CSCI 420 Computer Graphics
Lecture 15

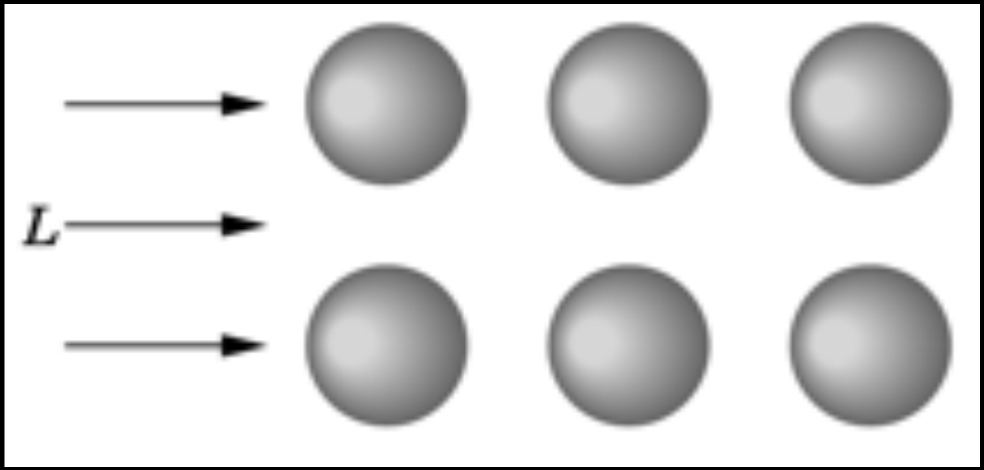
Ray Tracing

Ray Casting
Shadow Rays
Reflection and Transmission
[Angel Ch. 11]

Oded Stein
University of Southern California

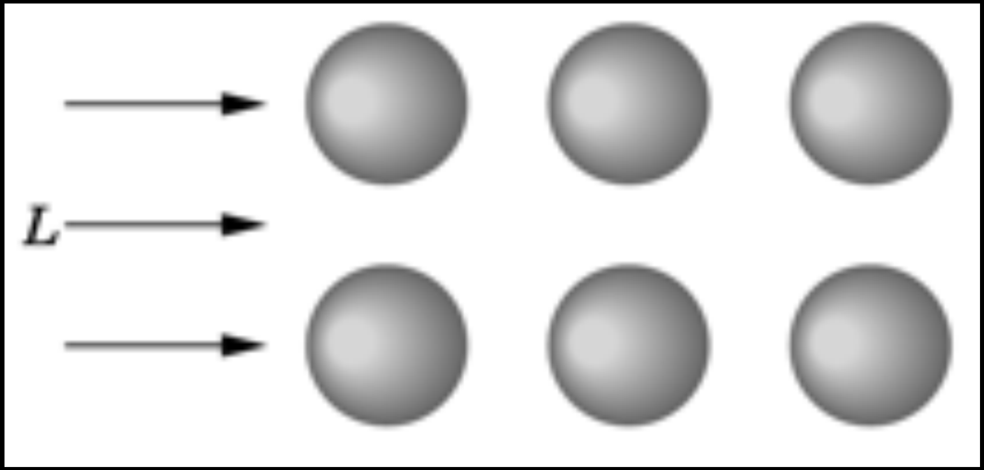
Local Illumination

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



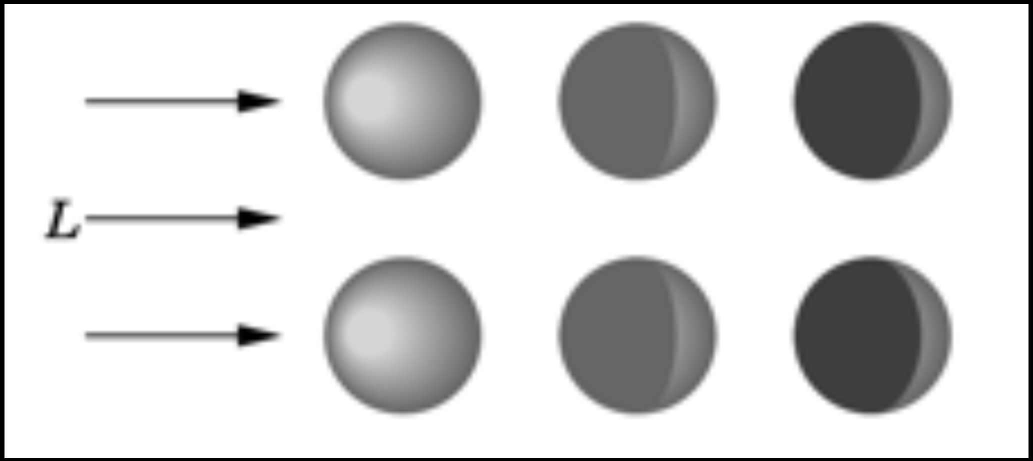
Local Illumination

- Each vertex shader runs independently
- Each fragment shader runs independently
- We have some interaction when depth buffering happens.
- All other object interactions require some kind of hack.



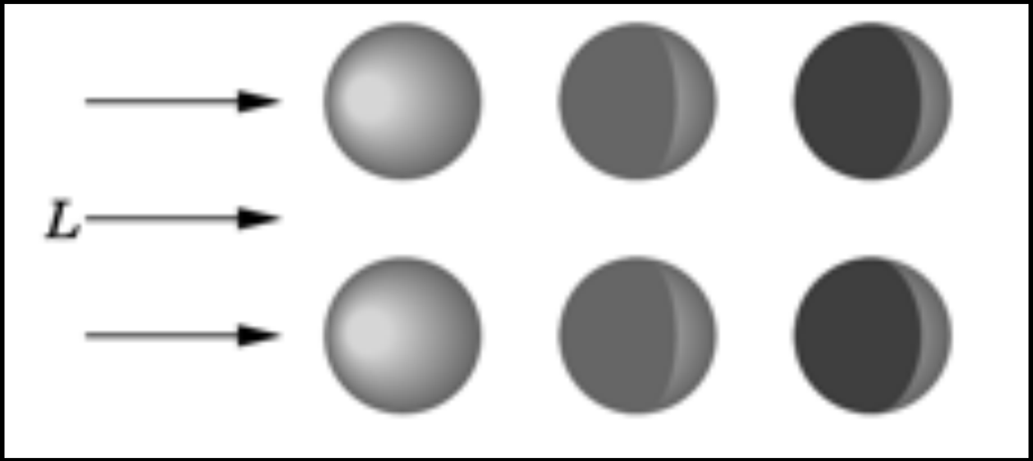
Global Illumination

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



Global Illumination

- In the real world...
  - Surfaces obstruct light.
  - Surfaces reflect light onto other surfaces.
  - Surfaces scatter light.



Object Space:

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

Image Space:

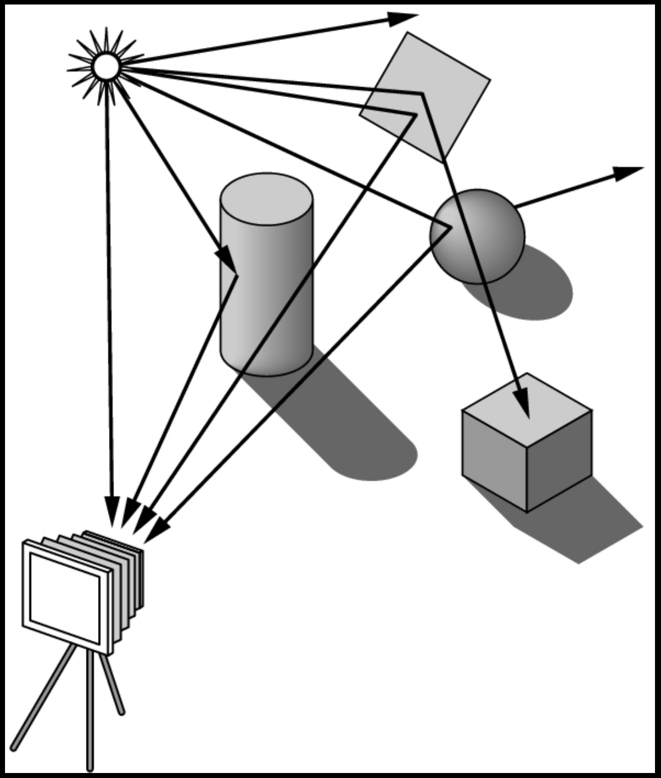
- Ray tracing: for each pixel, determine color
  - Pixel-level parallelism
  - Difficulty: very intensive computation, usually off-line

So let's simulate physics!

