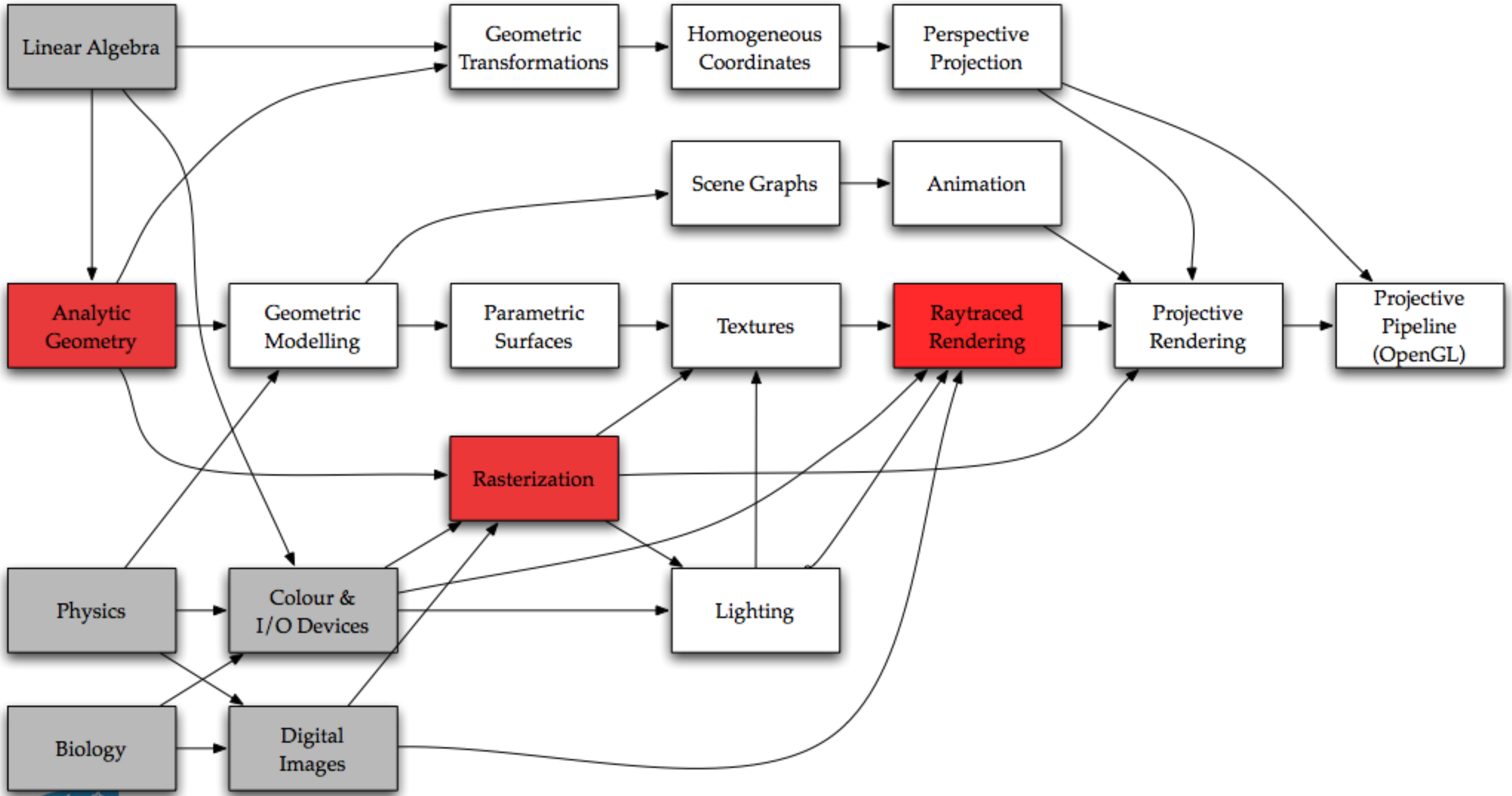
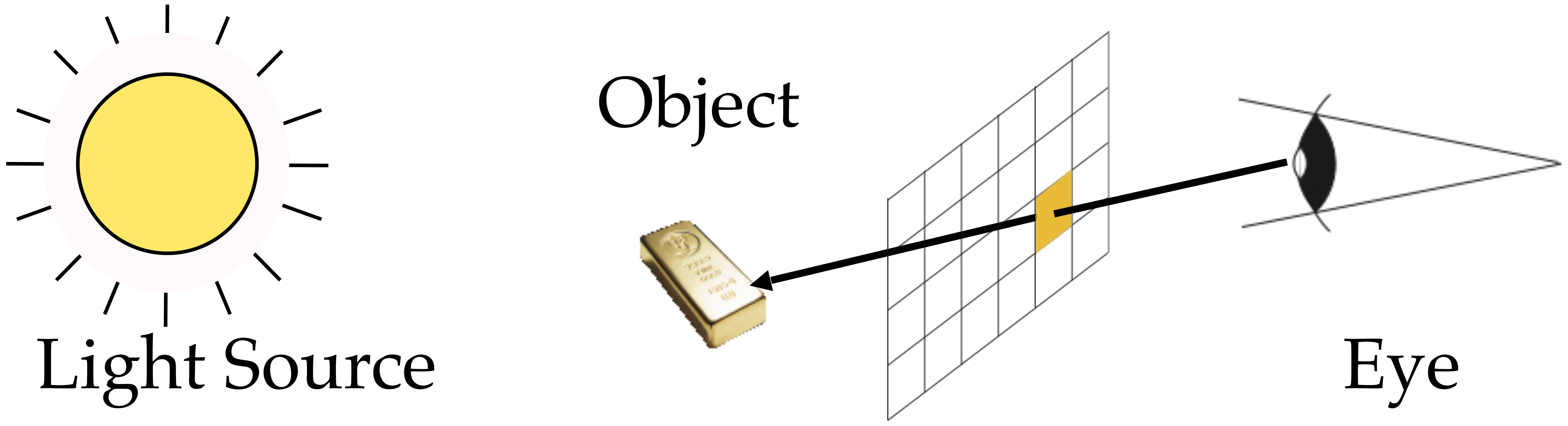
Lighting & Shadows

Where we Are


Linear Algebra
Geometric Transformations
Homogeneous Coordinates
Perspective Projection
Scene Graphs
Animation
Analytic Geometry
Geometric Modelling
Parametric Surfaces
Textures
Raytraced Rendering
Projective Rendering
Projective Pipeline (OpenGL)
Rasterization
Physics
Colour & I/O Devices
Lighting
Biology
Digital Images

Raytracing

- For each pixel
  - Start at eye
  - Trace a ray through image plane
  - Compute colour of object it hits



Reminder

- Light is:
  - *emitted* from a *source*
  - *reflected* from a *surface*
  - *absorbed* by a surface or object

