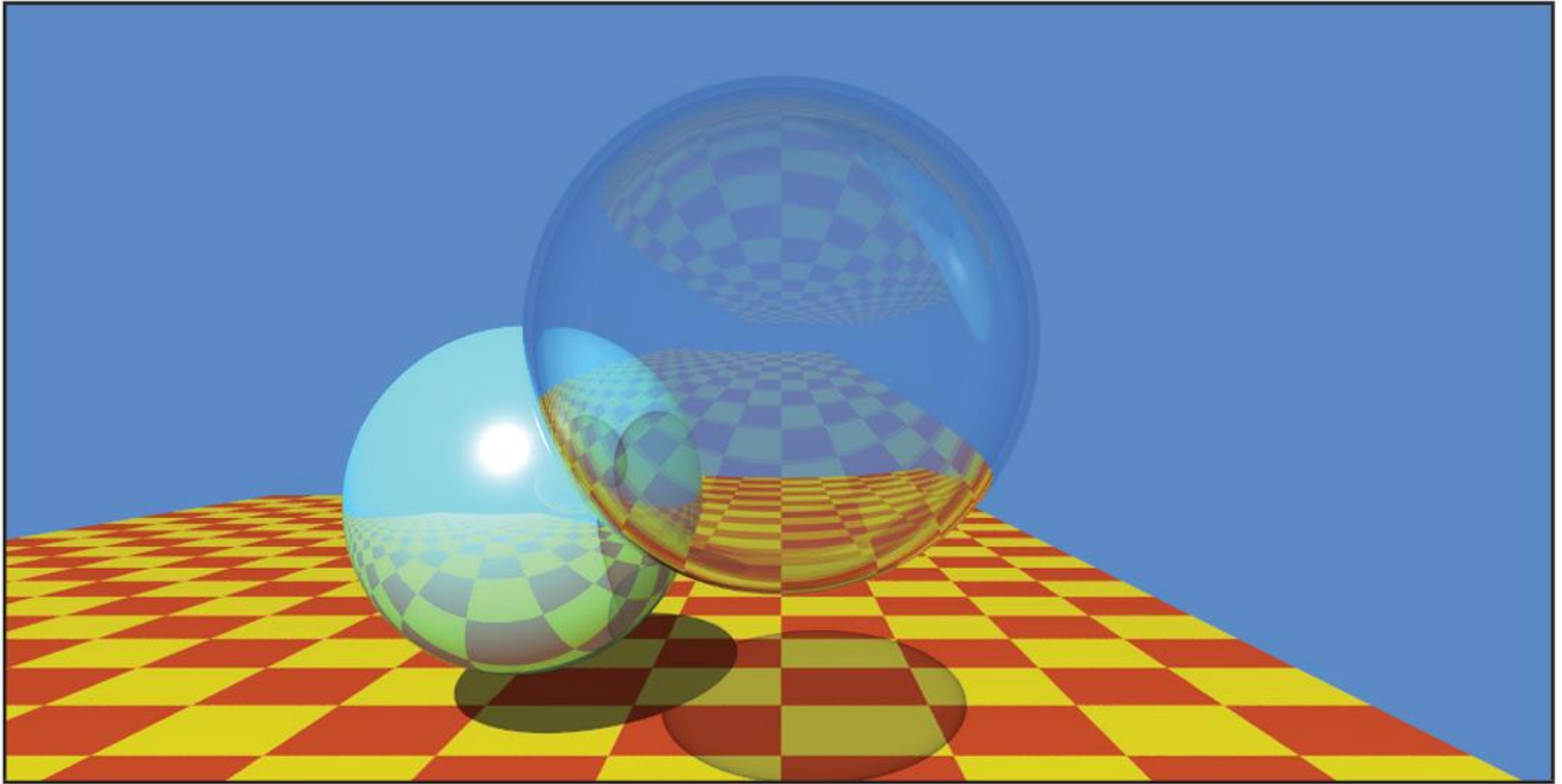
$$I_{\text{direct}} = \text{Shade}(N, L, u, R)$$

$$I_{\text{reflected}} = I(P, R)$$

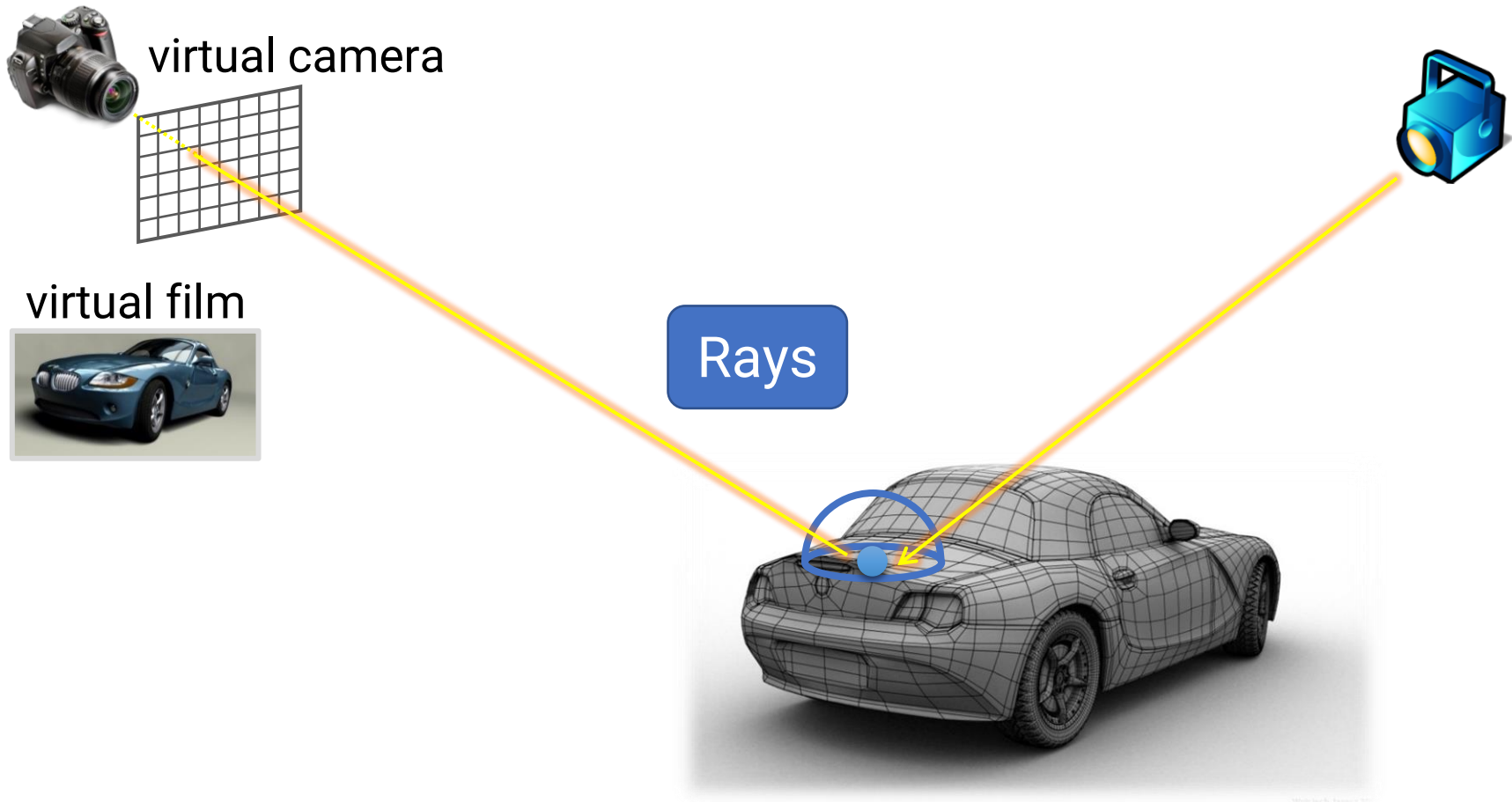
$$I_{\text{refracted}} = I(P, T)$$



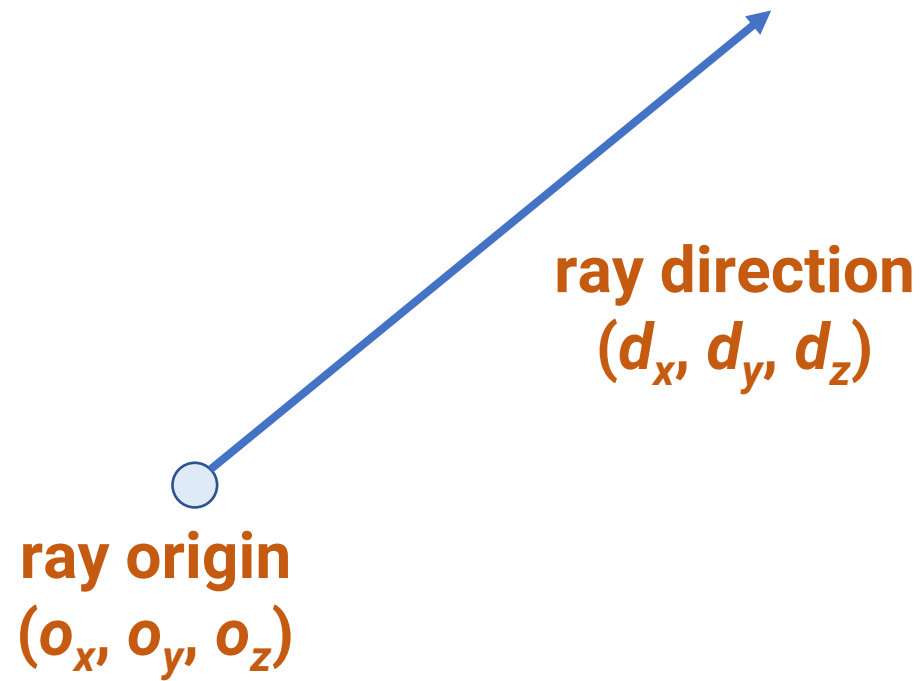
Whitted Ray Tracing (cont.)



Components of Ray Tracing

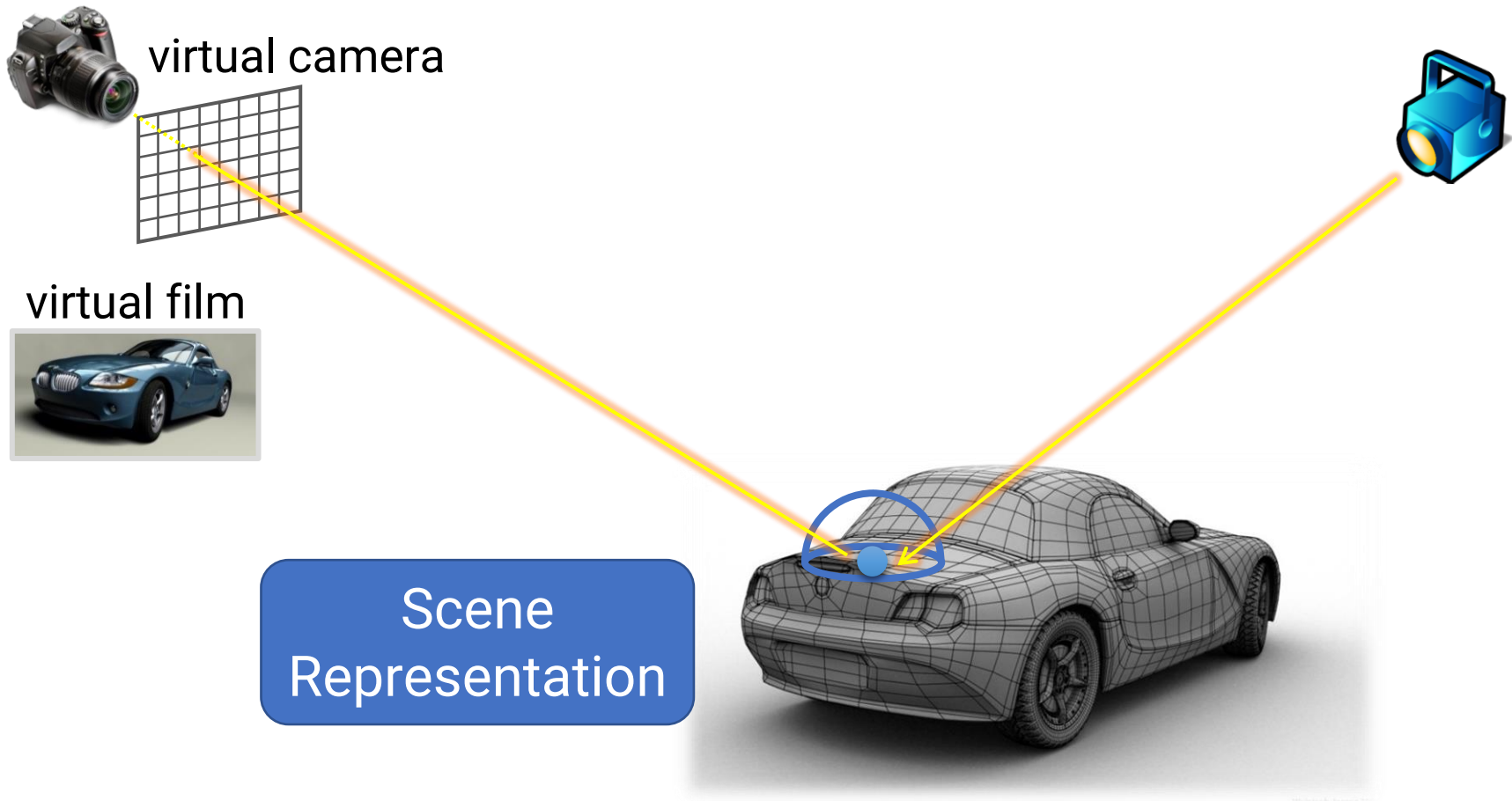


Rays



Components of Ray Tracing

- A united approach for different light transport paths

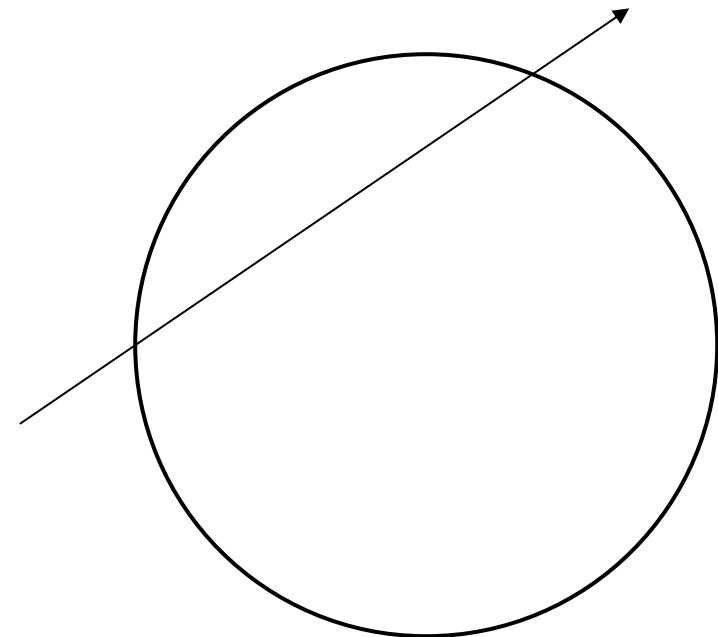
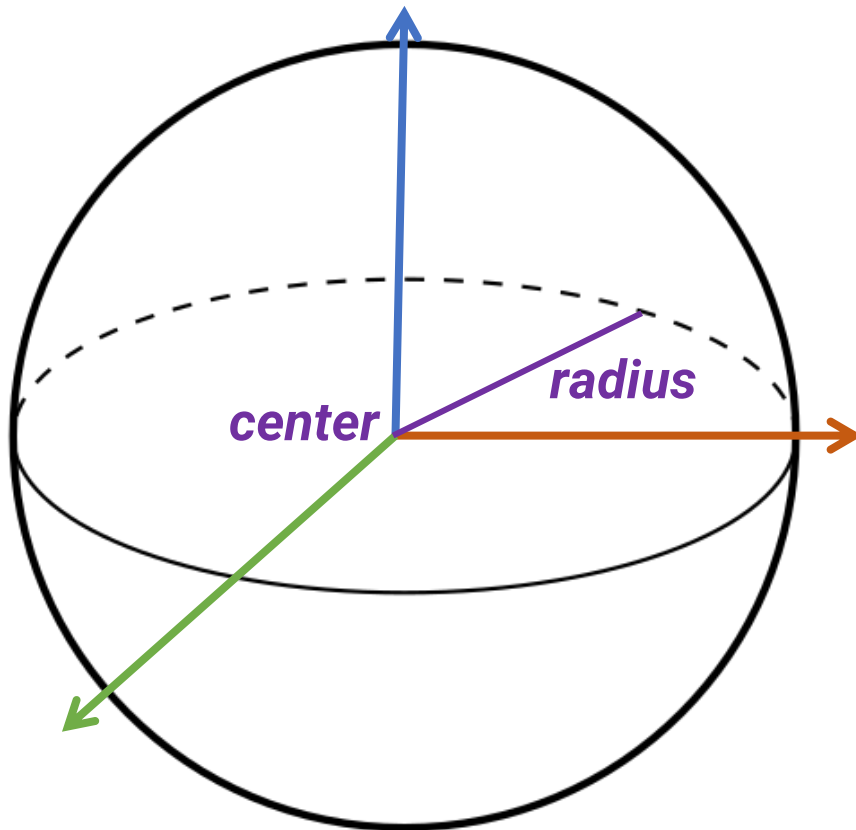


Scene Representation in Ray Tracing

- Basically, just like what you learned in rasterization
 - Also use the idea of object instancing (world transform)
 - Also use camera space (easier to generate rays)
- But **NOT** limited to triangles
- You can use **any** representation if and only if you can **find the intersection of a ray and the surface**

Scene Representation in Ray Tracing (cont.)