- Simulating physics is what computers are for!
- Physically trace every single ray of light.
- All light comes from light sources.
- Light that makes it to the camera film eventually gets written into the output image.

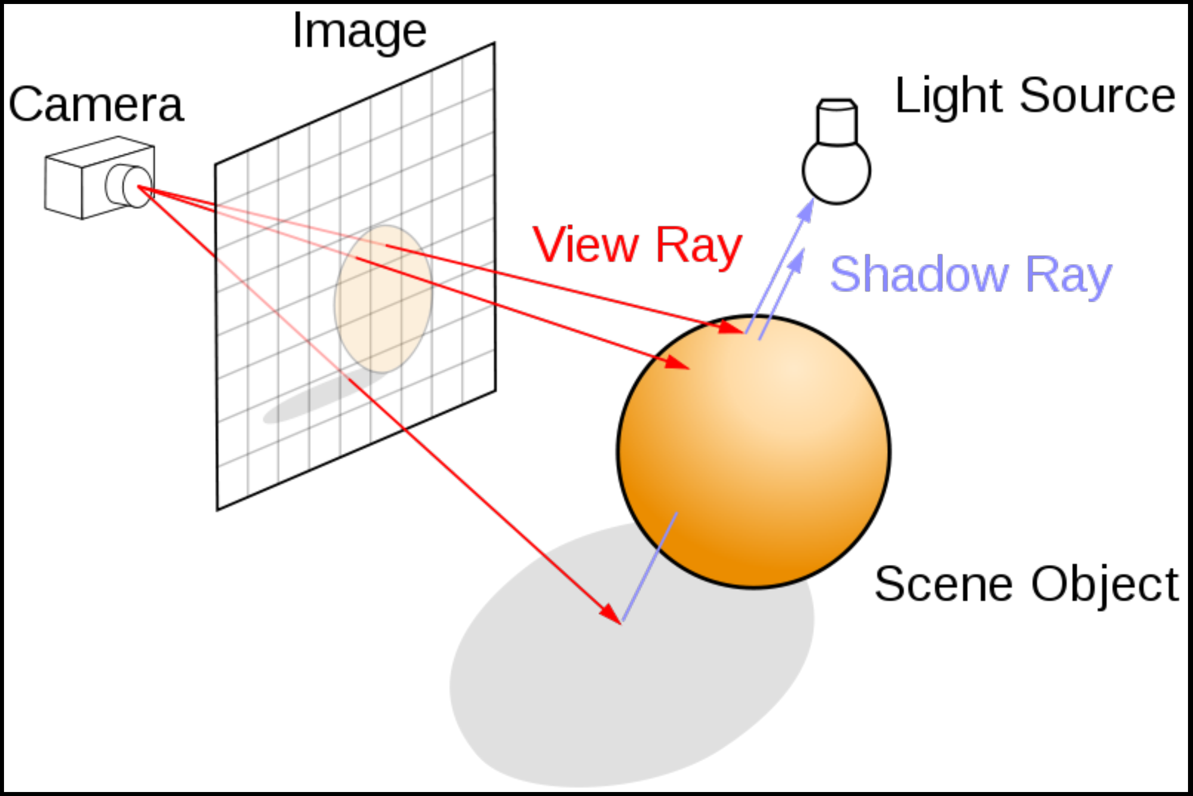
First idea: Forward Ray Tracing

- 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
  - Really, really not practical.



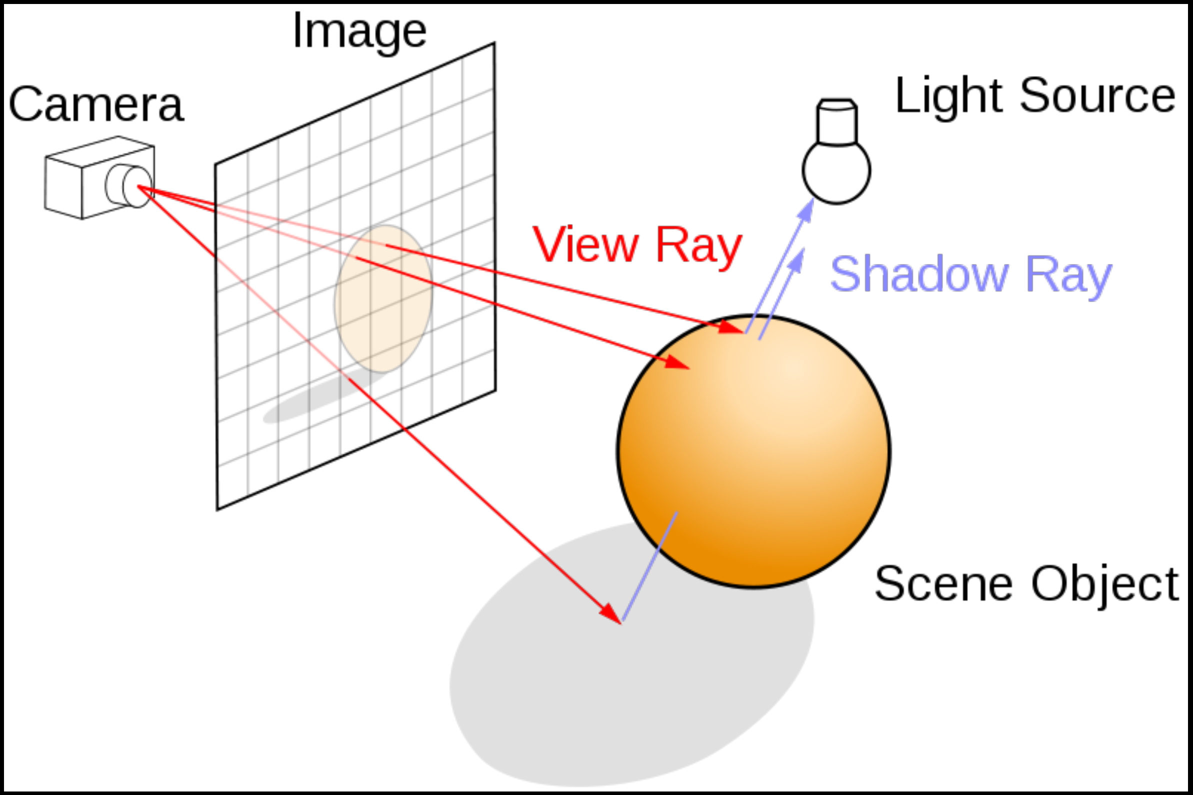
Backward Ray Tracing

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



Backward Ray Tracing

- Shoot one ray from camera through each pixel in image plane
  - ...what's the physical problem here?



Backward Ray Tracing

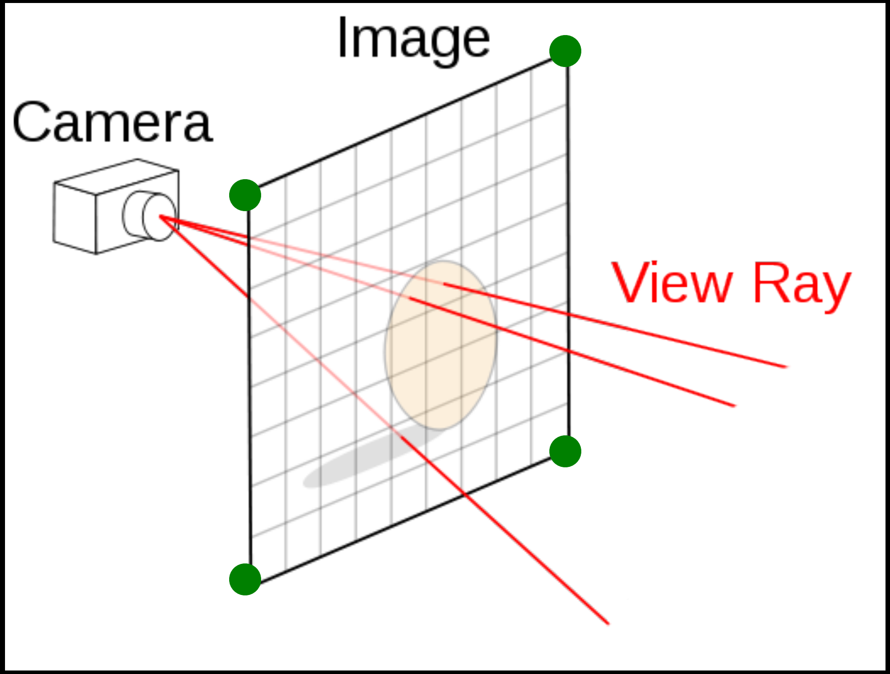
- Shoot one ray from camera through each pixel in image plane
  - ...what's the physical problem here?
  - ...if we actually simulated every single backward ray correctly, this would also not be computationally practical.
  - In practice, we simulate only up to a certain amount of bounces.