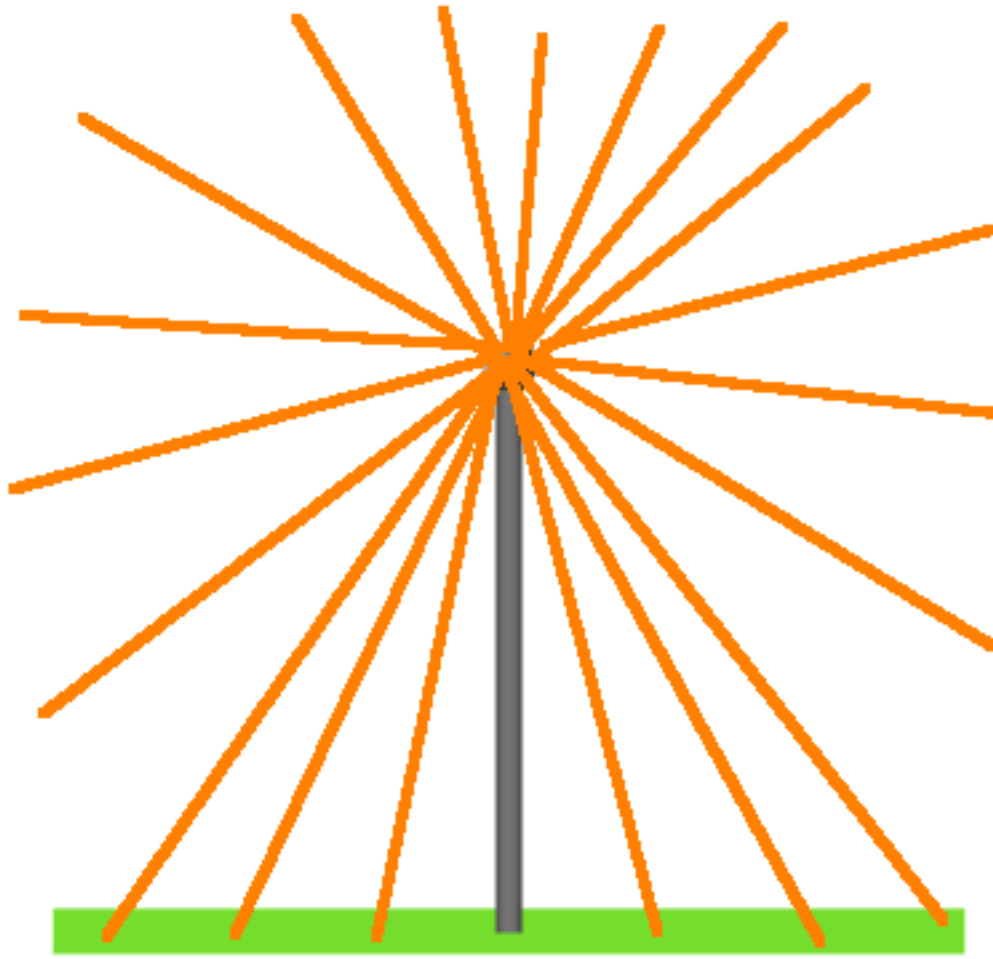
OpenGL models mostly reflected light

Light Sources

- Can be characterized in terms of:
  - where they are (*location*)
  - the light's *geometry*
  - how much light they emit (*intensity*)
  - the *spectral distribution* of the light

Point Sources

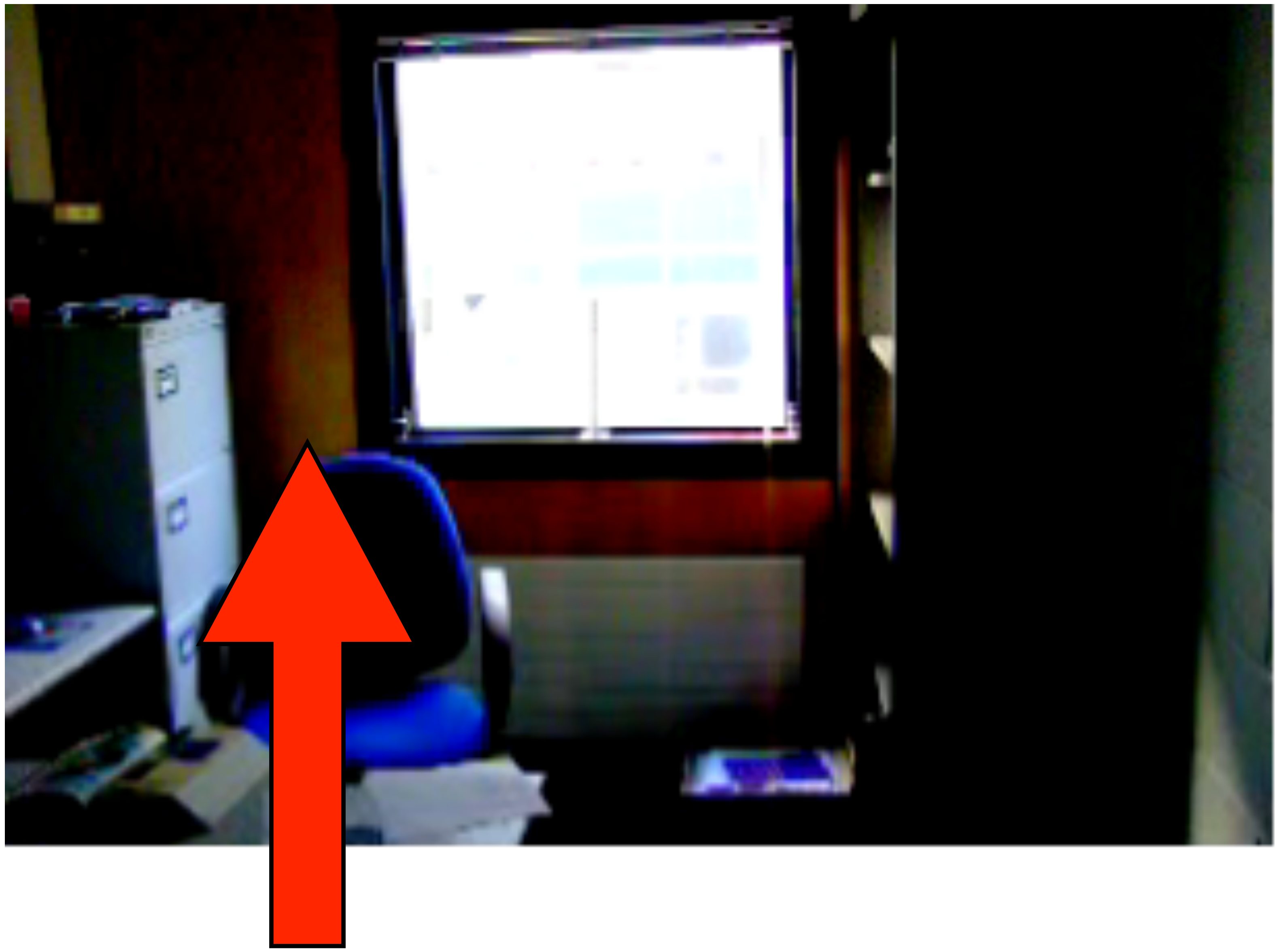
- *Point sources* assume that all the light radiates from a single point
  - easy to model & render
  - simple geometrically
- Distant lamps, the sun, incandescent bulbs, &c.