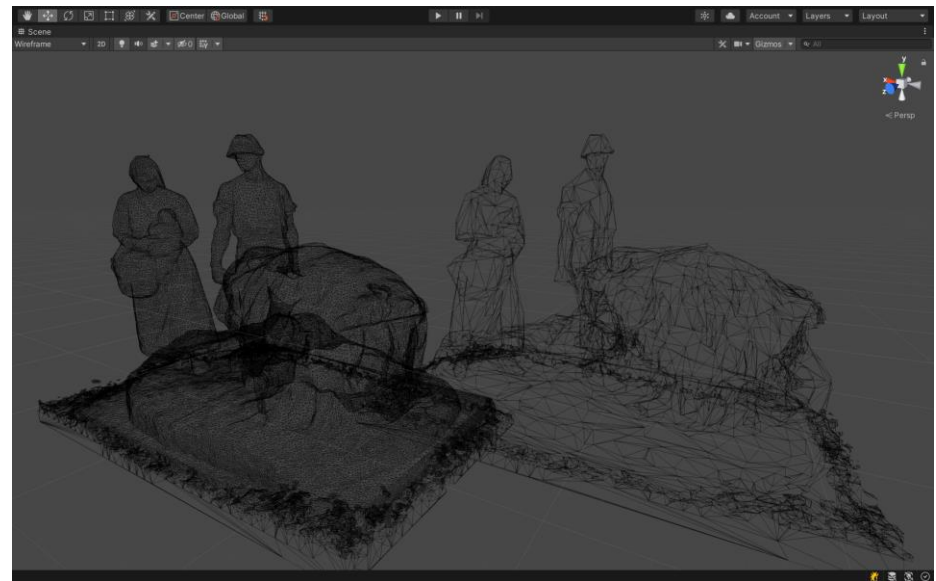
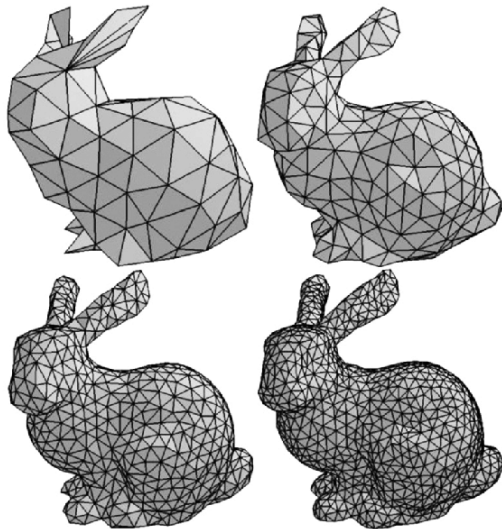
- For example, you can represent a sphere using its center and radius



Solve math for the intersection

Scene Representation in Ray Tracing (cont.)

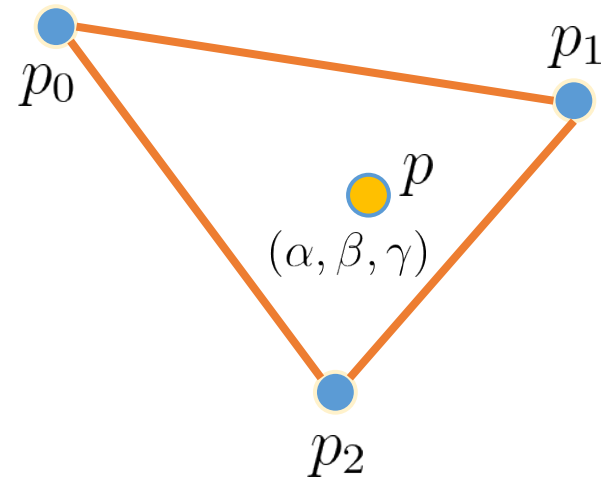
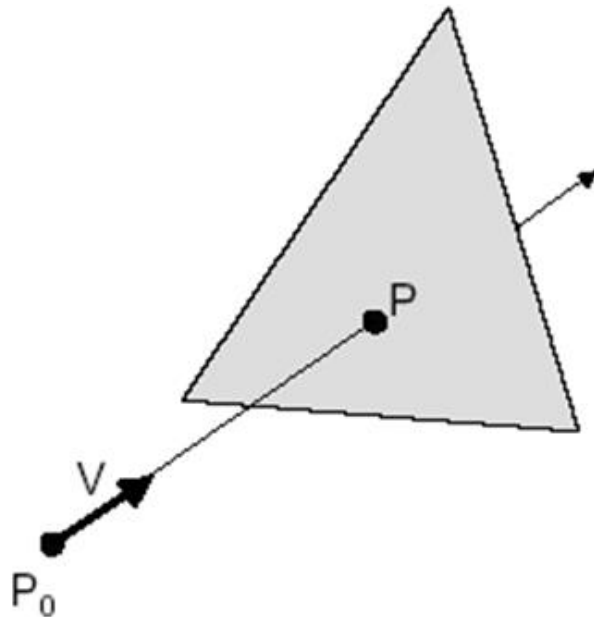
- Triangles are still the most commonly used representation because they can represent arbitrary shapes
- In offline rendering, we usually break up the triangles of objects and treat them “**triangle soup**”



Scene Representation in Ray Tracing (cont.)

• Ray-triangle intersection

- Intersect ray with the plane the triangle locates
- Check if the intersection point is inside the triangle
 - Can use **barycentric coordinate**

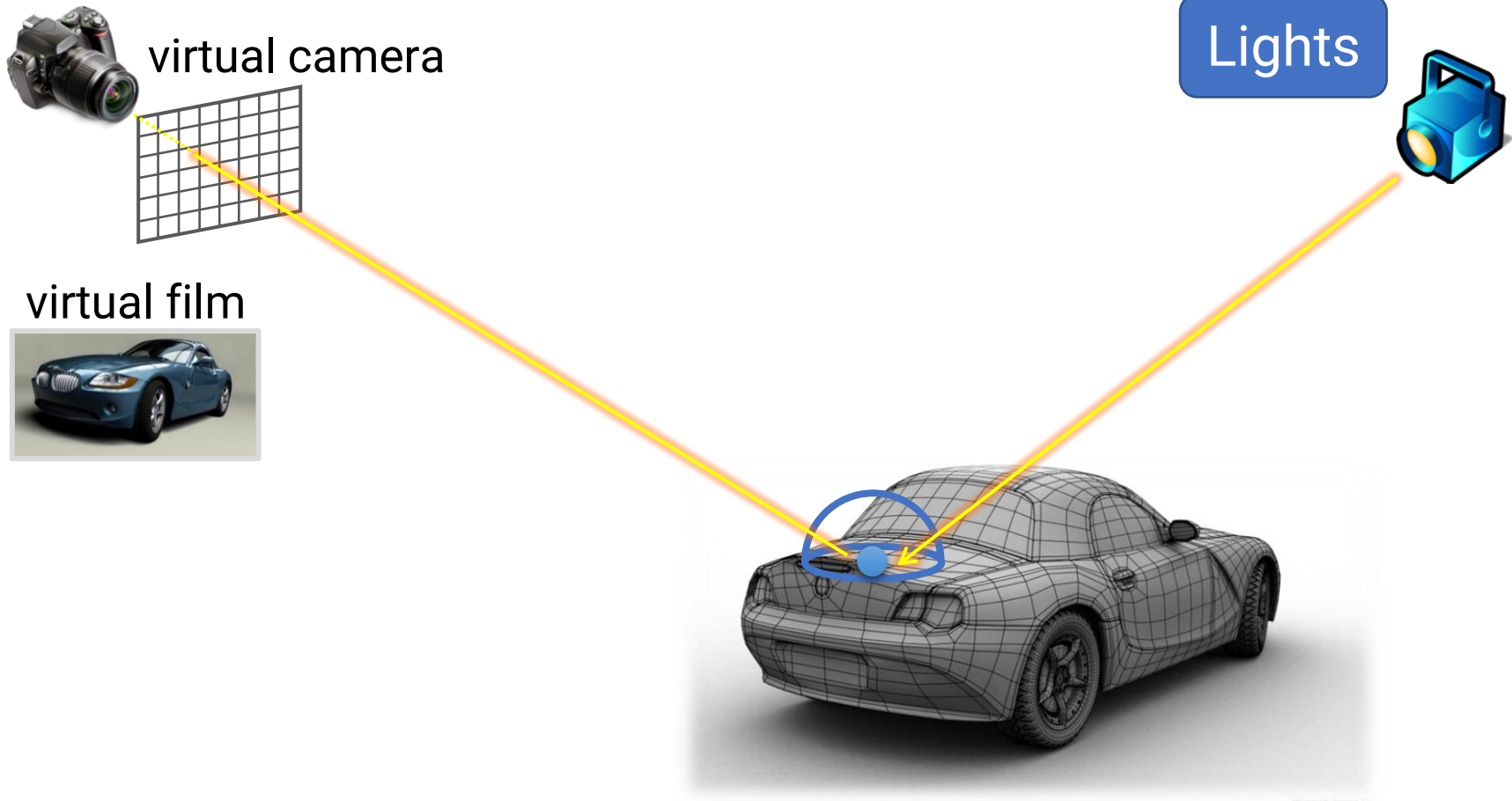


$$p = \alpha p_0 + \beta p_1 + \gamma p_2$$

The values $\alpha, \beta, \gamma \in [0, 1]$ **if and only if p is inside the triangle**

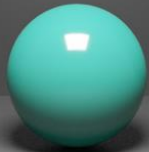
Components of Ray Tracing

- A united approach for different light transport paths

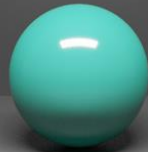


Lights in Ray Tracing

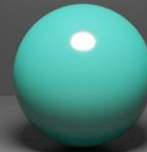
- Basically, just like what you learned in rasterization
- But more complex lights such as area lights and environment lighting are also used for photorealism
 - Estimate the lighting contribution by **sampling**



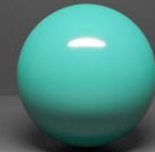
Square



Rectangle



Disk



Ellipse

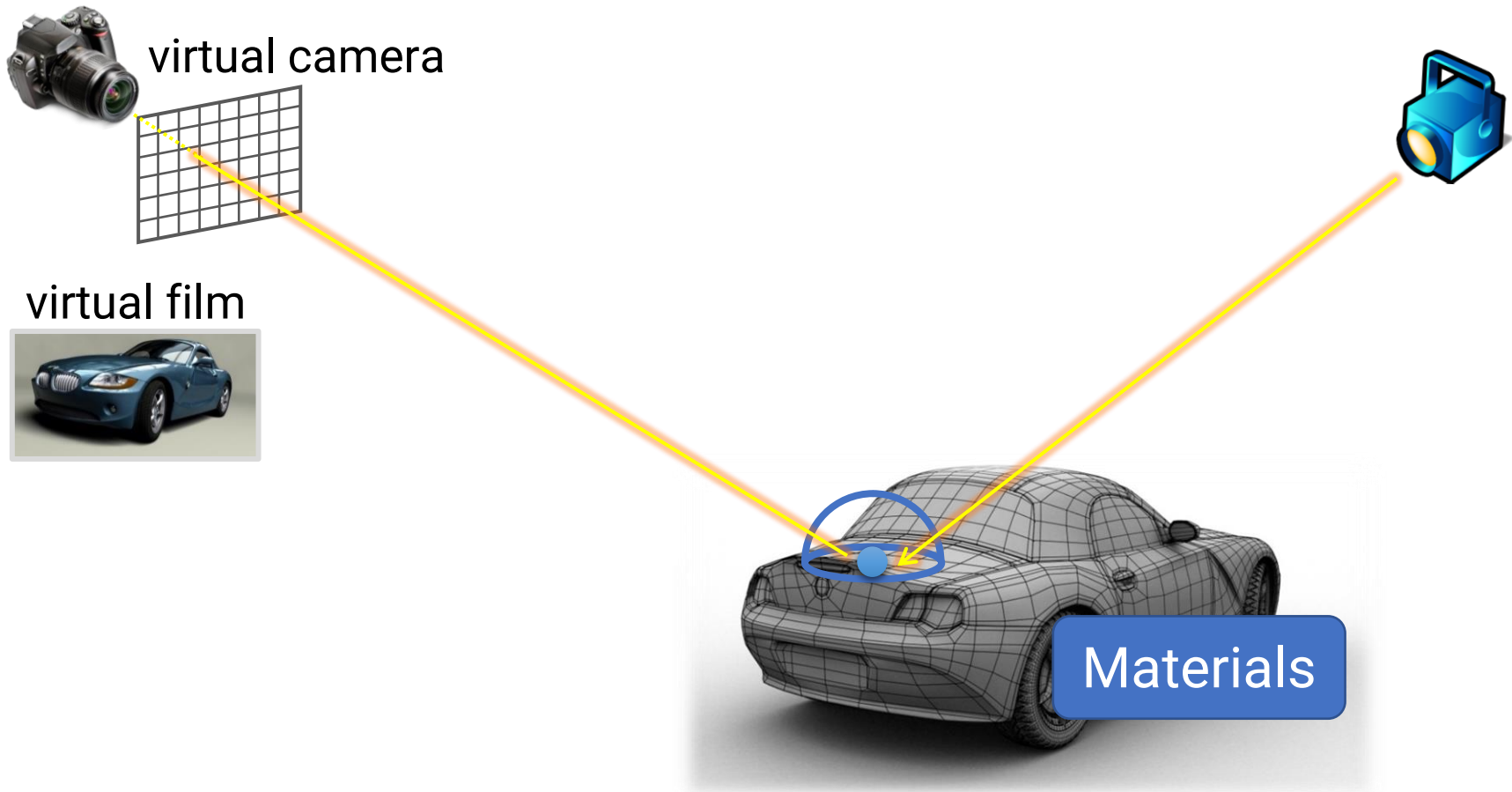
Lights in Ray Tracing

- Basically, just like what you learned in rasterization
- But more complex lights such as area lights and environment lighting are also used for photorealism
 - Estimate the lighting contribution by **sampling**



