Computer Graphics
-Ray Tracing

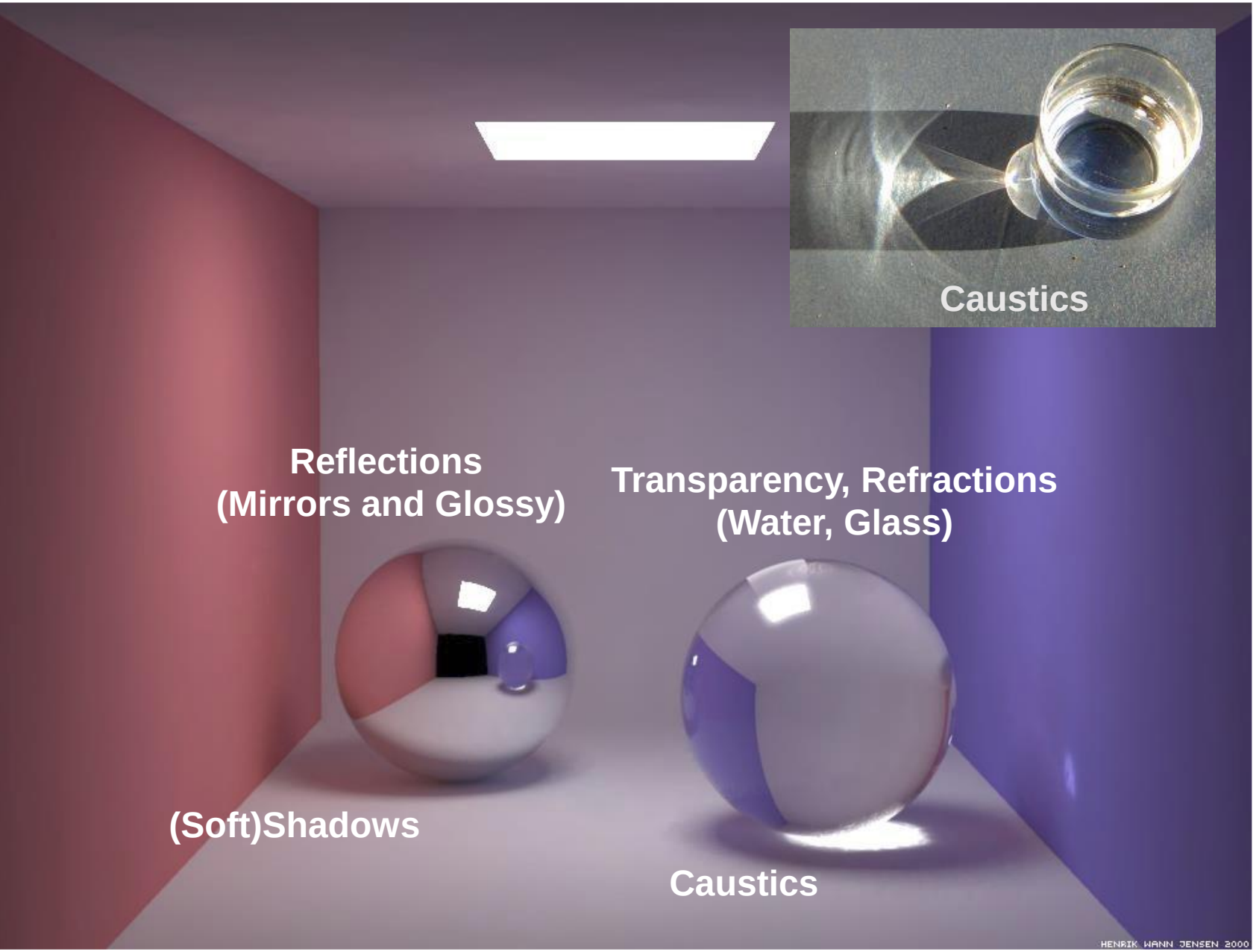
Junjie Cao @ DLUT

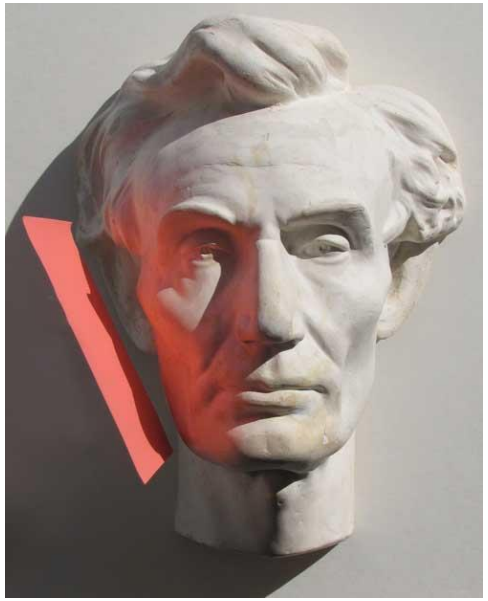
Spring 2018

http://jjcao.github.io/ComputerGraphics/

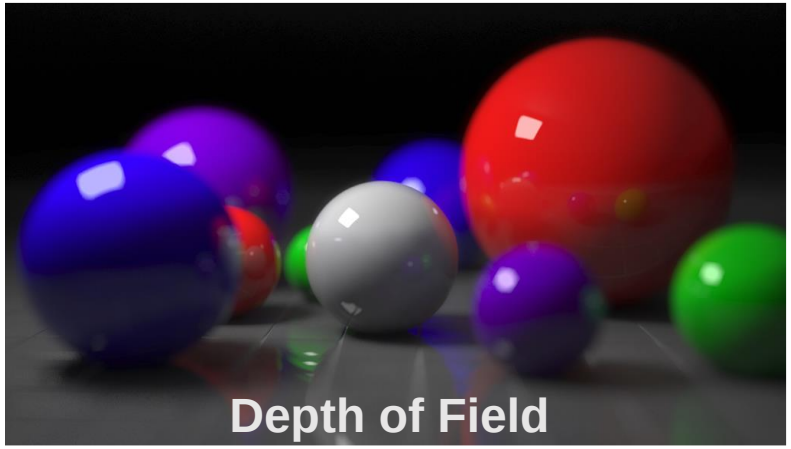
Motivation: Reflections

Motivation: Effects needed for Realism





Inter reflections (Color Bleeding)



Motivation: Effects needed for Realism

- (Soft) Shadows
- Reflections (Mirrors and Glossy)
- Transparency (Water, Glass)
- Inter reflections (Color Bleeding)
- Complex Illumination (Natural, Area Light)
- Realistic Materials (Velvet, Paints, Glass)
- …

Most of these effects are possible but very difficult to do using the OpenGL pipeline that we have studied so far.

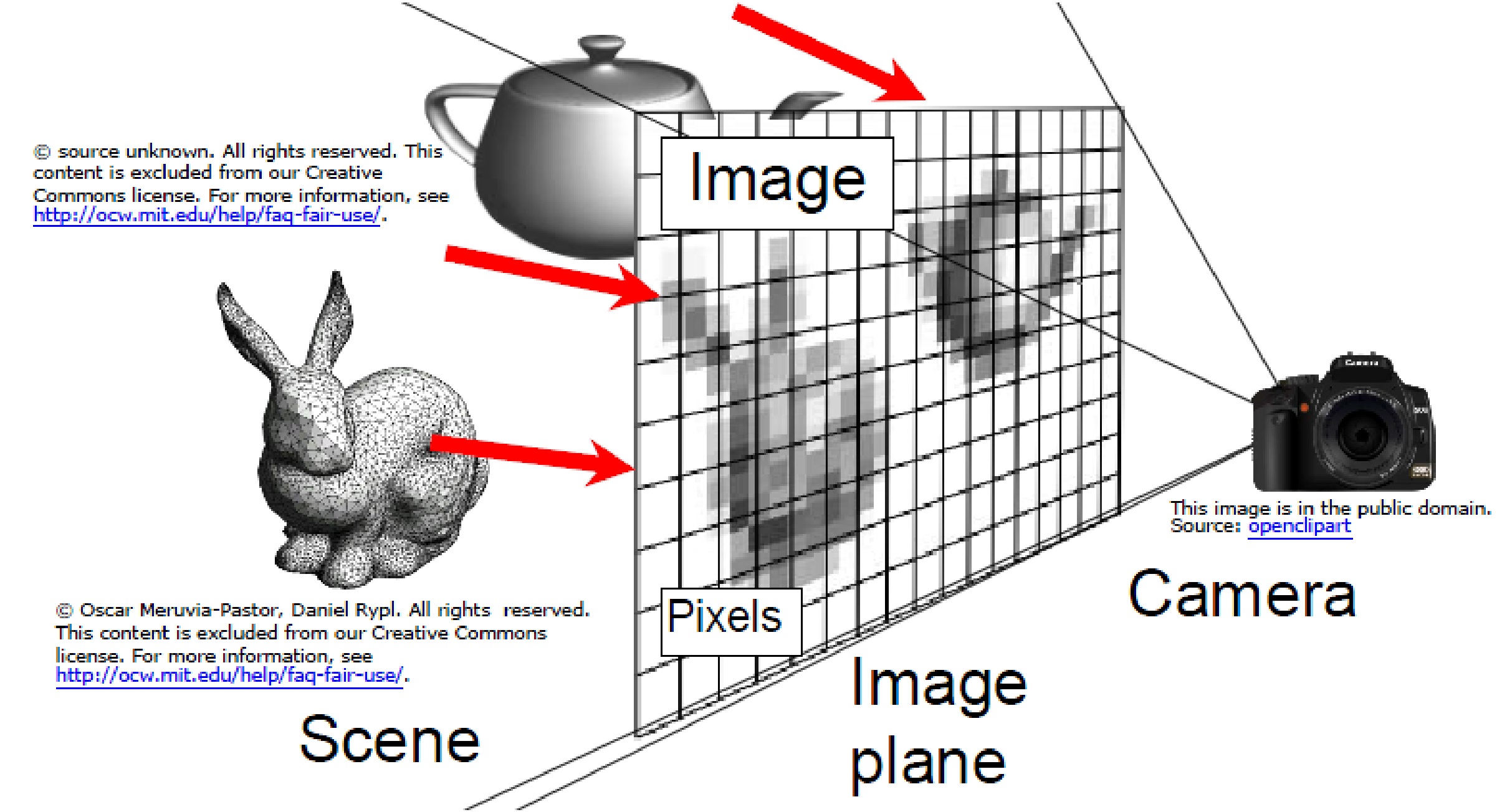
Ray Tracing

- Different Approach to Image Synthesis as compared to Hardware pipeline (OpenGL)
- Pixel by Pixel instead of Object by Object
- Easy to compute shadows/transparency/etc