How does backward ray tracing work?

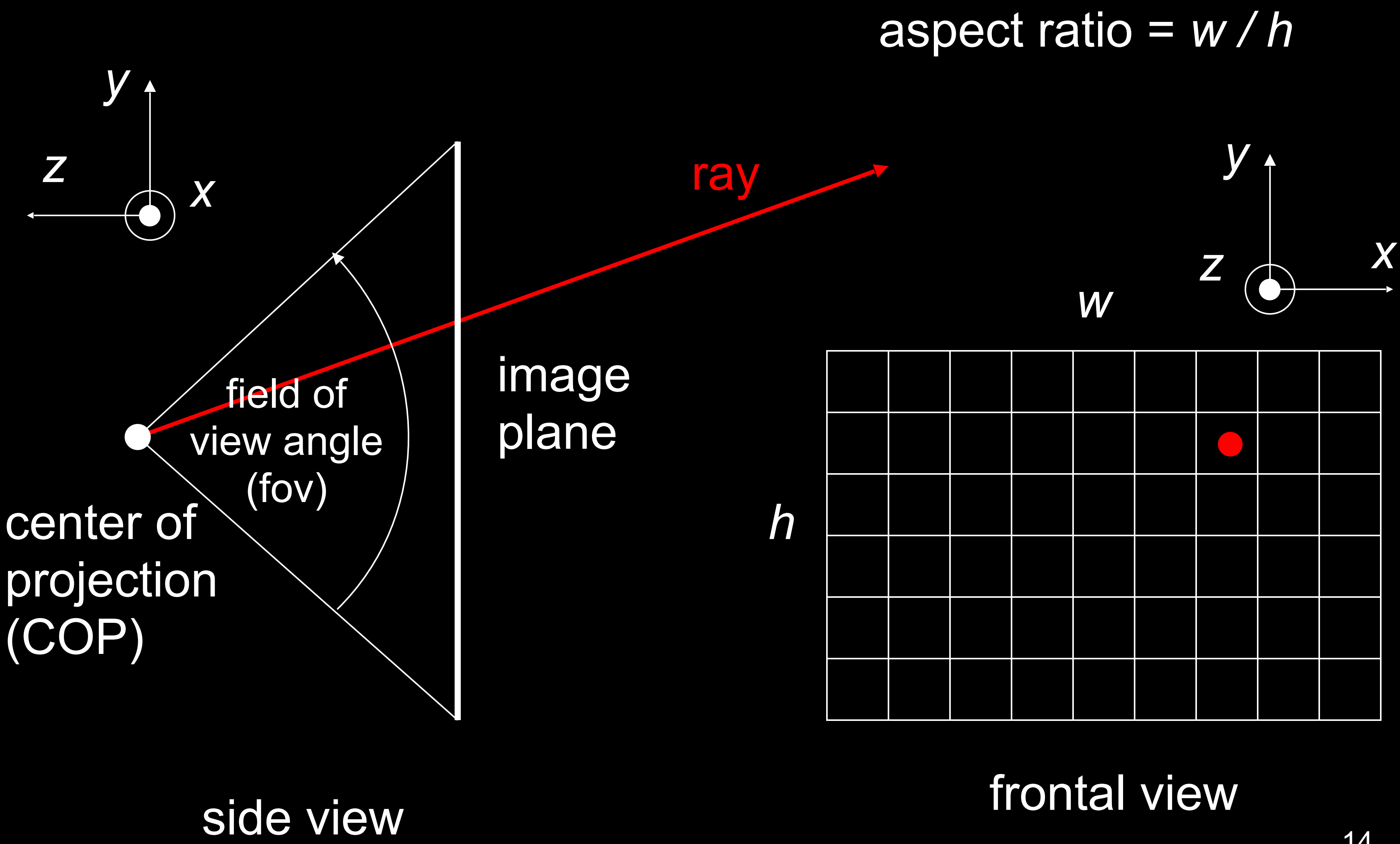
- You will implement a basic backward ray tracing renderer in your homework.
- This is an algorithm that requires almost no libraries, hardware, etc...
  - Unlike OpenGL
  - You only need to be able to write to an image file as output.

Generating Rays

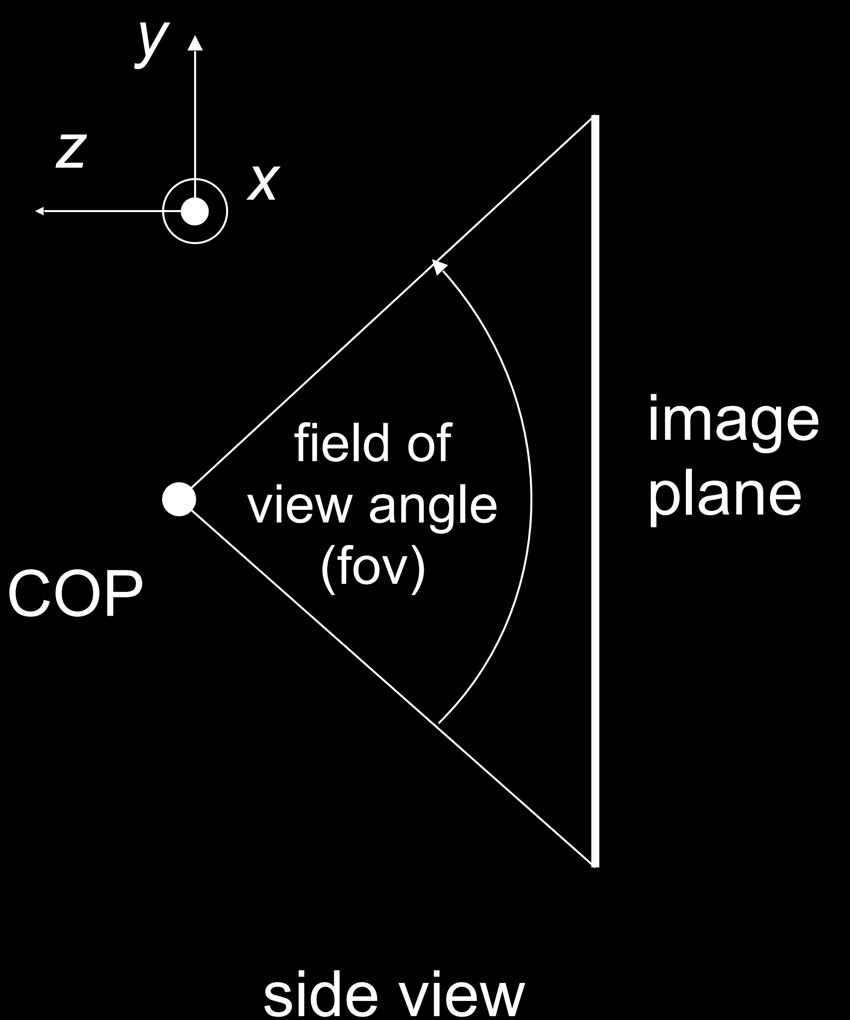
- Camera is at (0,0,0) and points in the negative z-direction
- Must determine coordinates of image corners in 3D

