# CS380: Computer Graphics
# Ray Tracing

Sung-Eui Yoon
(윤성의)

**Course URL:**
**http://sglab.kaist.ac.kr/~sungeui/CG/**

# Class Objectives

- **Understand overall algorithm of recursive ray tracing**
  - **Ray generations**
  - **Intersection tests and acceleration methods**
  - **Basic sampling methods**

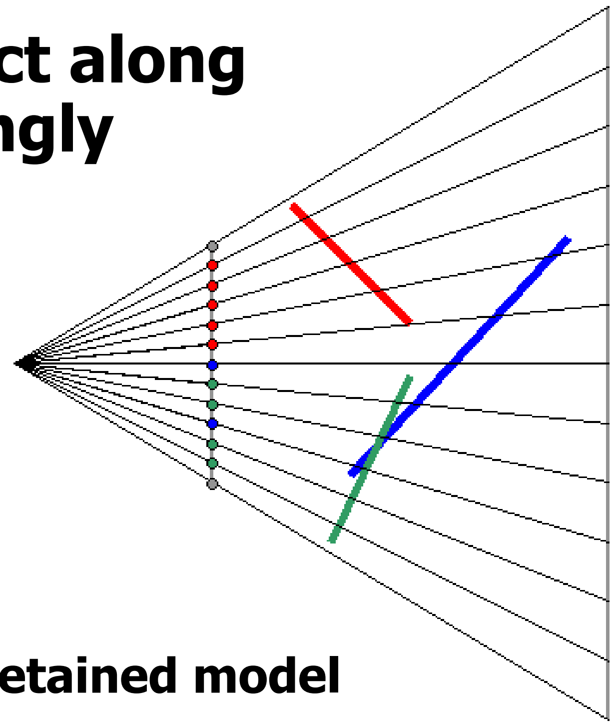
- **Related chapter**
  - **Part II, Ray Tracing**

# Various Visibility Algorithm

- Scan-line algorithm; briefly touched before
- Z-buffer
- Ray casting, etc.

# Ray Casting

- **For each pixel, find closest object along the ray and shade pixel accordingly**

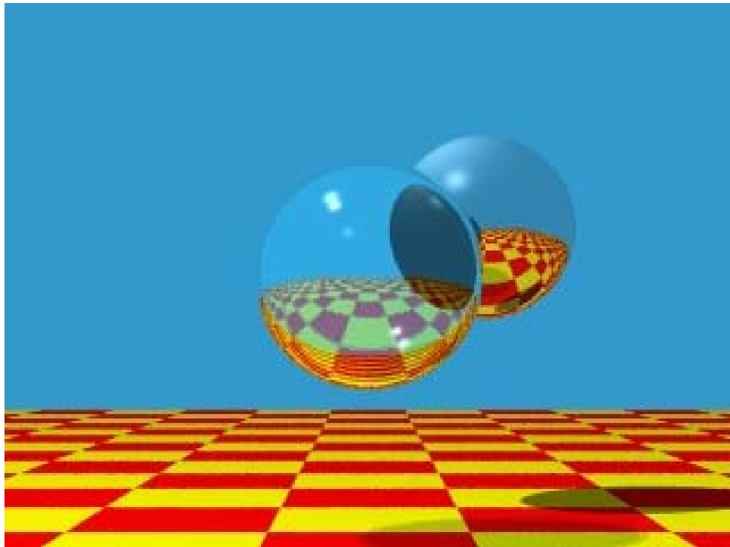


- **Advantages**
  - **Conceptually simple**
  - **Can be extended to handle global illumination effects**
- **Disadvantages**
  - **Renderer must have access to entire retained model**
  - **Hard to map to special-purpose hardware**
  - **Less efficient than rasterization in terms of utilizing spatial coherence**

# Recursive Ray Casting

- **Ray casting generally dismissed early on because of aforementioned problems**

- **Gained popularity in when Turner Whitted (1980) showed this image**
  - ***Show recursive*** **ray casting could be used for global illumination effects**

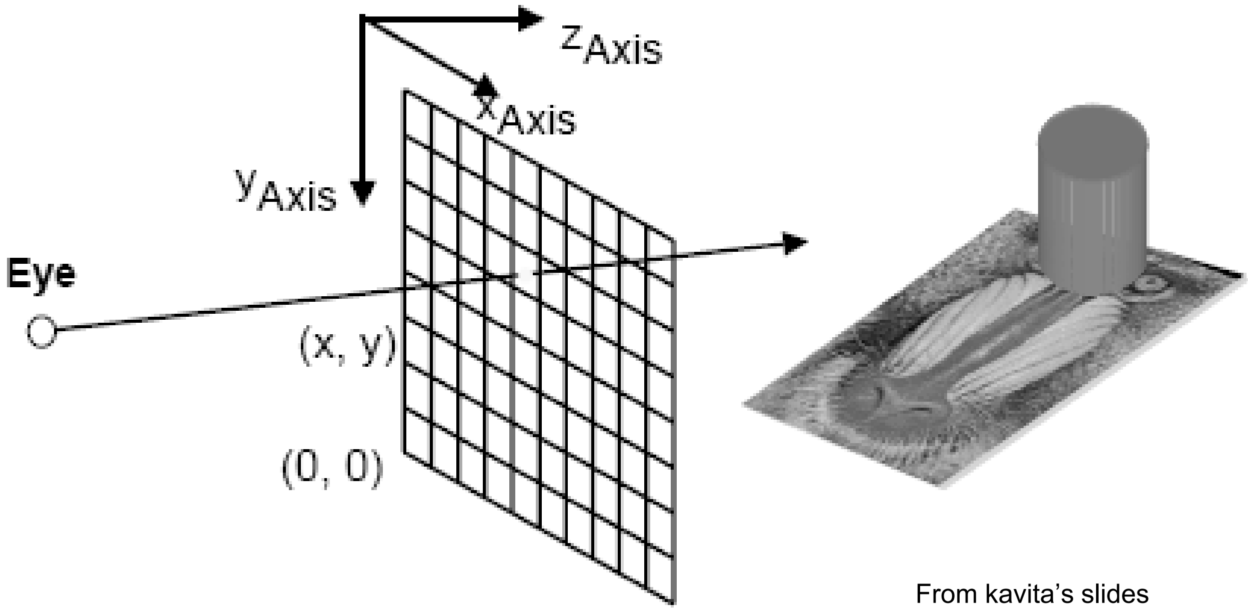


# Ray Casting and Ray Tracing

- Trace rays from eye into scene
  - Backward ray tracing
- Ray casting used to compute visibility at the eye
- Perform ray tracing for arbitrary rays needed for shading
  - Reflections
  - Refraction and transparency
  - Shadows

# Basic Algorithms of Ray Tracing

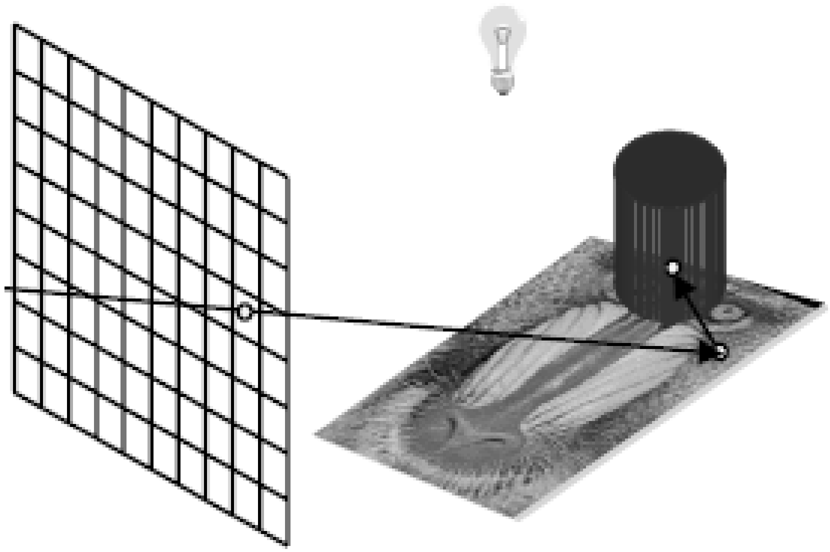
- **Rays are cast from the eye point through each pixel in the image**

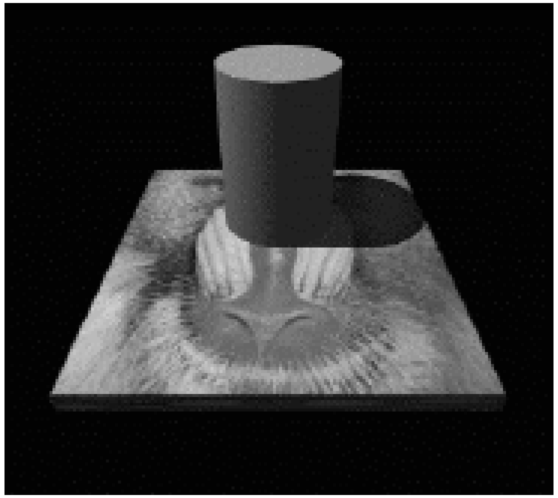



From kavita's slides

# Shadows

- **Cast ray from the intersection point to each light source**
  - **Shadow rays**