Other Light Sources

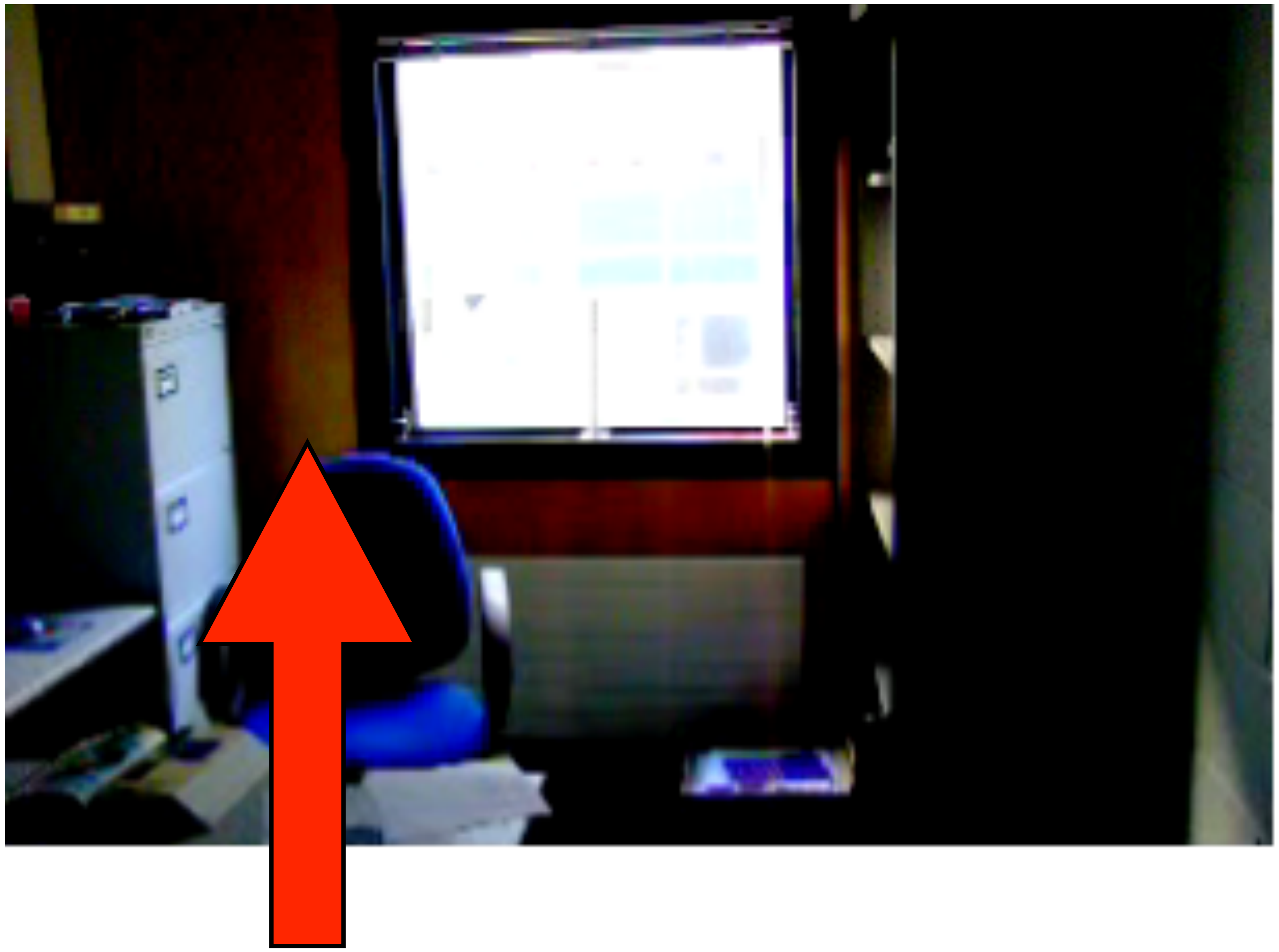
- A frosted bulb
- A fluorescent light
- A gas flame
- A ring on an electric cooker
- An incandescent bulb (close-up)
- We won’t worry about these (for now)

Lighting Example



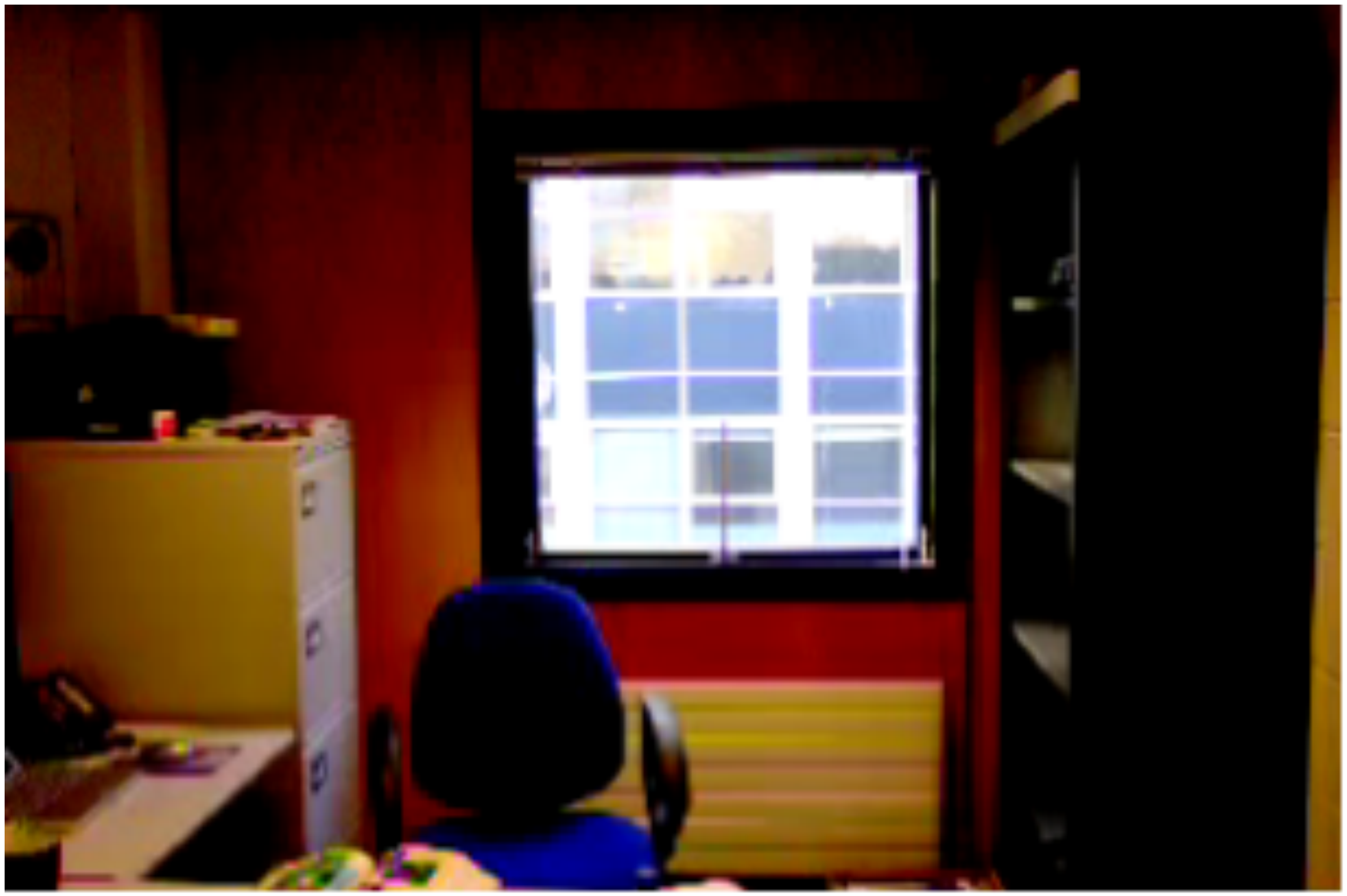
- Why is this patch lit if window is only source?