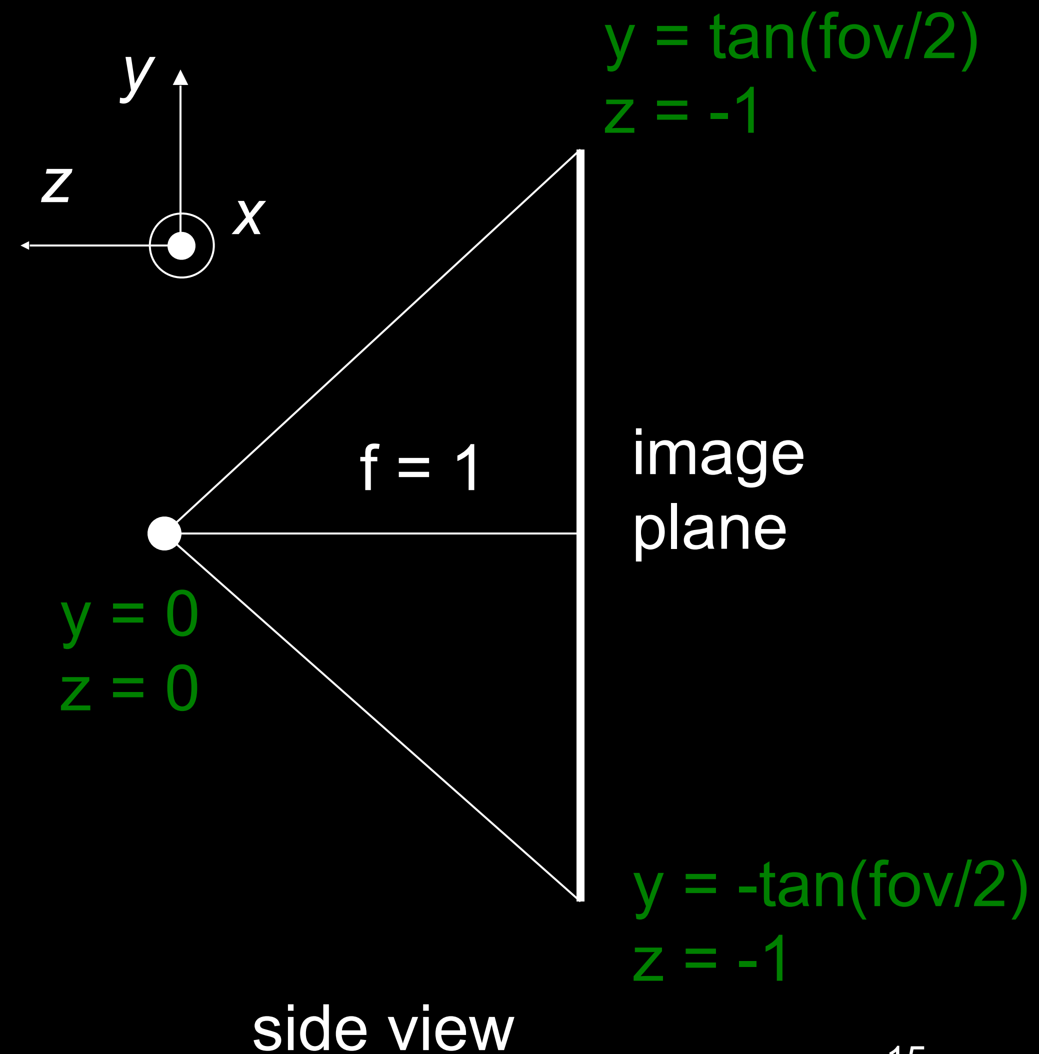


Generating Rays

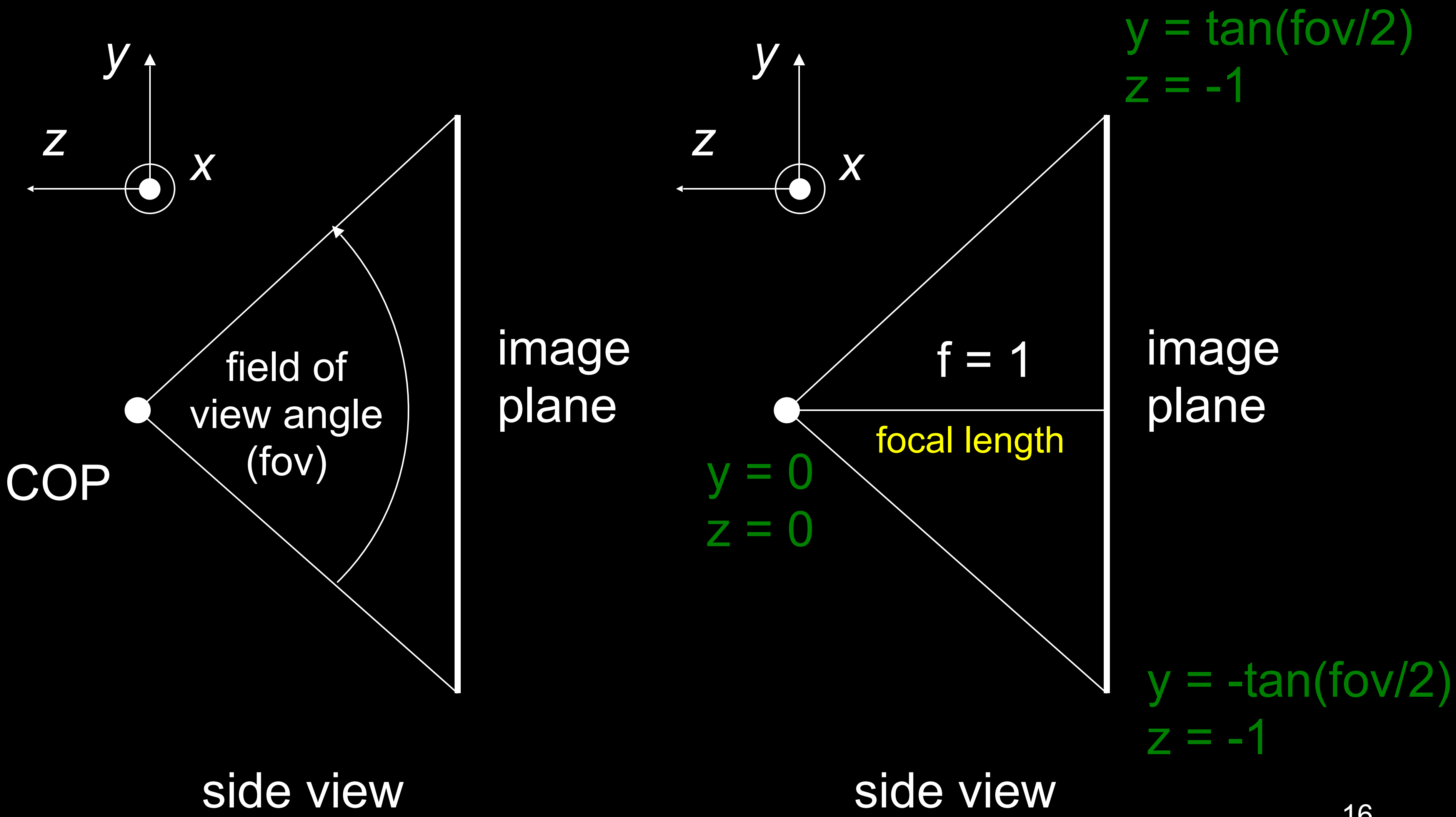


Generating Rays

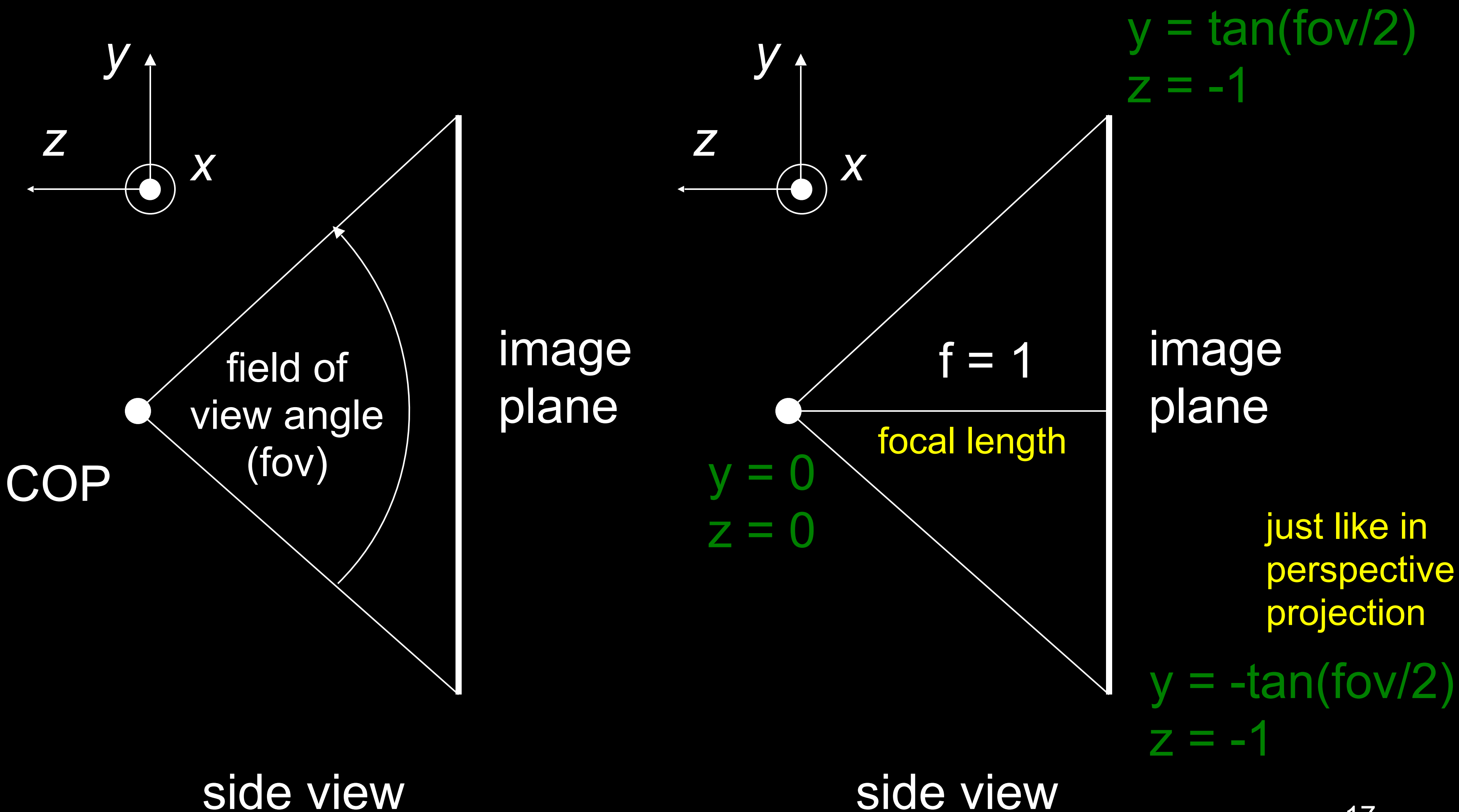




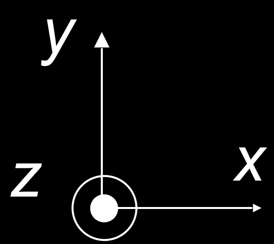
Generating Rays



Generating Rays



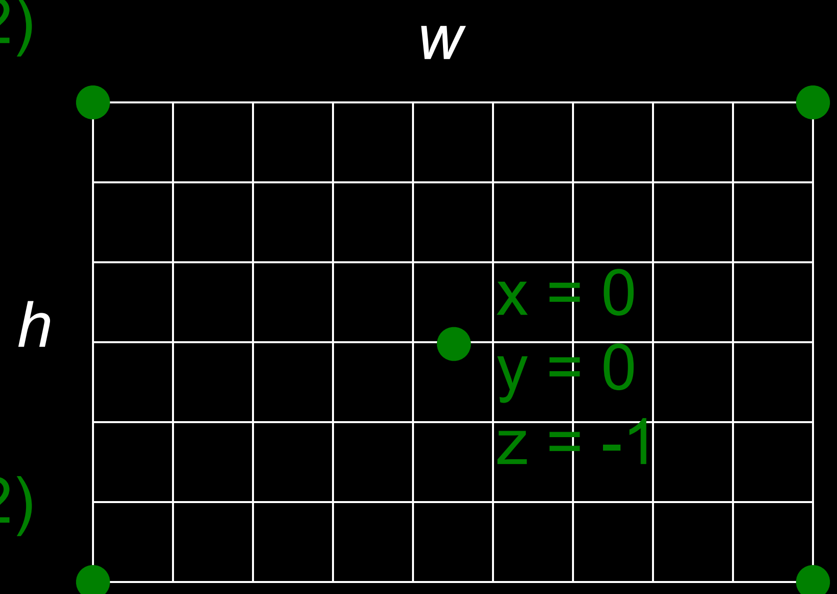
Generating Rays



a = aspect ratio = w / h

$x = -a \tan(fov/2)$
$y = \tan(fov/2)$
$z = -1$

$x = a \tan(fov/2)$