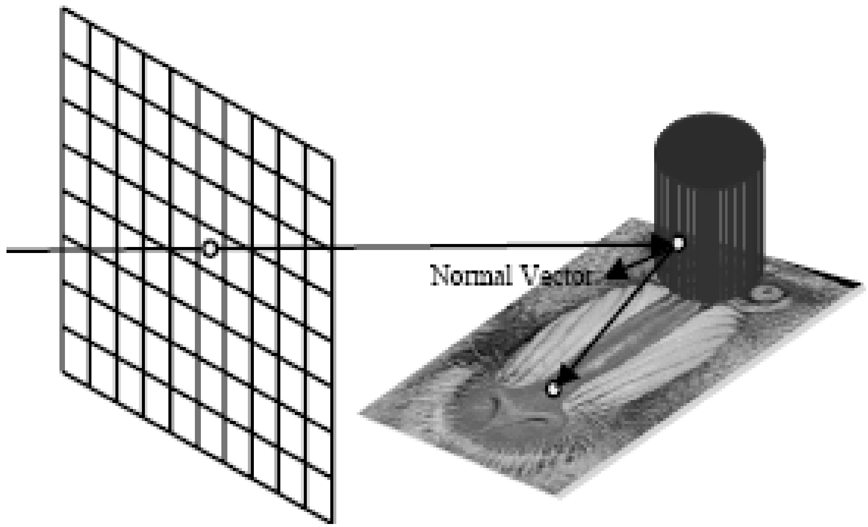


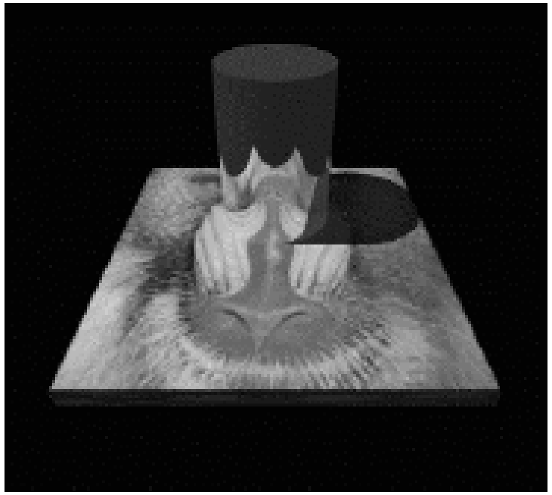


From kavita's slides

# Reflections

- **If object specular, cast secondary reflected rays**

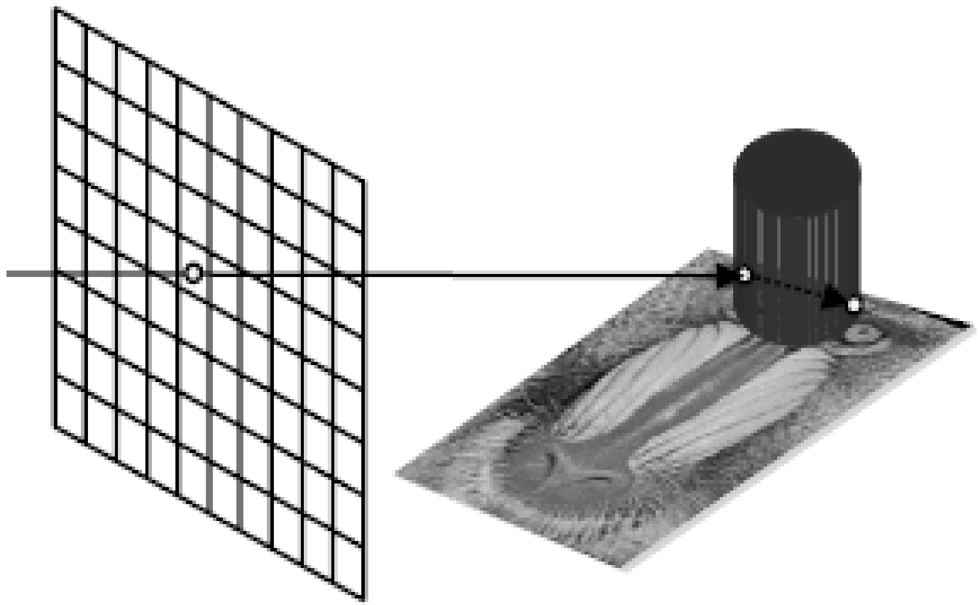


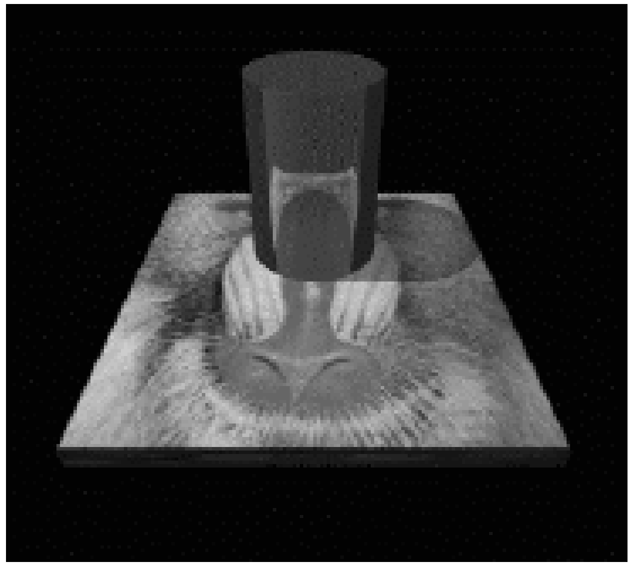



From kavita's slides

# Refractions

- **If object tranparent, cast secondary refracted rays**





From kavita's slides

# An Improved Illumination Model [Whitted 80]

- **Phong model**

$$I_r = \sum_{j=1}^{numLights} \left( k_a^j I_a^j + k_d^j I_d^j (\hat{N} \bullet \hat{L}_j) + k_s^j I_s^j (\hat{V} \bullet \hat{R})^{n_s} \right)$$

- **Whitted model**

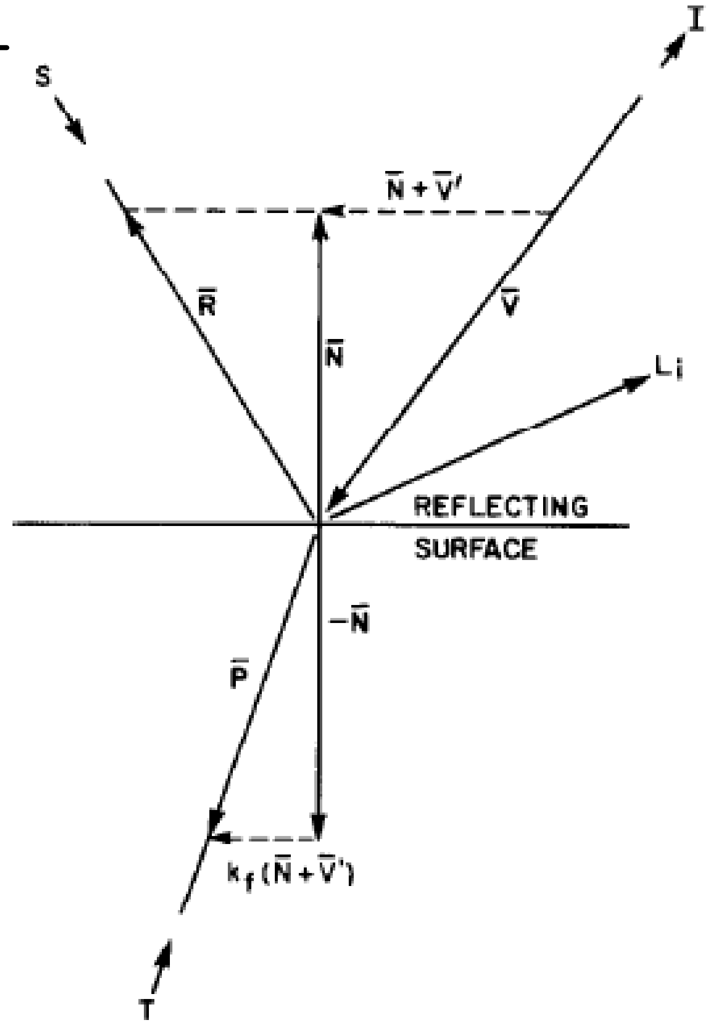
$$I_r = \sum_{j=1}^{num\_Visible\_Lights} \left( k_a^j I_a^j + k_d^j I_d^j (\hat{N} \bullet \hat{L}_j) \right) + k_s S + k_t T$$

  - **S and T are intensity of light from reflection and transmission rays**
  - **Ks and Kt are specular and transmission coefficient**

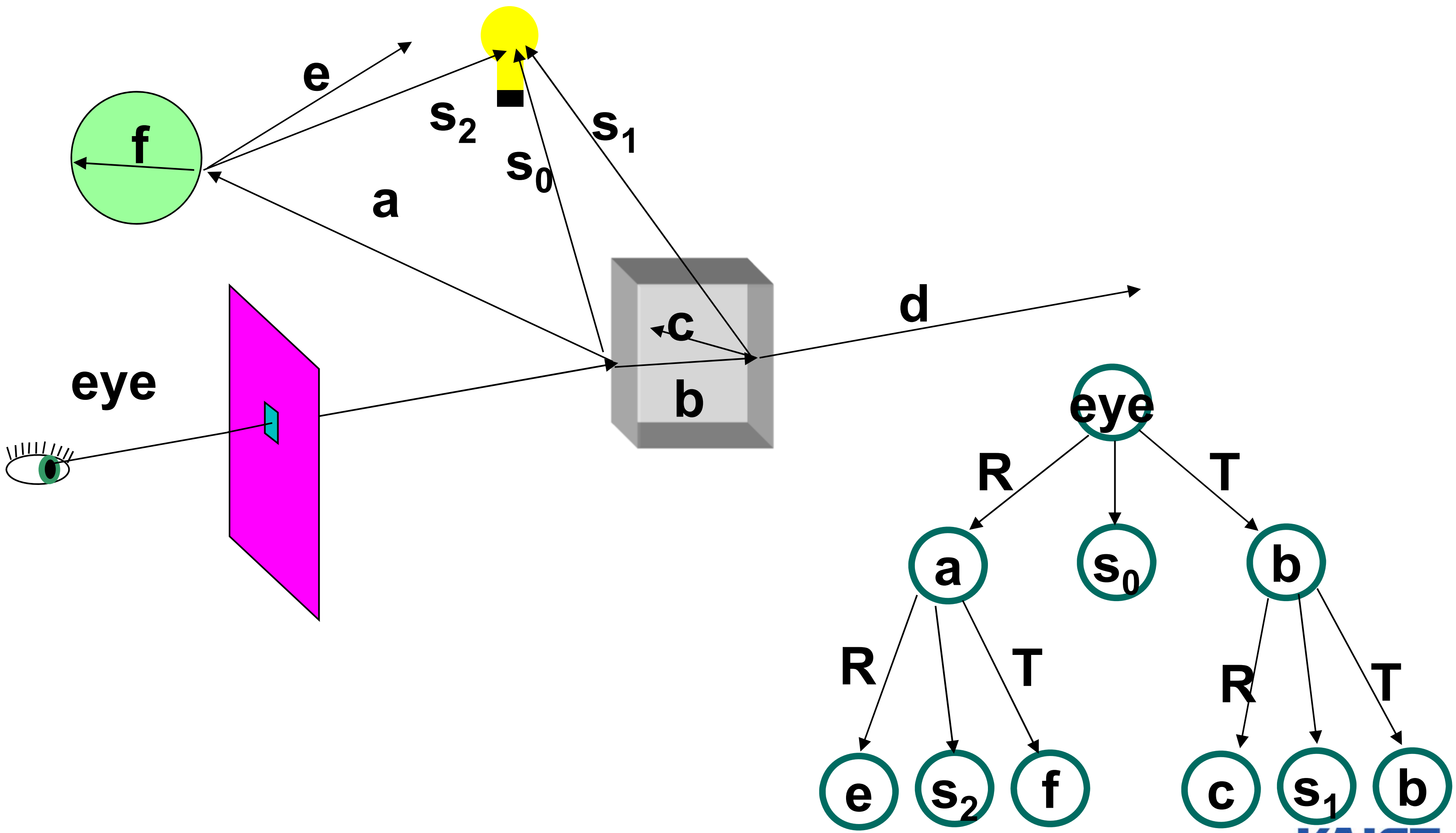
# An Improved Illumination Model [Whitted 80]