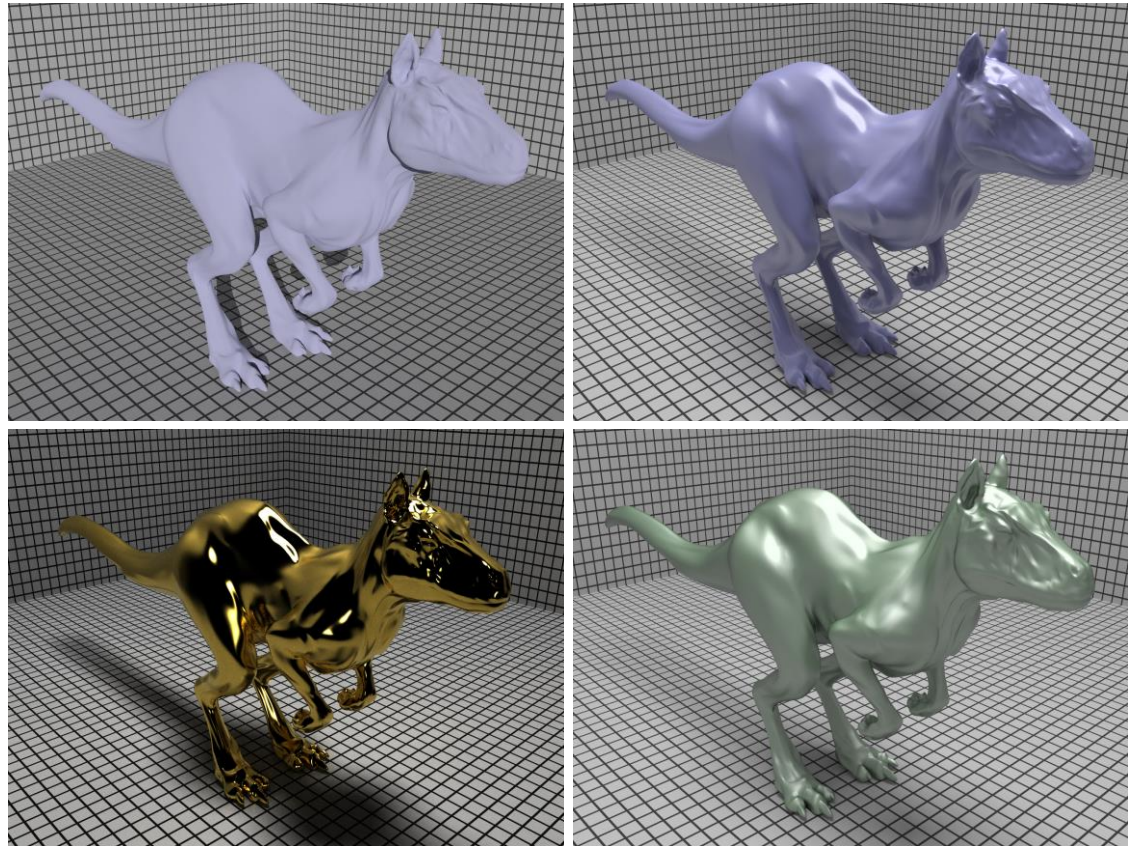
Components of Ray Tracing

- A united approach for different light transport paths



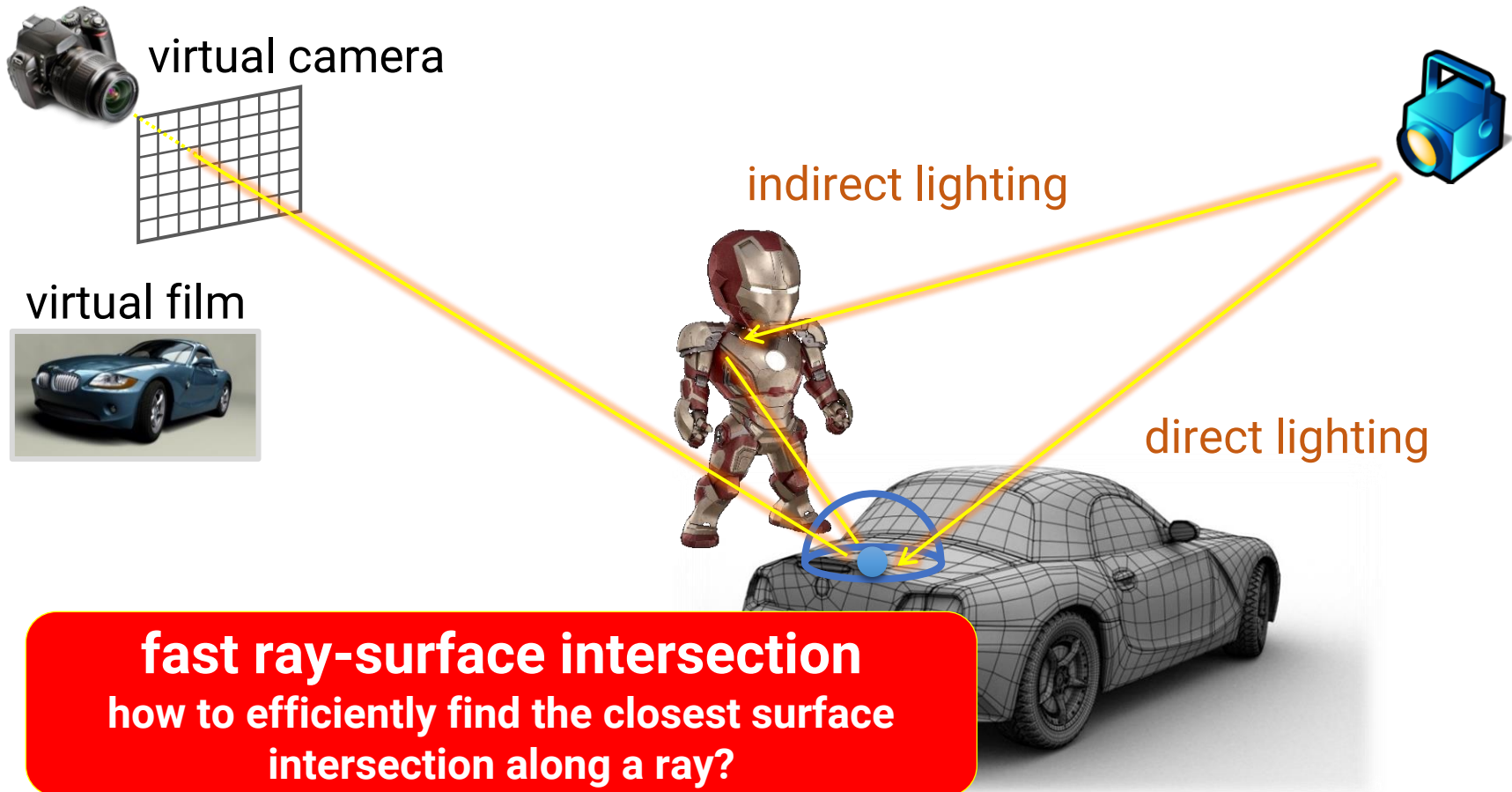
Materials in Ray Tracing

- Basically, just like what you learned in rasterization
- But more complex materials such as the microfacet models



Key: Fast Ray-Surface Intersection

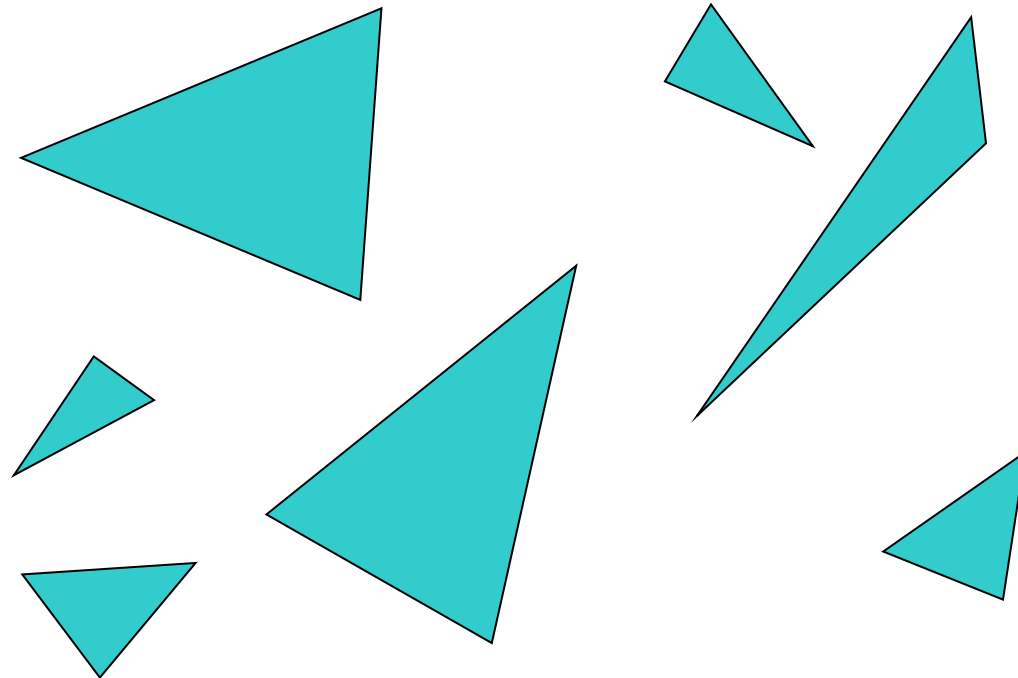
- A united approach for different light transport paths



Acceleration Structure

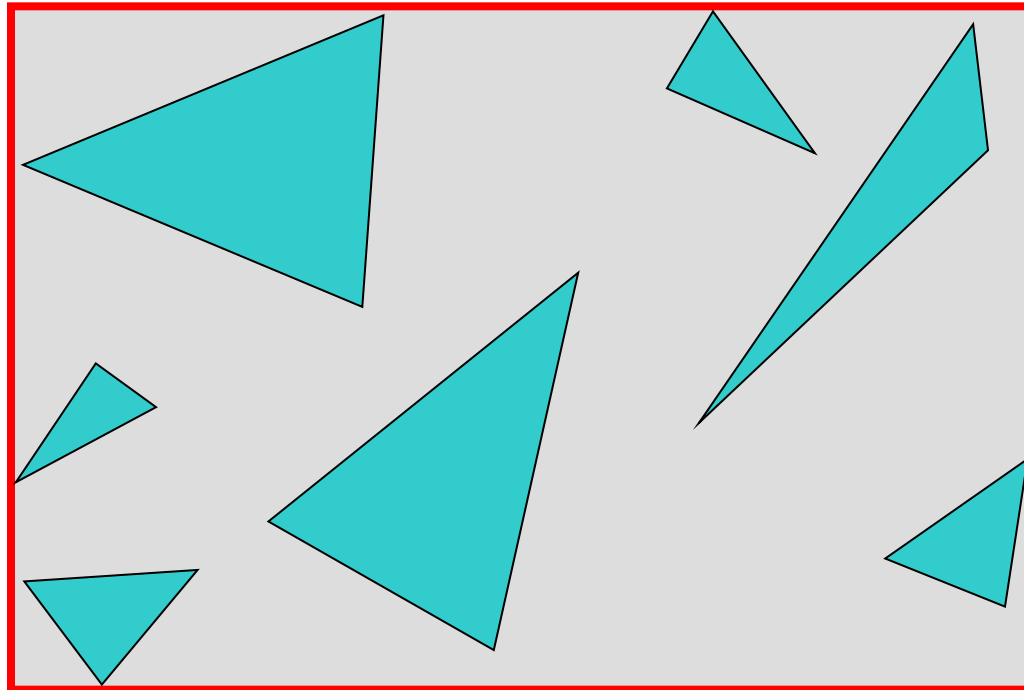
- **Reduce the required number of ray-surface intersection**
- Common acceleration structures
 - Bounding volume hierarchy (BVH)
 - Space subdivision

Bounding Volume Hierarchy



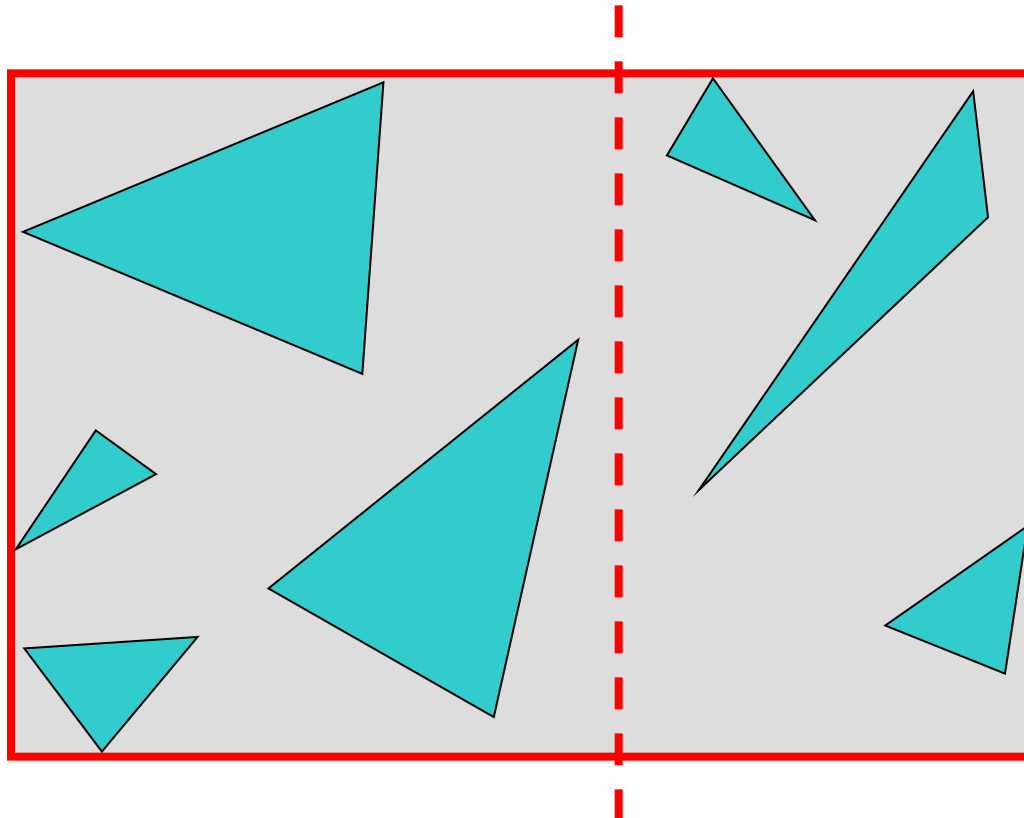
Bounding Volume Hierarchy (cont.)

- Find the bounding box of all objects



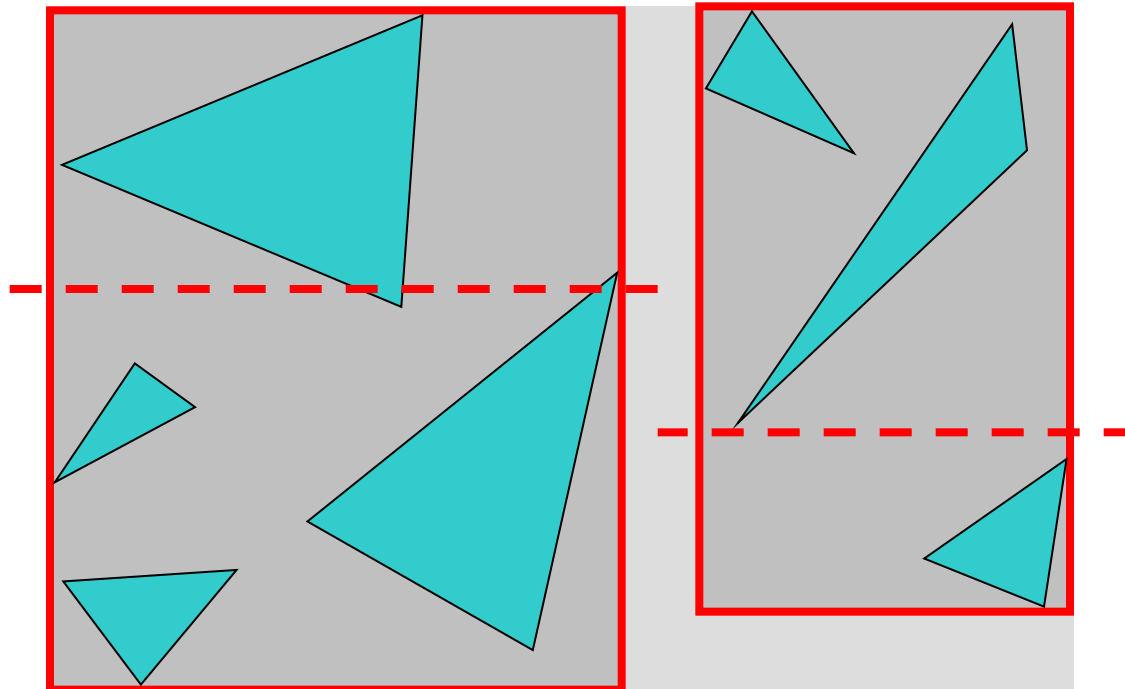
Bounding Volume Hierarchy (cont.)