$y = \tan(fov/2)$
$z = -1$



$x = -a \tan(fov/2)$
$y = -\tan(fov/2)$
$z = -1$

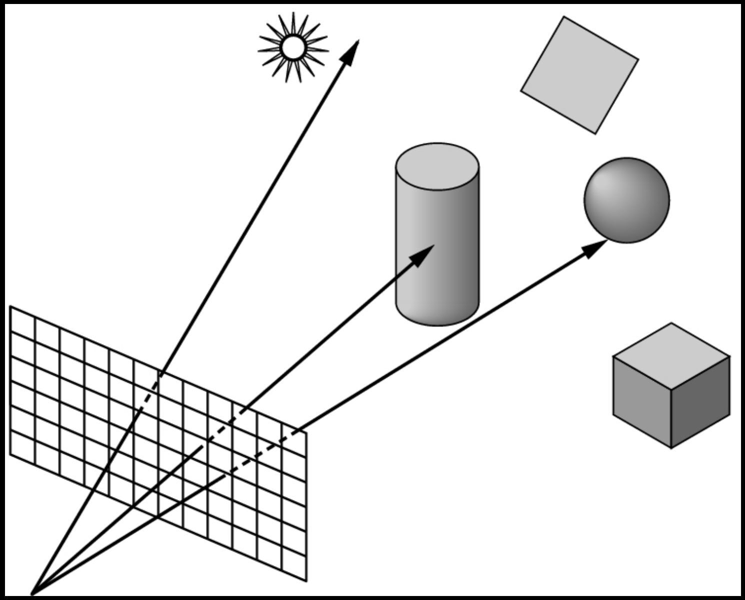
$x = a \tan(fov/2)$
$y = -\tan(fov/2)$
$z = -1$

frontal view

Determining Pixel Color

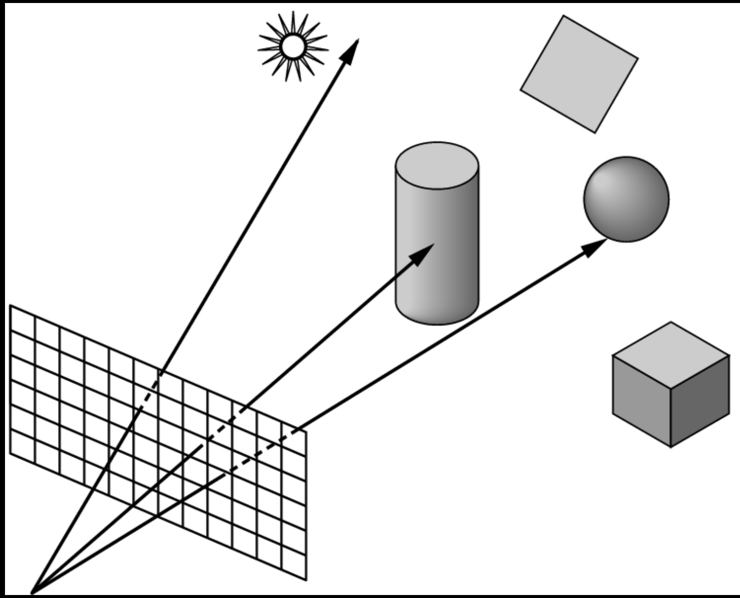
1. Phong model (local as before)
2. Shadow rays
3. Specular reflection
4. Specular transmission

Steps (3) and (4) require recursion.



