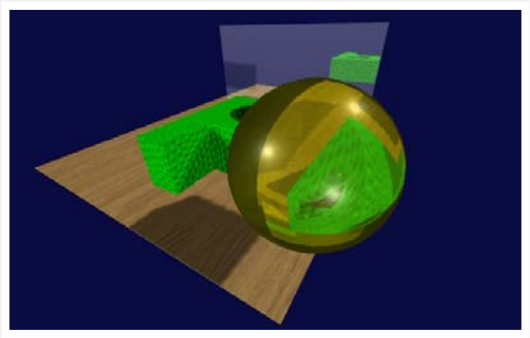
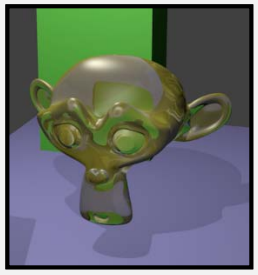
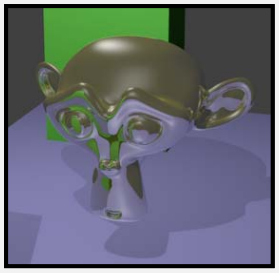
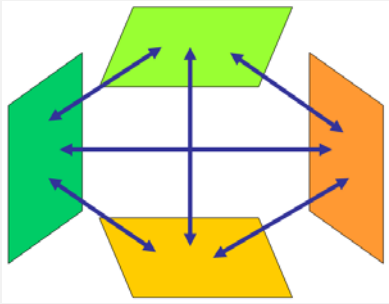
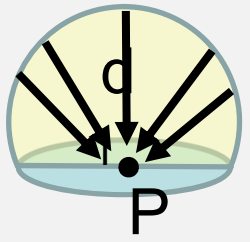
Rendering









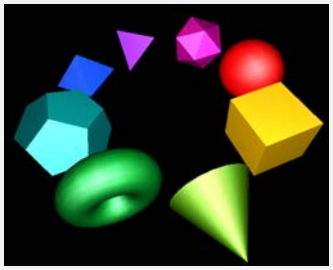


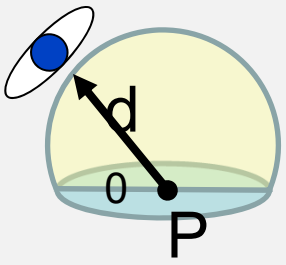
Oregon State University

Mike Bailey

mjb@cs.oregonstate.edu

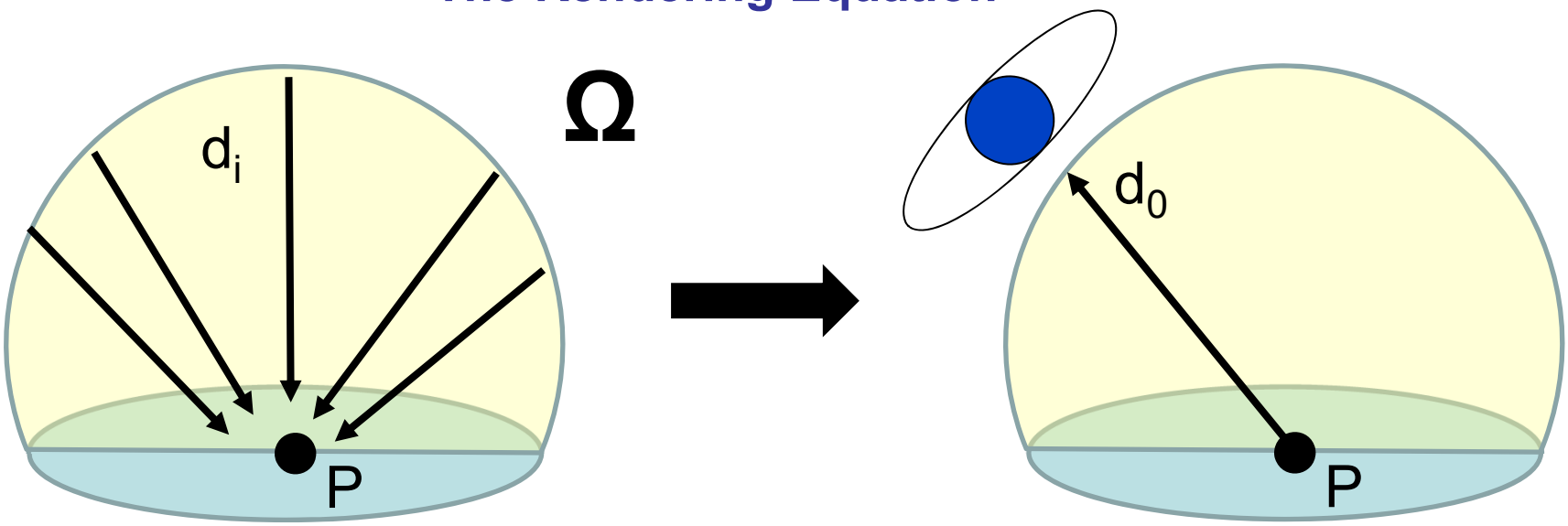
This work is licensed under a Creative Commons Attribution-NonCommercial-NoDerivatives 4.0 International License