- Find the bounding box of all objects
- Split shapes into two groups



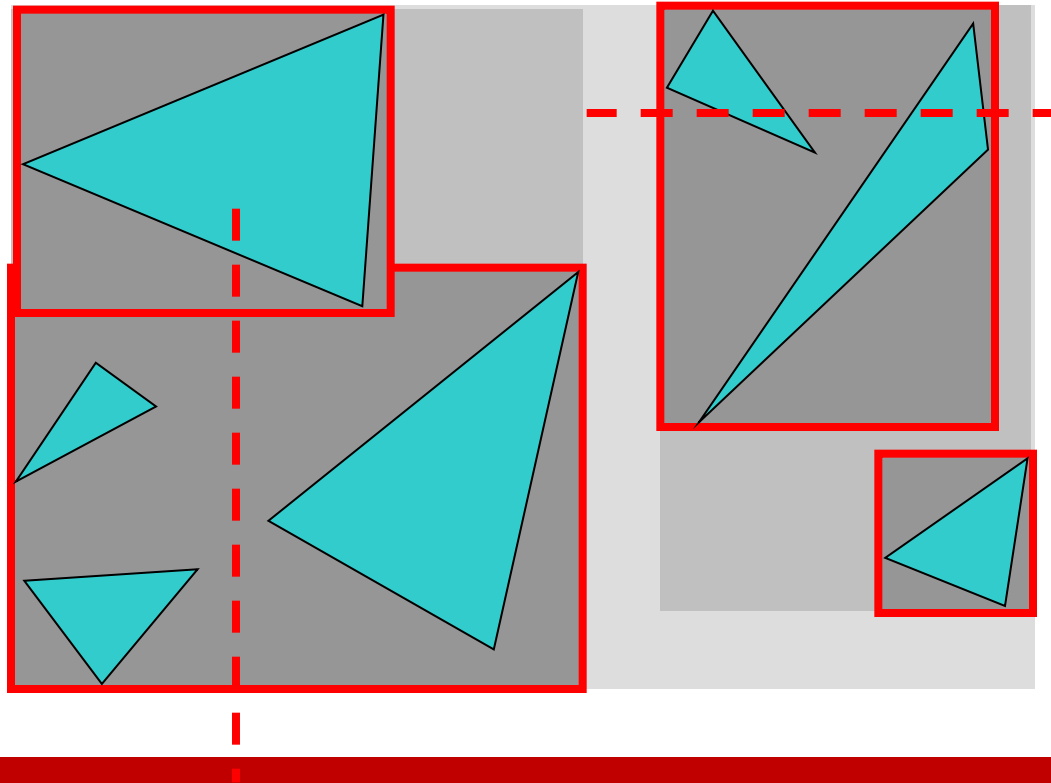
Bounding Volume Hierarchy (cont.)

- Find the bounding box of all objects
- Split shapes into two groups
- Recursive



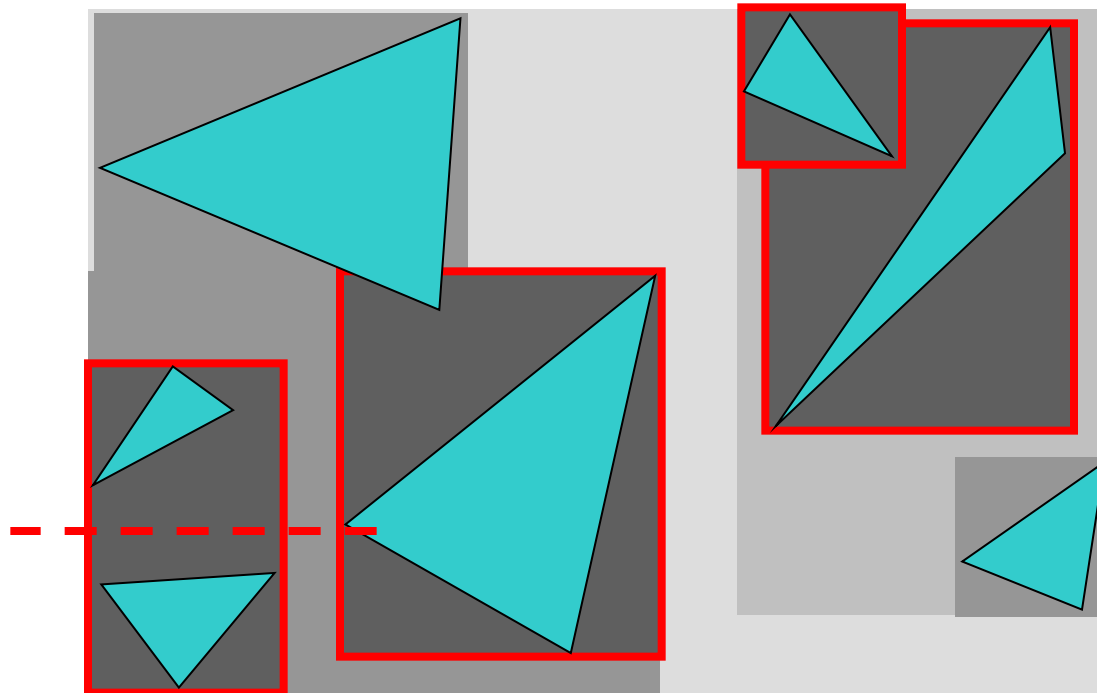
Bounding Volume Hierarchy (cont.)

- Find the bounding box of all objects
- Split shapes into two groups
- Recursive



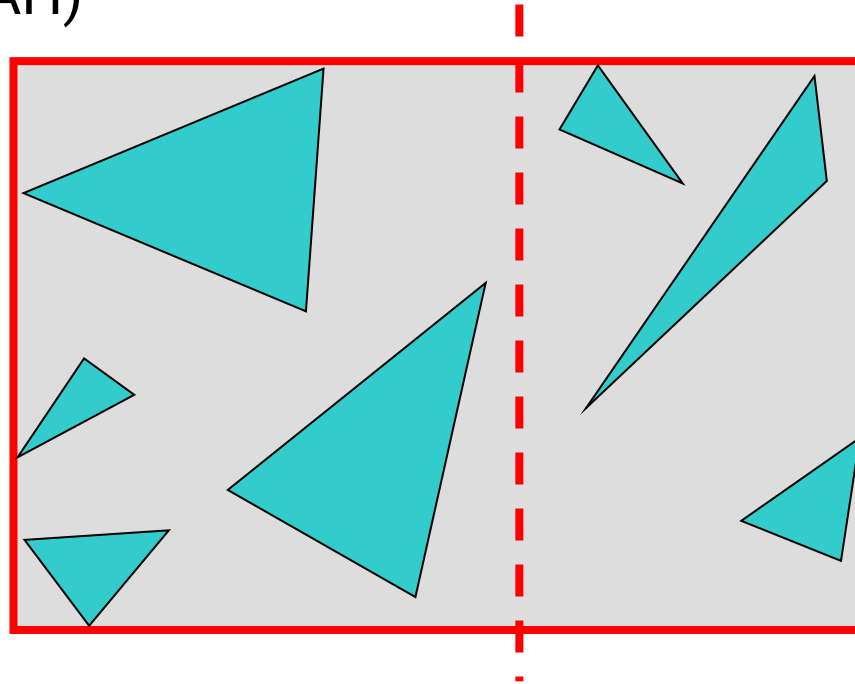
Bounding Volume Hierarchy (cont.)

- Find the bounding box of all objects
- Split shapes into two groups
- Recursive



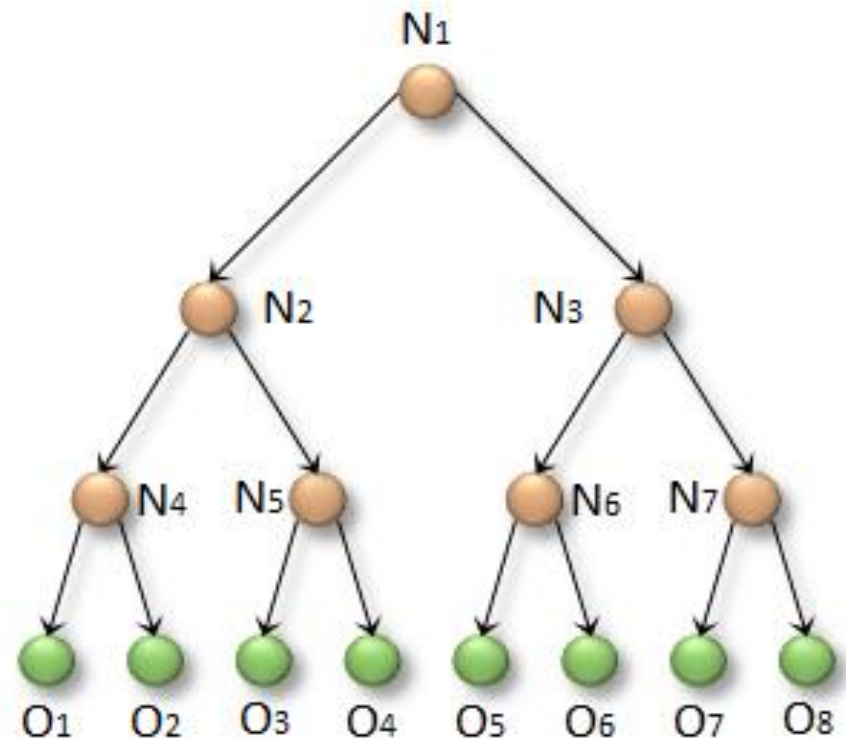
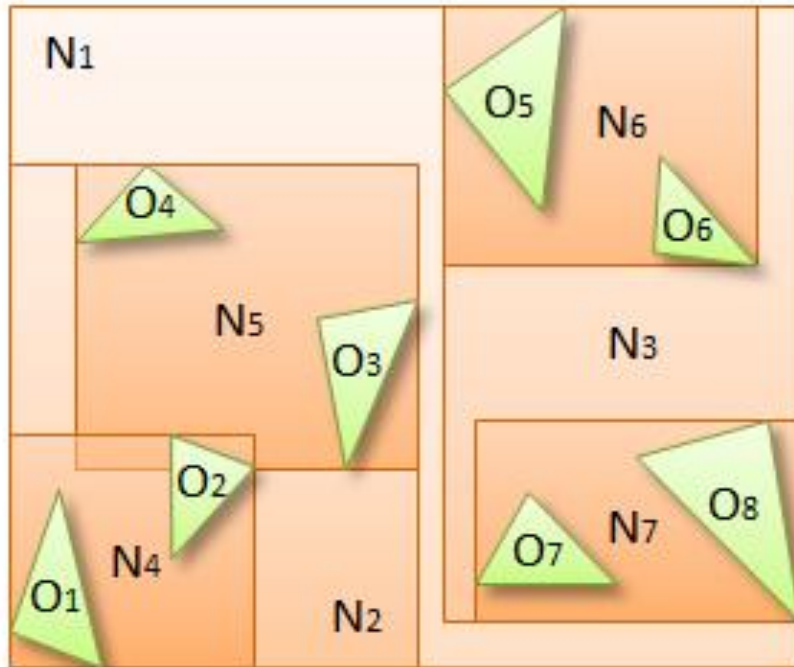
Bounding Volume Hierarchy (cont.)

- Where to split?
 - At midpoint
 - Put half of the shapes on each side
 - Use some objective functions, such as surface-area heuristic (SAH)



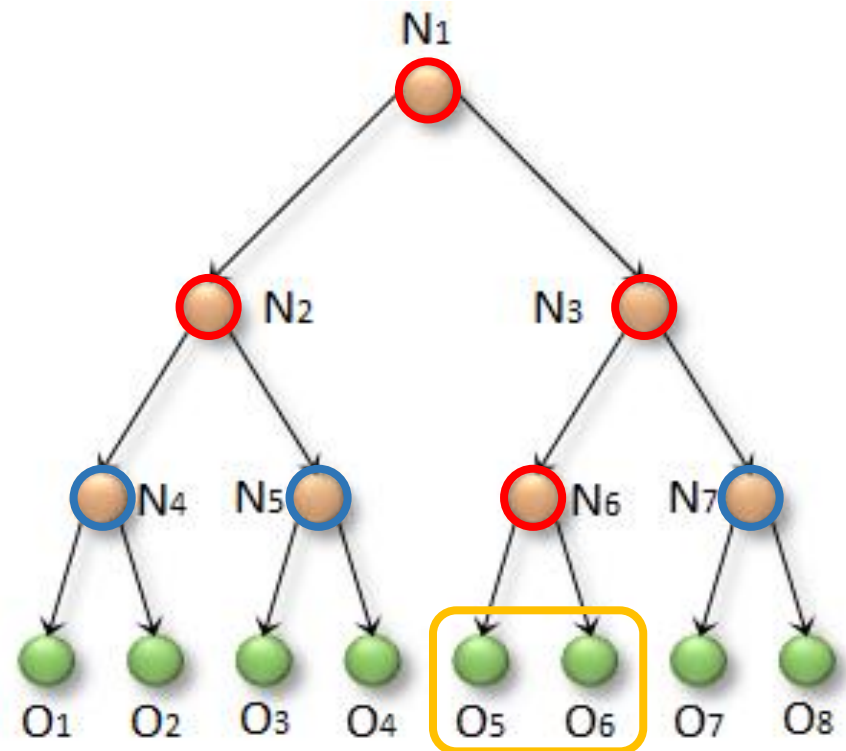
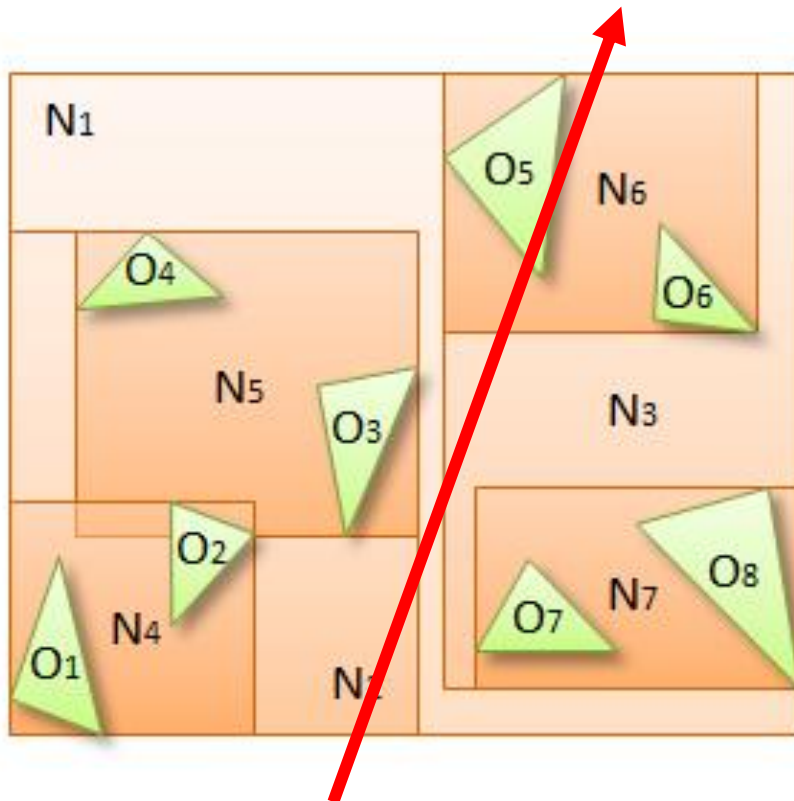
Bounding Volume Hierarchy (cont.)