Lighting Example



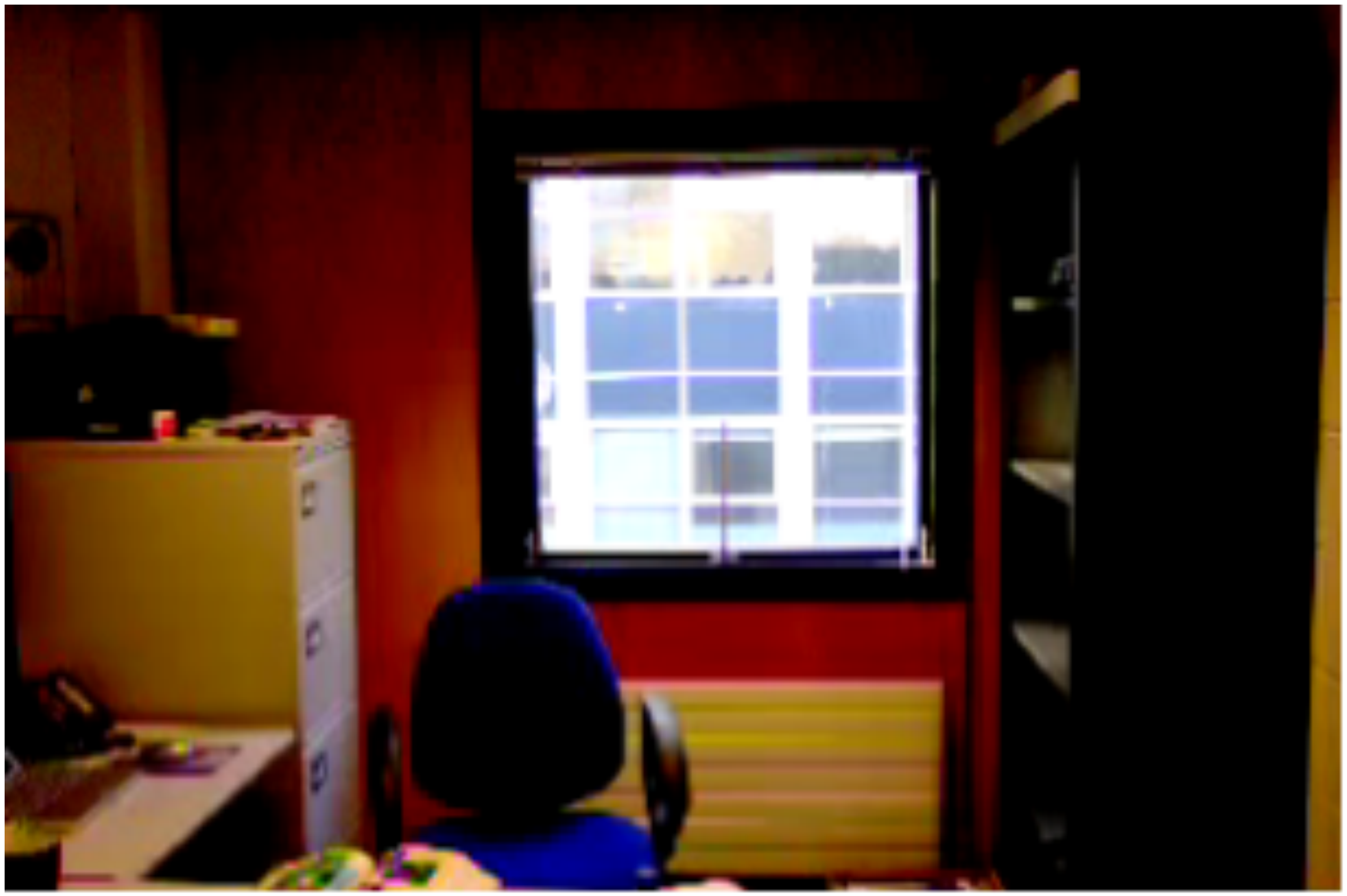
- Light here is reflected from somewhere else

Another Example



- What has changed here?

Another Example



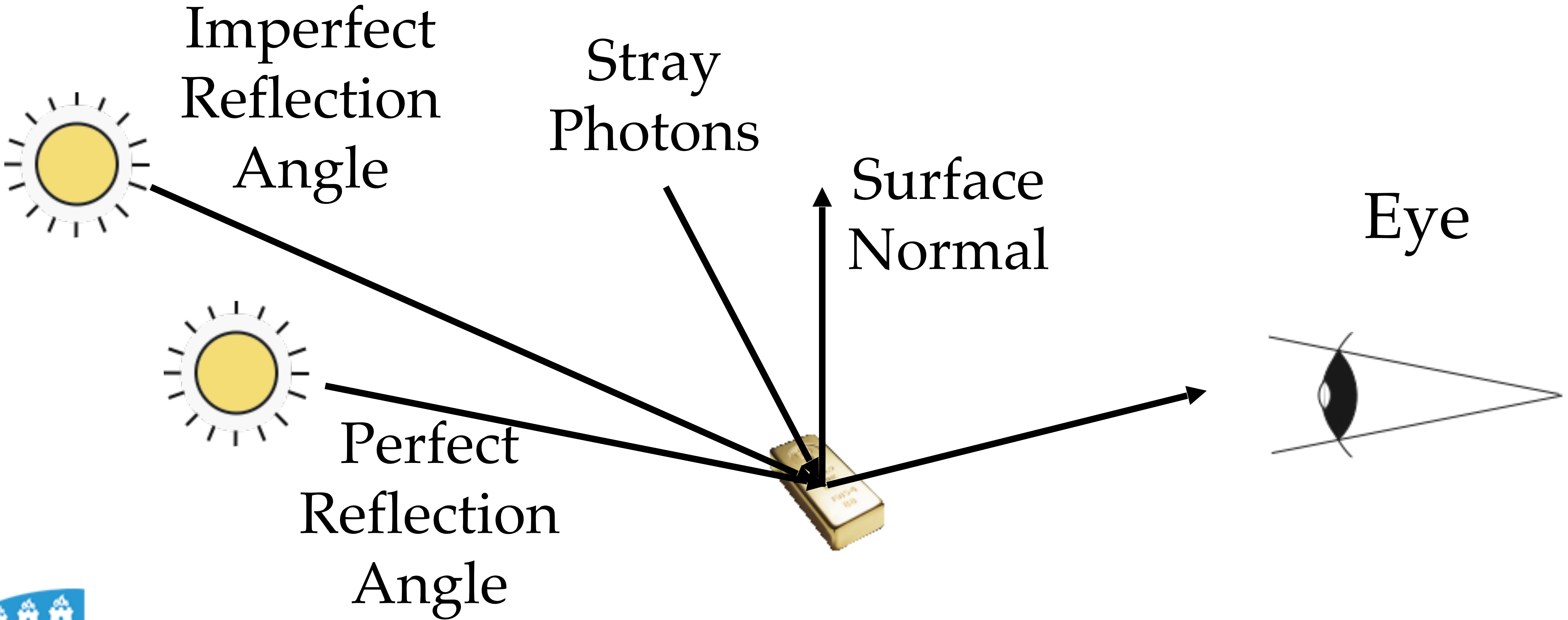
- Now there are multiple light sources

Incoming Light

- At the eye, some light is direct
    - i.e. it comes straight from the source
- The rest bounces around for a while
    - may bounce from ***<u>many</u>*** surfaces
    - may bounce from air molecules
- We need to simplify this behaviour

Origin of Photons

- For any point, light comes from all over



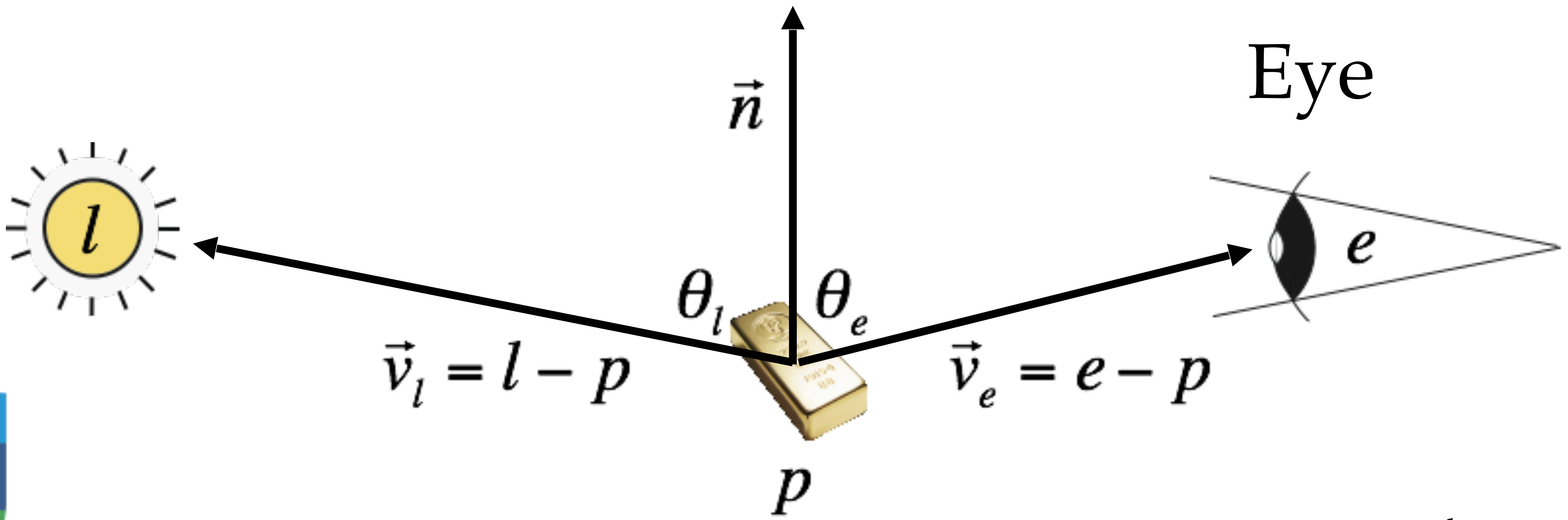
Phong Lighting Model

- Total lighting at a point is:
  - *specular* (shiny) reflection, plus
  - *diffuse* (matt) reflection, plus
  - *ambient* (background) reflection, plus
  - *emitted* light

$$I_{total}(p) = I_{specular}(p) + I_{diffuse}(p) + I_{ambient}(p) + I_{emitted}(p)$$