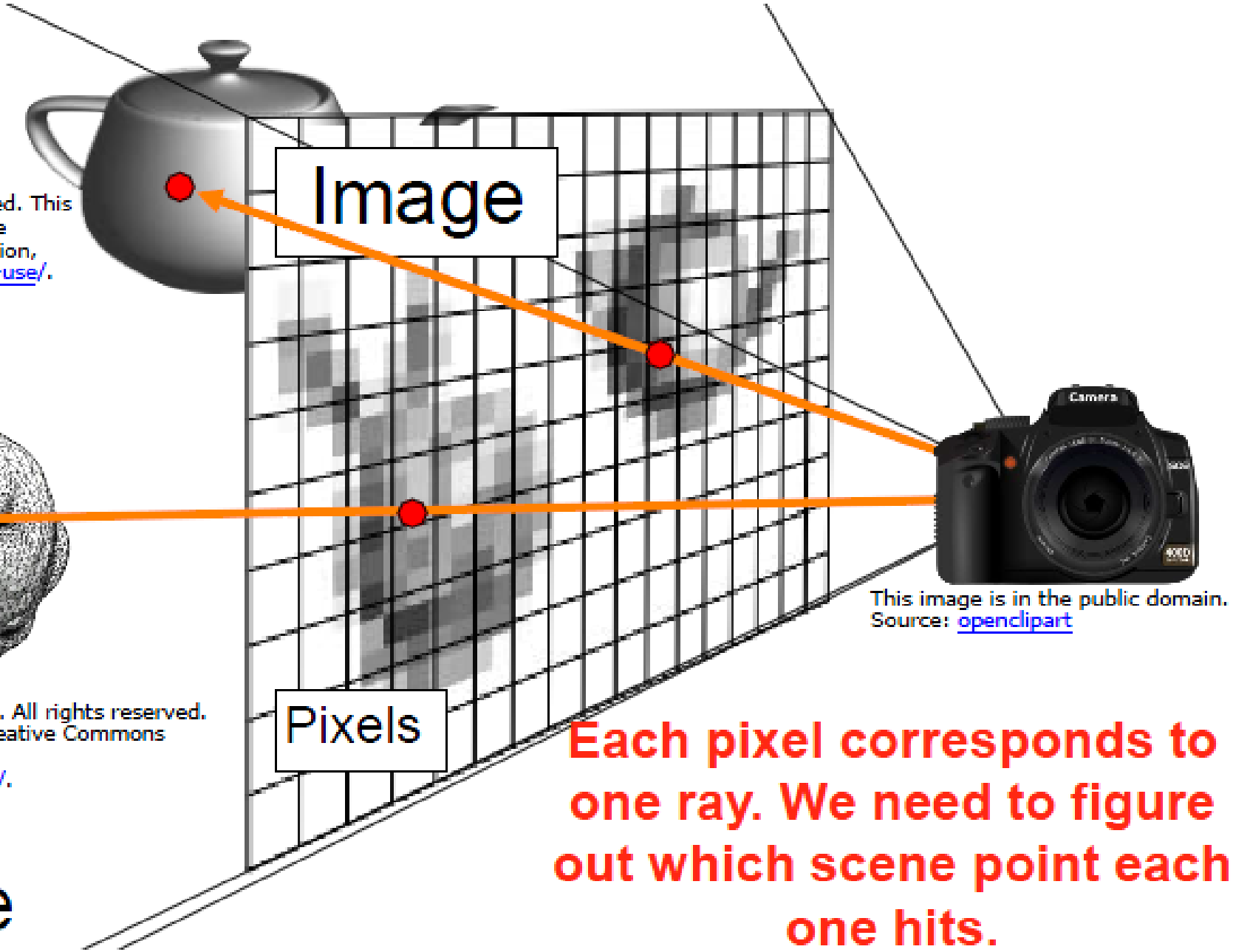
Today

- What does *rendering* mean?
- Ray casting
- Ray tracing

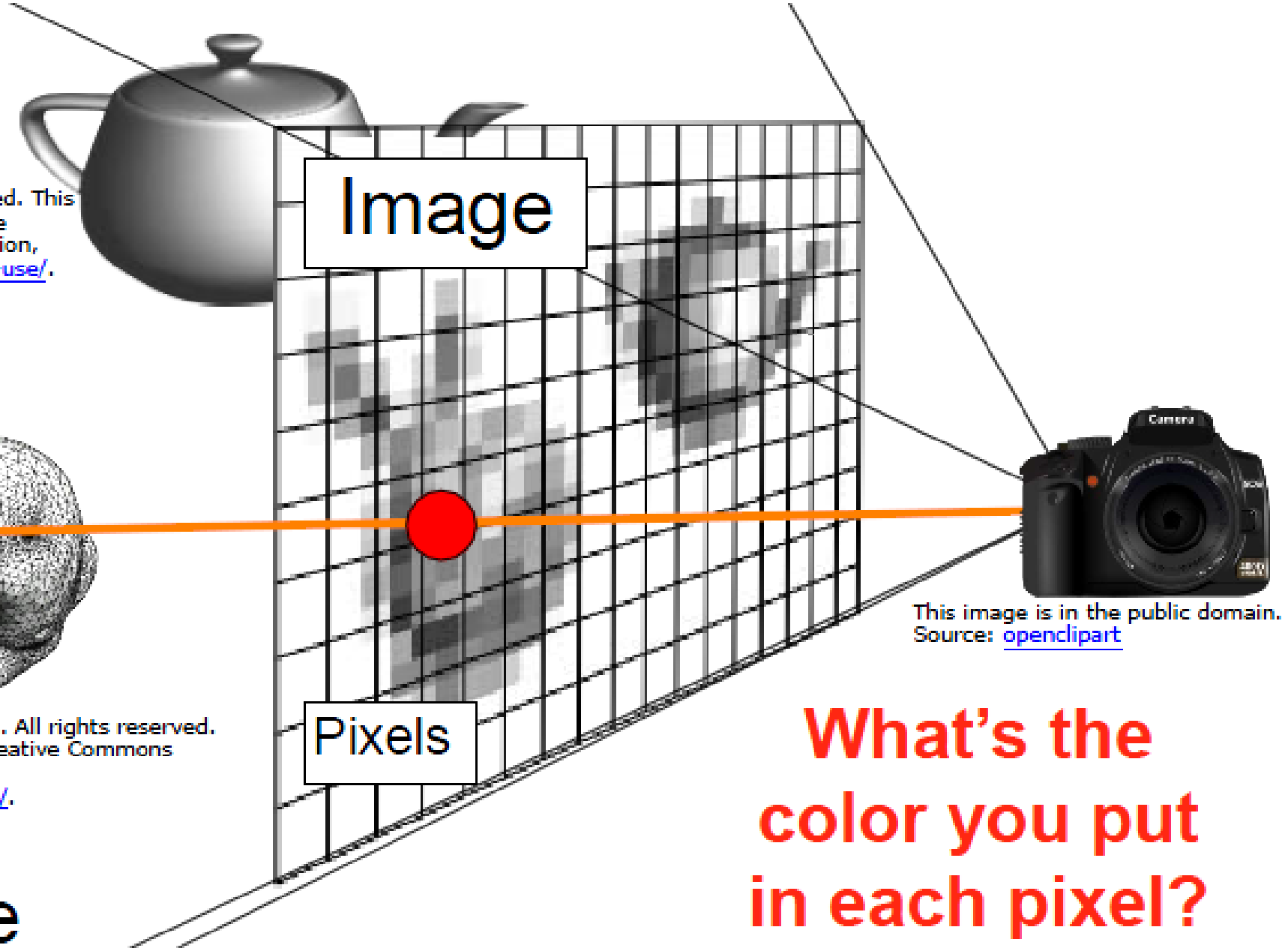
Rendering = Scene to Image



Rendering – Pinhole Camera

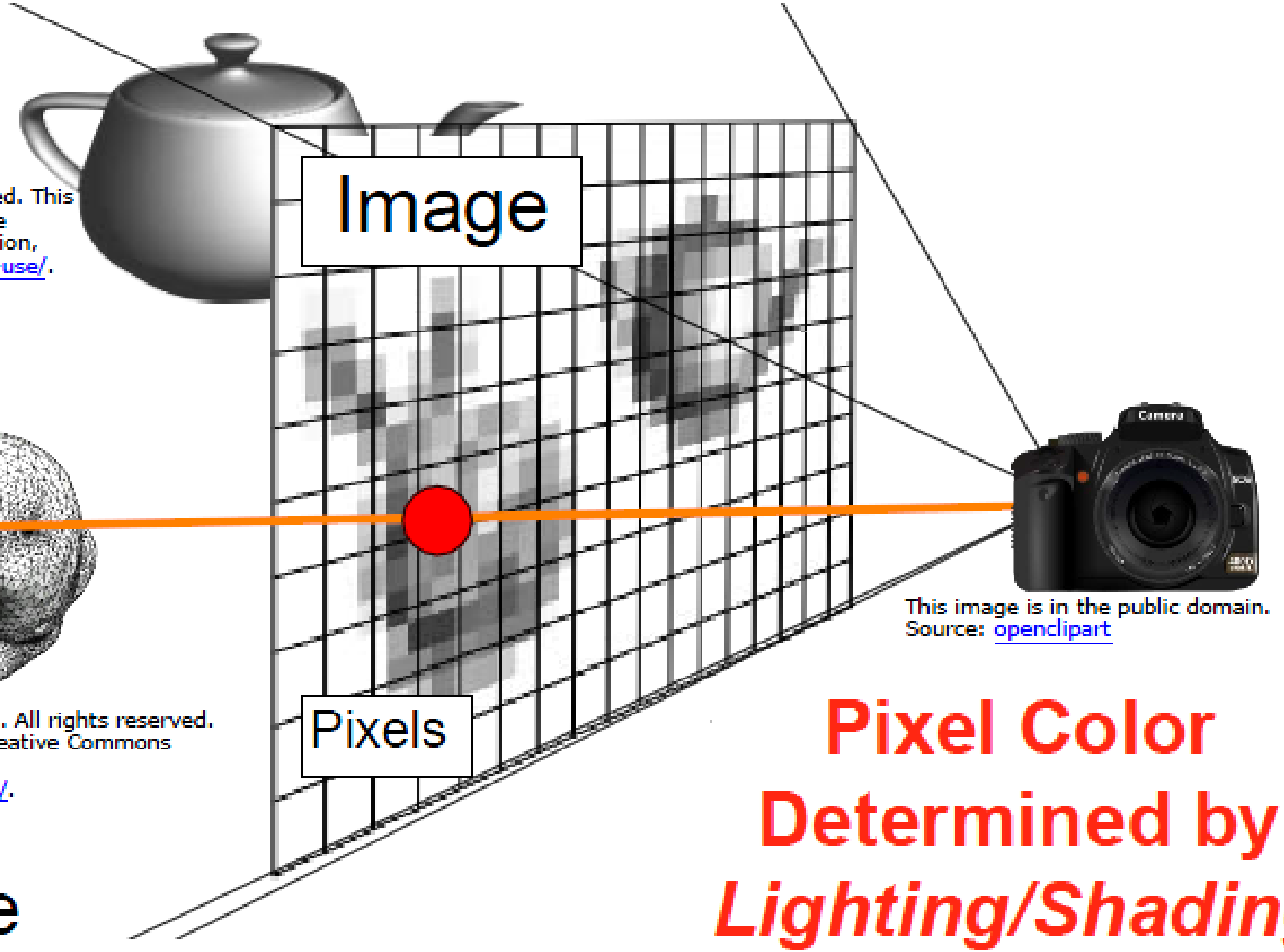


Rendering



What's the color you put in each pixel?

Rendering



Pixel Color Determined by *Lighting/Shading*

Rendering

- “Rendering” refers to the entire process that produces color values for pixels, given a 3D representation of the scene

- **Pixels correspond to rays**; need to figure out the **visible** scene point along each ray
  - Called “hidden surface problem” in older texts
  - “Visibility” is a more modern term

  - Also, we assume (for now) a single ray per pixel

Rendering

- “Rendering” refers to the entire process that produces color values for pixels, given a 3D representation of the scene

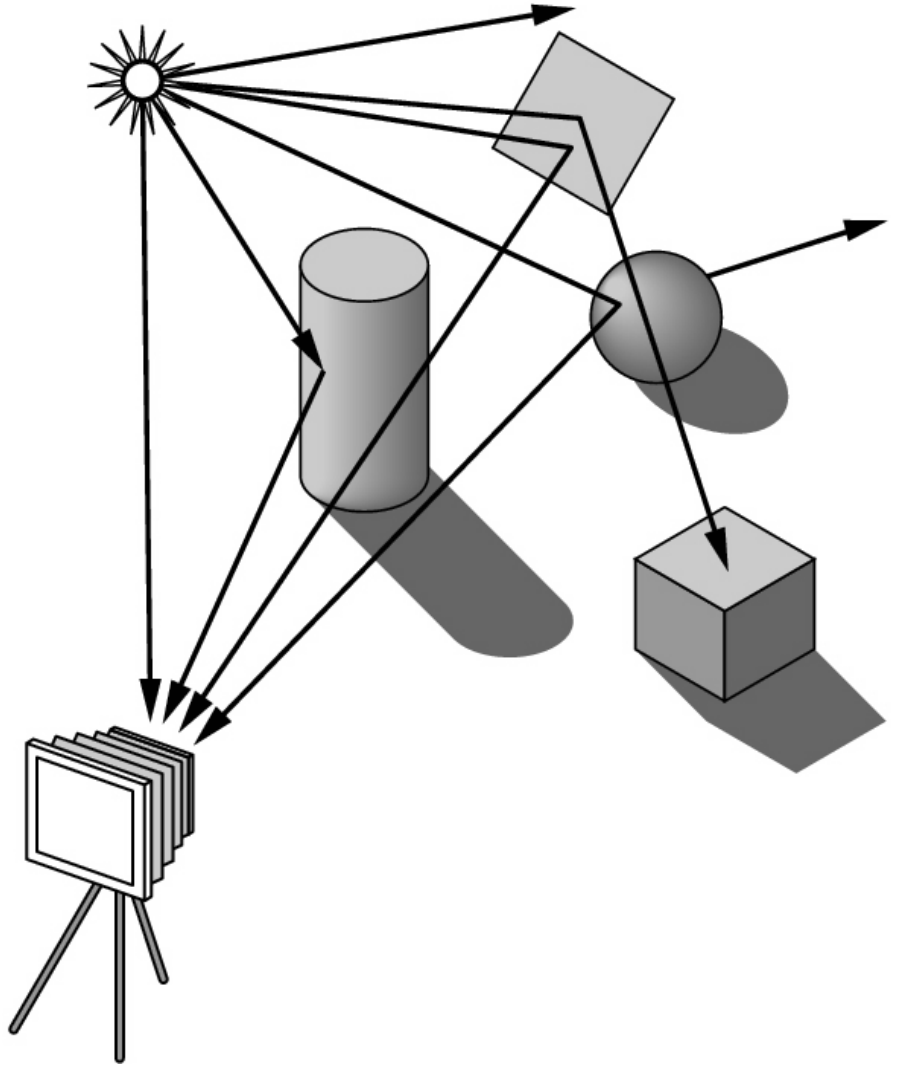
- **Pixels correspond to rays**; need to figure out the **visible** scene point along each ray
  - Called “hidden surface problem” in older texts
  - “Visibility” is a more modern term
  - Also, we assume (for now) a single ray per pixel

- Major algorithms: **Ray tracing and rasterization**

- Note: We are assuming a pinhole camera (for now)