- **Preprocess:** build a hierarchy of bounding volumes
 - The bounding volume of an interior node contains all children

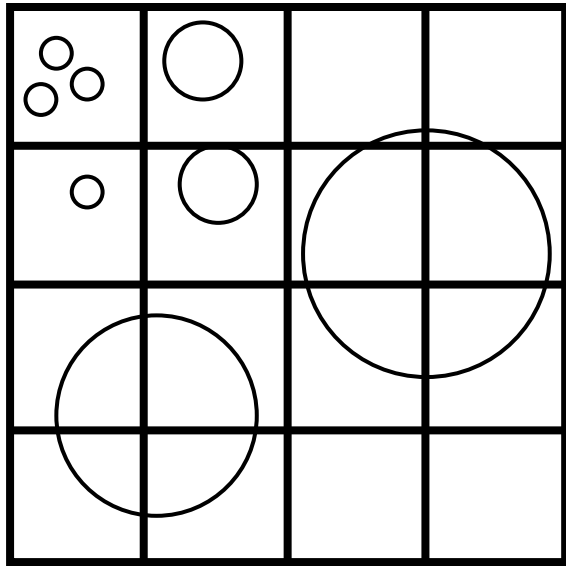


Bounding Volume Hierarchy (cont.)

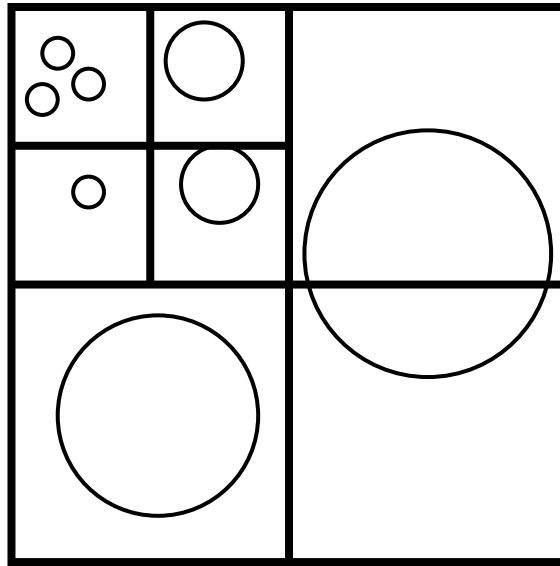
- **Rendering:** use the hierarchy to accelerate ray intersections
 - Test node contents only if the ray hits the bounding volume



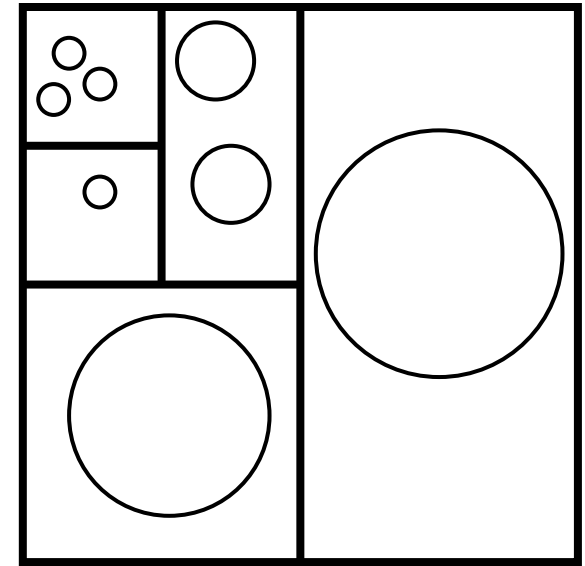
Space Subdivision Approaches



uniform grid

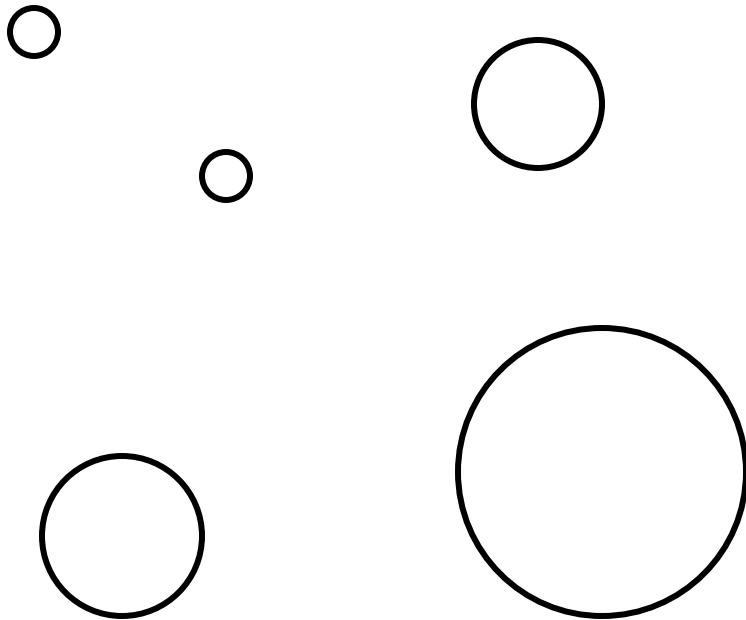


octree

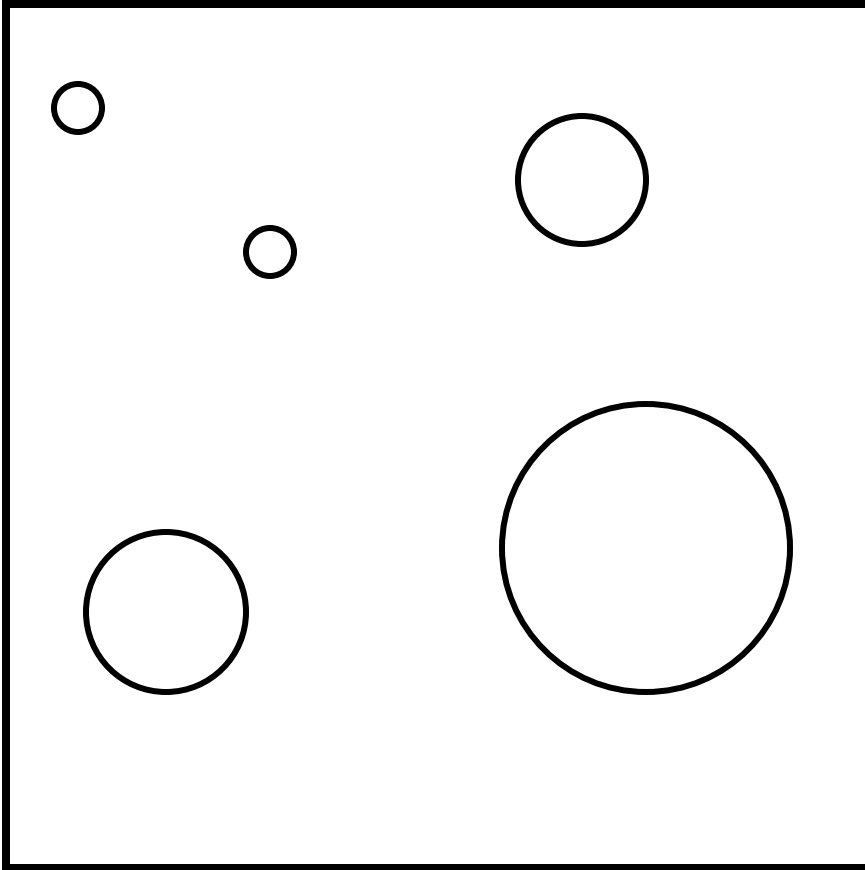


kd tree

Uniform Grid

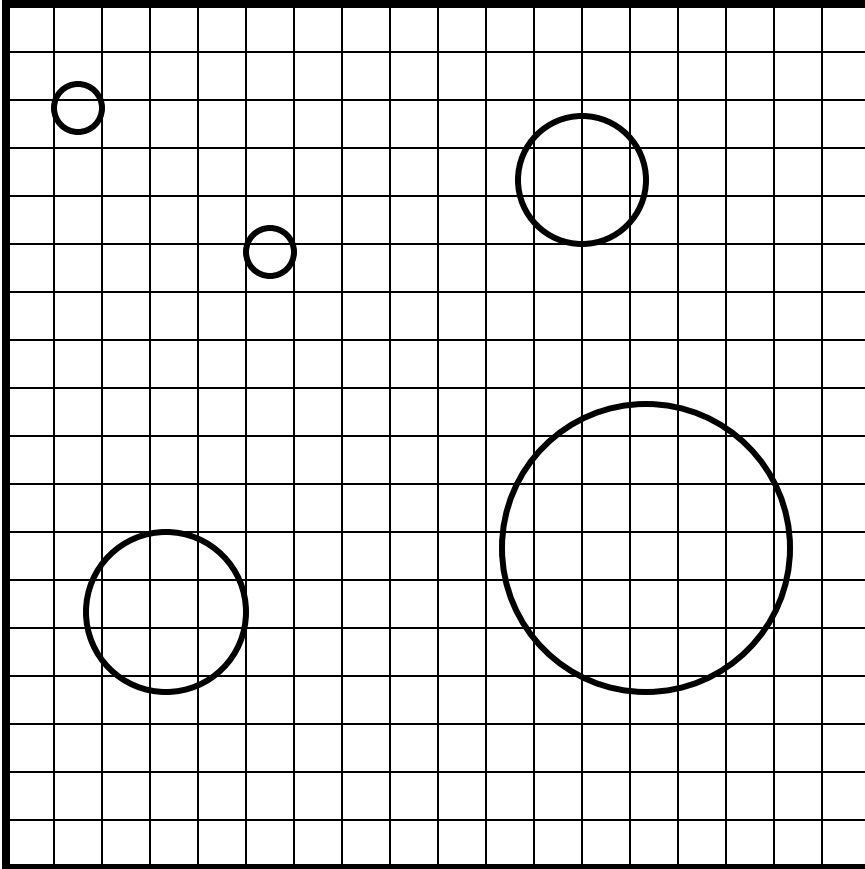


Uniform Grid (cont.)



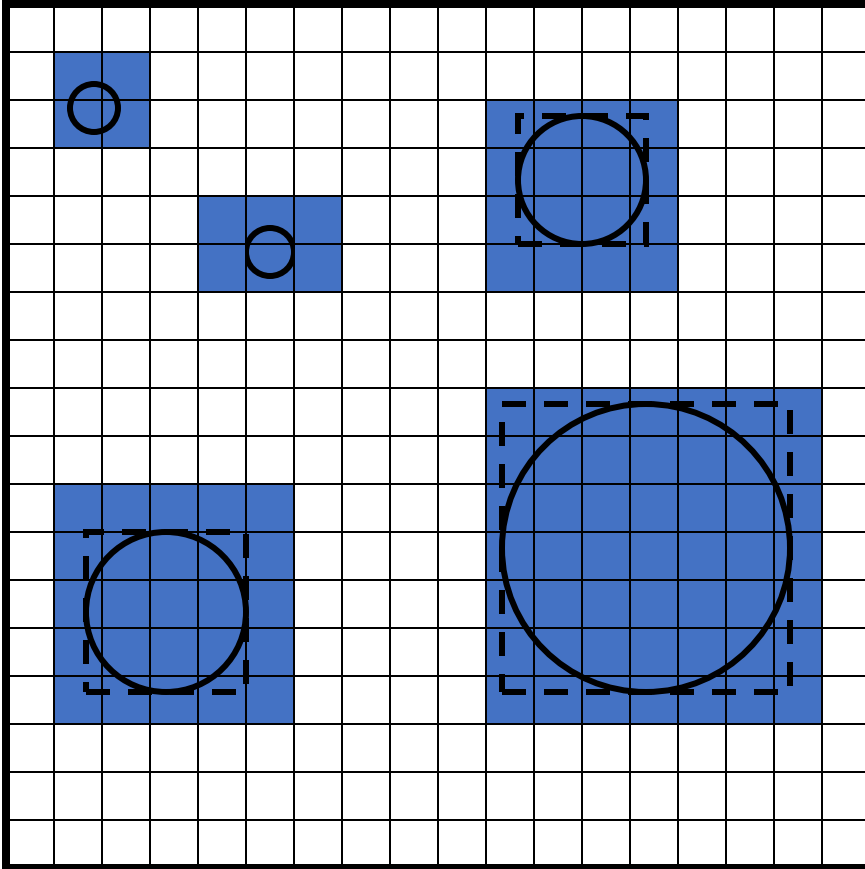
- **Preprocess**
 - Find the bounding box

Uniform Grid (cont.)



- **Preprocess**
 - Find the bounding box
 - Determine grid resolution

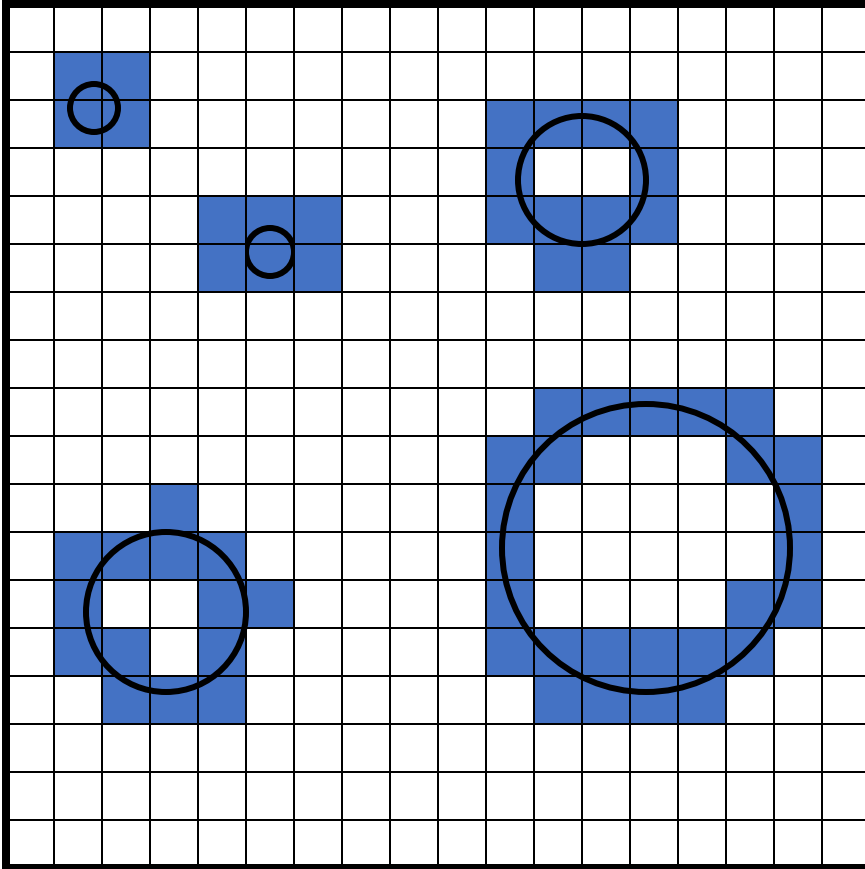
Uniform Grid (cont.)