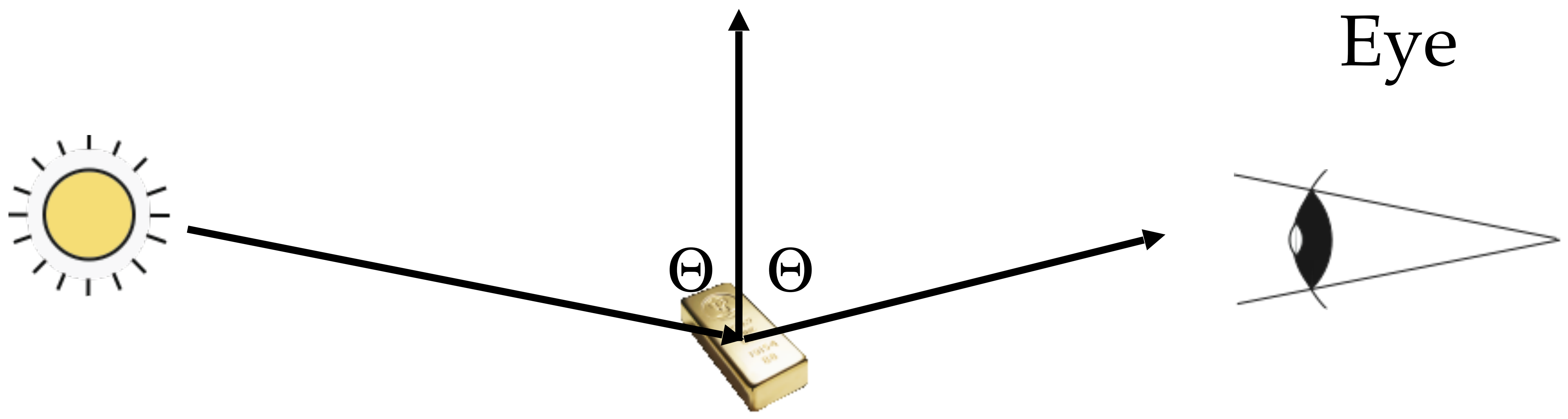
Surface Normal

- The *normal* of a surface is a vector that is perpendicular to the surface
- It is used to measure angle of reflection
- Assume vectors are all *units*



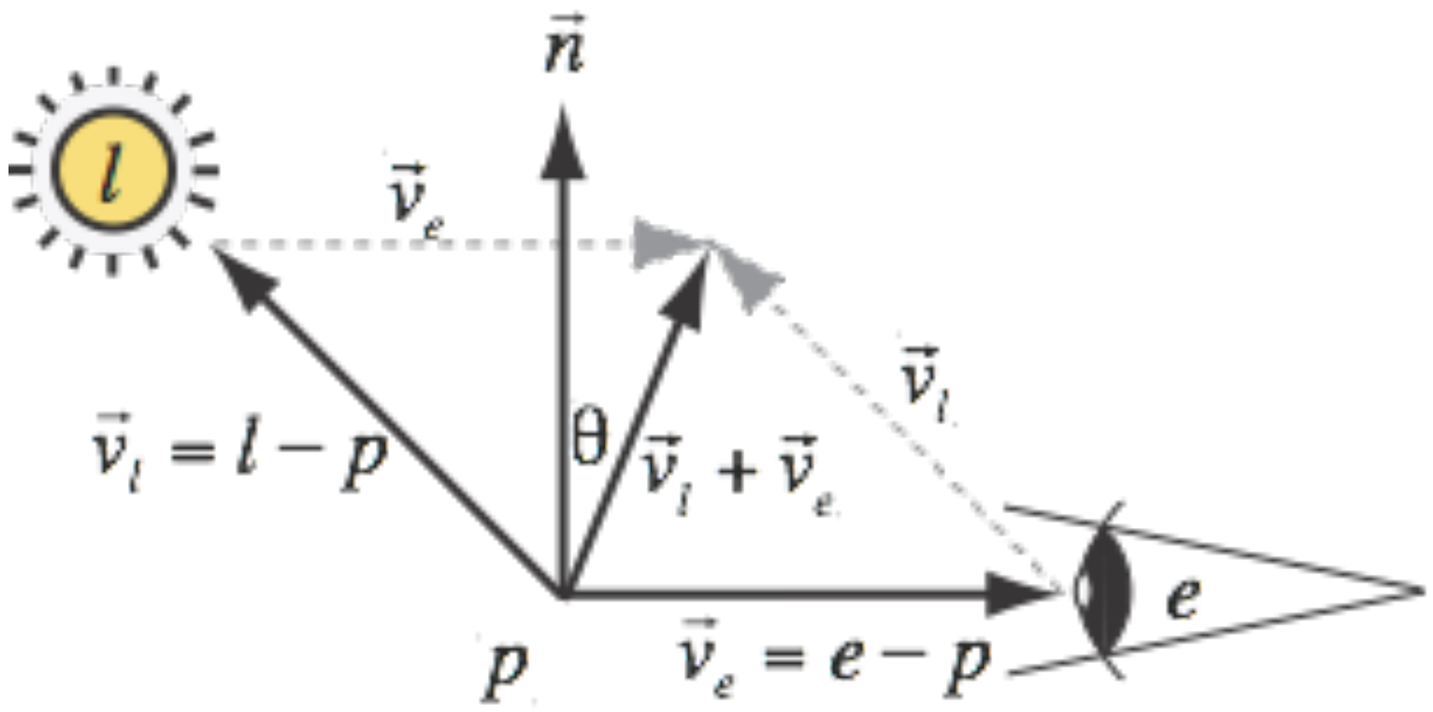
Perfect Reflection

- Angle of incidence = angle of reflection
  - so normal vector is *bisector*



Specular Reflection

- Specular light spreads out a *little* bit
- Reflects strongly for angles *close* to perfect
  - i.e. if the *bisector* is close to *n*