Oregon State University
Computer Graphics

The Rendering Equation



$$B(P, d_0, \lambda) = E(P, d_0, \lambda) + \int_{\Omega} B(P, d_i, \lambda) f(\lambda, d_i, d_0)(d_i \cdot \hat{n}) d\Omega$$

This is the true rendering situation. Essentially, it is an energy balance:

Light Shining from a point P =
Light emitted by that point +
Reflectivity * Σ(Light arriving from all other points)

But, this is time-consuming to solve "exactly".

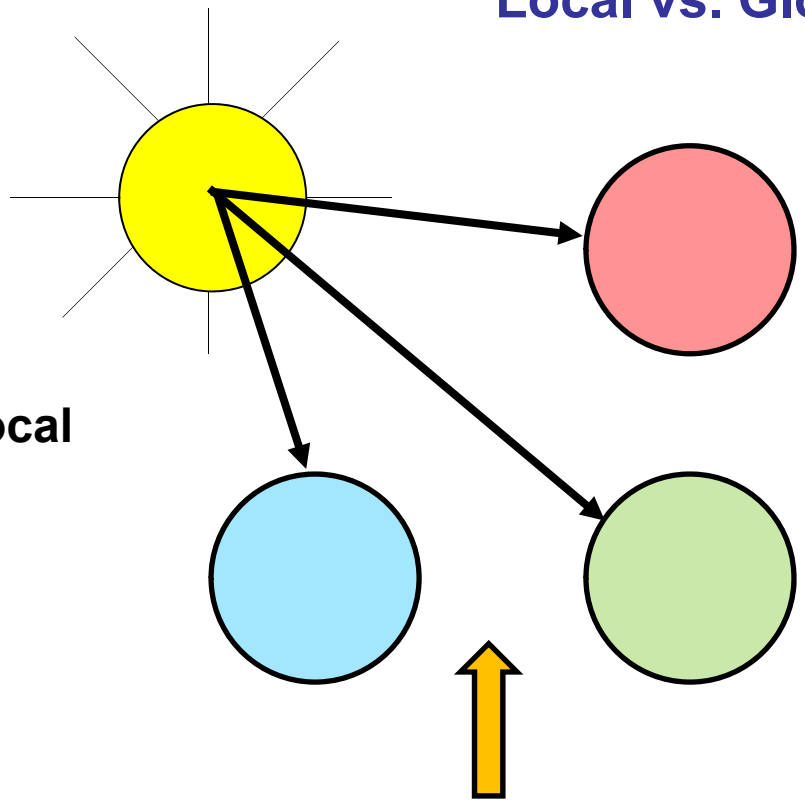
So, we need to know **how much of an approximation we need**

Rendering

Rendering is the process of creating an image of a geometric model. There are questions you need to ask:

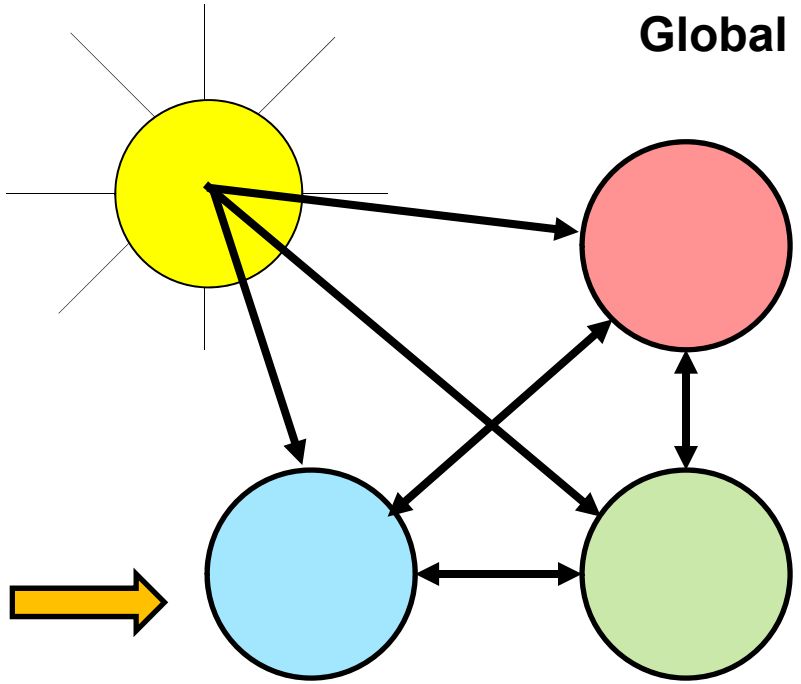
- For what purpose am I doing this?
- How realistic do I need this image to be?
- How much compute time do I have to create this image?
- Do I need to take lighting into account?
- Does the illumination need to be global or will local do?
- Do I need to create shadows?
- Do I need to create reflections and refractions?
- How good do the reflections and refractions need to be?

Local vs. Global Illumination

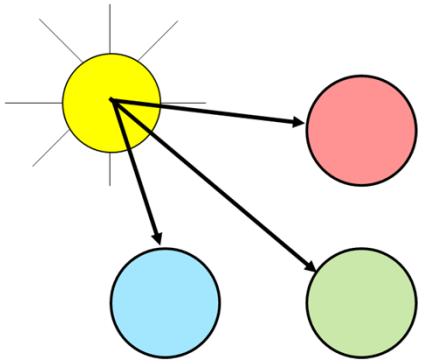


If the appearance of an object is only affected by its own characteristics and the characteristics of the light sources, then you have **Local Illumination.**

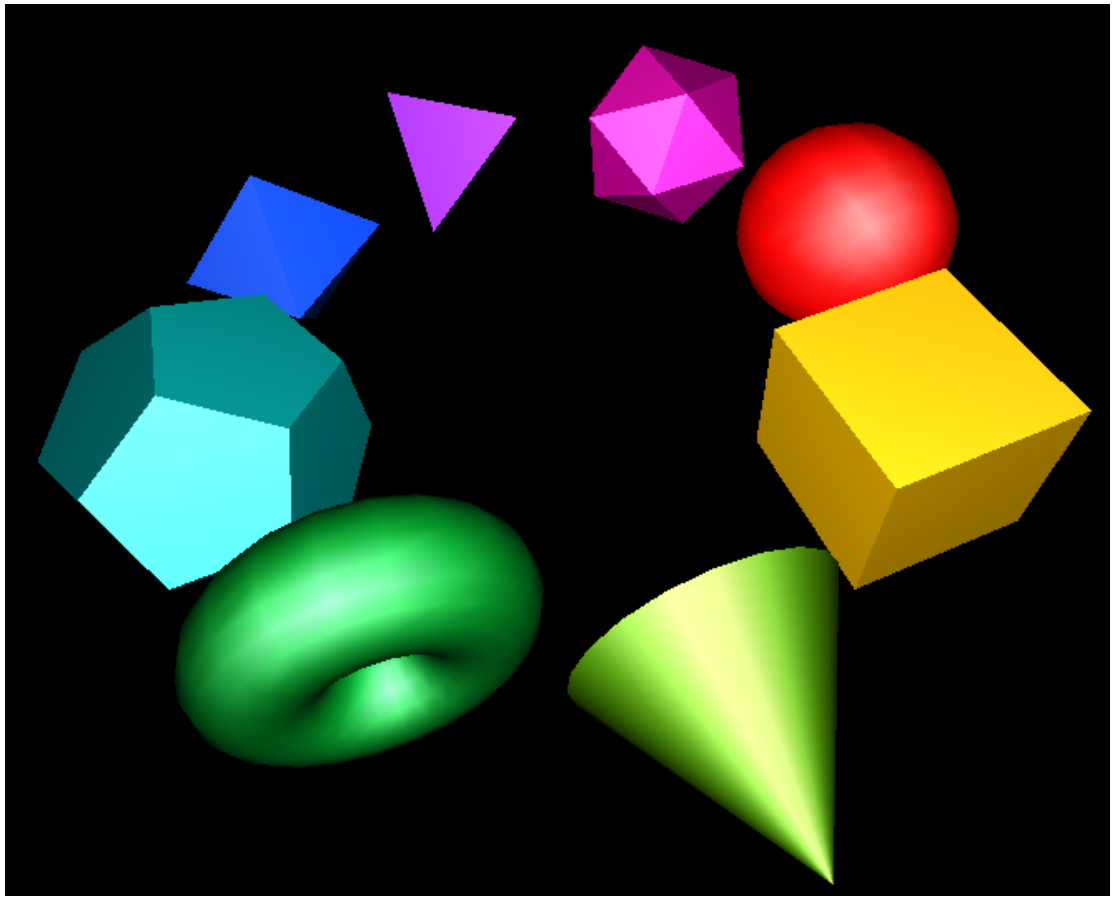
If the appearance of an object is also affected by the appearances of other objects, then you have **Global Illumination.**