- **Preprocess**

- Find the bounding box
- Determine grid resolution
- Place a shape in a cell if its bounding box overlaps the cell

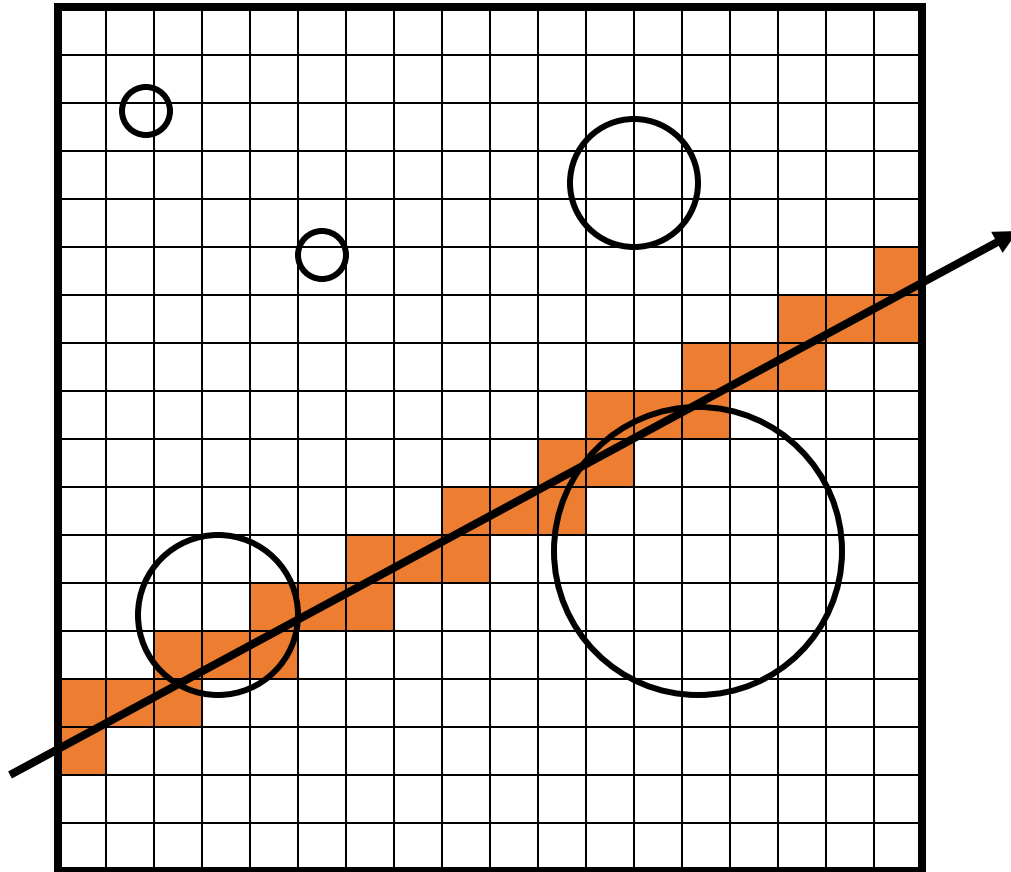
Uniform Grid (cont.)



- **Preprocess**

- Find the bounding box
- Determine grid resolution
- Place a shape in a cell if its bounding box overlaps the cell
- Check that if the shape overlaps the cell

Uniform Grid (cont.)



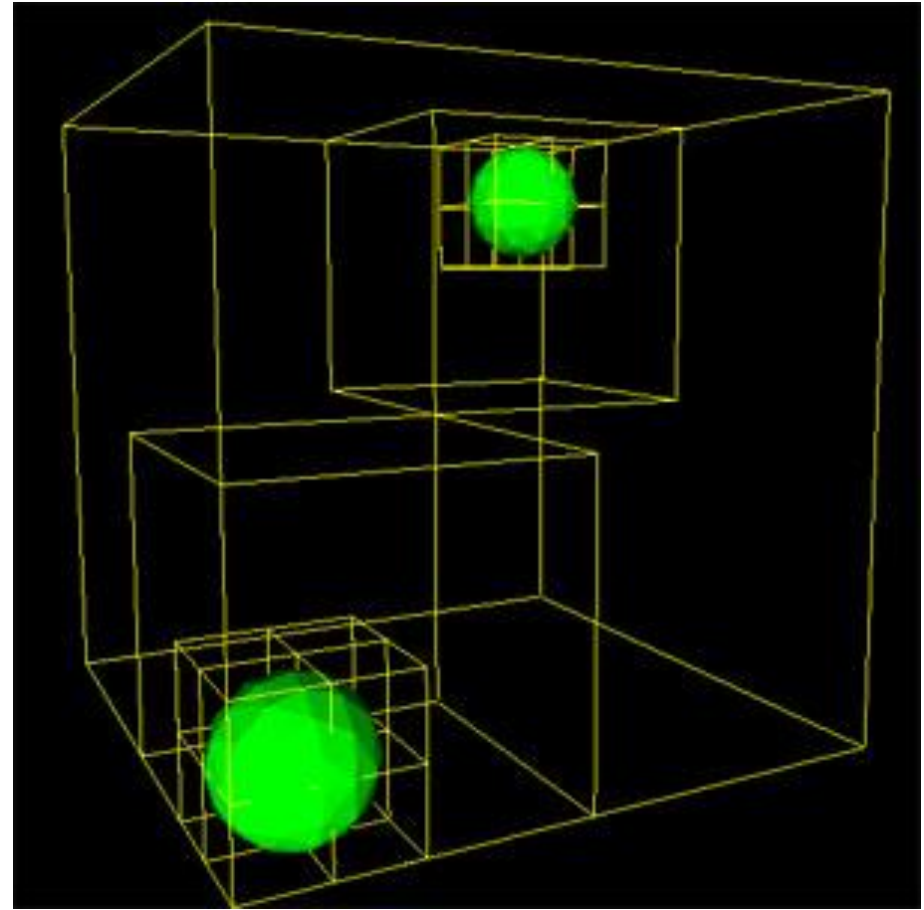
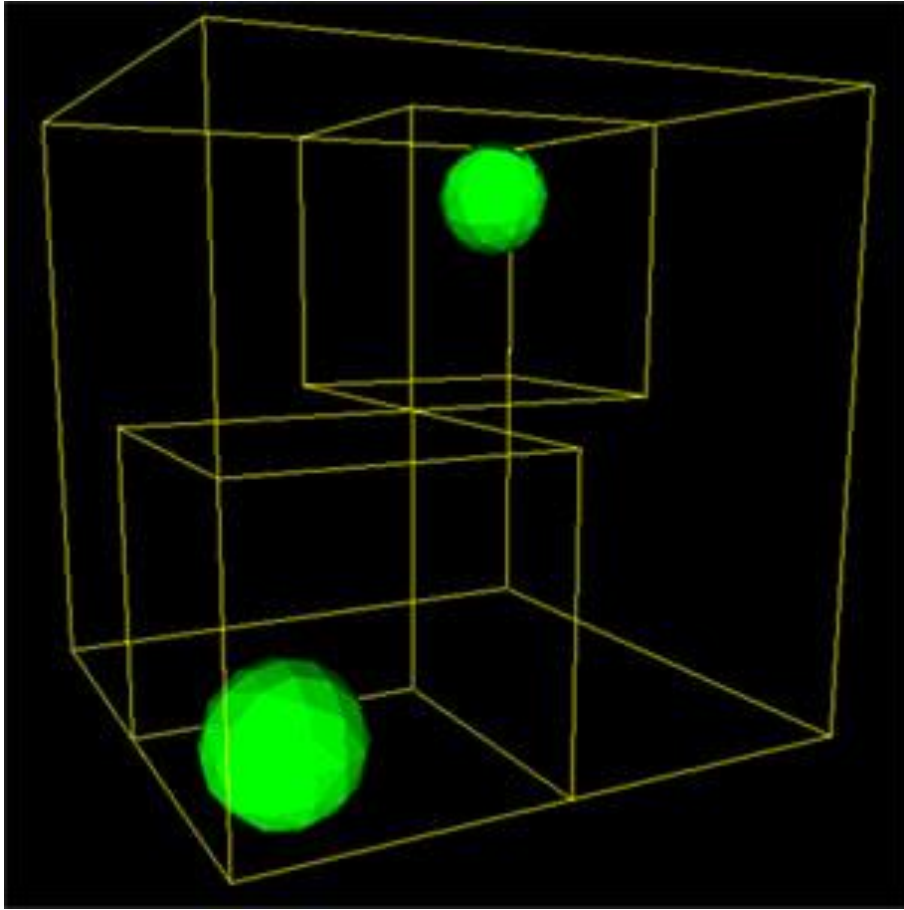
- **Preprocess**

- Find the bounding box
- Determine grid resolution
- Place a shape in a cell if its bounding box overlaps the cell
- Check that if the shape overlaps the cell

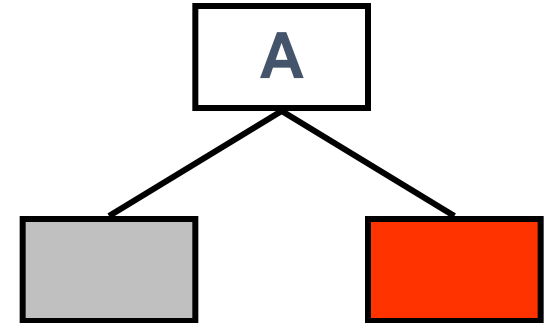
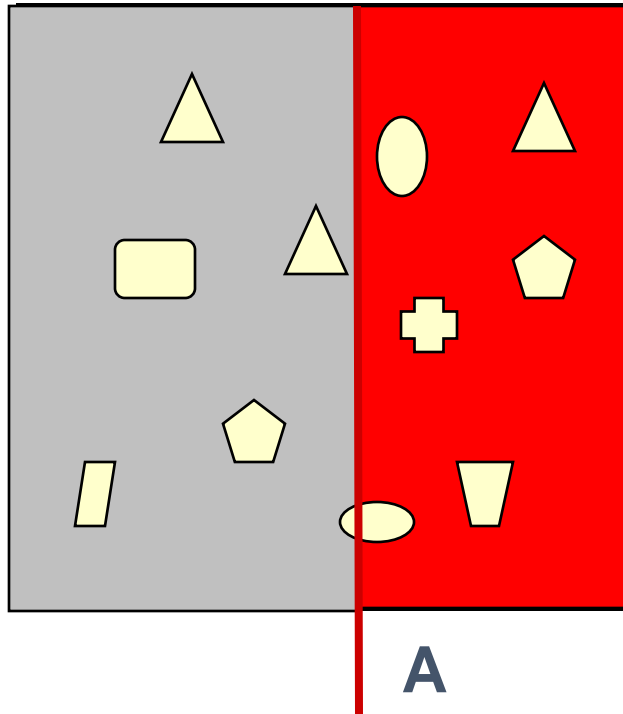
- **Rendering**

- Use 3D-DDA to traverse the grid

Octree

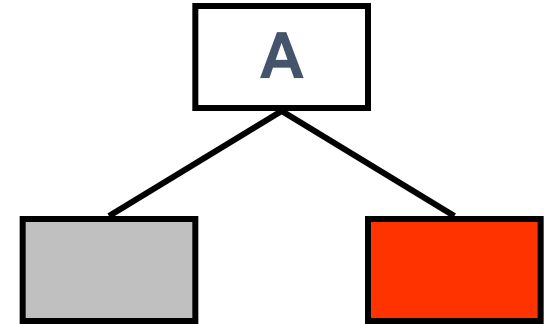
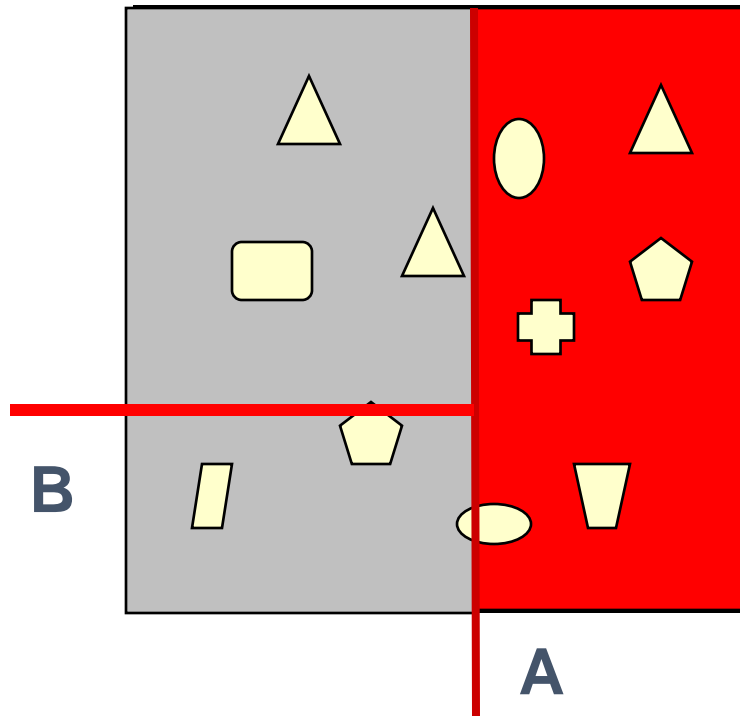


Kd Tree

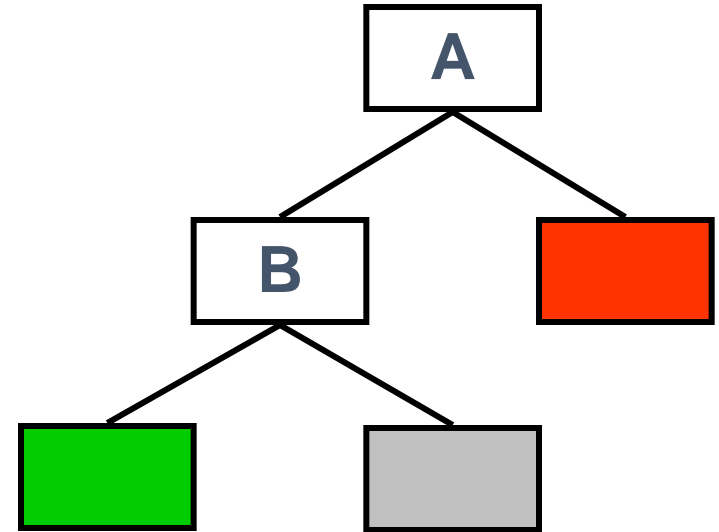
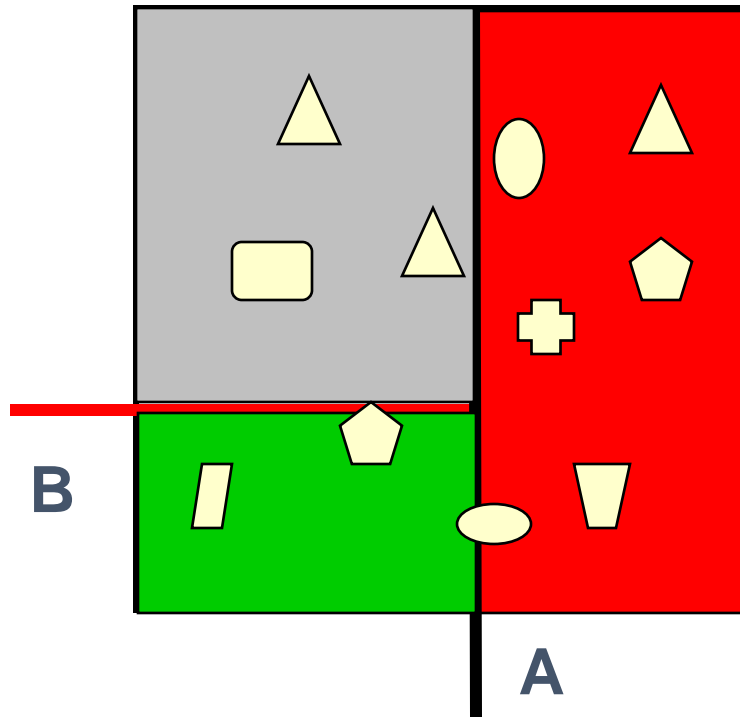