Specular Reflection

- Based on angle between normal and bisector

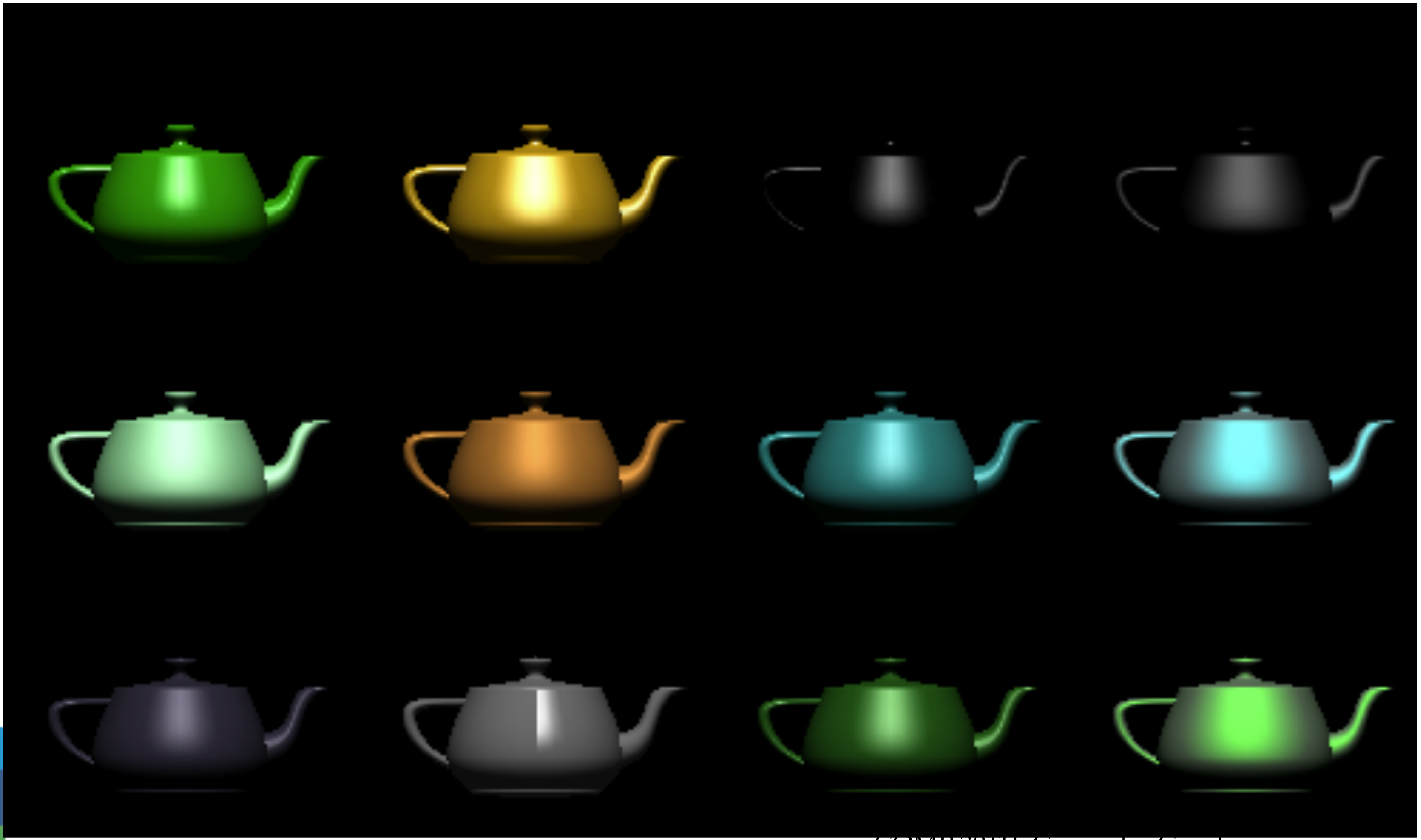
$$\cos\theta = \frac{\vec{n}\bullet\left(\vec{v}_b\right)}{\left\|\vec{n}\right\|\left\|\vec{v}_b\right\|}, \quad \vec{v}_b = \frac{\vec{v}_l + \vec{v}_e}{2}$$

- Raised to an exponent to exaggerate effect

$$I_{specular}\left(p\right) = k_{specular}\left(\frac{\vec{n}\bullet\left(\vec{v}_b\right)}{\left\|\vec{n}\right\|\left\|\vec{v}_b\right\|}\right)^{n_s}$$

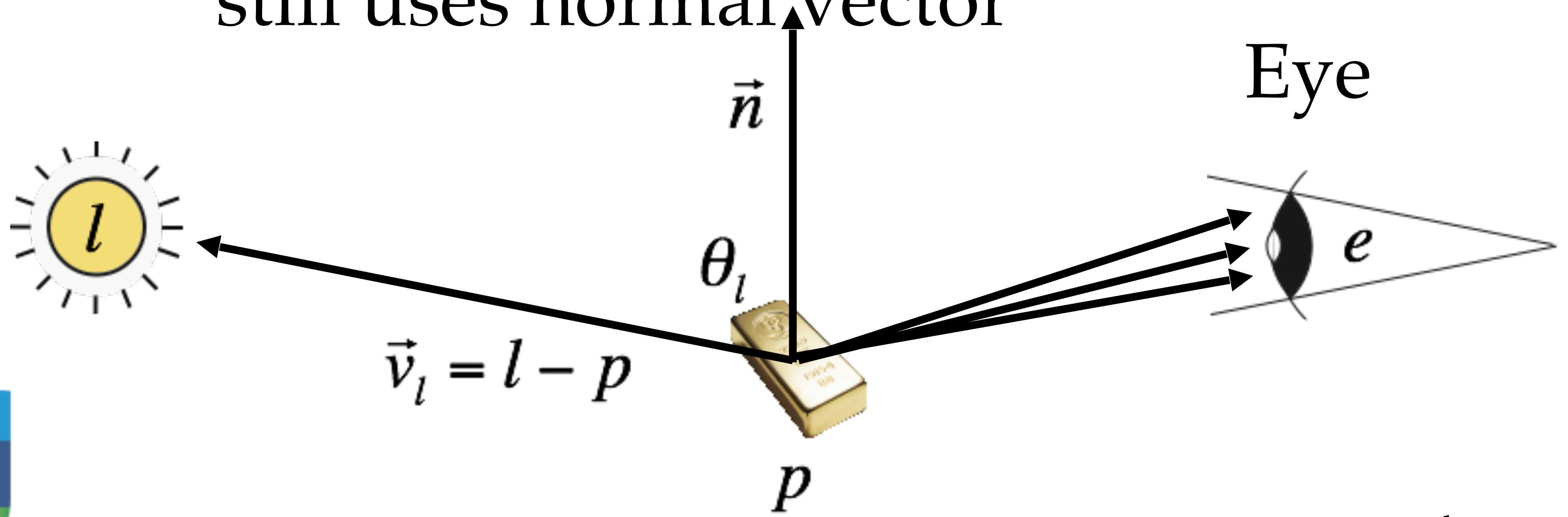
  - this adjusts the size of the highlight

Specular Highlights



Diffuse Light

- Diffuse light is from rough surfaces
  - rough at the microscopic scale
  - normal is essentially random
- Although surface is oriented, diffuse light still uses normal vector