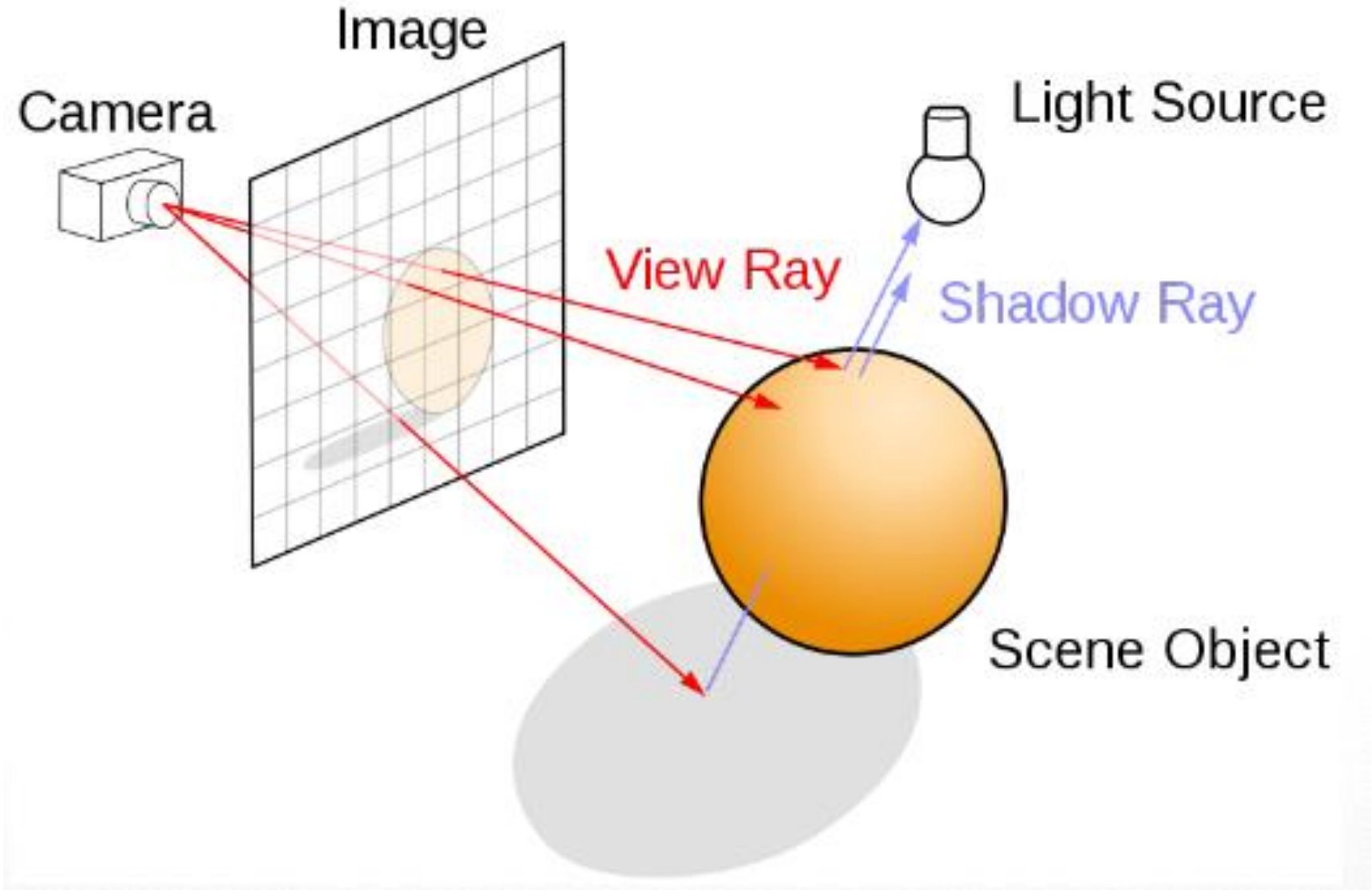
First idea: Forward Ray Tracing

- Shoot (many) light rays from each light source
- Rays bounce off the objects
- Simulates paths of photons
- Problem: many rays will miss camera and not contribute to image!
- This algorithm is not practical



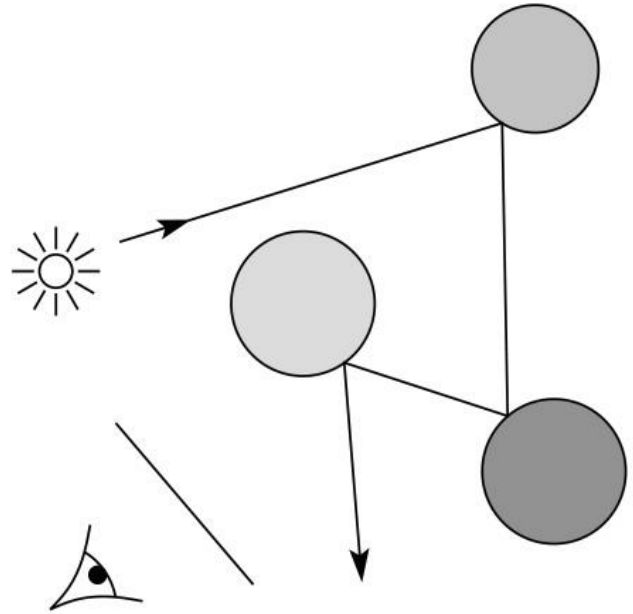
Backward Ray Tracing

- Shoot one ray from camera through each pixel in image plane

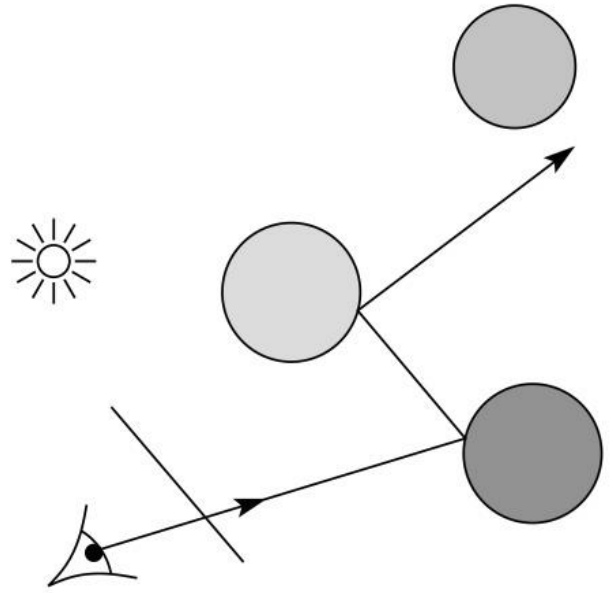


Eye vs. light ray tracing

- Where does light begin?
- At the light: light ray tracing (a.k.a., **forward** ray tracing or photon tracing)



- At the eye: eye ray tracing (a.k.a., **backward** ray tracing)



- We will generally follow rays from the eye into the scene.

Dürer’s Ray Casting Machine

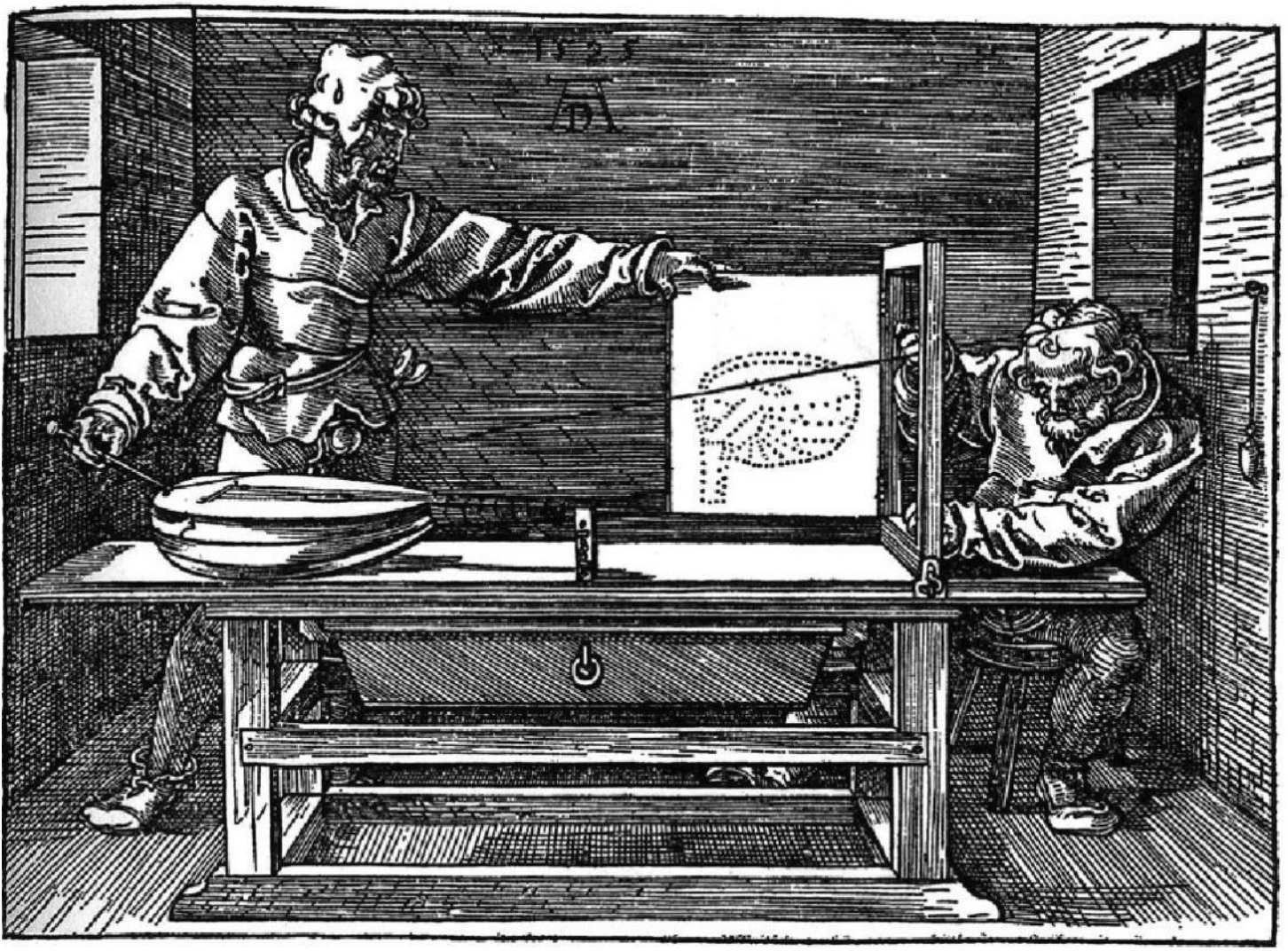
- Albrecht Dürer, 16th century

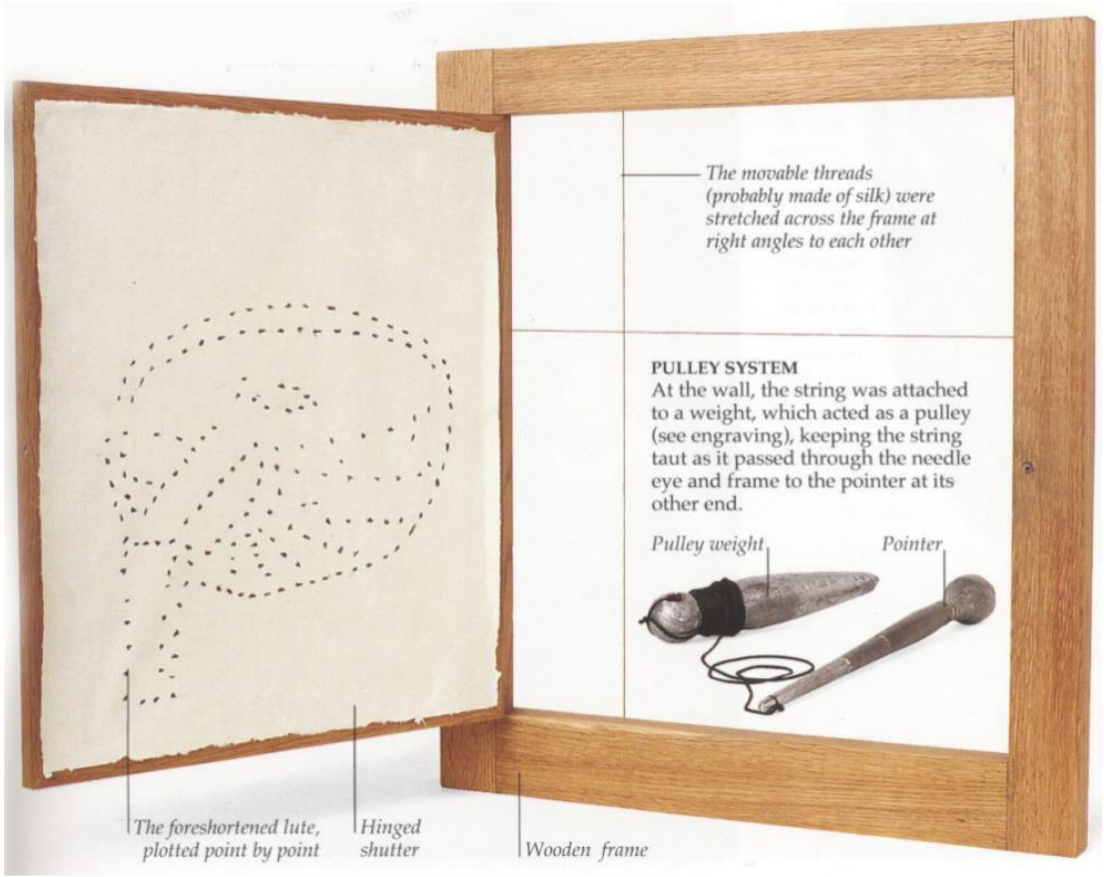
Dürer’s Ray Casting Machine

- Albrecht Dürer, 16th century

Dürer’s Ray Casting Machine

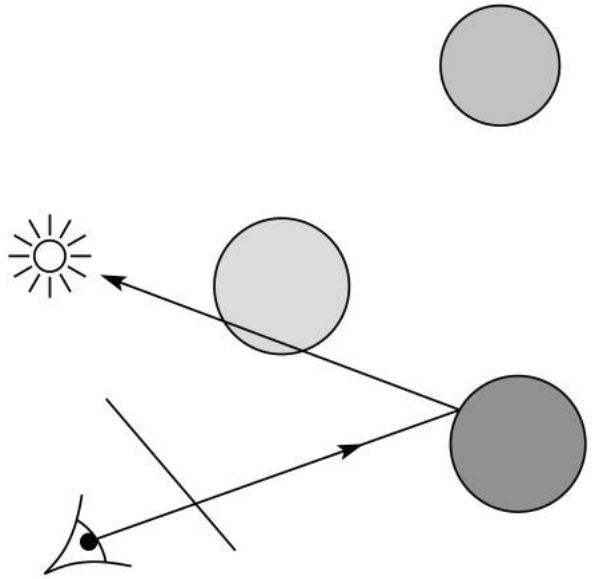
- Albrecht Dürer, 16th century



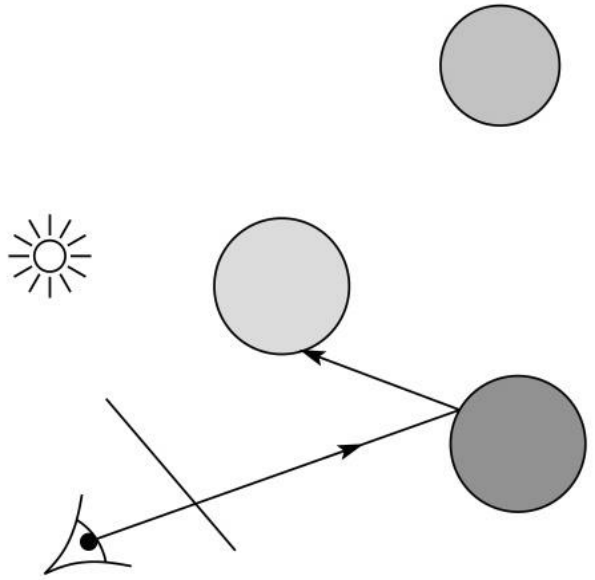


Precursors to ray tracing

- **Local illumination**
  - Cast one eye ray,
    - then shade according to light