$$I_r = \sum_{j=1}^{numLights} \left(k_a^j I_a^j + k_d^j I_d^j (\hat{N} \bullet \hat{L}_j)\right) + k_s S + k_t T$$

**Computing reflection and transmitted/refracted rays is based on Snell's law**



# Ray Tree



# Overall Algorithm of Ray Tracing

- **Per each pixel, compute a ray, R**

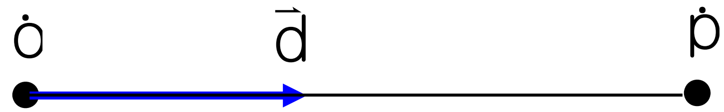
**Def function RayTracing (R)**

- **Compute an intersection against objects**
- **If no hit,**
  - **Return the background color**
- **Otherwise,**
  - **Compute shading, c**
  - **General secondary ray, R′**
  - **Perform c′ = RayTracing (R′)**
  - **Return c+c′**

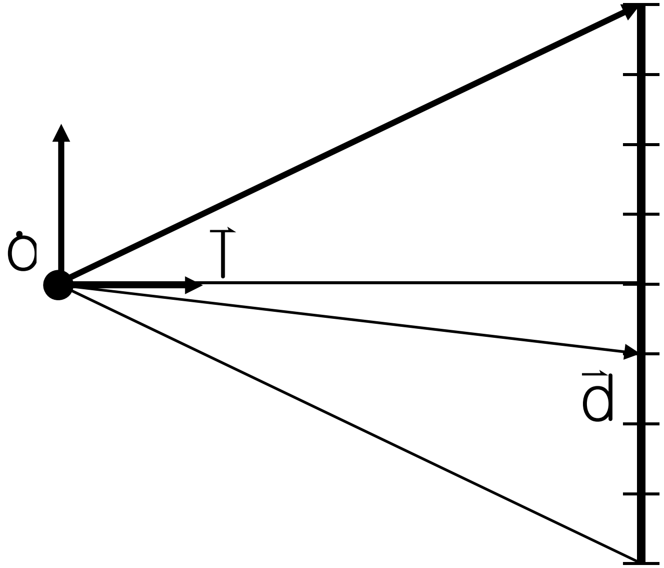
# Ray Representation

- **We need to compute the first surface hit along a ray**
  - **Represent ray with origin and direction**
  - **Compute intersections of objects with ray**
  - **Return the closest object**

$$\dot{p}(t) = \dot{o} + t\vec{d}$$



# Generating Primary Rays

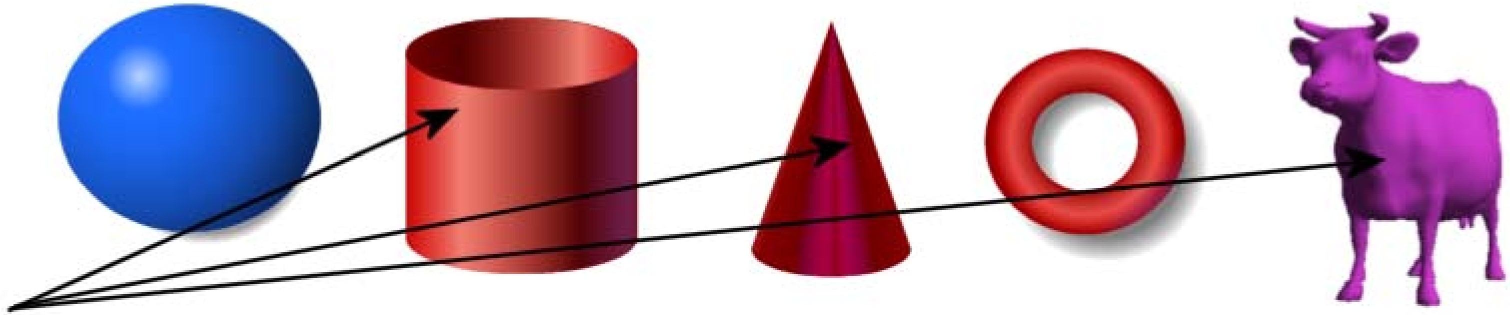


# Generating Secondary Rays

- **The origin is the intersection point $p_0$**
- **Direction depends on the type of ray**
  - **Shadow rays – use direction to the light source**
  - **Reflection rays – use incoming direction and normal to compute reflection direction**
  - **Transparency/refraction – use snell's law**

# Intersection Tests

Go through all of the objects in the scene to determine the one closest to the origin of



Strategy: Solve of the intersection of the Ray with a mathematical description of the object

# Simple Strategy

- **Parametric ray equation**
  - **Gives all points along the ray as a function of the parameter**

$$\dot{p}(t) = \dot{o} + t\vec{d}$$

- **Implicit surface equation**
  - **Describes all points on the surface as the zero set of a function**

$$f(\dot{p}) = 0$$

- **Substitute ray equation into surface function and solve for t**

$$f(\dot{o} + t\vec{d}) = 0$$

# Ray-Plane Intersection

- **Implicit equation of a plane:**

$$\vec{n} \cdot \dot{p} - d = 0$$

- **Substitute ray equation:**

$$\vec{n} \cdot (\dot{o} + t\,\vec{d}) - d = 0$$

- **Solve for t:**

$$t\,(\vec{n} \cdot \vec{d}) = d - \vec{n} \cdot \dot{o}$$

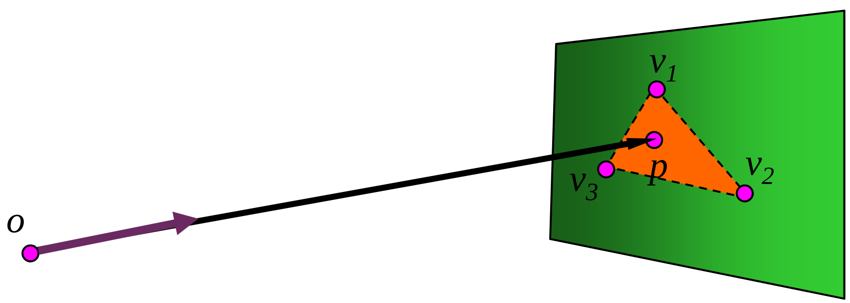
$$t = \frac{d - \vec{n} \cdot \dot{o}}{\vec{n} \cdot \vec{d}}$$



KAIST

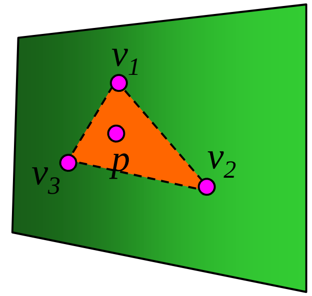
# Generalizing to Triangles

- Find of the point of intersection on the plane containing the triangle
- Determine if the point is inside the triangle
  - Barycentric coordinate method
  - Many other methods

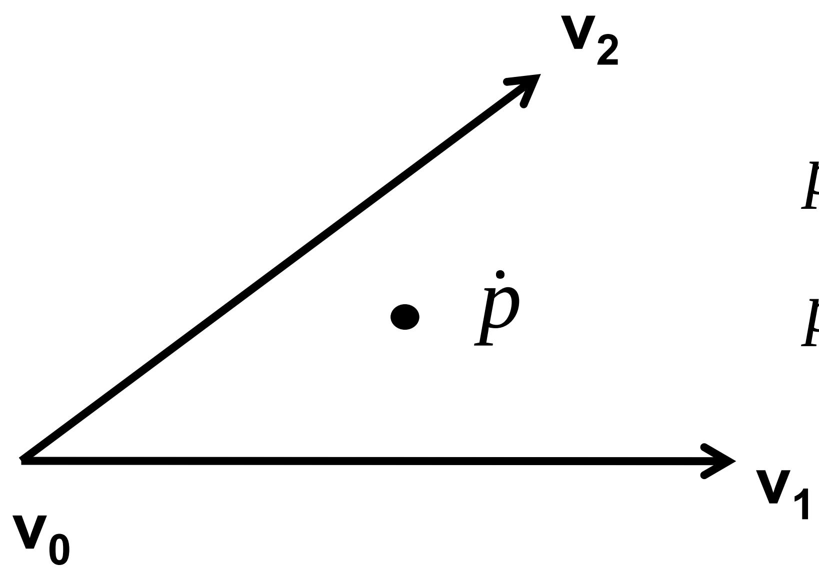


# Barycentric Coordinates

- **Points in a triangle have positive barycentric coordinates:**




$$\dot{p} = \alpha\dot{v}_0 + \beta\dot{v}_1 + \gamma\dot{v}_2$$ **,where** $$\alpha + \beta + \gamma = 1$$