Local Illumination at Work

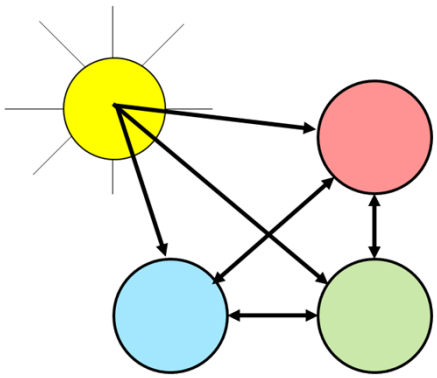


"If the appearance of an object is only affected by its own characteristics and the characteristics of the light sources, then you have **Local Illumination.**"



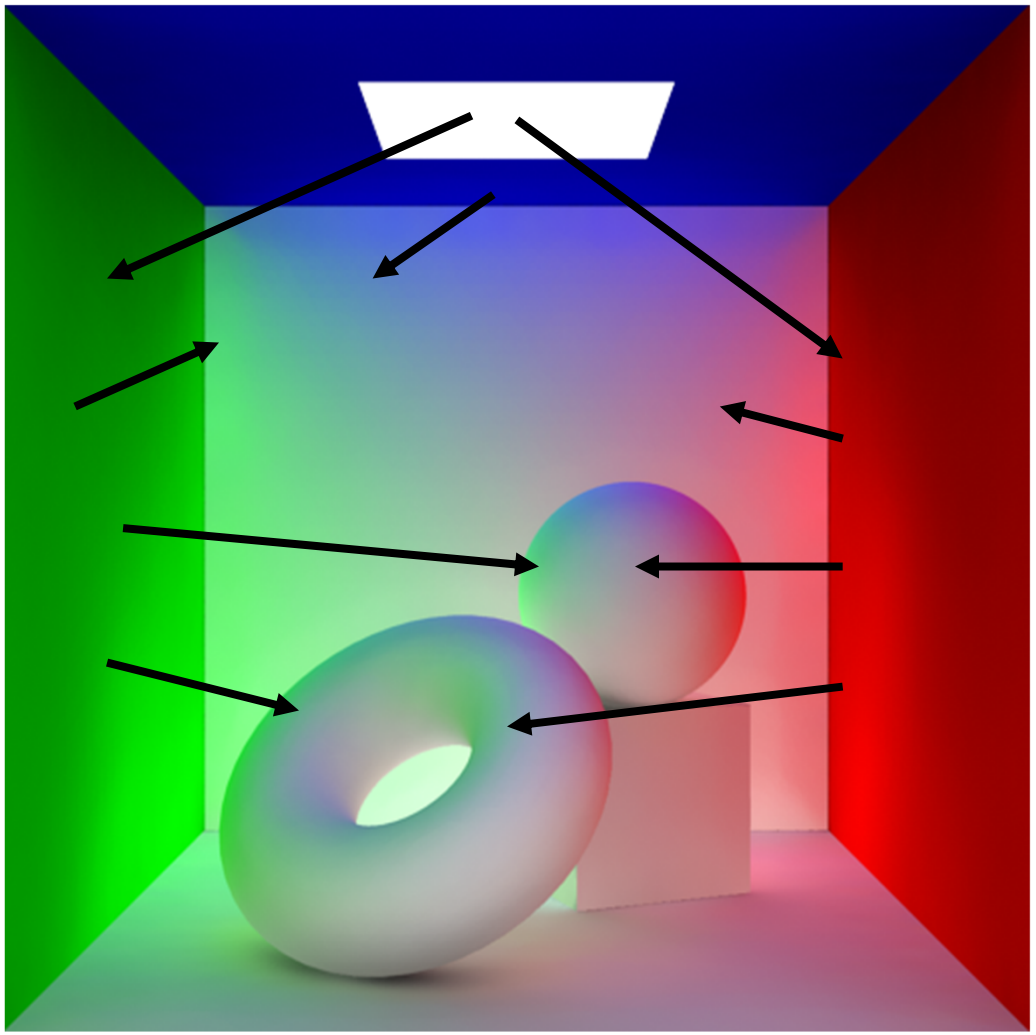
OpenGL rendering uses Local Illumination.