Determining Pixel Color

1. Phong model (local as before)
2. Shadow rays
3. Specular reflection
4. Specular transmission



Later:

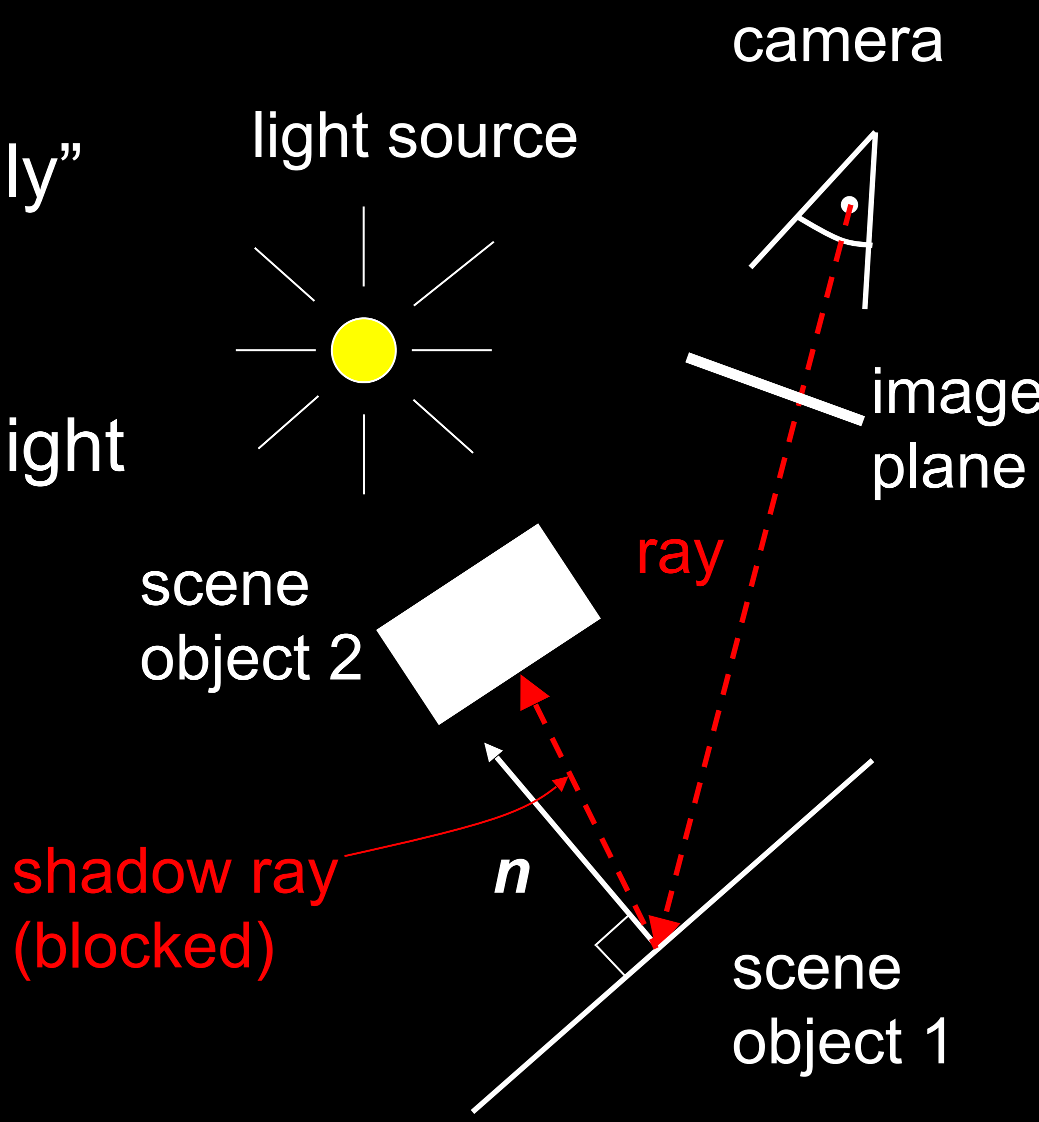
5. Metallic reflection
6. Scattering (subsurface)

Shadow Rays

- Previously, in local OpenGL illumination...
- At every point, we loop through all lights and add their contribution with the Blinn/Phong lighting model.
- But now that we raytrace, we can check whether each light *actually* hits the point we are drawing.

Shadow Rays

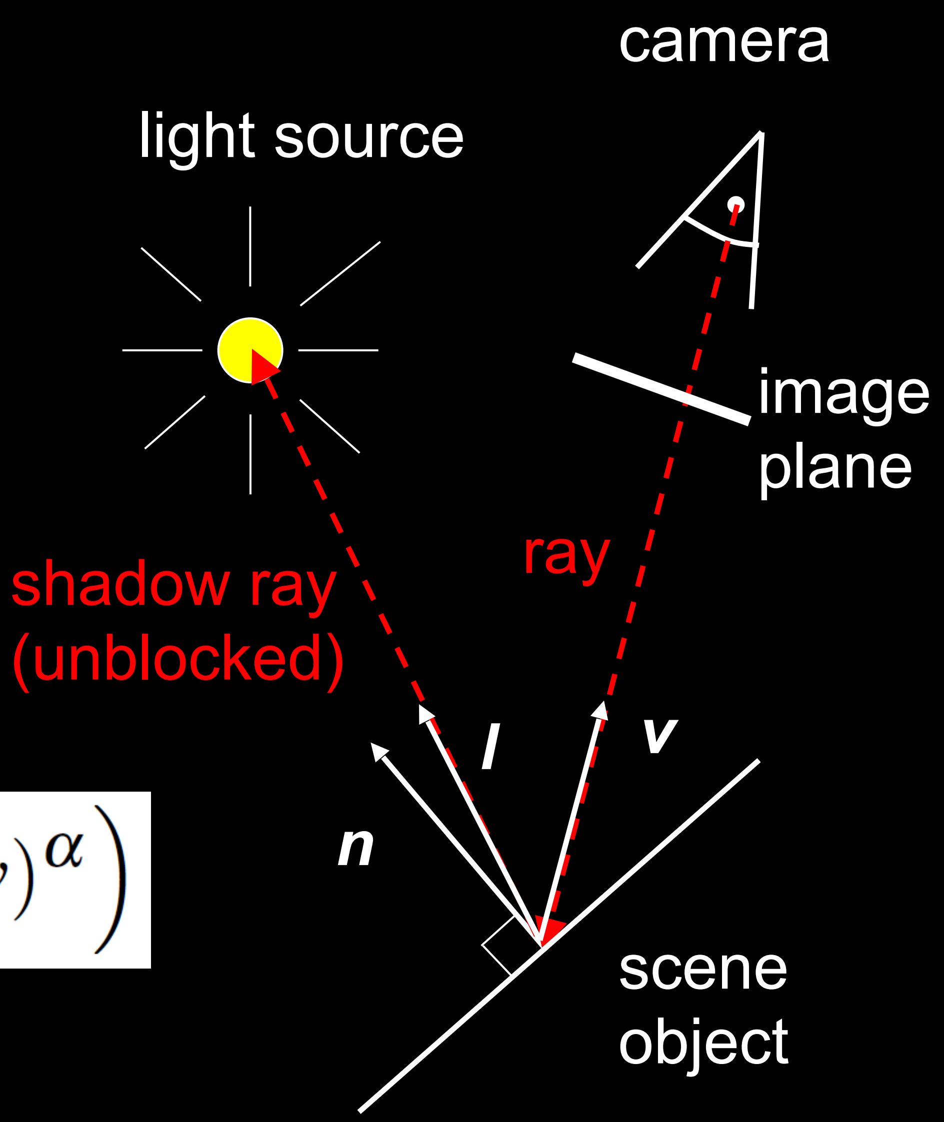
- Determine if light “really” hits surface point
- Cast shadow ray from surface point to each light
- If shadow ray hits opaque object, no contribution from that light
- This is essentially improved diffuse reflection