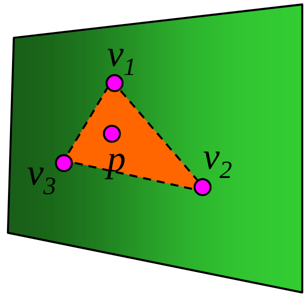

$$\dot{p} = \dot{v}_0 + \beta(\dot{v}_1 - \dot{v}_0) + \gamma(\dot{v}_2 - \dot{v}_0)$$

$$\dot{p} = (1 - \beta - \gamma)\dot{v}_0 + \beta\dot{v}_1 + \gamma\dot{v}_2$$

# Barycentric Coordinates

- **Points in a triangle have positive barycentric coordinates:**

$$\dot{p} = \alpha\dot{v}_0 + \beta\dot{v}_1 + \gamma\dot{v}_2 \quad \text{,where} \quad \alpha + \beta + \gamma = 1$$




- **Benefits:**
  - **Barycentric coordinates can be used for interpolating vertex parameters (e.g., normals, colors, texture coordinates, etc)**

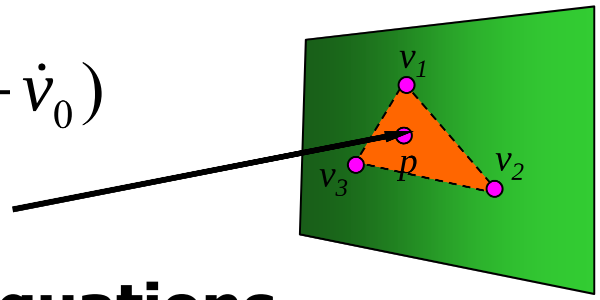


KAIST

# Ray-Triangle Intersection

- **A point in a ray intersects with a triangle**

$$\dot{p}(t) = \dot{v}_0 + \beta(\dot{v}_1 - \dot{v}_0) + \gamma(\dot{v}_2 - \dot{v}_0)$$

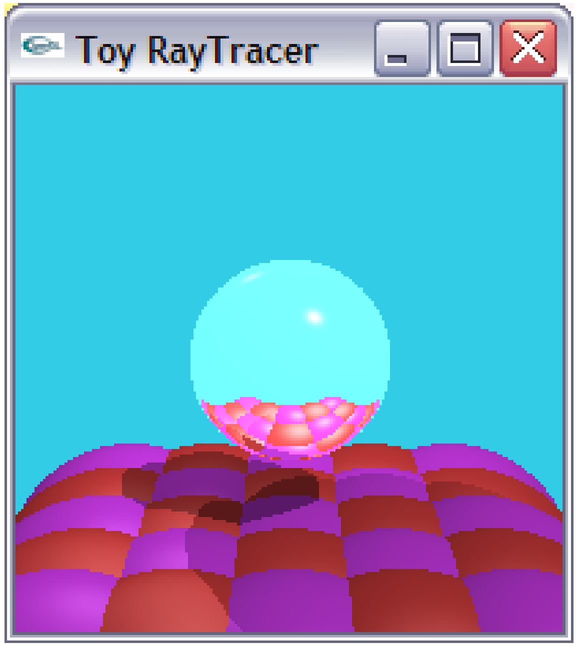



  - **Three unknowns, but three equations**
  - **Compute the point based on t**
  - **Then, check whether the point is on the triangle**

# Pros and Cons of Ray Tracing

**Advantages of Ray Tracing:**

- **Very simple design**
- **Improved realism over the graphics pipeline**

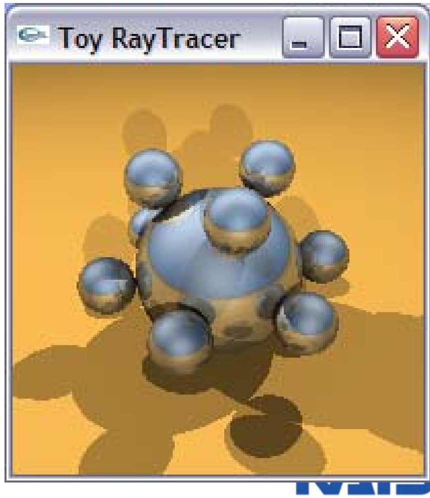



**Disadvantages:**

- **Very slow per pixel calculations**
- **Only approximates full global illumination**
- **Hard to accelerate with special-purpose H/W**

# Acceleration Methods

- **Rendering time for a ray tracer depends on the number of ray intersection tests per pixel**
  - **The number of pixels X the number of primitives in the scene**
- **Early efforts focused on accelerating the ray-object intersection tests**
- **More advanced methods required to make ray tracing practical**
  - **Bounding volume hierarchies**
  - **Spatial subdivision**



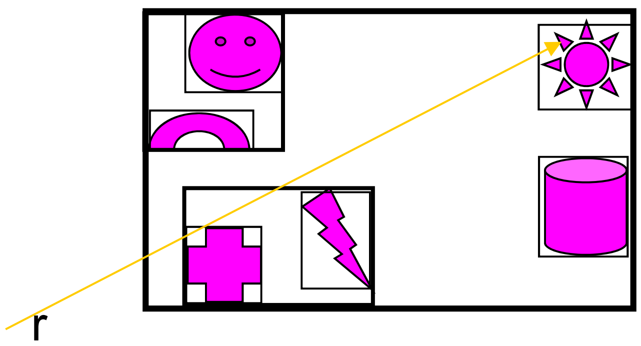
# Bounding Volumes

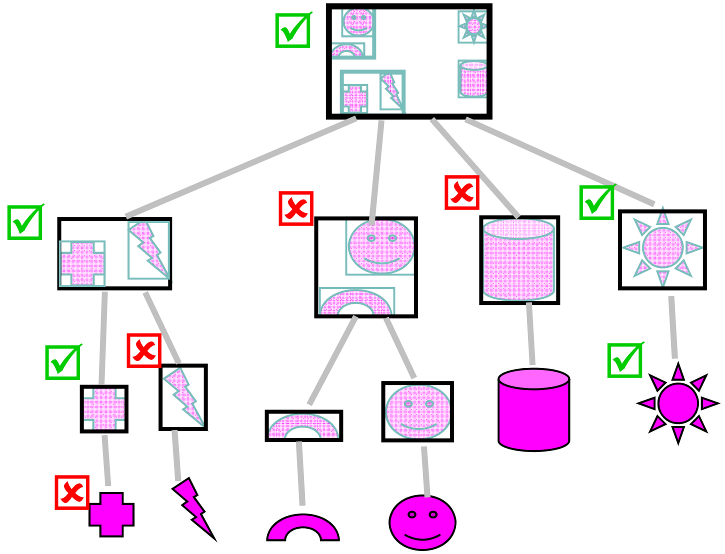
- **Enclose complex objects within a simple-to-intersect objects**
  - **If the ray does not intersect the simple object then its contents can be ignored**
  - **The likelihood that it will strike the object depends on how tightly the volume surrounds the object.**


**Potentially tighter fit,**
**but with higher computation**

# Hierarchical Bounding Volumes

- Organize bounding volumes as a tree
- Each ray starts with the root BV of the tree and traverses down through the tree