- Appel (1968)
  - Cast one eye ray + secondary ray to light

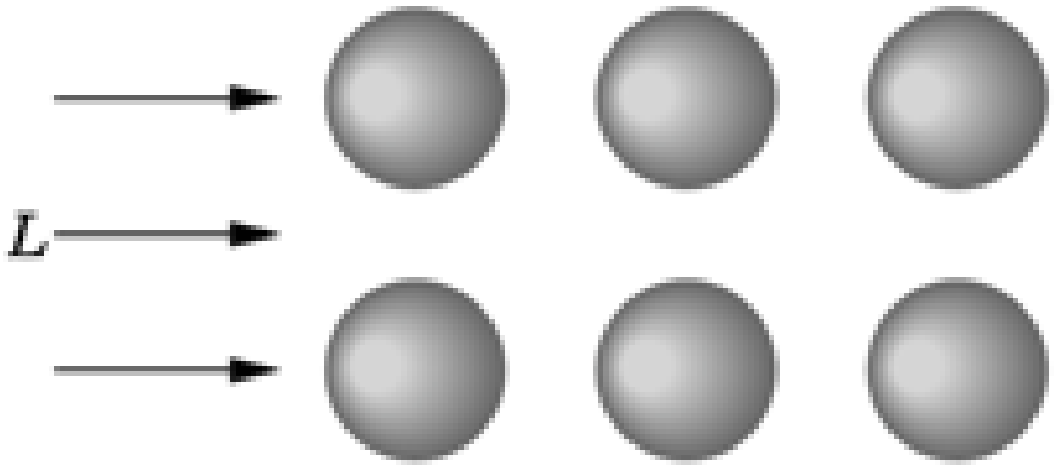


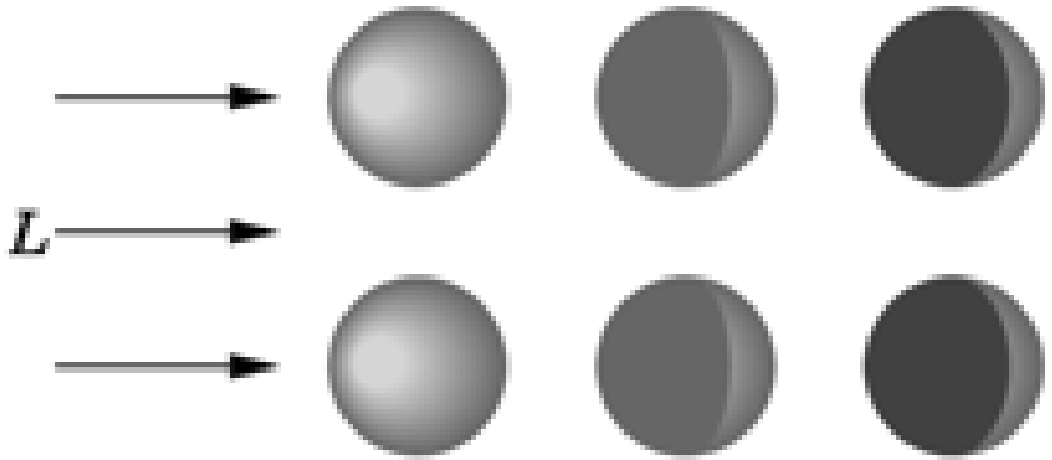
- **Ray casting = eye rays only, tracing = also secondary**

Local v.s. Global Illumination

- Object illuminations are independent
- No light scattering between objects
- No real shadows, reflection, transmission
- OpenGL pipeline uses this

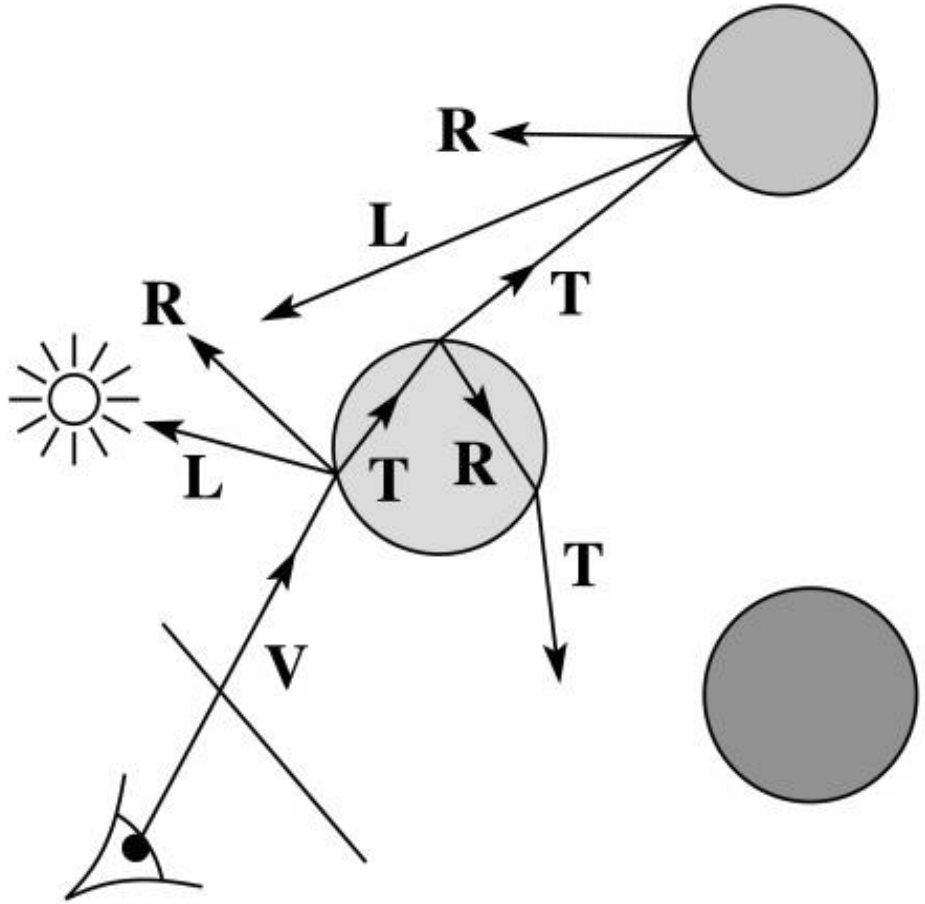
- **Ray tracing** (highlights, reflection, transmission)
- Radiosity (surface inter reflections)
- Photon mapping
- Precomputed Radiance Transfer (PRT)





Whitted ray-tracing algorithm

- In 1980, Turner Whitted introduced ray tracing to the graphics community.
  - Combines eye ray tracing + rays to light
  - Recursively traces rays



- Algorithm:

1. trace a **primary ray** in direction **V** to the first visible surface.

2. For each intersection, trace **secondary rays**:

Shadow rays in directions **L** to light sources

Reflected ray in direction **R**.

Transmitted ray in direction **T**.

Raytracing History

- “An improved illumination model for shaded display” by T. Whitted, CACM 1980
- 512*512, VAX 11/780
- 74 min, today real-time

Ray Tracing: History

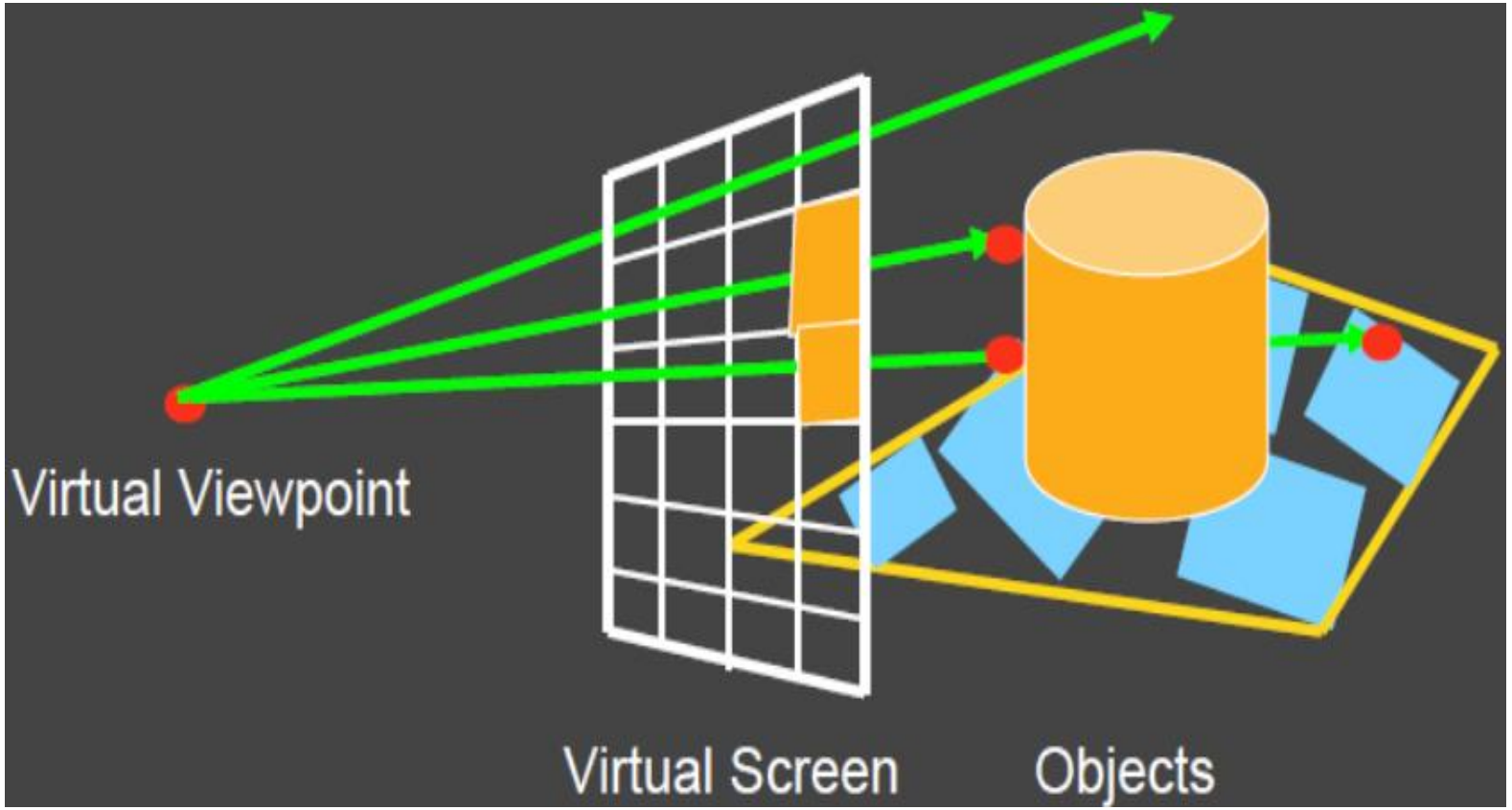
- Appel 68
- **Whitted 80 [recursive ray tracing]**
  - **Landmark in computer graphics**
- Lots of work on various geometric primitives
- Lots of work on accelerations
- Current Research
  - Real-time raytracing (historically, slow technique)
  - Ray tracing architecture

one of the most significant developments in the history of CG

Outline

- Rendering & History of ray tracing
- Basic Ray Casting (instead of rasterization)
  - Comparison to hardware scan conversion
- *Camera Ray Casting (choose ray directions)*
- Shadows / Reflections (core algorithm)
- Ray-Surface Intersection
  - Ray-object intersections
  - Ray-tracing transformed objects
- Optimizations
- Lighting calculations

Ray Casting



- Ray misses all objects: Pixel colored black
- Ray intersects object: shade using color
- Multiple intersections: Use closet one (as does OpenGL), lights, materials

- Produce same images as with OpenGL
- Visibility per pixel instead of Z-buffer
- Find nearest object by shooting rays into scene
- Shade it as in standard OpenGL

Outline in Code

```
Image Raytrace (Camera cam, Scene scene, int width, int height)
{
    Image image = new Image (width, height) ;
    for (int i = 0 ; i < height ; i++)
        for (int j = 0 ; j < width ; j++) {
            Ray ray = RayThruPixel (cam, i, j) ;
            Intersection hit = Intersect (ray, scene) ;
            image[i][j] = FindColor (hit) ;
        }
    return image ;
}
```

Object Space v.s. Image Space

Graphics pipeline: for each object, render

- Efficient pipeline architecture, real-time
- Difficulty: object interactions (shadows, reflections, etc.)