Leaf nodes correspond to unique regions in space

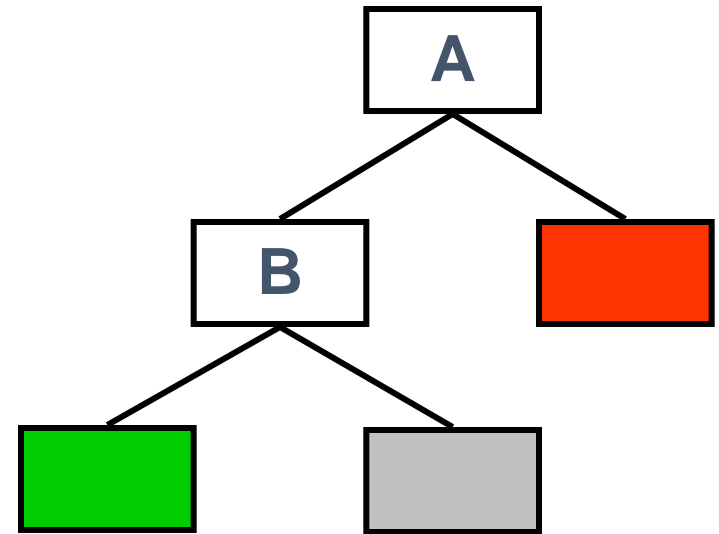
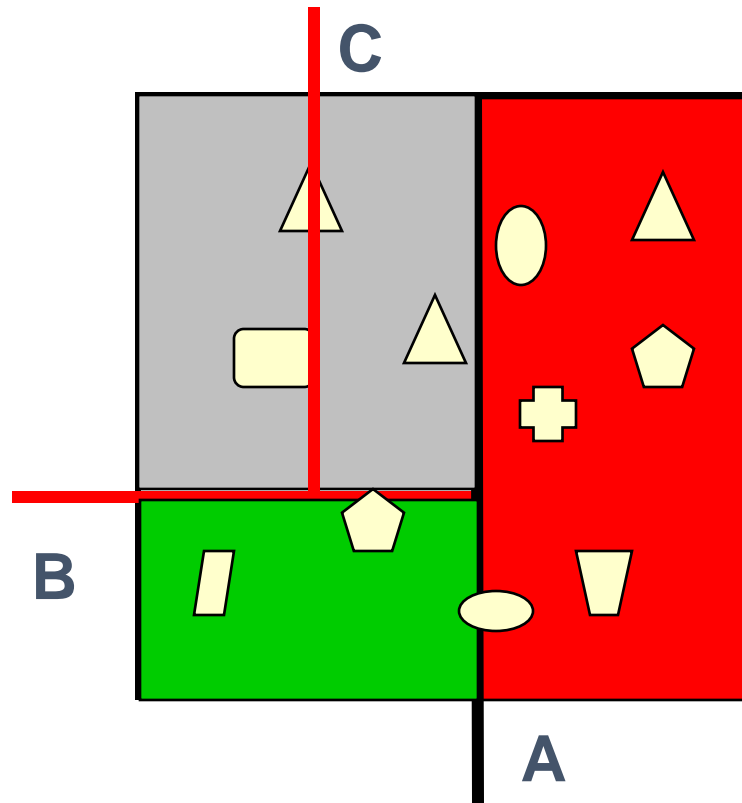
Kd Tree



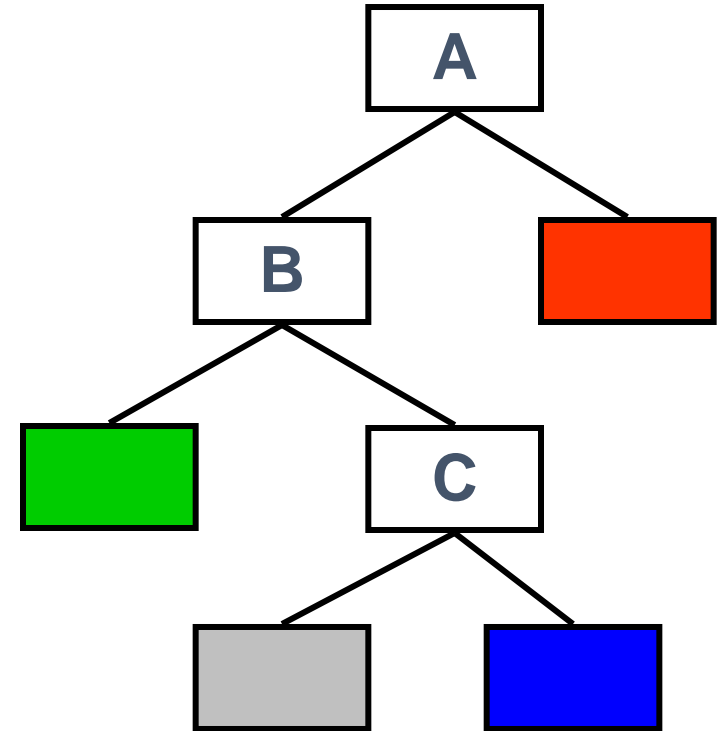
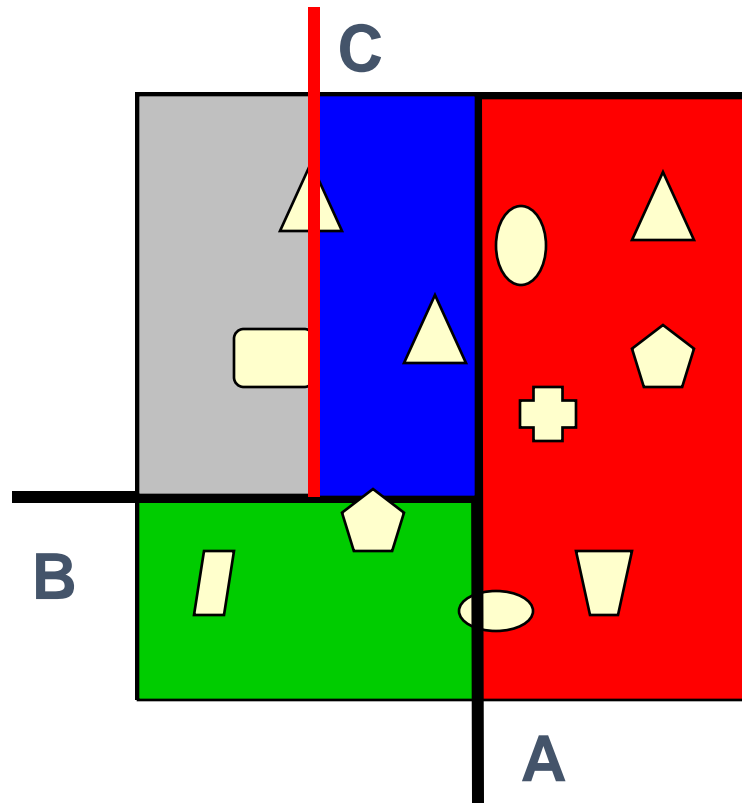
Kd Tree



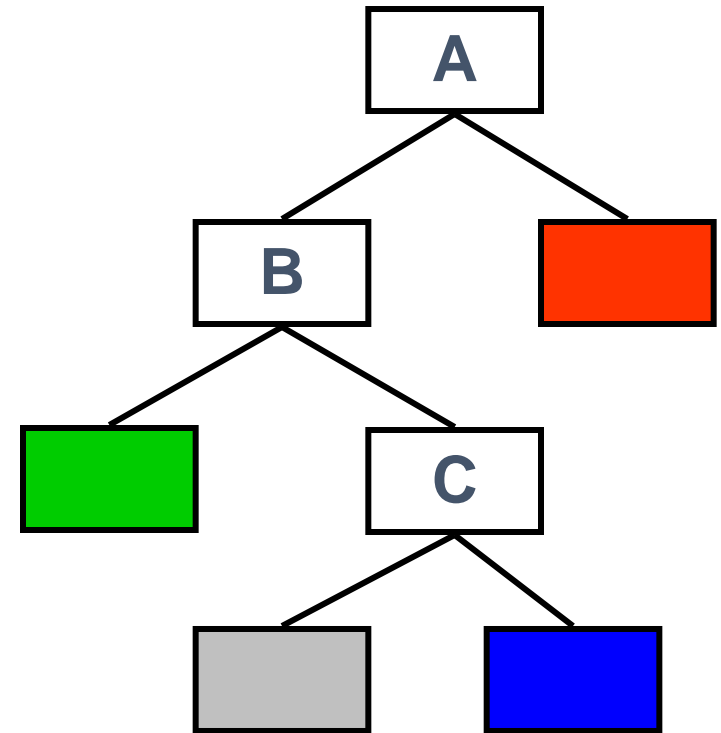
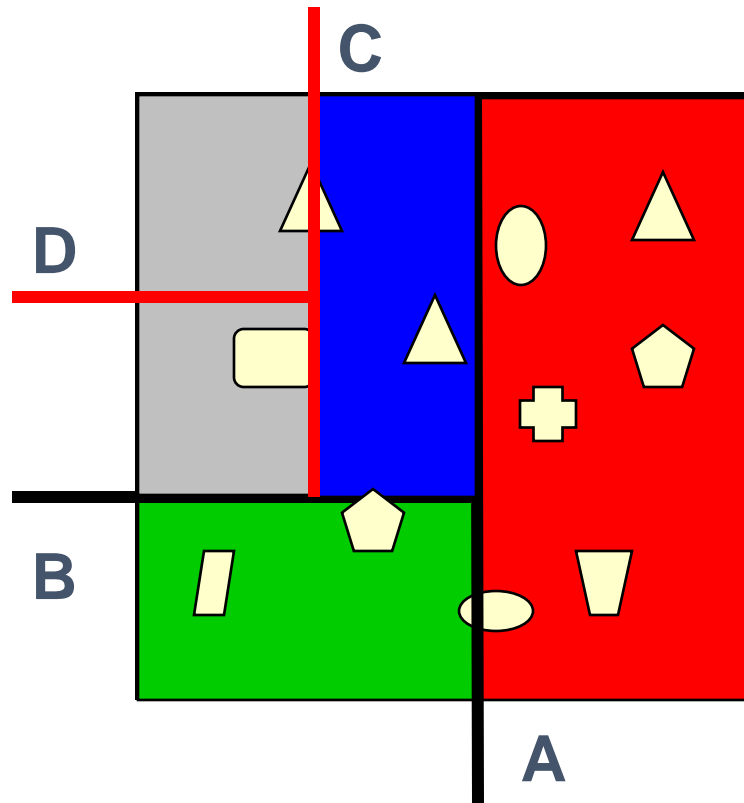
Kd Tree



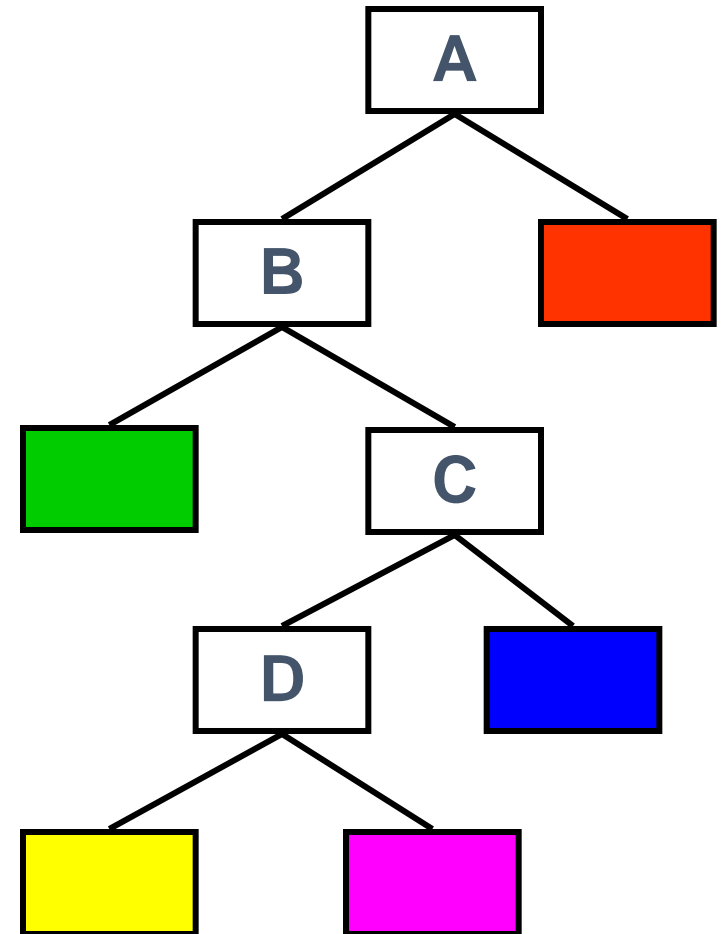
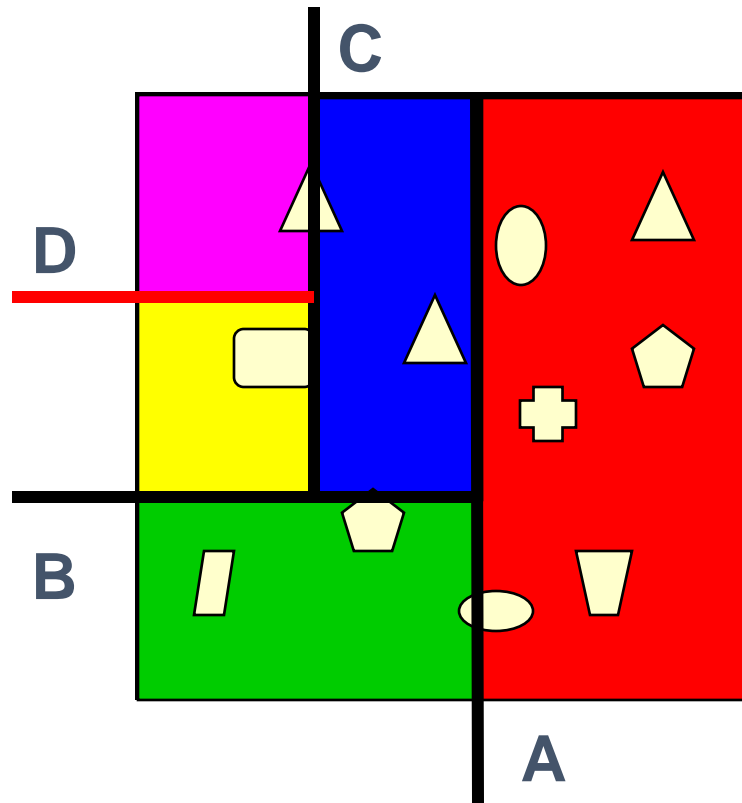
Kd Tree