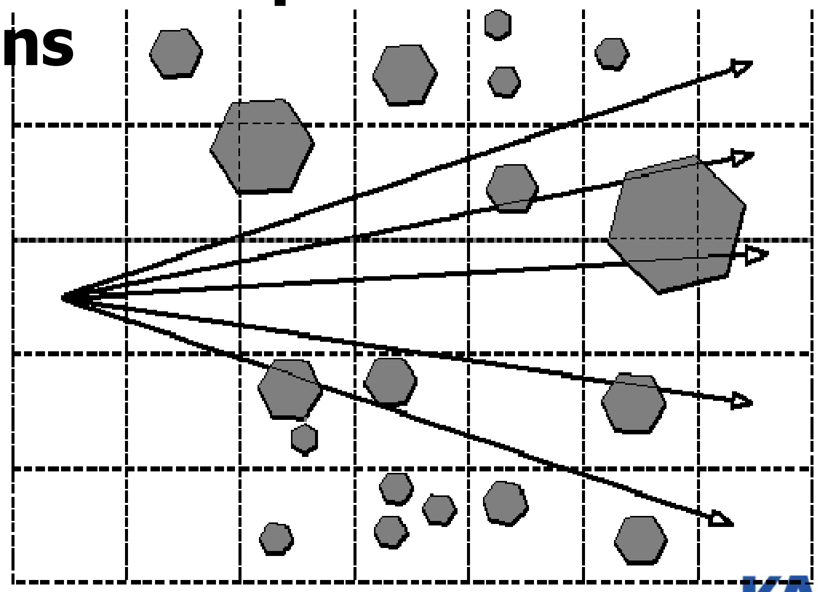


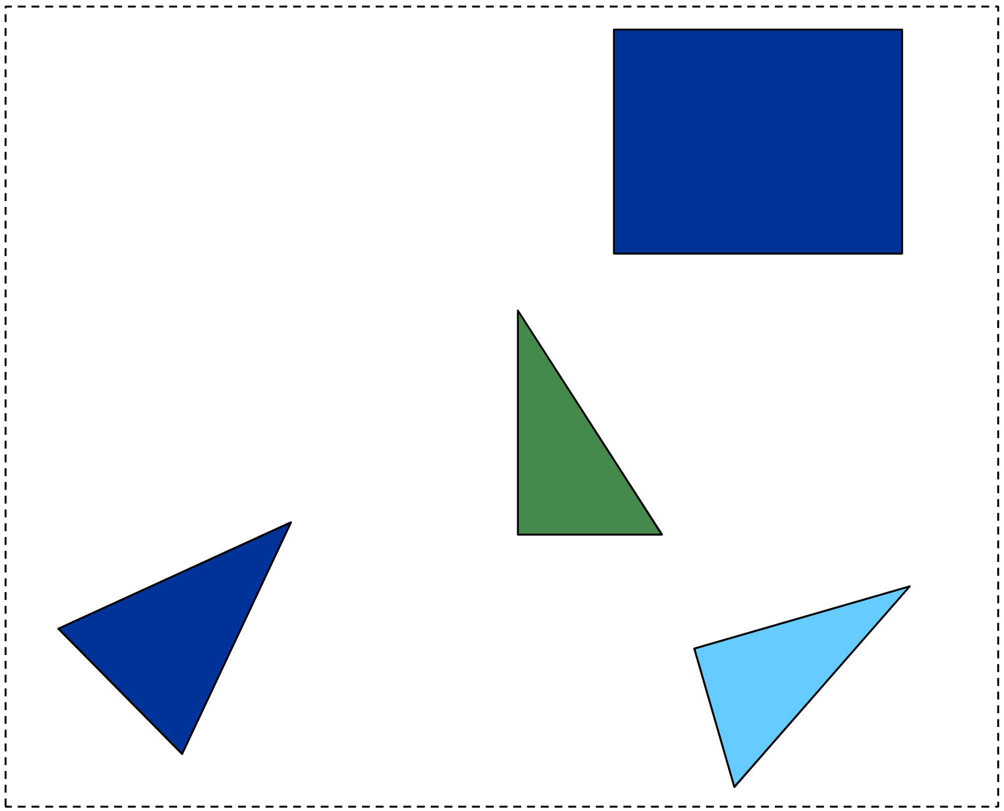
# Spatial Subdivision

**Idea: Divide space in to subregions**

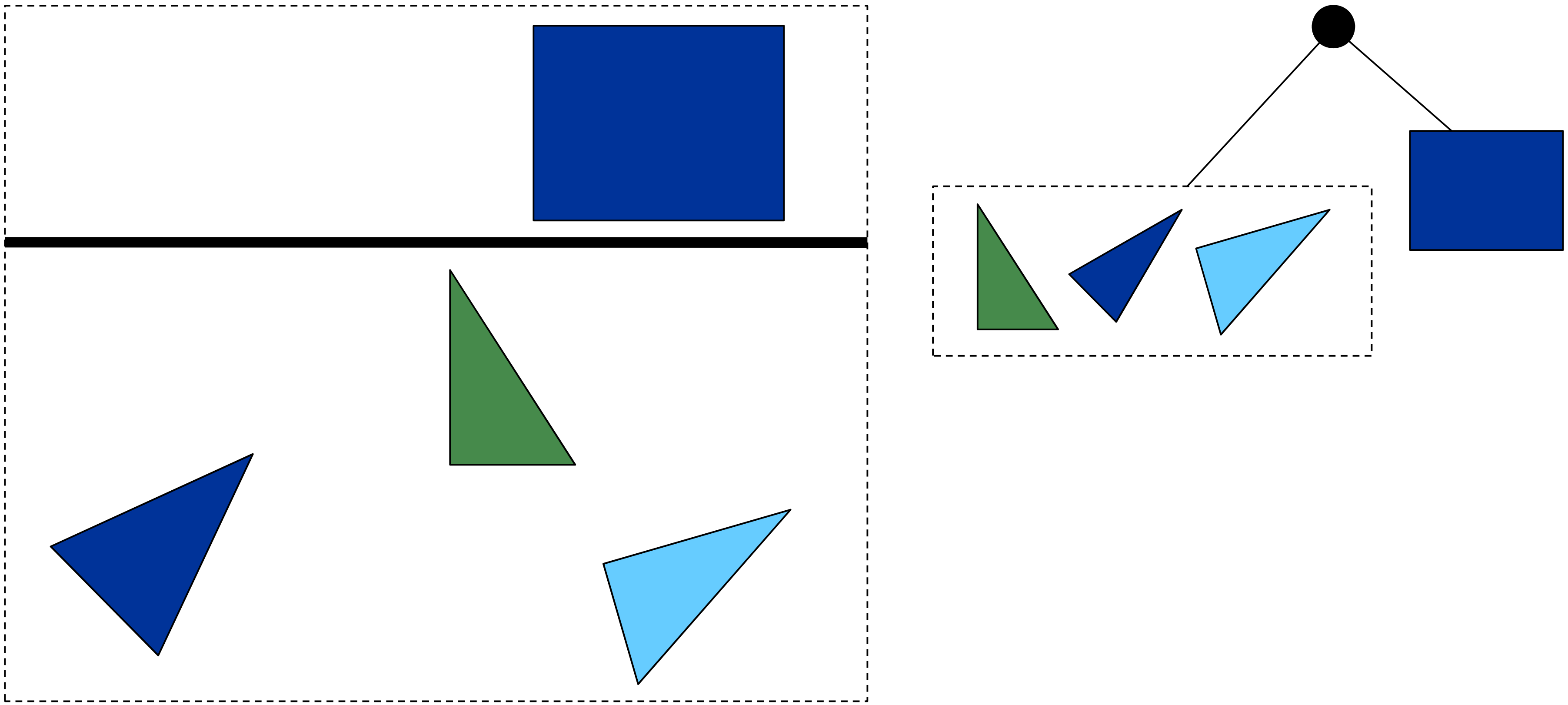
- **Place objects within a subregion into a list**
- **Only traverse the lists of subregions that the ray passes through**
- **"Mailboxing" used to avoid multiple test with objects in multiple regions**
- **Many types**
  - **Regular grid**
  - **Octree**
  - **BSP tree**
  - **kd-tree**