Ray tracing: for each pixel, determine color

- Pixel-level parallelism
- Difficulty: very intensive computation, usually off-line

Object Space v.s. Image Space

- Per-pixel evaluation, per-pixel rays (not scan-convert each object).
  - Ray tracing: : |pixels| * |objs| => costly
  - Rasterization: |objs| * |pixels| (number of pixels a given object covers, which could be 10 or 20, or 1 or 2 pixels)

- But good for walkthroughs of extremely large models
  - have billions of objects, but only millions of pixels
  - never hit them in the raytracer (acceleration structures)

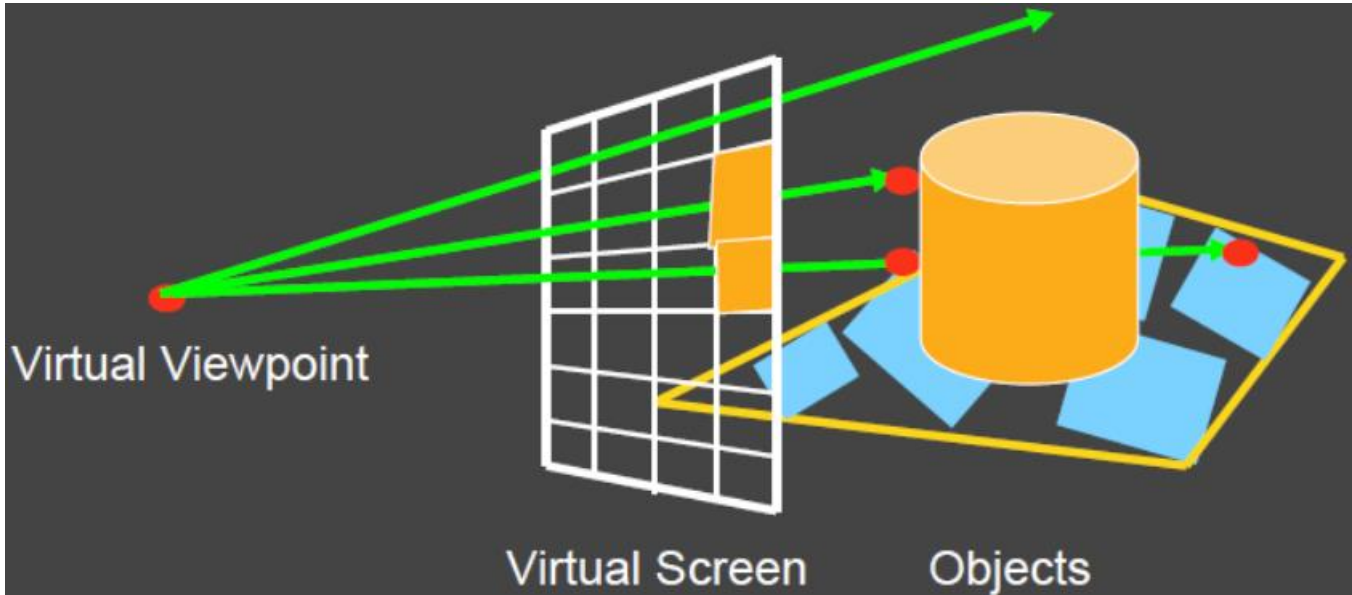
- More complex shading, lighting effects possible

Outline

- History
- Basic Ray Casting (instead of rasterization)
  - Comparison to hardware scan conversion
- Camera Ray Casting *(choose ray directions)*
- Shadows / Reflections (core algorithm)
- Ray-Surface Intersection
  - Ray-object intersections
  - Ray-tracing transformed objects
- Optimizations
- Lighting calculations

Finding Ray Direction

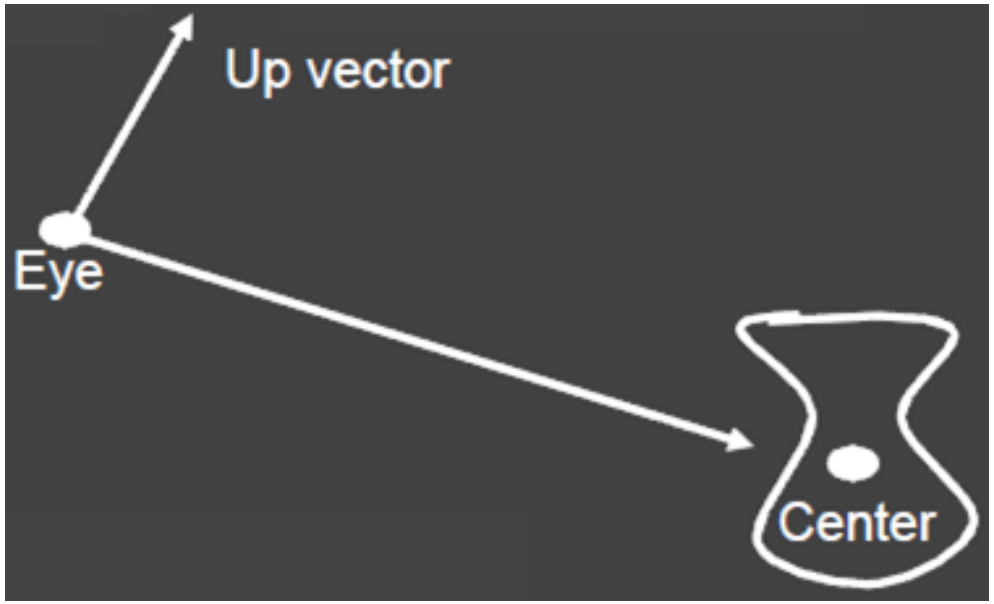
- Goal is to find ray direction for given pixel i and j
- Many ways to approach problem
  - Objects in world coord, find dirn of each ray (we do this)
  - Camera in canonical frame, transform objects (OpenGL)
- Basic idea
  - Ray has origin (camera center) and direction
  - Find direction, given camera params and i and j
- Camera params as in gluLookAt
  - Lookfrom[3], LookAt[3], up[3], fov



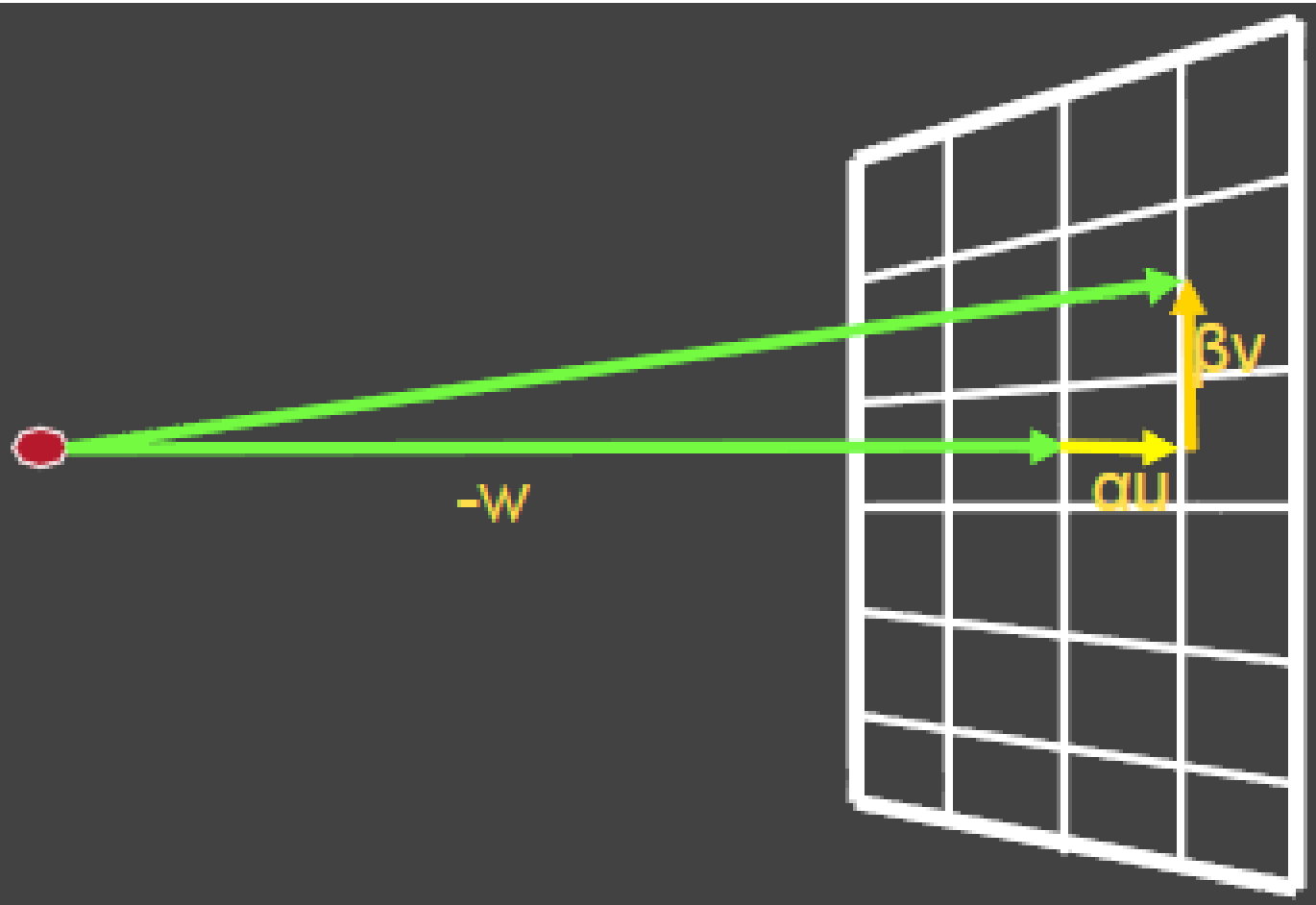
Constructing a coordinate frame?

- We want to position camera at origin, looking down –Z dirn
- Hence, vector **a** is given by **eye** – **center**
- The vector **b** is simply the **up** vector

$$w = \frac{a}{\|a\|} \qquad u = \frac{b \times w}{\|b \times w\|} \qquad v = w \times u$$



Canonical viewing geometry



$$ray = eye + \frac{\alpha u + \beta v - w}{|\alpha u + \beta v - w|}$$

$$\alpha = \tan\left(\frac{fovx}{2}\right) \times \left(\frac{j - (width\,/\,2)}{width\,/\,2}\right)$$

$$\beta = \tan\left(\frac{fovy}{2}\right) \times \left(\frac{(height\,/\,2) - i}{height\,/\,2}\right)$$

Outline

- History
- Basic Ray Casting (instead of rasterization)
  - Comparison to hardware scan conversion
- Camera Ray Casting *(choose ray directions)*
- Shadows / Reflections (core algorithm)
- Ray-Surface Intersection
  - Ray-object intersections
  - Ray-tracing transformed objects
- Optimizations
- Lighting calculations

Whitted algorithm

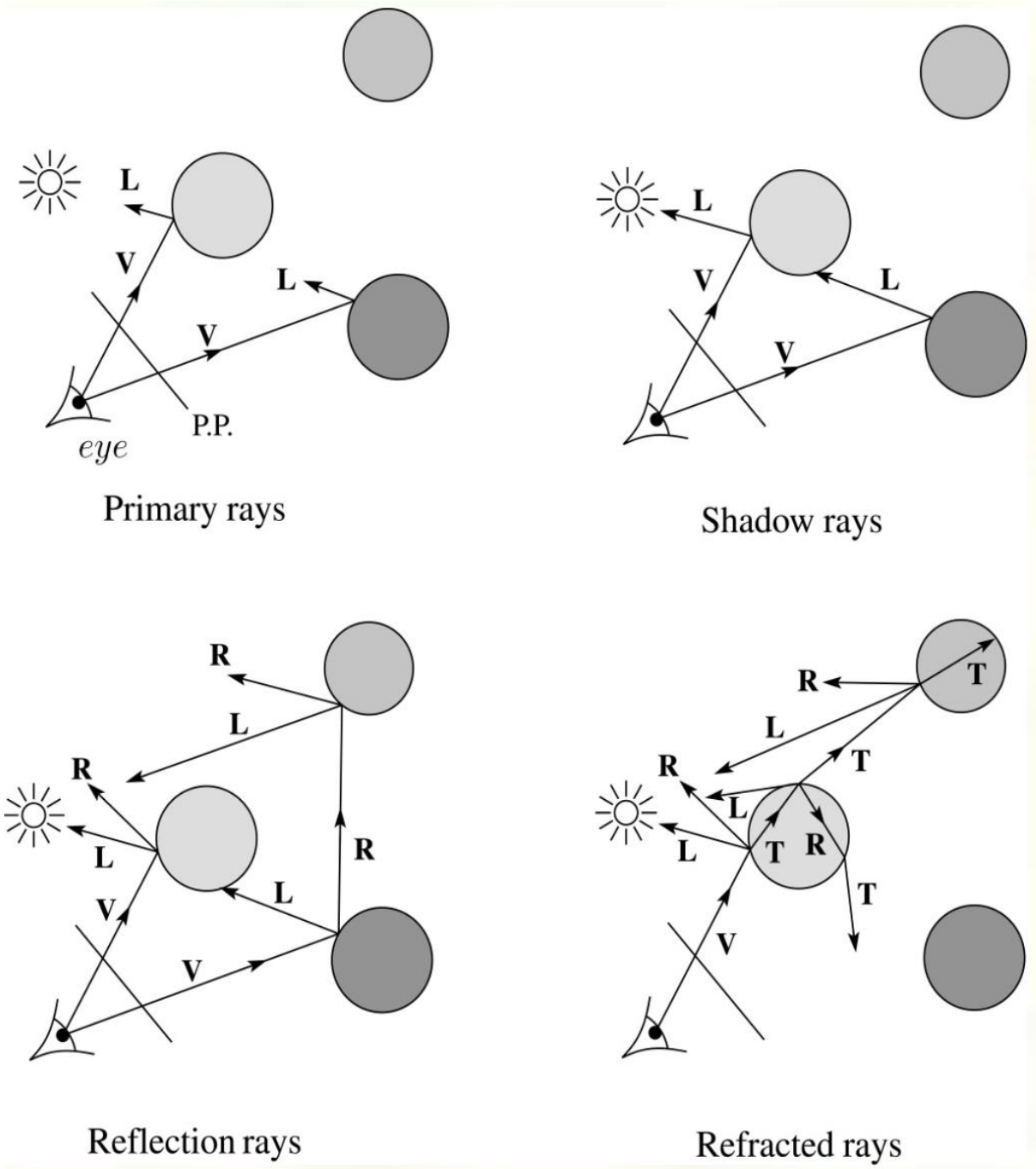
1. Phong model (local as before)