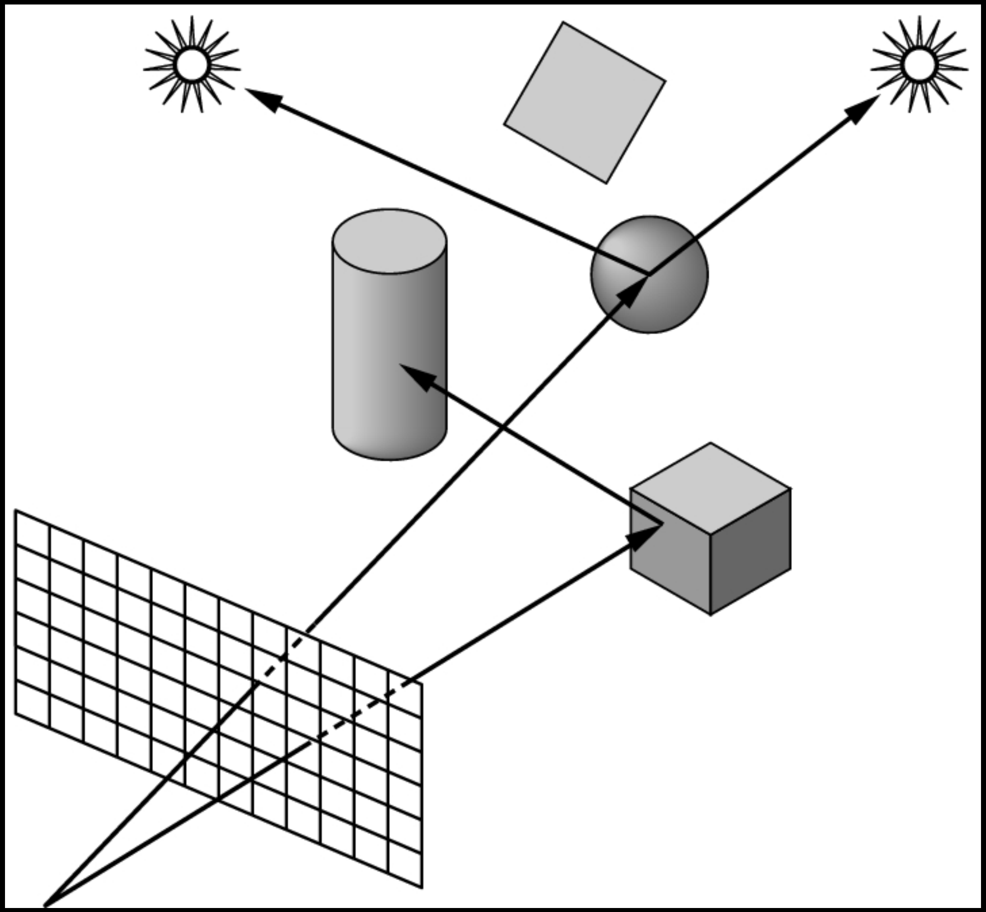
Phong Model

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

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

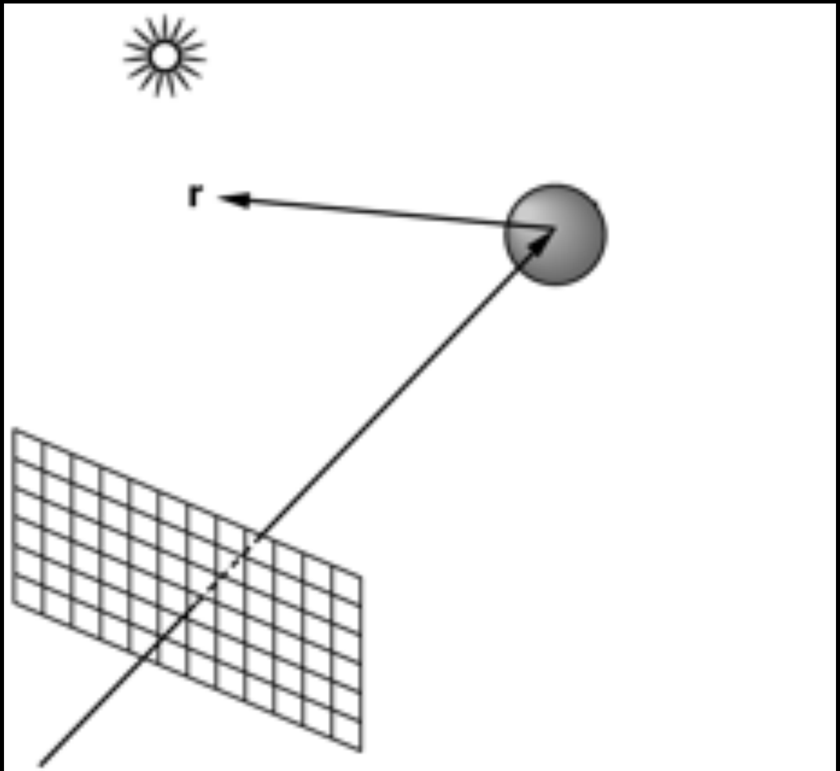


Where is Phong model applied
in this example?
Which shadow rays are blocked?



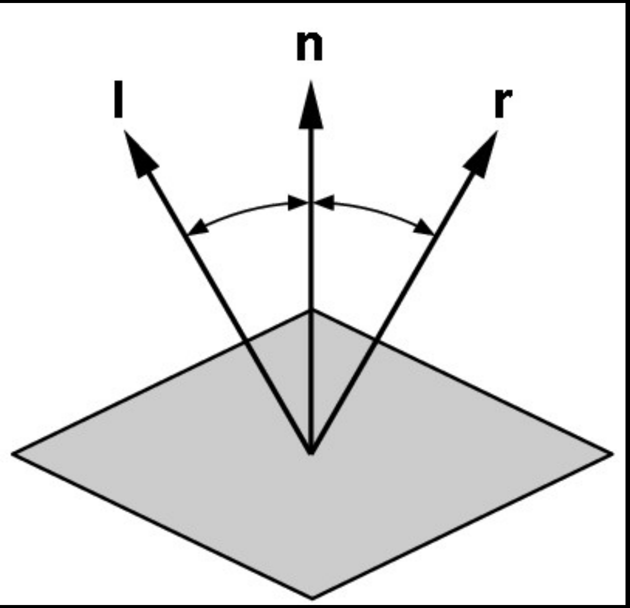
Reflection Rays

- For specular component of illumination
- Compute reflection ray (recall: backward!)
- Call ray tracer recursively to determine color

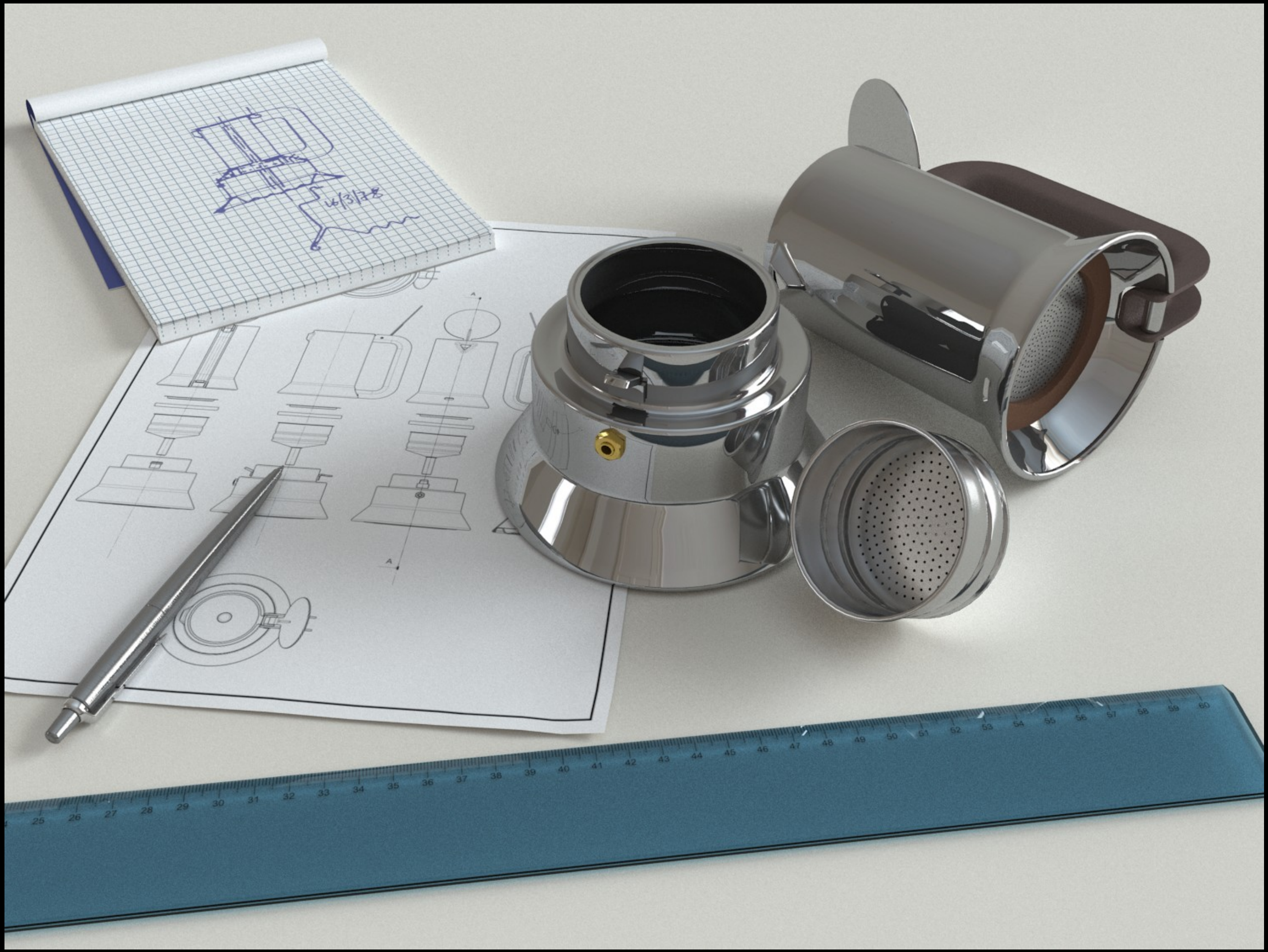


Angle of Reflection

- Recall: incoming angle = outgoing angle
- $\mathbf{r} = 2(\mathbf{l} \cdot \mathbf{n})\, \mathbf{n} - \mathbf{l}$
- Compute only for surfaces that are reflective