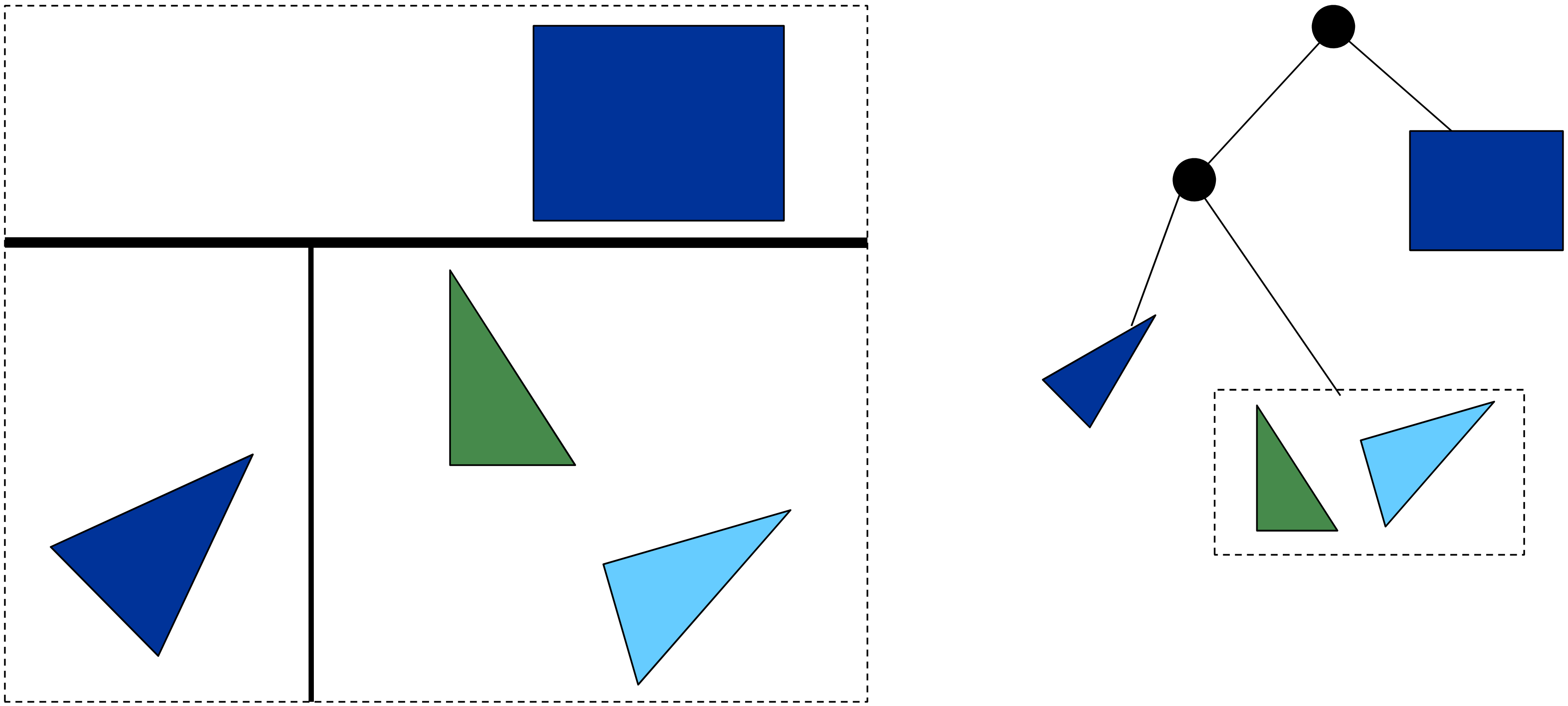
# Kd-tree: Example



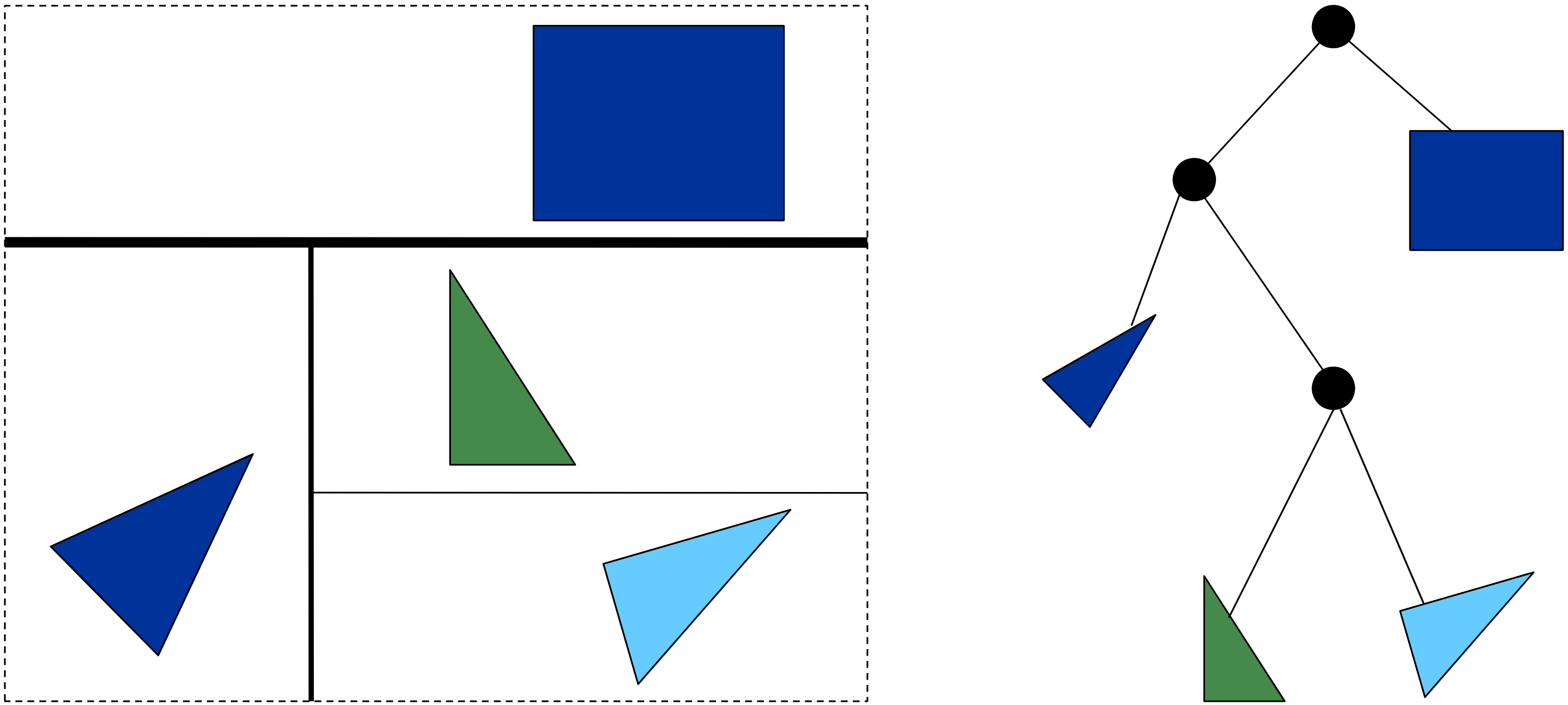
# Kd-tree: Example



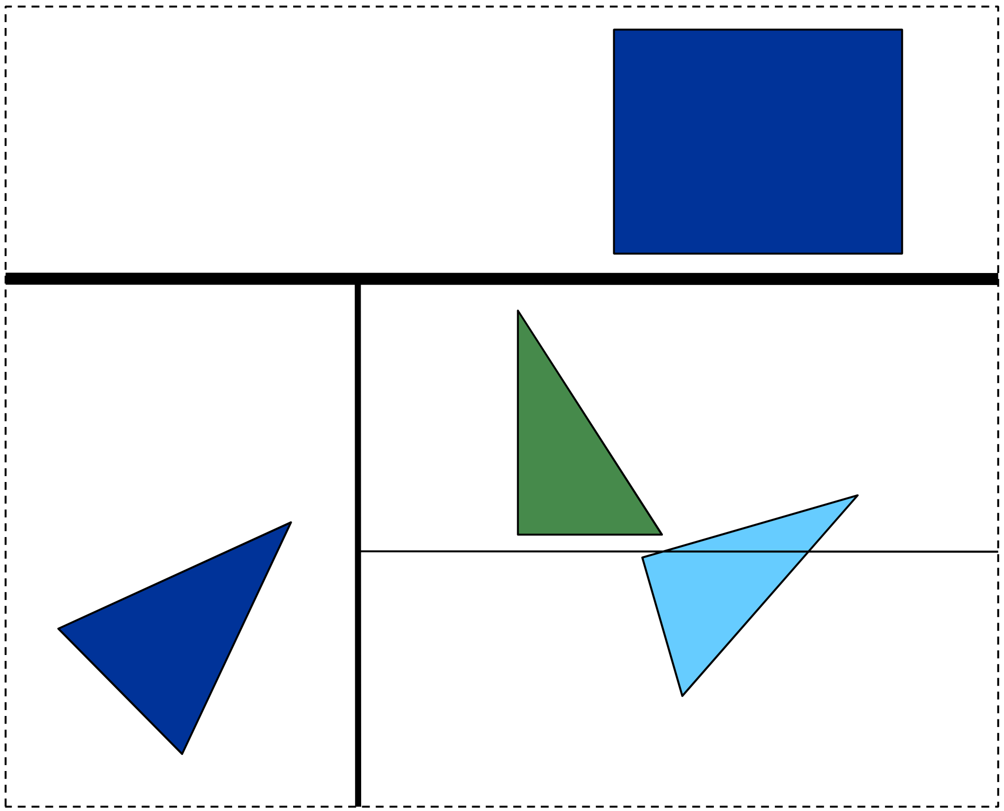
# Kd-tree: Example



# Example



# Kd-tree: Example

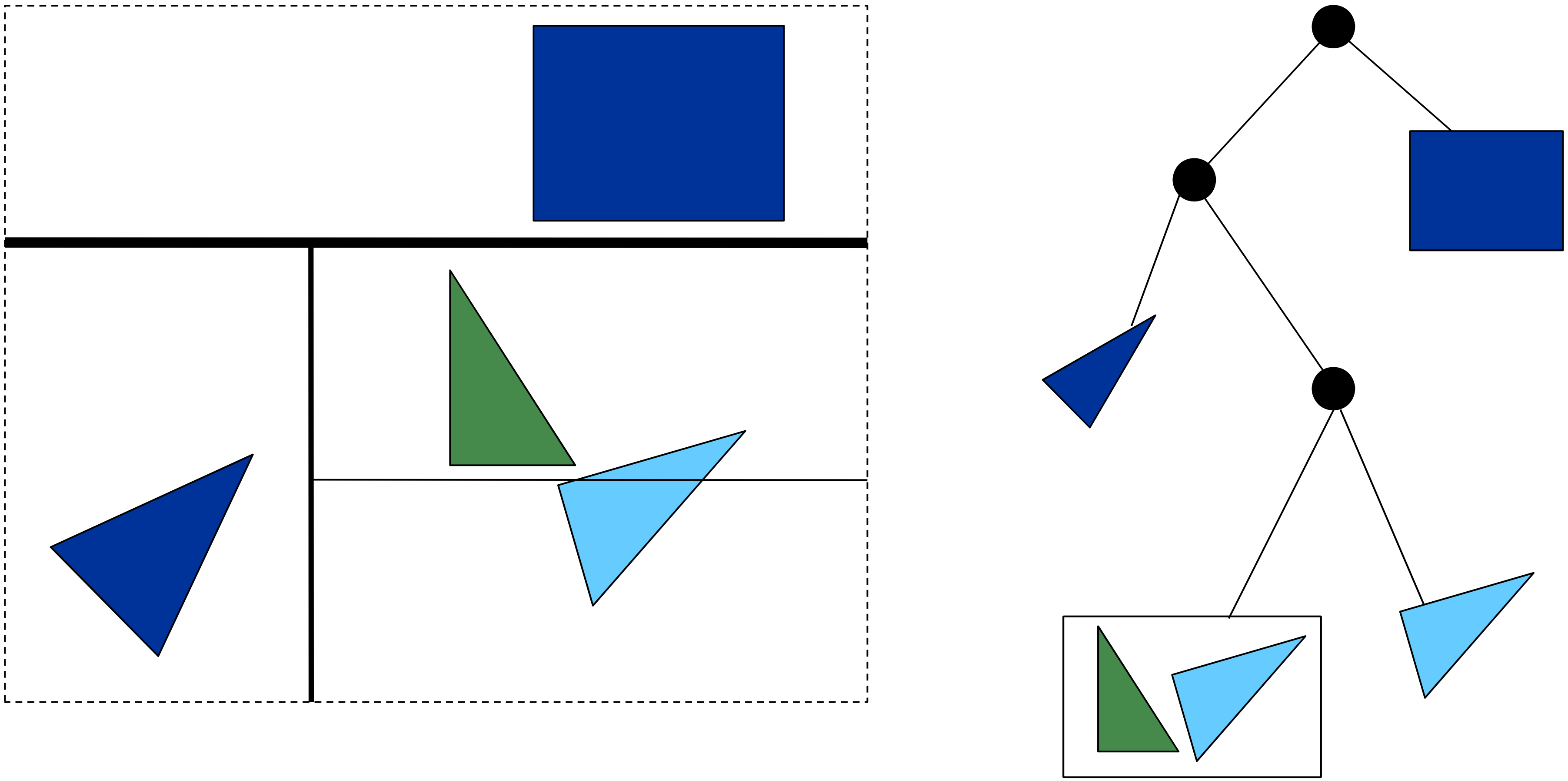


**What about triangles overlapping the split?**



KAIST

# Kd-tree: Example

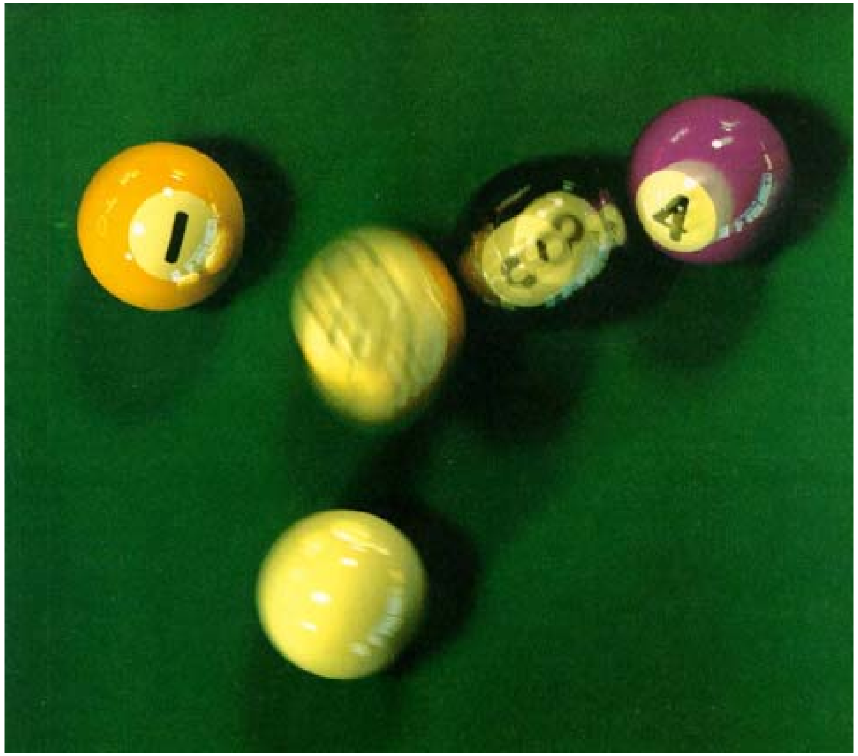


# Other Optimizations

- **Shadow cache**
- **Adaptive depth control**
- **Lazy geometry loading/creation**

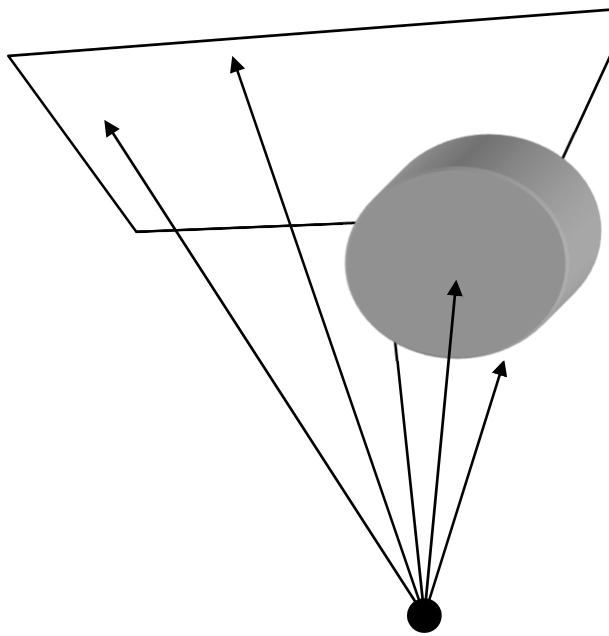
# Distributed Ray Tracing [Cook et al. 84]

- **Cook et al. realized that ray-tracing, when combined with randomized sampling, which they called "jittering", could be adapted to address a wide range of rendering problems:**



# Soft Shadows

- **Take many samples from area light source and take their average**