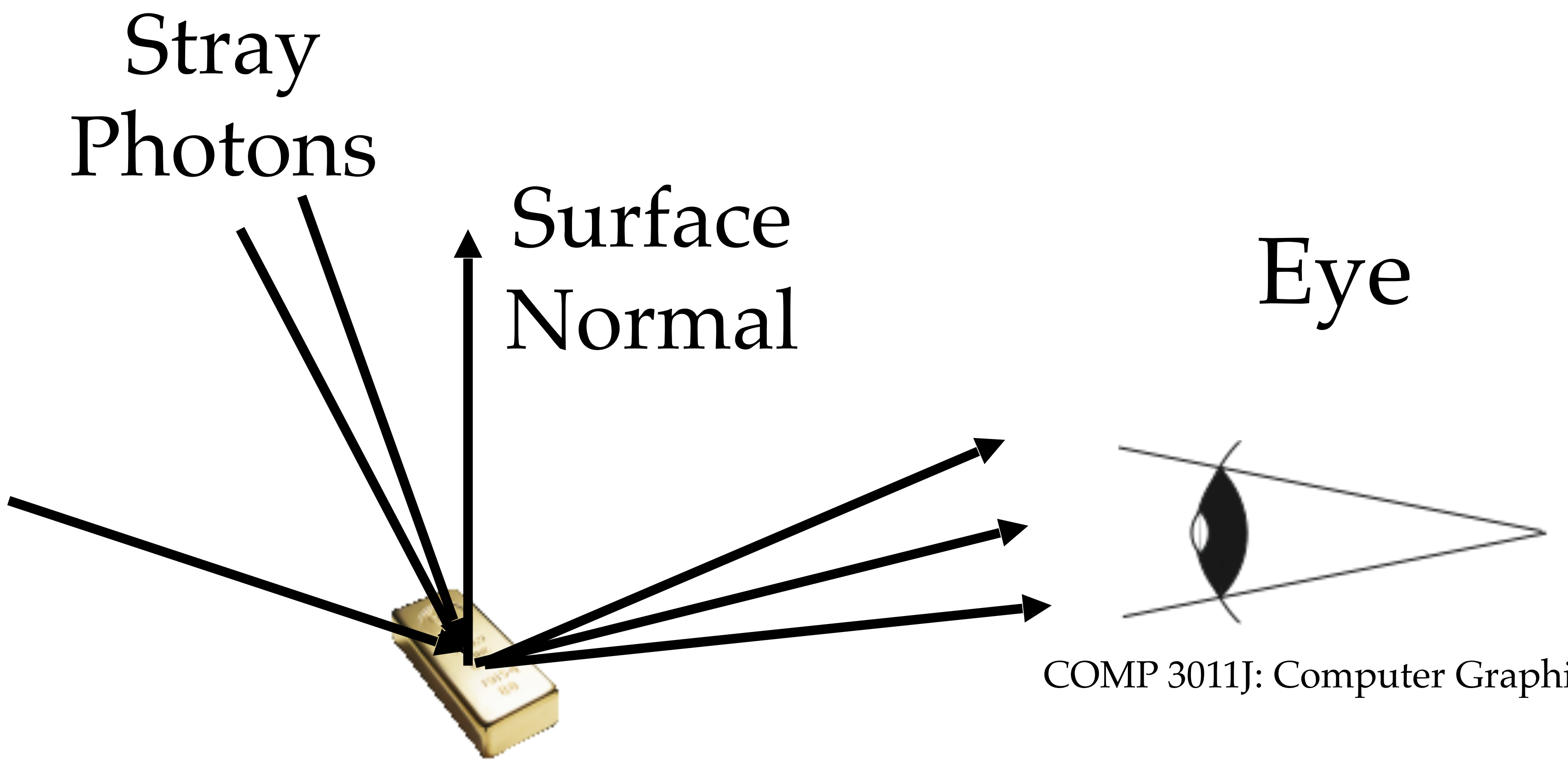
Diffuse Computation

- Light is *spread* over surface
  - depending on *incident* angle
  - reflects *uniformly* in all directions
  - not dependent on *reflection* angle

$$I_{diffuse}(p) = k_{diffuse} \cos\theta_l$$
$$= k_{diffuse} \frac{\vec{n} \bullet \vec{v}_l}{\|\vec{n}\| \|\vec{v}_l\|}$$

Ambient Lighting

- Some photons have bounced around
  - Hard to identify their source
  - Roughly same number everywhere



Ambient Light

- Ambient light is *uniform*
  - constant amount
  - doesn't always work
- Here, fewer reflections onto side of cabinet than onto wall

$$I_{ambient}(p) = k_{ambient}$$

Emitted Light

- Light from a *glowing* object
- For simplicity, uniform in *all* directions
- Not affected by *incoming* light

$$I_{emitted}(p) = k_{emitted}$$

Surface Modulation

- Terms *k* depend on:
  - intensity of light
  - reflectivity (*albedo*) of surface

$$k_{specular} = l_{specular} r_{specular}$$

where

$$l_{specular} = \text{specular intensity of light}$$

$$r_{specular} = \text{specular albedo of surface}$$

Putting it Back Together

$$I_{total}(p) = I_{specular}(p) + I_{diffuse}(p) + I_{ambient}(p) + I_{emitted}(p)$$

$$= l_{specular} r_{specular} \left(\frac{\vec{n} \bullet (\vec{v}_b)}{\|\vec{n}\| \|\vec{v}_b\|} \right)^{n_s}$$

$$+ l_{diffuse} r_{difffuse} \frac{\vec{n} \bullet \vec{v}_l}{\|\vec{n}\| \|\vec{v}_l\|}$$

$$+ l_{ambient} r_{ambient}$$

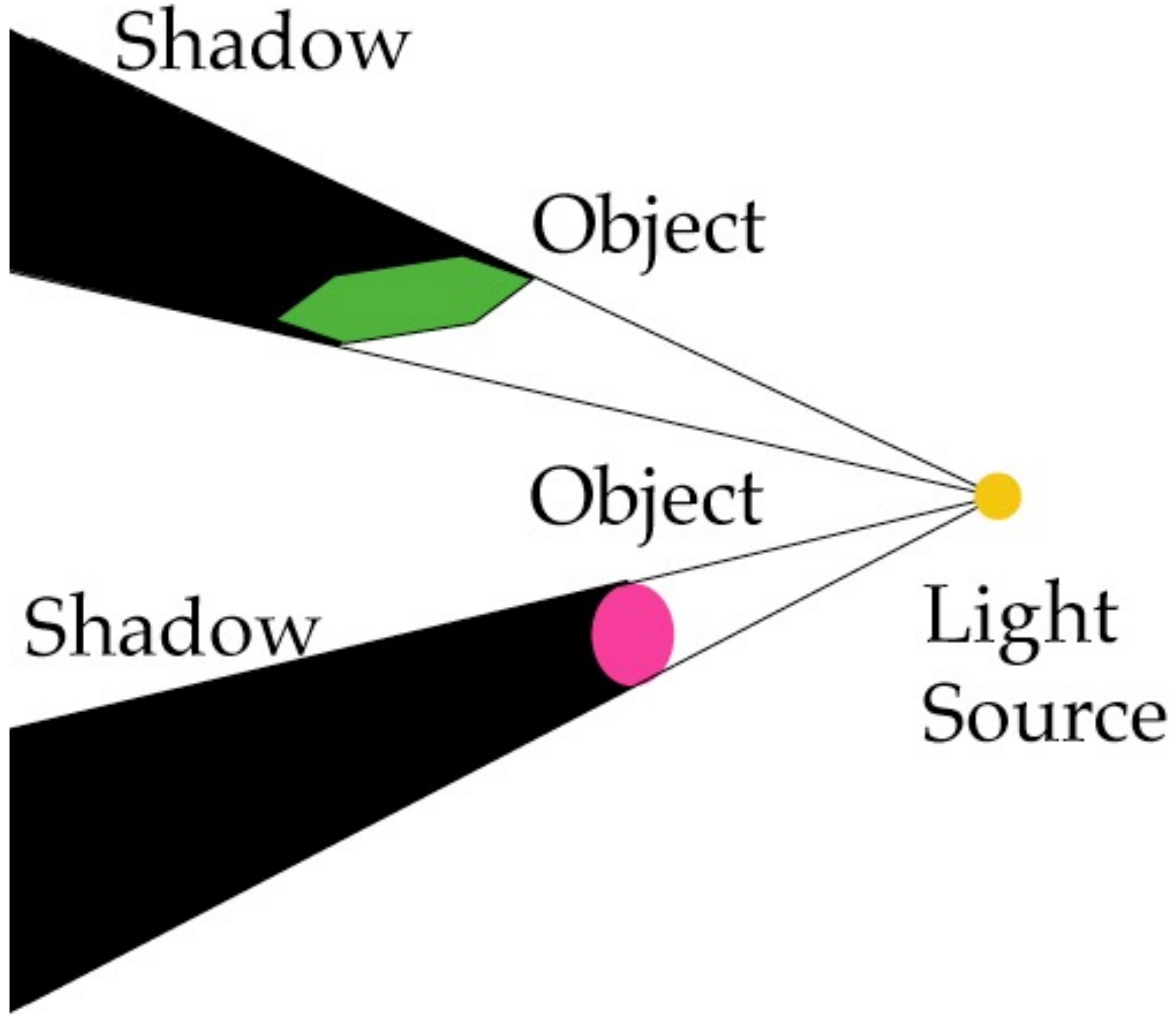
$$+ k_{emitted}$$

Effects of Colour Light

- To add coloured lights, treat R, G, B separately
  - Evaluate lighting intensity for each color channel (R, G, B), calculate the light intensity using separate constants. These constants represent the strength or influence of the light in each channel, allowing you to adjust the light's color and effect on the surface.
  - Perform lighting calculations three times, once for each color channel.
  - Combine Results:

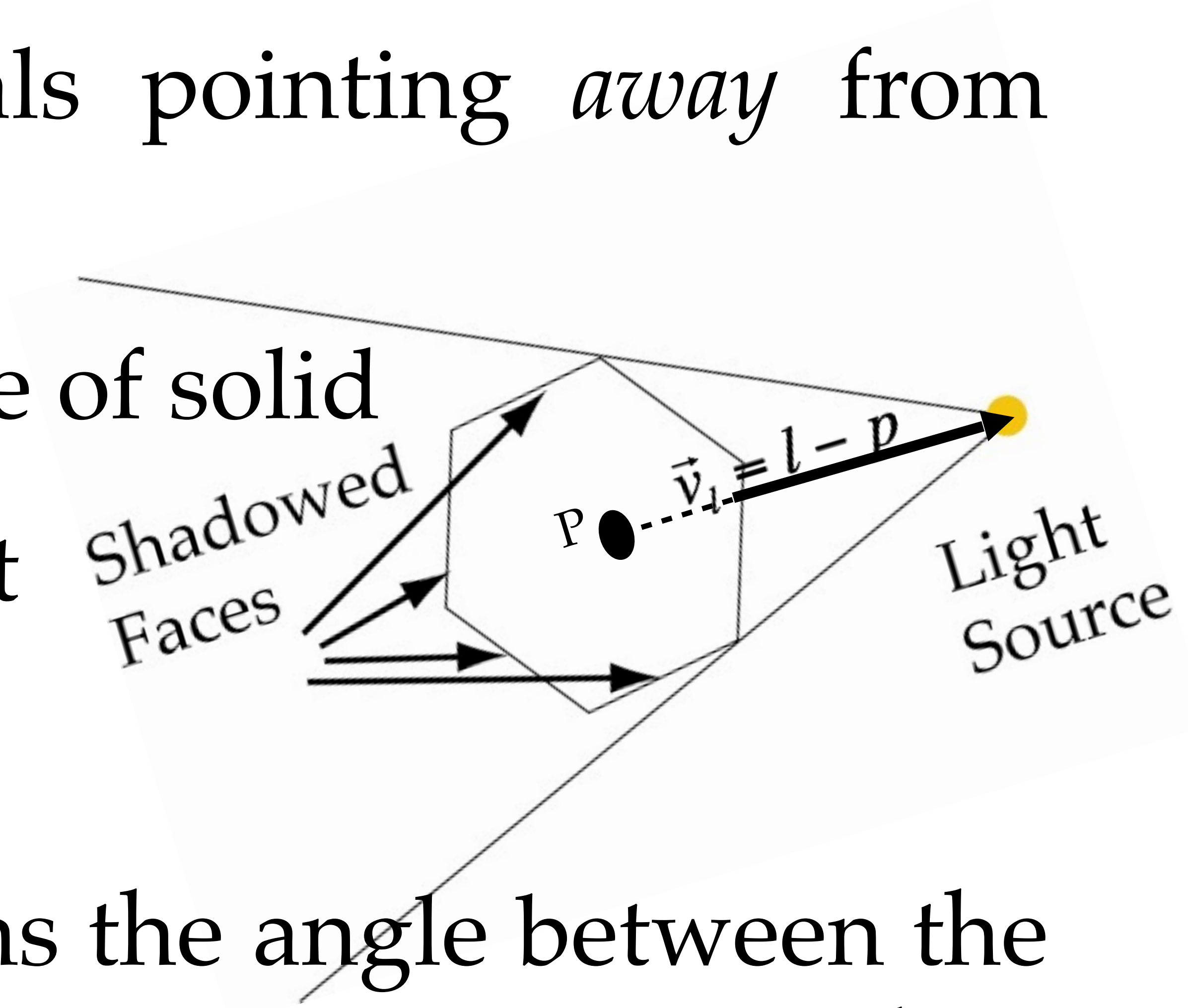
Shadows

- What is a *shadow*?
  - It's not an object
  - It's an *absence* of light
    - behind a solid object
  - Important depth cue
- How do we do this?
  - Two approaches , one using Ray tracing / other using rasterization.



Shadowed Faces

- *Shadowed faces* have normals pointing *away* from light source
- Light is blocked by other face of solid
    - no diffuse or specular light
    - just ambient and emissive
- So $\vec{n} \cdot \vec{v}$ is negative it means the angle between the light direction and the surface normal is greater than 90 degrees, indicating that the surface is facing away from the light source and therefore lies in shadow.



How to handle Shadows in OpenGL

- **OpenGL will not by default handle shadows for you**

Early Techniques: Simple Shadow Blobs

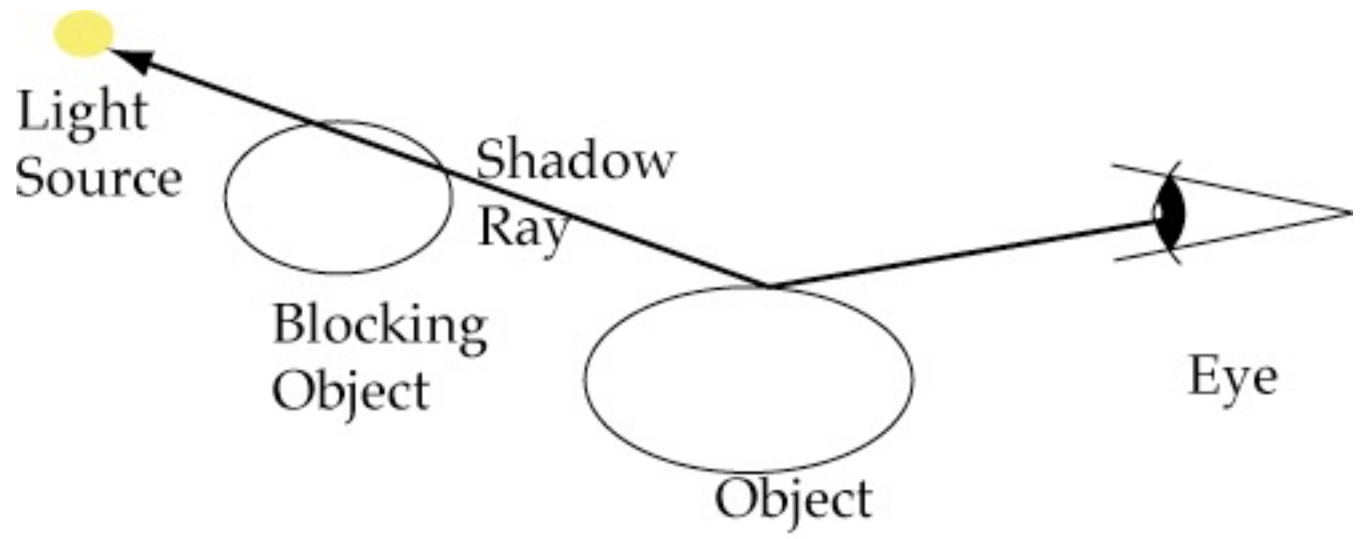
- Early 3D games used a simple approach, where they would render a flat, darkened circle (often referred to as a "shadow blob") beneath characters or objects. This approach was computationally cheap and added some depth but wasn't very realistic, as the shadow didn't account for the actual shape of the object or the lighting conditions.

Shadow Mapping

- In modern 3D graphics, shadow mapping is a popular technique:
  - **Create a Shadow Map**: First, render the scene from the perspective of the light source. This generates a depth map (shadow map), which records the distance of the closest object to the light at each point.
  - **Compare Depths**: In the main rendering pass (viewed from the camera), compare the depth of each point on objects with the depth recorded in the shadow map. If a point is farther from the light than the recorded depth at that location, it's in shadow.
  - **Texture-Based Shadow Rendering**: The shadow map acts as a texture, determining which areas are in shadow when rendered from the main camera's perspective.

Raytracing Example : Shadow Rays

- When raytracer hits a surface
  - Draw a ray towards the light
  - If it hits anything else, it's shadowed



High-Level Raytracer

- For each pixel, generate a ray
- Find intersection of ray with closest object
- Generate shadow ray towards light
    - if it doesn’t intersect any other object
    - and normal points *toward* light
    - compute diffuse & specular light
- Always add in ambient & emitted light