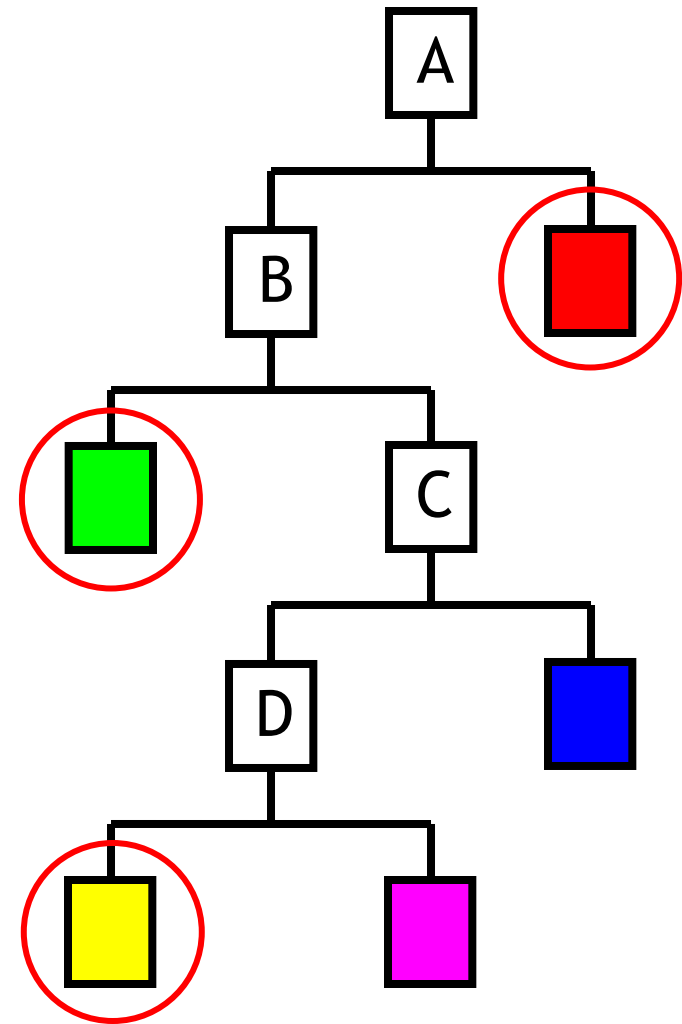
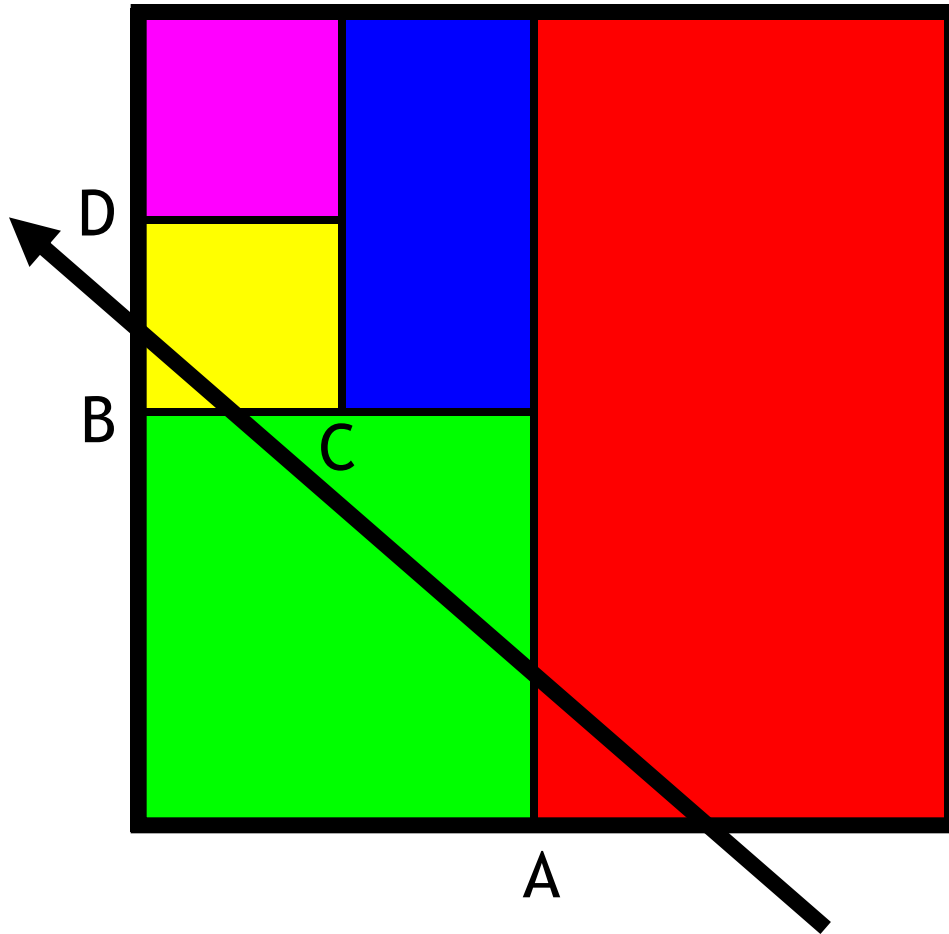
Kd Tree



Kd Tree



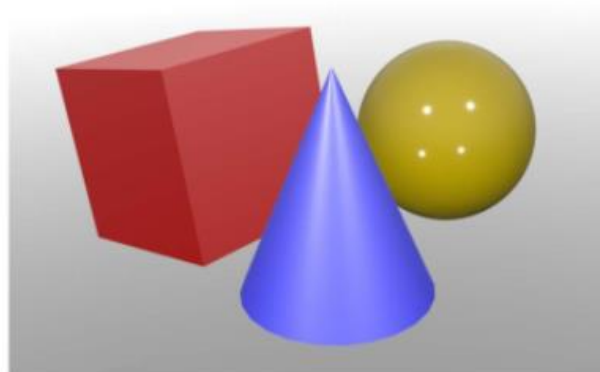
Kd Tree



Ray Tracing v.s. Rasterization

- Rasterization is **more friendly to hardware** and usually has higher parallelism
- But when we need to **interact with other triangles**, it is much more difficult to simulate effects such as reflection, refraction, shadows, and global illumination
 - Need specialized algorithms

RASTERIZATION



RAY TRACING



Ray Tracing v.s. Rasterization

- **Transparency**

- **Rasterization**

- Render the object in order (distant objects first) and blend with the previous result in the color buffer

- **Ray-tracing:**

- Trace a secondary (refracted) ray through the object's surface



Ray Tracing v.s. Rasterization

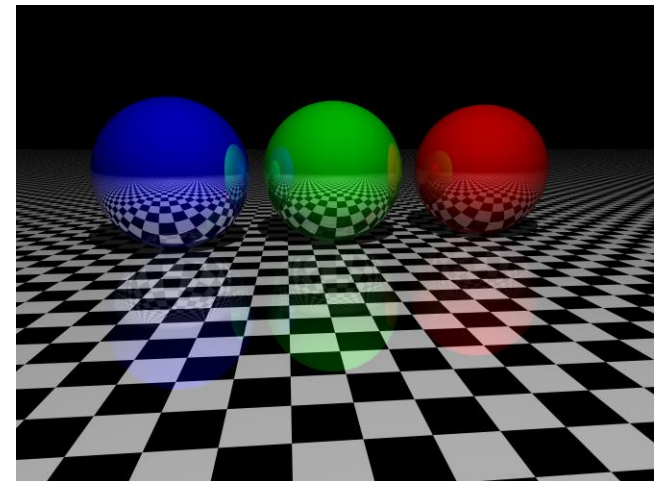
- **Reflection**

- **Rasterization**

- Render the scene into an environment map
 - Look up the environment map in the fragment shader

- **Ray-tracing:**

- Trace a secondary (reflected) ray from the object's surface



Ray Tracing v.s. Rasterization

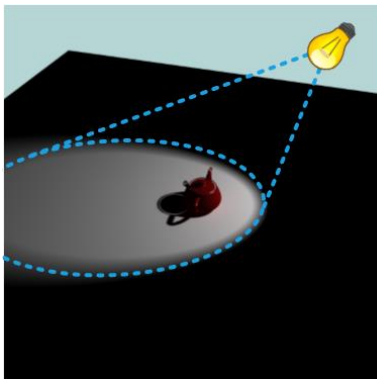
- **Shadow**

- **Rasterization**

- Render a **shadow map** to record the closest surface from each light
 - Look up the map to determine whether a surface point is in shadow or not in the second pass

- **Ray-tracing**

- Trace a shadow ray to see if the lighting direction is occluded



Ray Tracing v.s. Rasterization

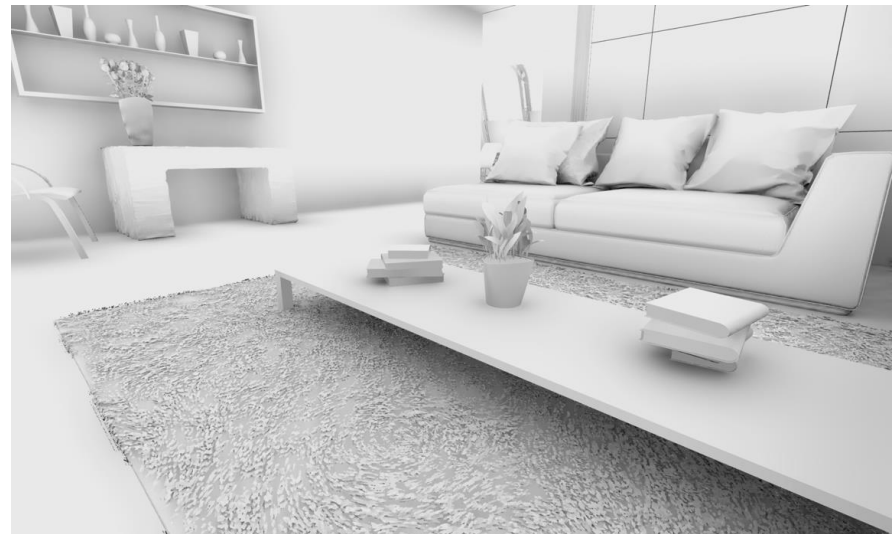
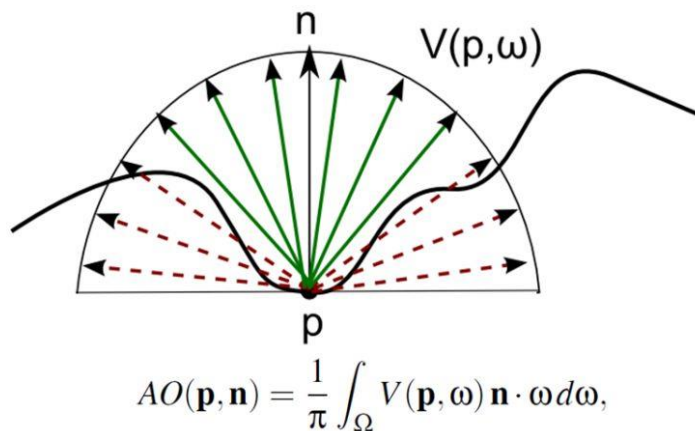
- **Ambient occlusion**

- **Rasterization**

- Use the **depth map** to find nearby occluders in screen space

- **Ray-tracing**

- Trace shadow rays to see if a direction is occluded



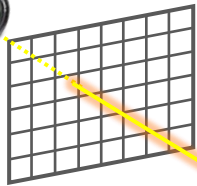
Ray Tracing v.s. Rasterization

- **Global illumination**

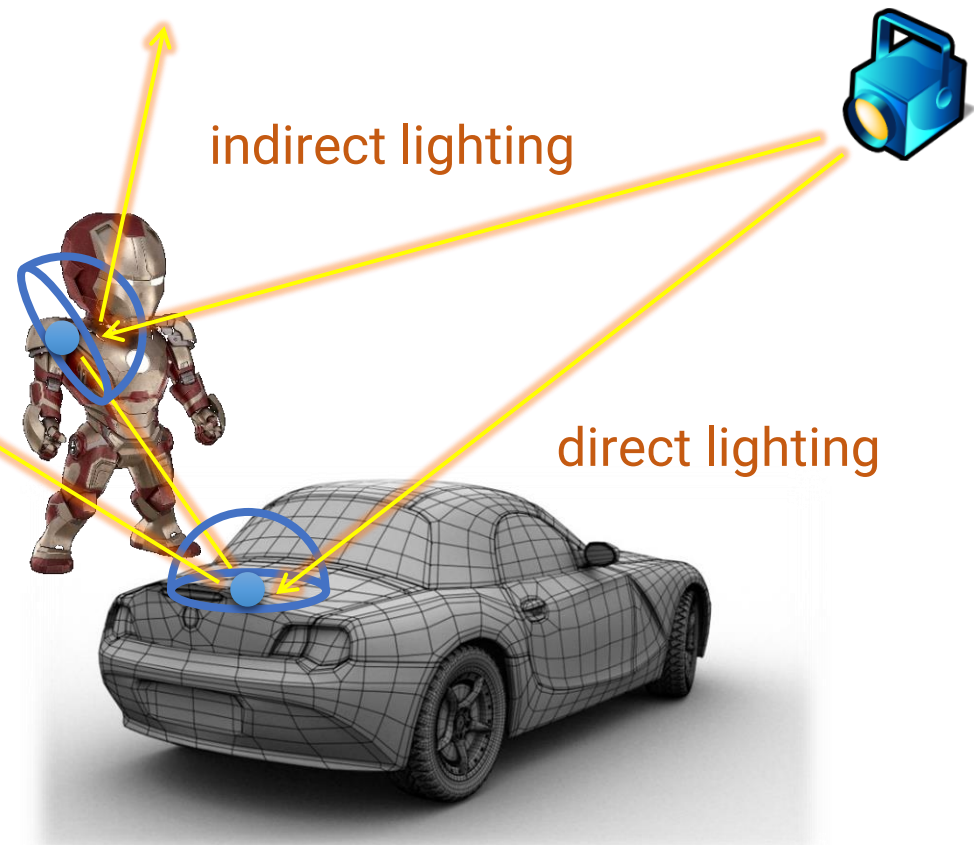
- **Ray-tracing**



virtual camera



virtual film



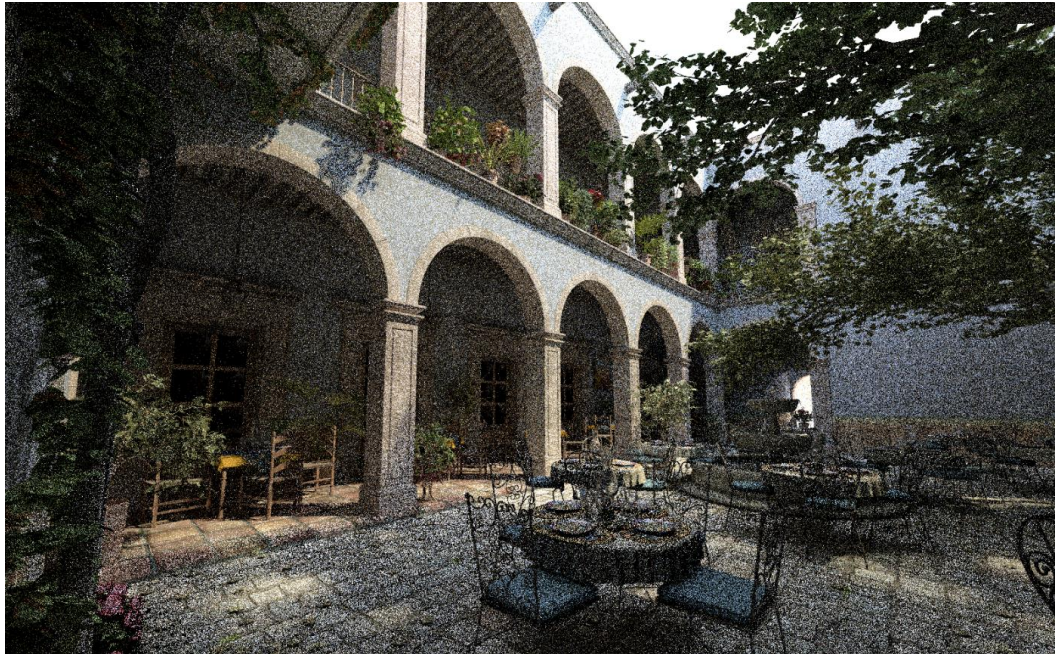
Ray Tracing v.s. Rasterization

- **Problems with ray tracing**

- Ray tracing is more general
- However, its simulator usually has a slow convergence rate and produces lots of noise when samples are not enough

- Solution

- More rays
- Filtering



Real-time Ray Tracing

- Recently some GPU ray tracers achieve real-time frame rates by incorporating filtering techniques
 - NVIDIA OptiX
 - <https://developer.nvidia.com/rtx/ray-tracing/optix>
 - Unreal Engine
 - <https://docs.unrealengine.com/5.1/en-US/hardware-ray-tracing-in-unreal-engine/>
 - DirectX
 - <https://microsoft.github.io/DirectX-Specs/d3d/Raytracing.html>
- It is believed to replace rasterization in the future
 - Not that sure now ...

Real-time Ray Tracing

- Unreal Engine Ray Tracing Demo
 - <https://www.youtube.com/watch?v=J3ue35ago3Y>