  - **Computes fractional visibility leading to penumbra**

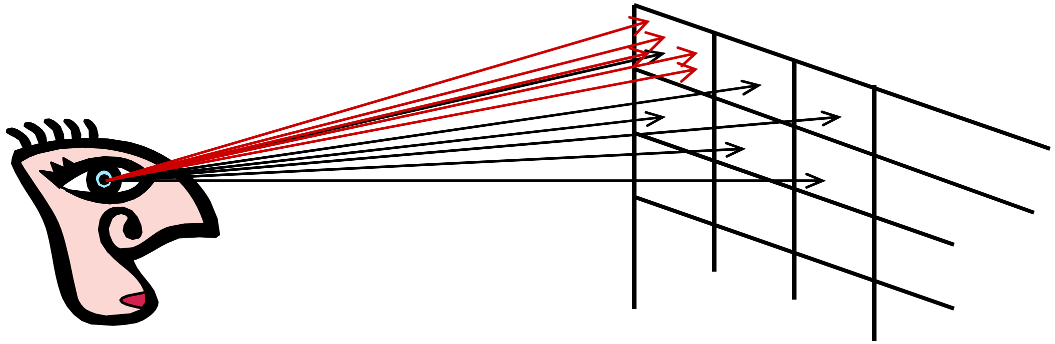


# Antialiasing

- **The need to sample is problematic because sampling leads to aliasing**
- **Solution 1: super-sampling**
  - **Increases sampling rate, but does not completely eliminate aliasing**
  - **Difficult to completely eliminate aliasing without prefiltering because the world is not band-limited**

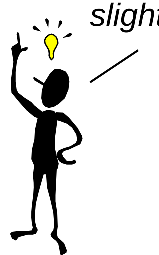
# Antialiasing

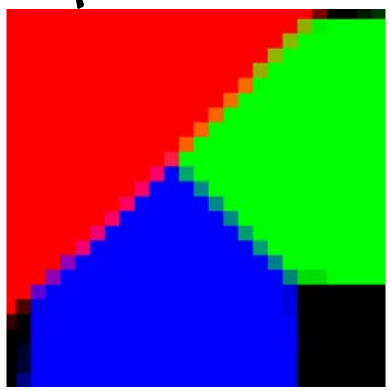
- **Solution 2: distribute the samples randomly**
  - **Converts the aliasing energy to noise which is less objectionable to the eye**



*Instead of casting one ray per pixel, cast several sub- sampling.*

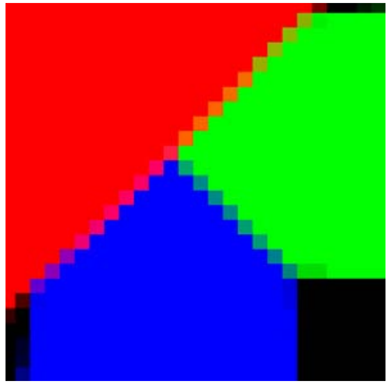
*Instead of uniform sub-sampling, jitter the pixels slightly off the grid.*