Global Illumination at Work



- The left wall is green.
- The right wall is red.
- The back wall is white.
- The ceiling is blue with a light source in the middle of it.
- The objects sitting on the floor are white.

"If the appearance of an object is also affected by the appearances of other objects, then you have **Global Illumination.**"



http://www.swardson.com/unm/tutorials/mentalRay3/

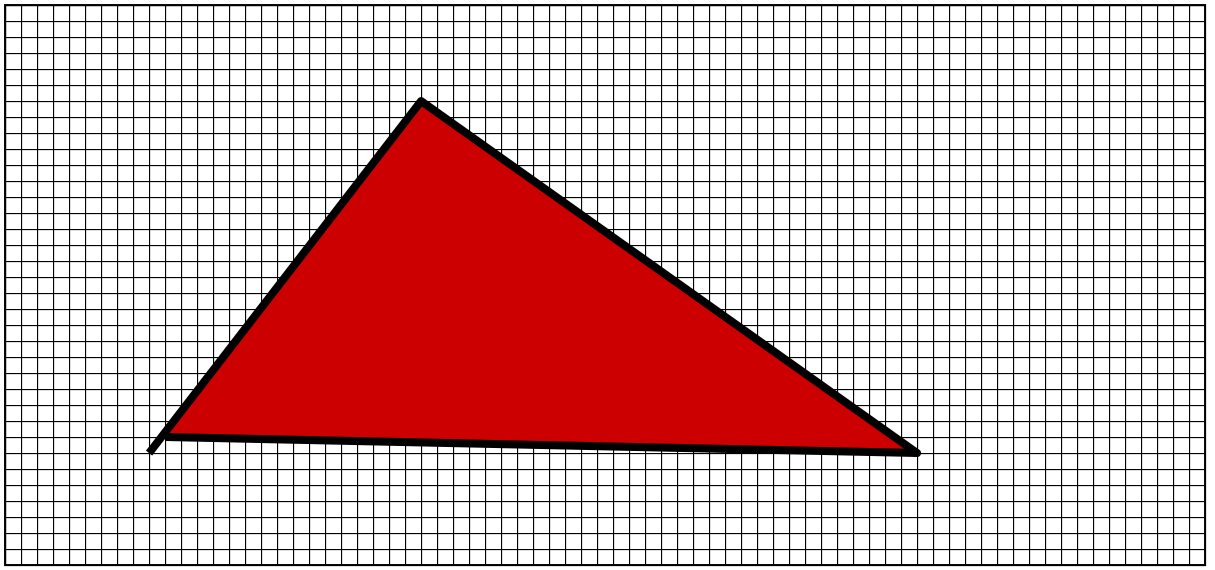
Two Directions for the Rendering to Happen

1. Starts at the object, works towards the eye

2. Starts at the eye, works towards the object

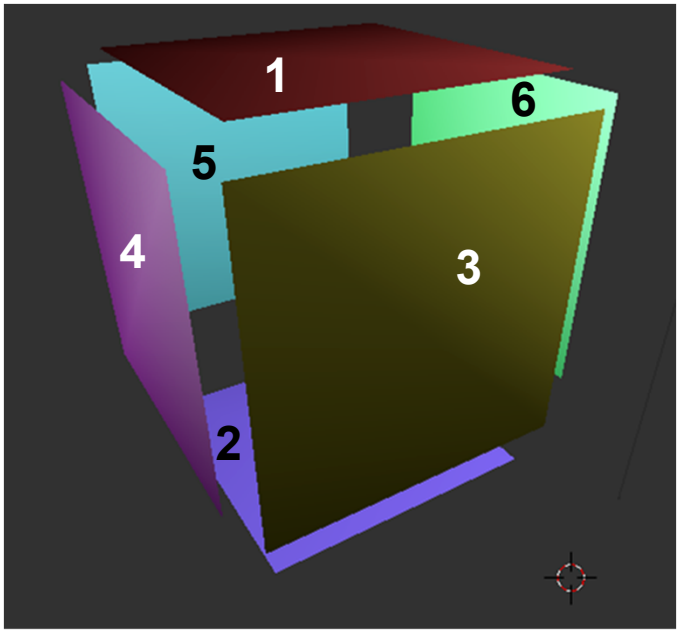
Starts at the Object, Works Towards the Eye