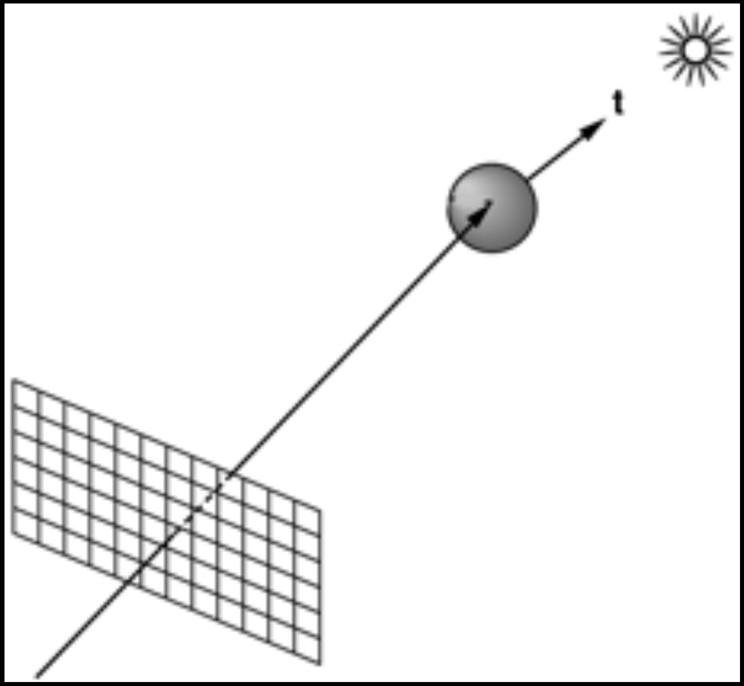
Reflections Example



www.yafaray.org

Transmission Rays

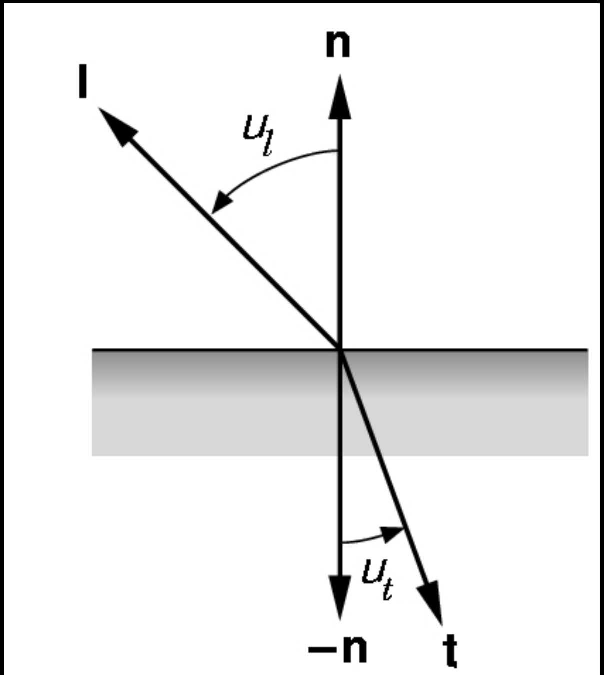
- Calculate light transmitted through surfaces
- Example: water, glass
- Compute transmission ray
- Call ray tracer recursively to determine color



Transmitted Light

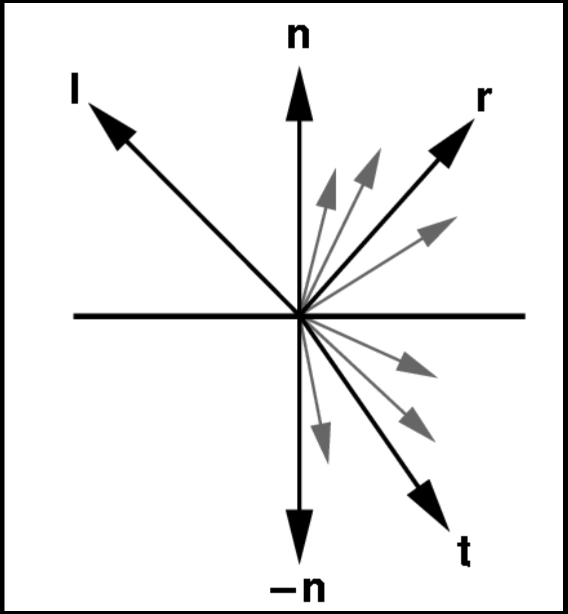
- Index of refraction is speed of light, relative to speed of light in vacuum
  - Vacuum: 1.0 (per definition)
  - Air: 1.000277 (approximate to 1.0)
  - Water: 1.33
  - Glass: 1.49
- Compute t using Snell's law
  - η_l = index for upper material
  - η_t = index for lower material

$$\frac{\sin(u_l)}{\sin(u_t)} = \frac{\eta_t}{\eta_l} = \eta$$



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

The Ray Casting Algorithm

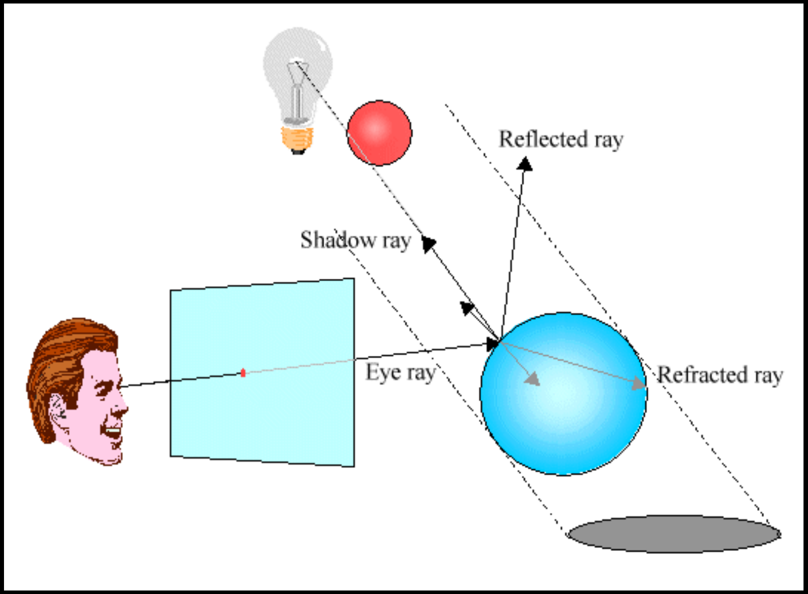
- Simplest case of ray tracing

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**
   - Calculate surface normal
   - For each light source, fire shadow ray
   - 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



Ray Tracing Assessment

- Global illumination method
- Image-based
- Pluses
  - Relatively accurate shadows, reflections, refractions
- Minuses
  - Slow (per pixel parallelism, not pipeline parallelism)
  - Aliasing
  - Inter-object diffuse reflections require many bounces

Raytracing Example I

www.yafaray.org

Raytracing Example II

www.povray.org

Raytracing Example III

www.yafaray.org

Raytracing Example IV

www.povray.org

Summary

- Ray Casting
- Shadow Rays and Local Phong Model
- Reflection
- Transmission

- Next lecture: Geometric queries