2. Shadow rays

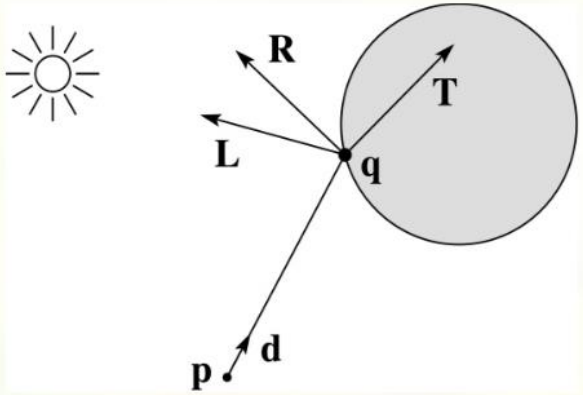
3. Specular reflection

4. Specular transmission

Steps (3) and (4) require recursion



Shading



- A ray is defined by an origin **P** and a unit direction **d** and is parameterized by t:

$$P + t\mathbf{d}$$

- Let $I(P, \mathbf{d})$ be the intensity seen along that ray. Then:

$$I(P, \mathbf{d}) = I_{\text{direct}} + I_{\text{reflected}} + I_{\text{transmitted}}$$

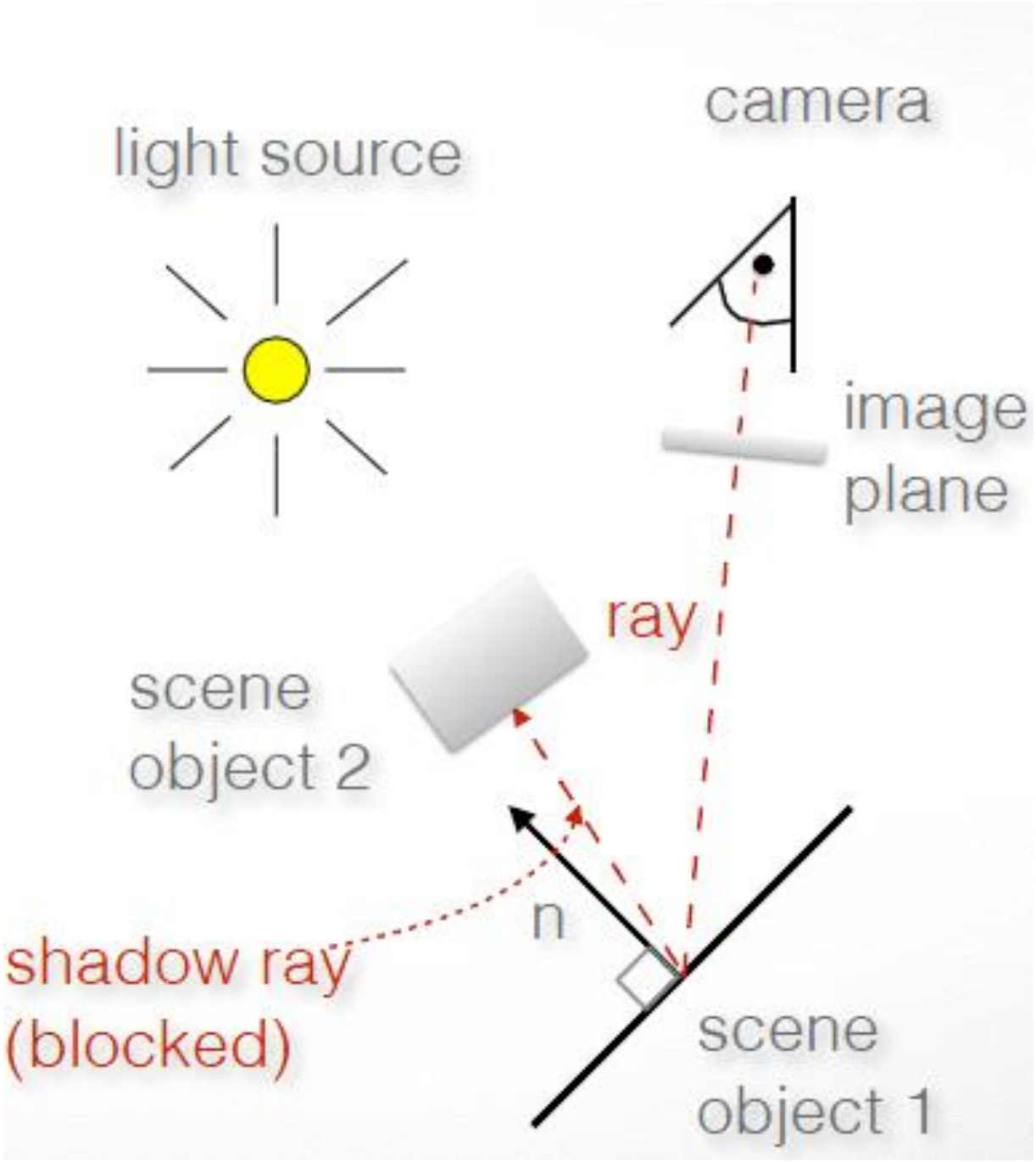
- where
  - I_{direct} is computed from the Phong model
  - $I_{\text{reflected}} = k_r I(Q, \mathbf{R})$
  - $I_{\text{transmitted}} = k_t I(Q, \mathbf{T})$

- Typically, we set $k_r = k_s$ and $k_t = 1 - k_s$.

Shadow Rays

- Shadow ray to light is unblocked: object visible

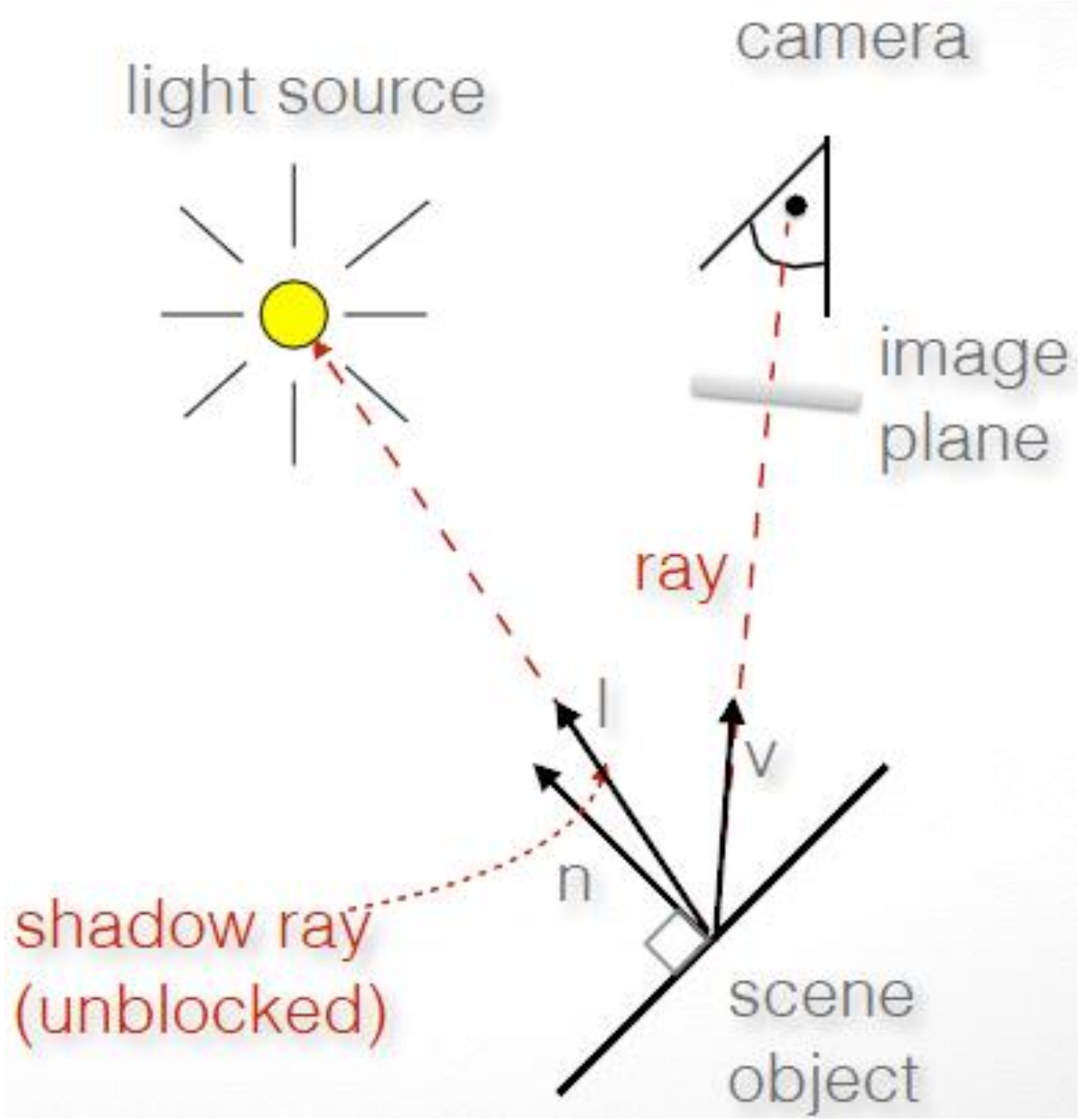
- Shadow ray to light is blocked: object in shadow



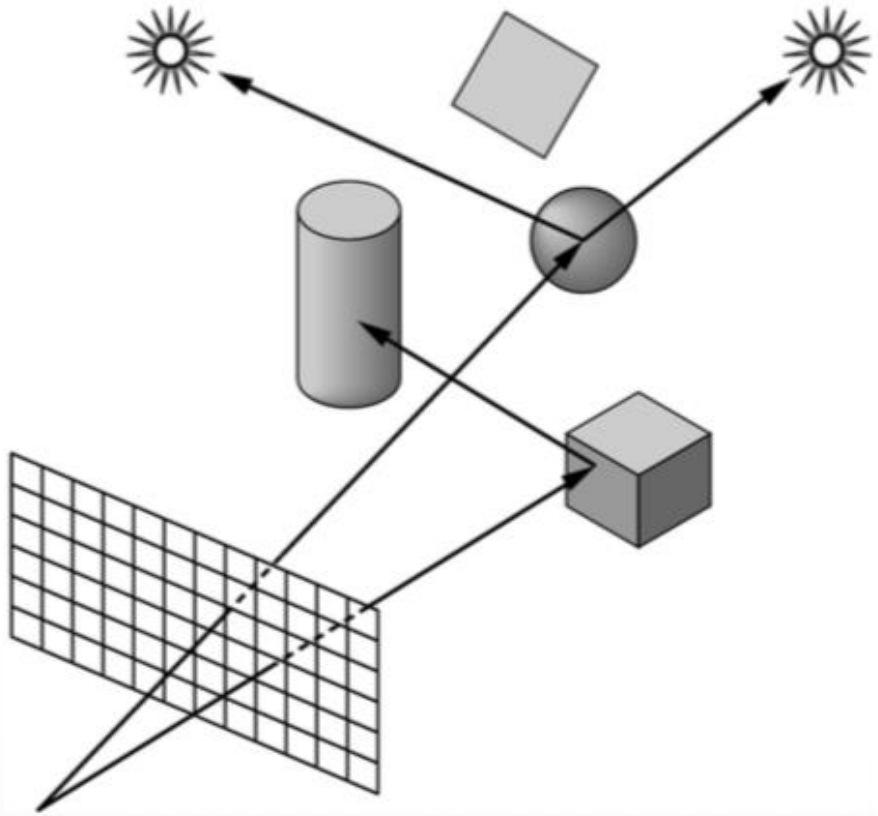
Phong Model

- If shadow ray can reach to the light, apply a standard Phong model

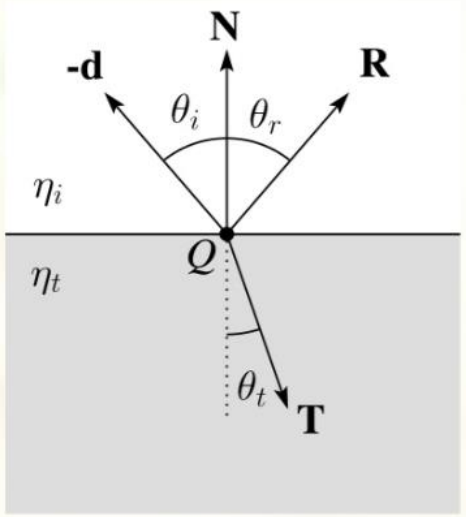
$$I = L\left(k_d(l \cdot n) + k_s(r \cdot v)^{\alpha}\right)$$



Where is Phong model applied in this example?
Which shadow rays are blocked?



Reflection and transmission



Index of refraction is speed of light, relative to speed of light in vacuum = c/v, c is speed in vacuum

Vacuum: 1.0
Air: 1.000277
Water: 1.33
Glass: 1.49

- Law of reflection:

$$\theta_i = \theta_r$$

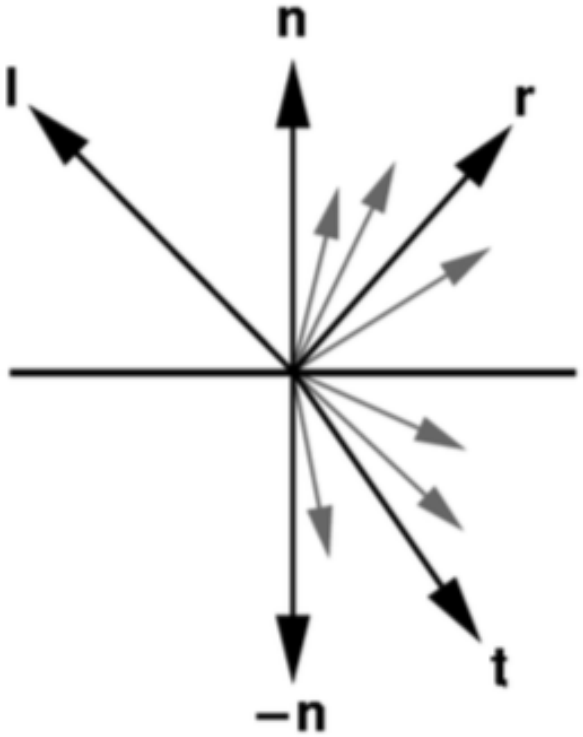
- Snell's law of refraction:

$$\eta_i \sin\theta_I = \eta_t \sin\theta_t$$

- where η_i , η_t are **indices of refraction**.