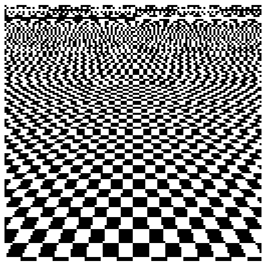


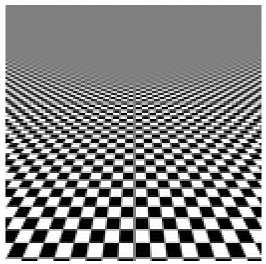


# Jittering Results for Antialiasing

2x2
sub-sampling

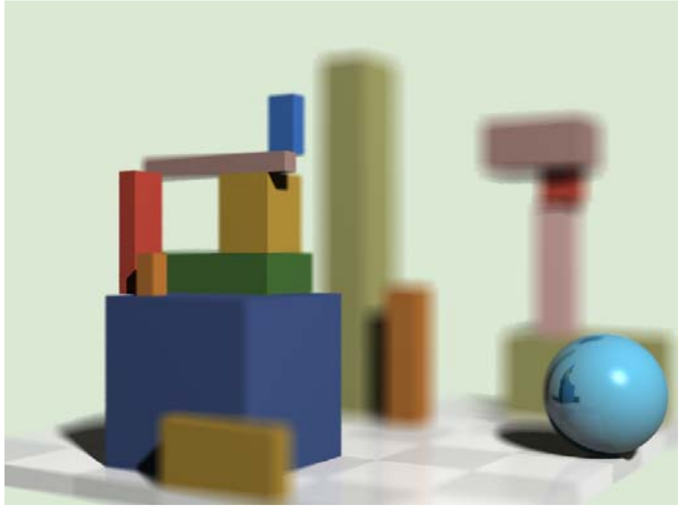




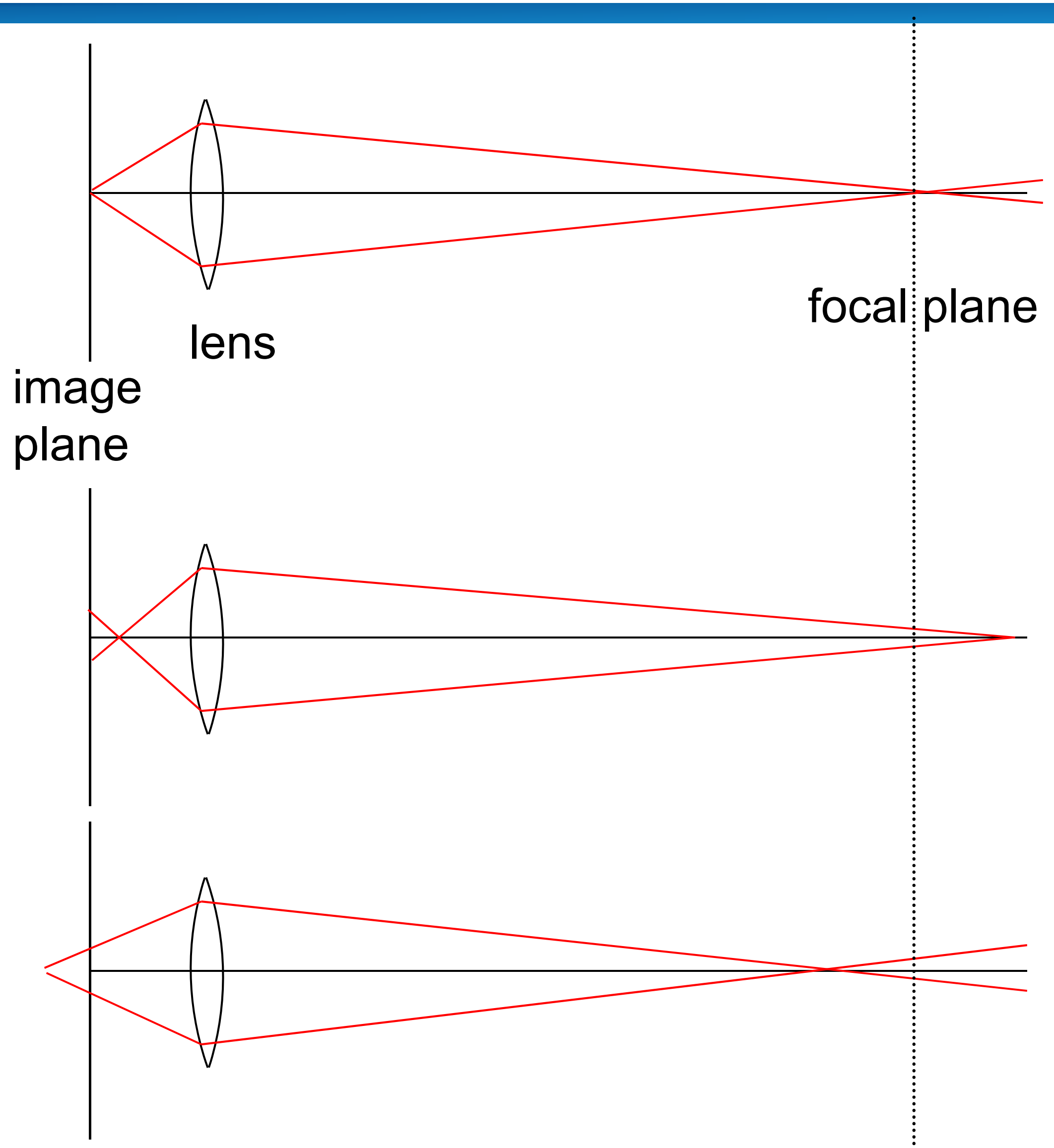


# Depth-of-Field

- Rays don't have to all originate from a single point.
- Real cameras collects rays over an aperture
  - Can be modeled as a disk
  - Final image is blurred away from the focal plane
  - Gives rise to depth-of-field effects

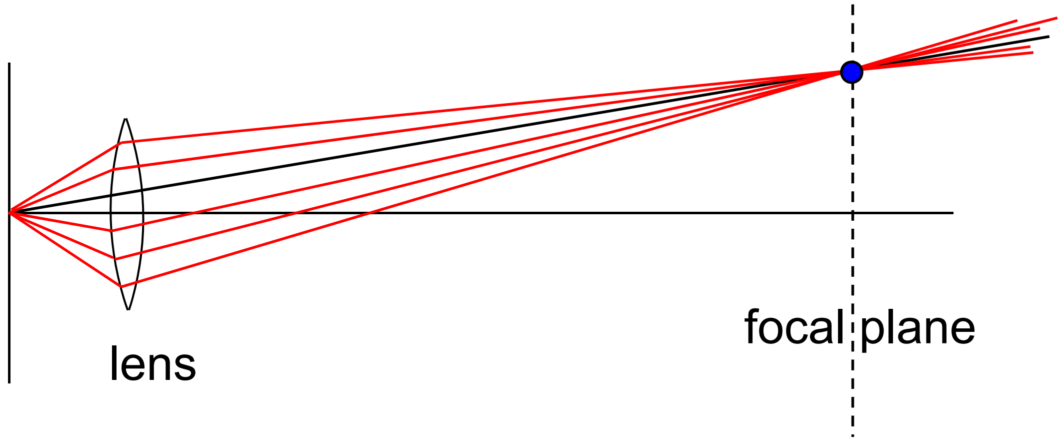


# Depth of Field

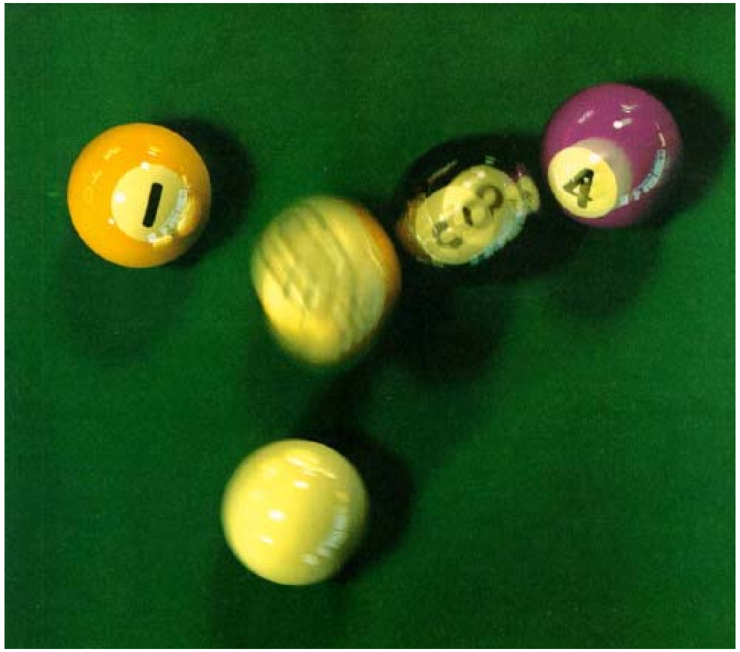


# Depth of Field

- Start with normal eye ray and find intersection with focal plane
- Choose jittered point on lens and trace line from lens point to focal point

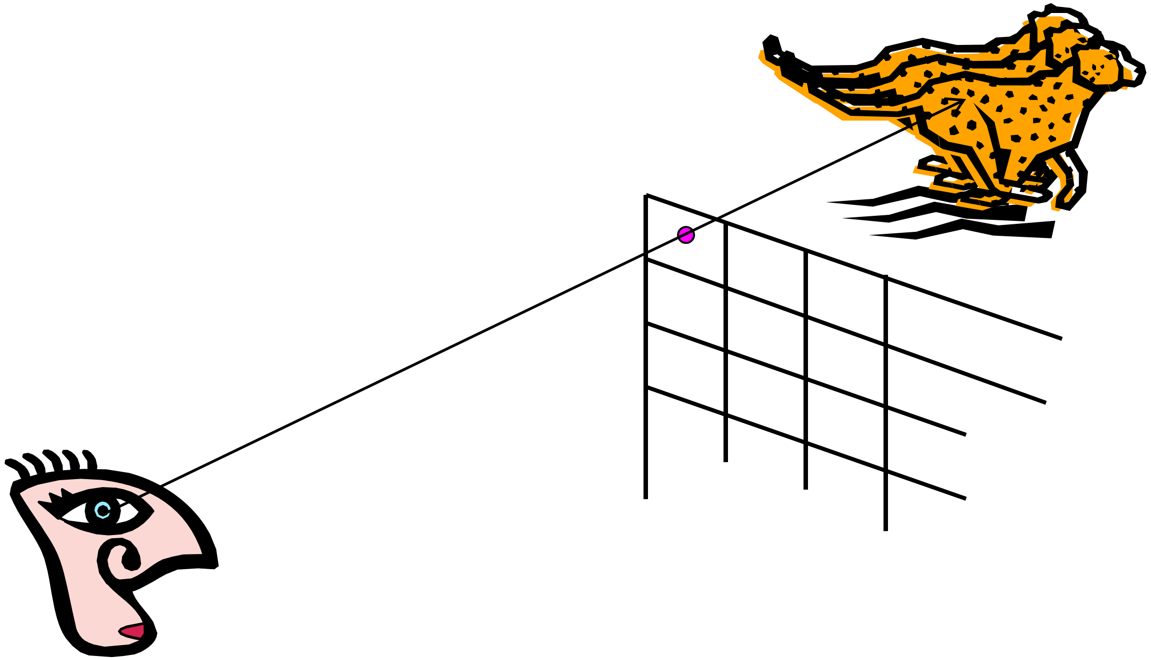


# Motion Blur



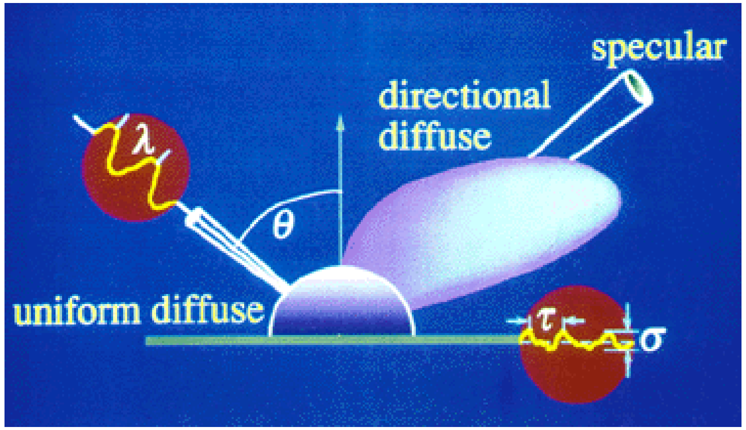
- Jitter samples through time
  - Simulate the finite interval that a shutter is open on a real camera

# Motion Blur



# Complex Interreflection

- **Model true reflection behavior as described by a full BRDF**
- **Randomly sample rays over the hemisphere, weight them by their BRDF value, and average them together**
  - **This technique is called "Monte Carlo Integration"**



# Related Courses

- **CS580: Advanced Computer Graphics**
  - **Focus on rendering techniques that generate photo-realistic images**

- **CS482: Interactive Computer Graphics**
  - **Interactive global illumination implemented by rasterization approaches**
  - **Techniques used in recent games**
  - **I'll teach it at Fall of 2018**