Translucency

- Most real objects are not transparent, but blur the background image
- Scatter light on other side of surface
-



- Use stochastic sampling (called distributed ray tracing)

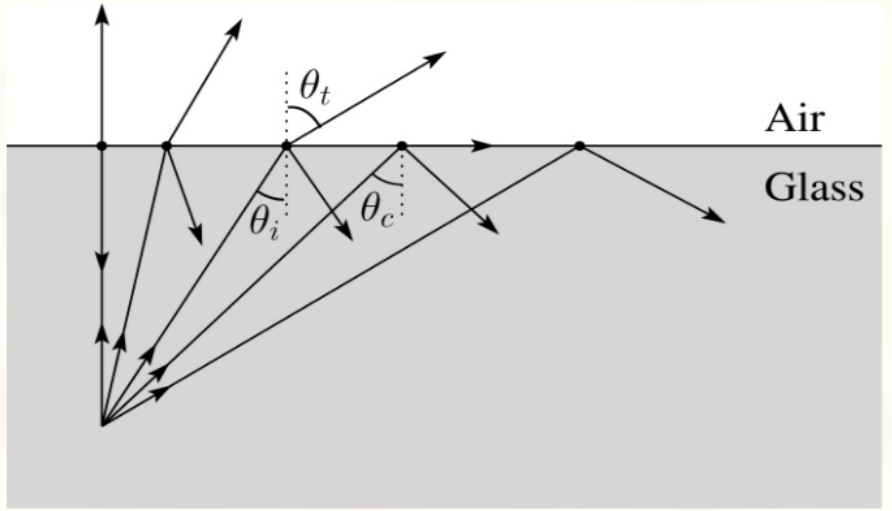
Transmission + Translucency Example

www.povray.org

Total Internal Reflection

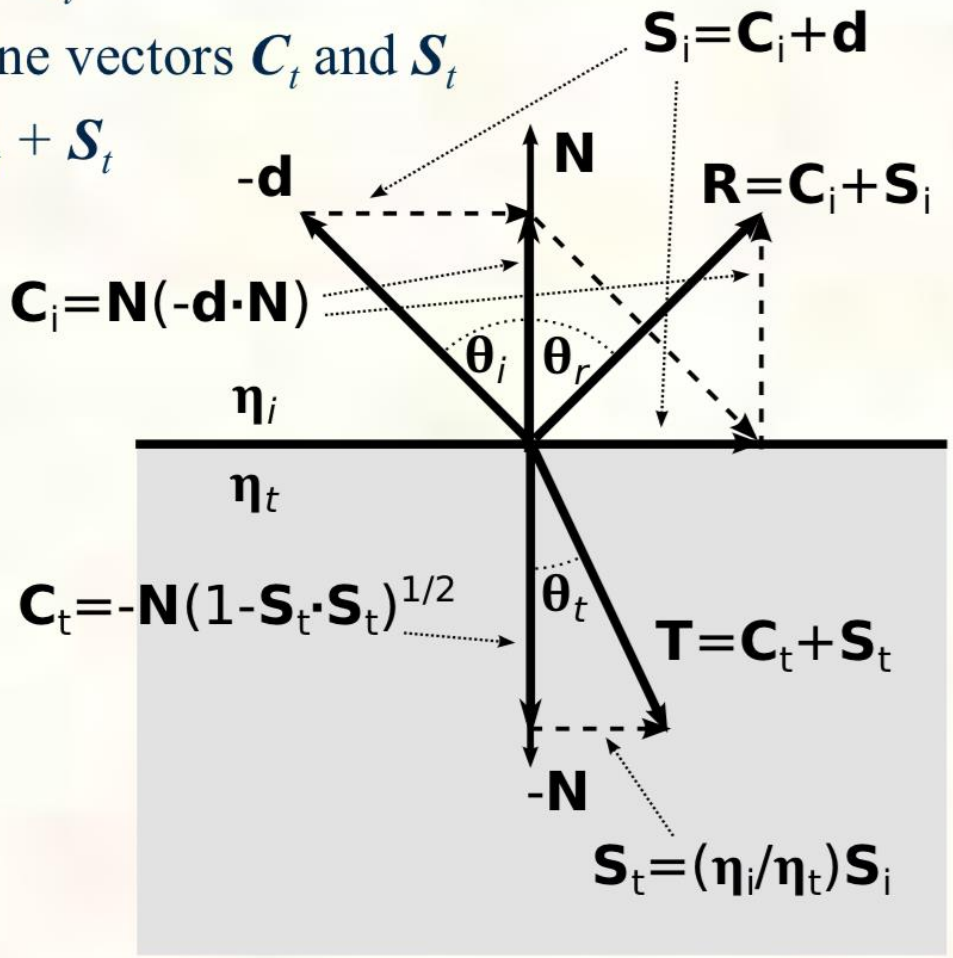
- The equation for the angle of refraction can be computed from Snell's law:

- What happens when $\eta_i > \eta_t$?
- When θ_t is exactly 90°, we say that θ_I has achieved the "critical angle" θ_c.
- For $\theta_I > \theta_c$, *no rays are transmitted*, and only reflection occurs, a phenomenon known as "total internal reflection" or TIR.



Reflected and transmitted rays

- For incoming ray $P(t)=P+td$
  - Compute input cosine and sine vectors $\boldsymbol{C}_i$ and $\boldsymbol{S}_i$
  - Reflected ray vector $\boldsymbol{R} = \boldsymbol{C}_i + \boldsymbol{S}_i$
  - Compute output cosine and sine vectors $\boldsymbol{C}_t$ and $\boldsymbol{S}_t$
  - Transmitted ray vector $\boldsymbol{T} = \boldsymbol{C}_t + \boldsymbol{S}_t$



The Ray Casting Algorithm

- Simplest case of ray tracing

1. For each pixel (x,y), fire a ray from COP through (x,y)
2. For each ray & object, calculate closest intersection
3. For closest intersection point **p**
    1. Calculate surface normal
    2. For each light source, fire shadow ray
    3. For each unblocked shadow ray, evaluate local Phong model for that light, and add the result to pixel color

- Critical operations
    - Ray-surface intersections
    - Illumination calculation

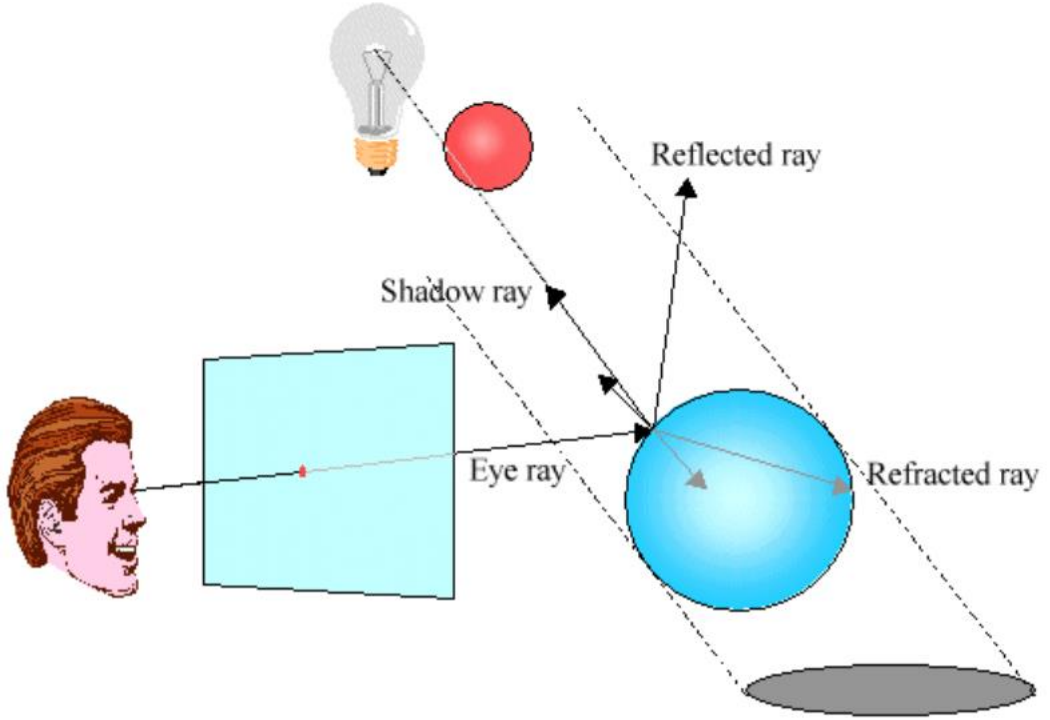
Recursive Ray Tracing

- Also calculate specular component
  - Reflect ray from eye on specular surface
  - Transmit ray from eye through transparent surface

- Determine color of incoming ray by recursion

- Trace to fixed depth

- Cut off if contribution below threshold



Raytracing Examples

www.yafaray.org

Effects needed for Realism

- (Soft) Shadows
- Reflections (Mirrors and Glossy)
- Transparency (Water, Glass)
- Inter reflections (Color Bleeding)
- Complex Illumination (Natural, Area Light)

- Discussed in this lecture
- Not discussed but possible with distribution ray tracing
- Hard (but not impossible) with ray tracing; radiosity methods

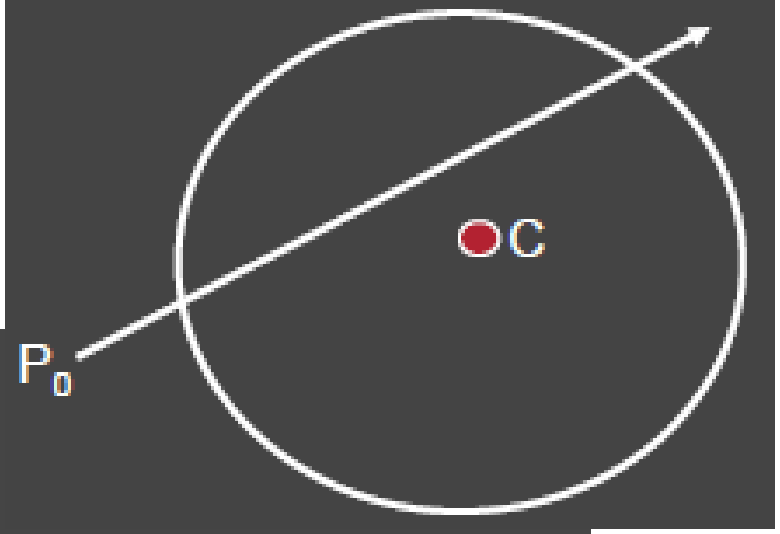
Outline

- History
- Basic Ray Casting (instead of rasterization)
  - Comparison to hardware scan conversion
- Camera Ray Casting *(choose ray directions)*
- Shadows / Reflections (core algorithm)
- Ray-Surface Intersection
  - Ray-object intersections
  - Ray-tracing transformed objects
- Optimizations
- Lighting calculations

Ray/Object Intersections

- Heart of Ray Tracer
  - One of the main initial research areas
  - Optimized routines for wide variety of primitives
- Various types of info
  - Shadow rays: Intersection/No Intersection
  - Primary rays: Point of intersection, material, normals
  - Texture coordinates
- Work out examples
  - Triangle, sphere, polygon, general implicit surface

Ray-Sphere Intersection



$$ray \equiv \vec{P} = \vec{P}_0 + \vec{P}_1 t$$

$$sphere \equiv (\vec{P} - \vec{C}) \bullet (\vec{P} - \vec{C}) - r^2 = 0$$

Substitute

$$ray \equiv \vec{P} = \vec{P}_0 + \vec{P}_1 t$$

$$sphere \equiv (\vec{P}_0 + \vec{P}_1 t - \vec{C}) \bullet (\vec{P}_0 + \vec{P}_1 t - \vec{C}) - r^2 = 0$$

Simplify

$$t^2 (\vec{P}_1 \bullet \vec{P}_1) + 2t\, \vec{P}_1 \bullet (\vec{P}_0 - \vec{C}) + (\vec{P}_0 - \vec{C}) \bullet (\vec{P}_0 - \vec{C}) - r^2 = 0$$

Ray-Sphere Intersection

$$t^2(\vec{P}_1 \bullet \vec{P}_1) + 2t\,\vec{P}_1 \bullet (\vec{P}_0 - \vec{C}) + (\vec{P}_0 - \vec{C}) \bullet (\vec{P}_0 - \vec{C}) - r^2 = 0$$

Solve quadratic equations for t

- 2 real positive roots: pick smaller root
- Both roots same: tangent to sphere
- One positive, one negative root: ray origin inside sphere (pick + root)
- Complex roots: no intersection (check discriminant of equation first)

Ray-Triangle Intersection

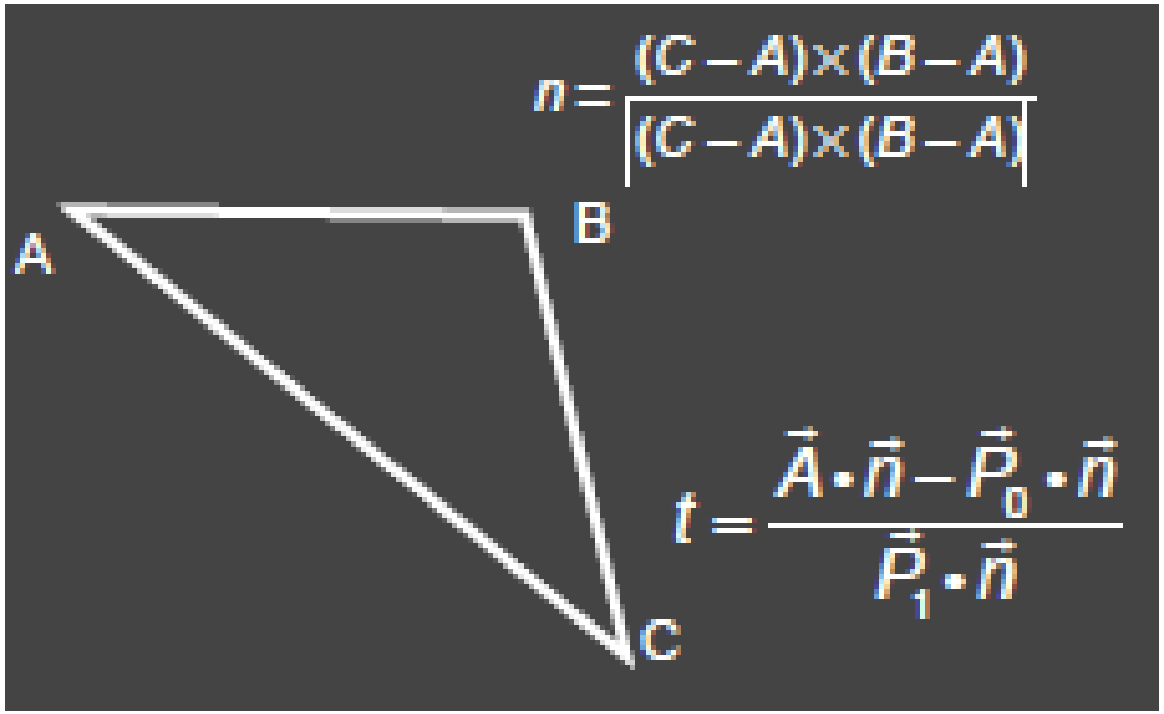
- One approach: Ray-Plane intersection, then check if inside triangle
- Plane equation:

$$plane \equiv \vec{P} \bullet \vec{n} - \vec{A} \bullet \vec{n} = 0$$

- Combine with ray equation

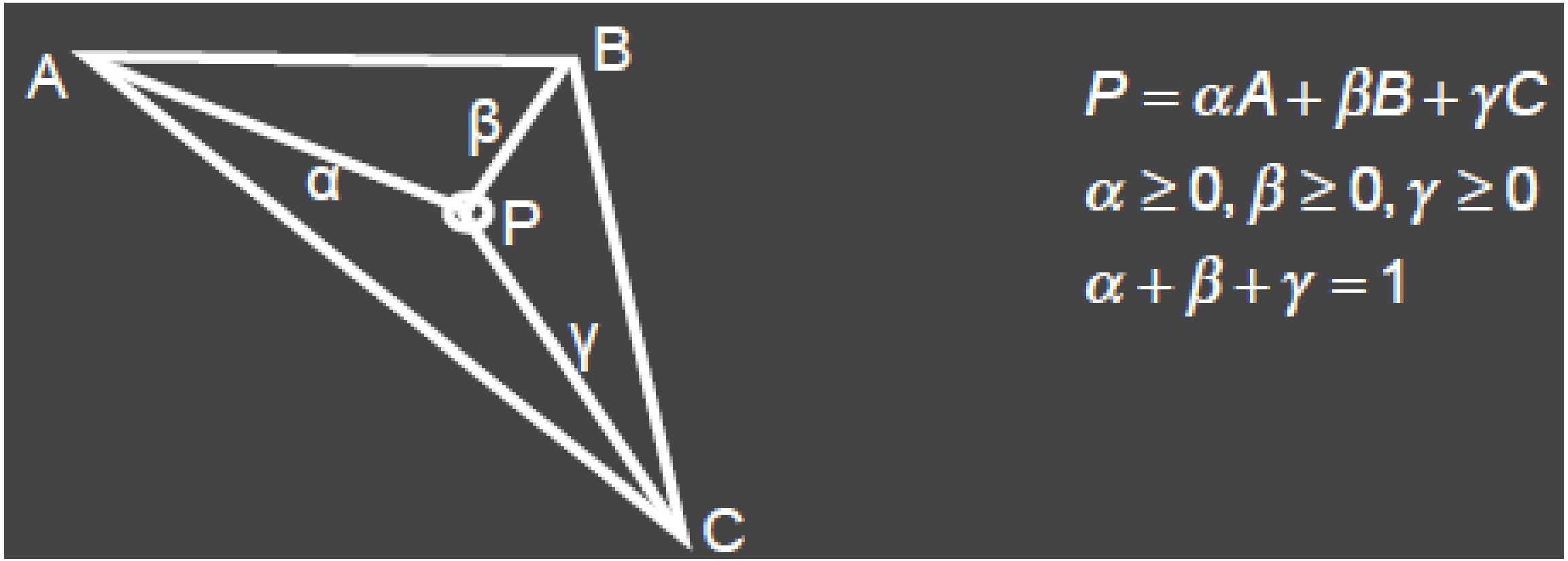
$$ray \equiv \vec{P} = \vec{P}_0 + \vec{P}_1 t$$

$$(\vec{P}_0 + \vec{P}_1 t) \bullet \vec{n} = \vec{A} \bullet \vec{n}$$



Ray inside Triangle

- Once intersect with plane, need to find if in triangle
- Many possibilities for triangles, general polygons
- We find parametrically [barycentric coordinates]. Also useful for other applications (texture mapping)



Other primitives

- Much early work in ray tracing focused on ray-primitive intersection tests
- Cones, cylinders, ellipsoids
- Boxes (especially useful for bounding boxes)
- General planar polygons
- Many more

Outline

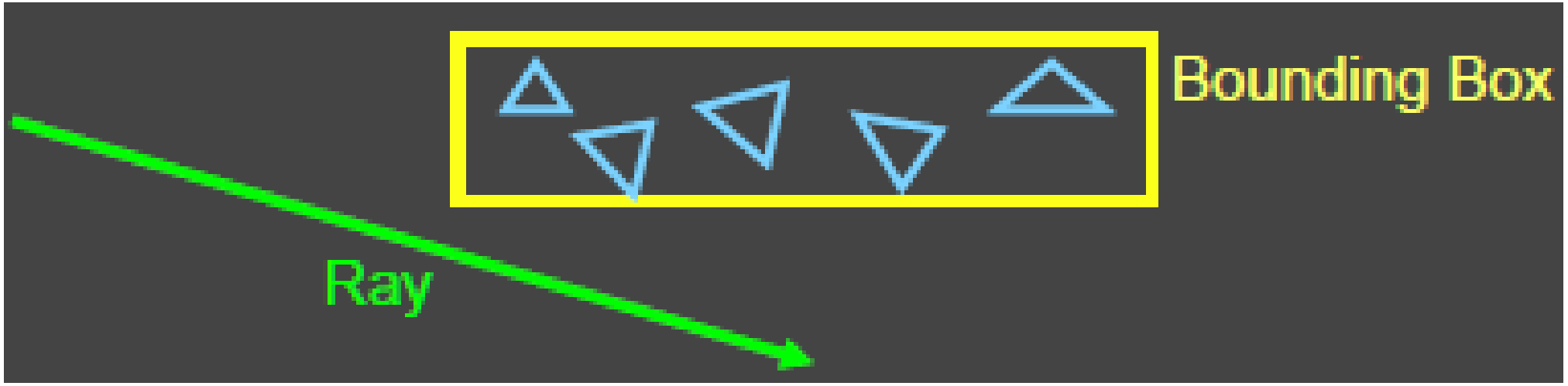
- History
- Basic Ray Casting (instead of rasterization)
  - Comparison to hardware scan conversion
- Camera Ray Casting *(choose ray directions)*
- Shadows / Reflections (core algorithm)
- Ray-Surface Intersection
  - Ray-object intersections
  - Ray-tracing transformed objects
- Optimizations
- Lighting calculations

Acceleration

- Testing each object for each ray is slow
  - Fewer Rays
    - Adaptive sampling, depth control
  - Generalized Rays
    - Beam tracing, cone tracing, pencil tracing etc.
  - Faster Intersections (more on this later)
    - Optimized Ray-Object Intersections
    - ***Fewer Intersections***

Acceleration Structures

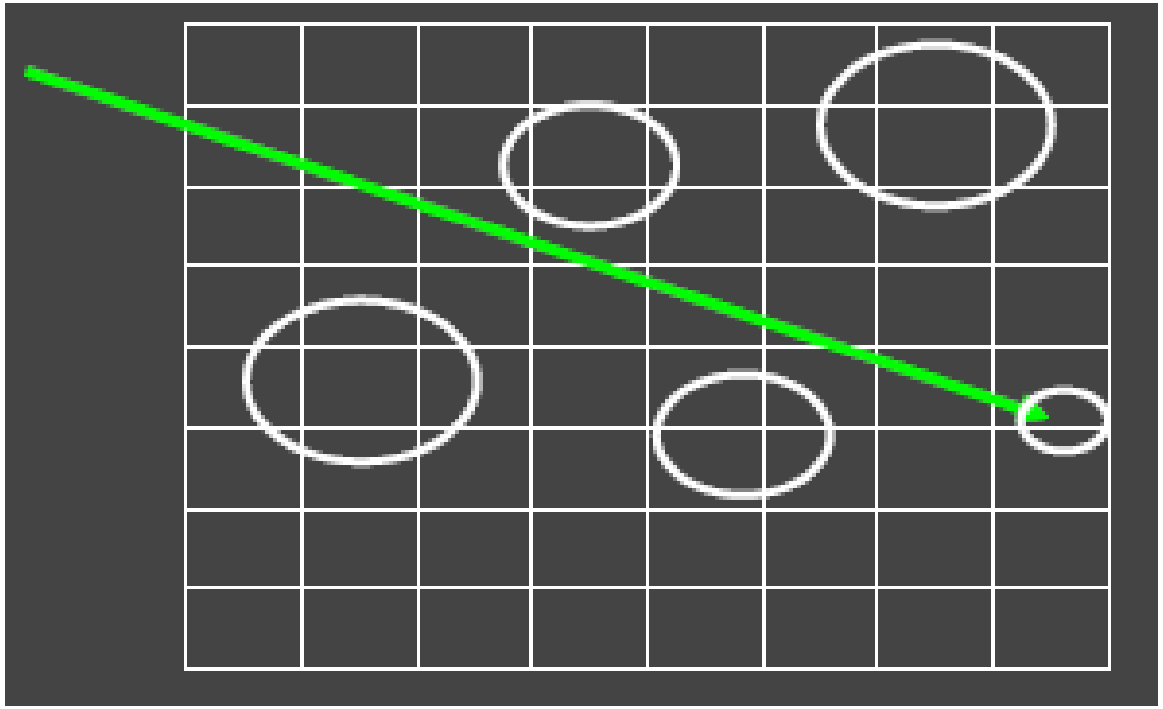
- Bounding boxes (possibly hierarchical)
  - If no intersection bounding box, needn't check objects



- Spatial Hierarchies (Oct-trees, kd trees, BSP trees)

Acceleration and Regular Grids

- Simplest acceleration, for example 5x5x5 grid
- For each grid cell, store overlapping triangles
- March ray along grid (need to be careful with this), test against each triangle in grid cell
- More sophisticated: kd-tree, oct-tree bsp-tree
- Or use (hierarchical) bounding boxes



Outline

- History
- Basic Ray Casting (instead of rasterization)
  - Comparison to hardware scan conversion
- Camera Ray Casting *(choose ray directions)*
- Shadows / Reflections (core algorithm)
- Ray-Surface Intersection
  - Ray-object intersections
  - Ray-tracing transformed objects
- Optimizations
- Lighting calculations

Lighting Model

- Similar to OpenGL
- Lighting model parameters (global)
  - Ambient r g b
  - Attenuation const linear quadratic
- Per light model parameters
  - Directional light (direction, RGB parameters)
  - Point light (location, RGB parameters)
  - Some differences from HW 2 syntax

$$L = \frac{L_0}{const + lin * d + quad * d^2}$$

Material Model

- Diffuse reflectance (r g b)
- Specular reflectance (r g b)
- Shininess s
- Emission (r g b)
- All as in OpenGL

Shading Model

$$\boldsymbol{I = k_a L_a + L_e + \sum_{i=1}^{n} L_i (k_d max(l \cdot n, 0) + k_s max(r \cdot v, 0)^{\alpha})}$$

- Global ambient term, emission from material
- For each light, diffuse specular terms
- Note visibility/shadowing for each light (not in OpenGL)
- Evaluated per pixel per light (not per vertex)

l = unit vector to light; v = l reflected about n; n = surface normal; v = vector to viewer

Recursive Ray Tracing

- For each pixel
  - Trace Primary Eye Ray, find intersection
  - Trace Secondary Shadow Ray(s) to all light(s)
    - Color = Visible ? Illumination Model : 0 ;
  - Trace Reflected Ray
    - Color += reflectivity * Color of reflected ray

Recursive Shading Model

$$I = k_a L_a + L_e + \sum_{i=1}^{n} L_i (k_d max(l \cdot n, 0) + k_s max(r \cdot v, 0)^{\alpha})$$

$$I_{final} = I + k_s L_r + k_T L_T$$

- Highlighted terms are recursive specularities [mirror reflections] and transmission (latter is extra)
- Trace secondary rays for mirror reflections and refractions, include contribution in lighting model
- GetColor calls RayTrace recursively (the I values in equation above of secondary rays are obtained by recursive calls)