- This is the kind of rendering you get on a graphics card (e.g., OpenGL).
- You have been doing this all along.
- Start with the geometry and project it onto the pixels.



How do things in front look like they are *really* in front?

Your application might draw this cube's polygons in 1-2-3-4-5-6 order, but 1, 3, and 4 still need to look like they were drawn last:



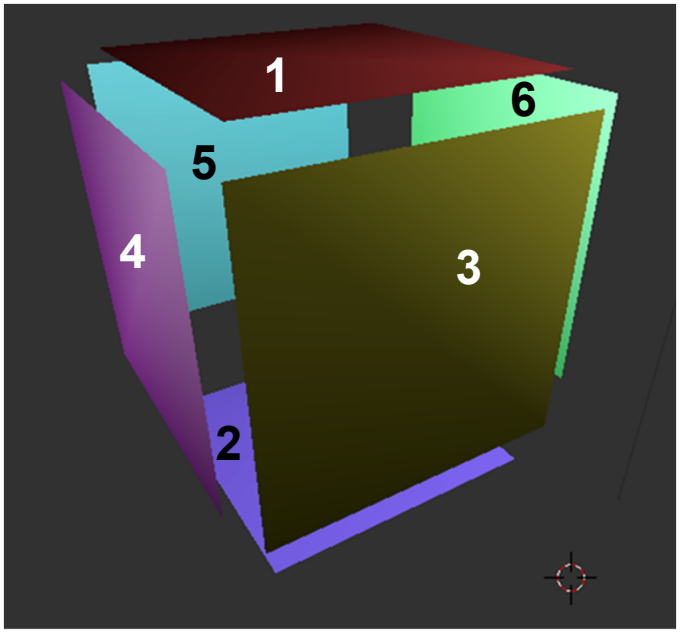
Solution #1: Sort your polygons in 3D by depth and draw them back-to-front.

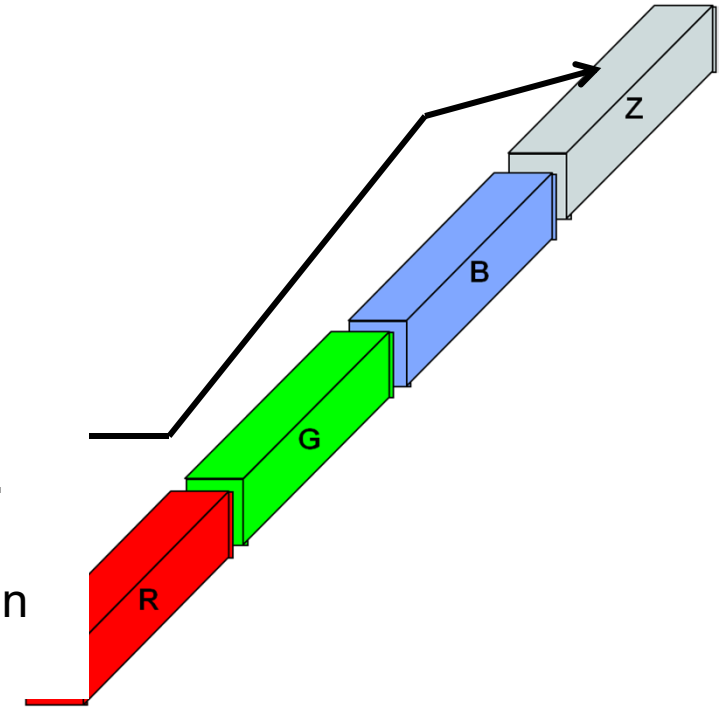
In this case 1-2-3-4-5-6 becomes 5-6-2-4-1-3.

This is called the **Painter's Algorithm**. It sucked to have to do things this way.

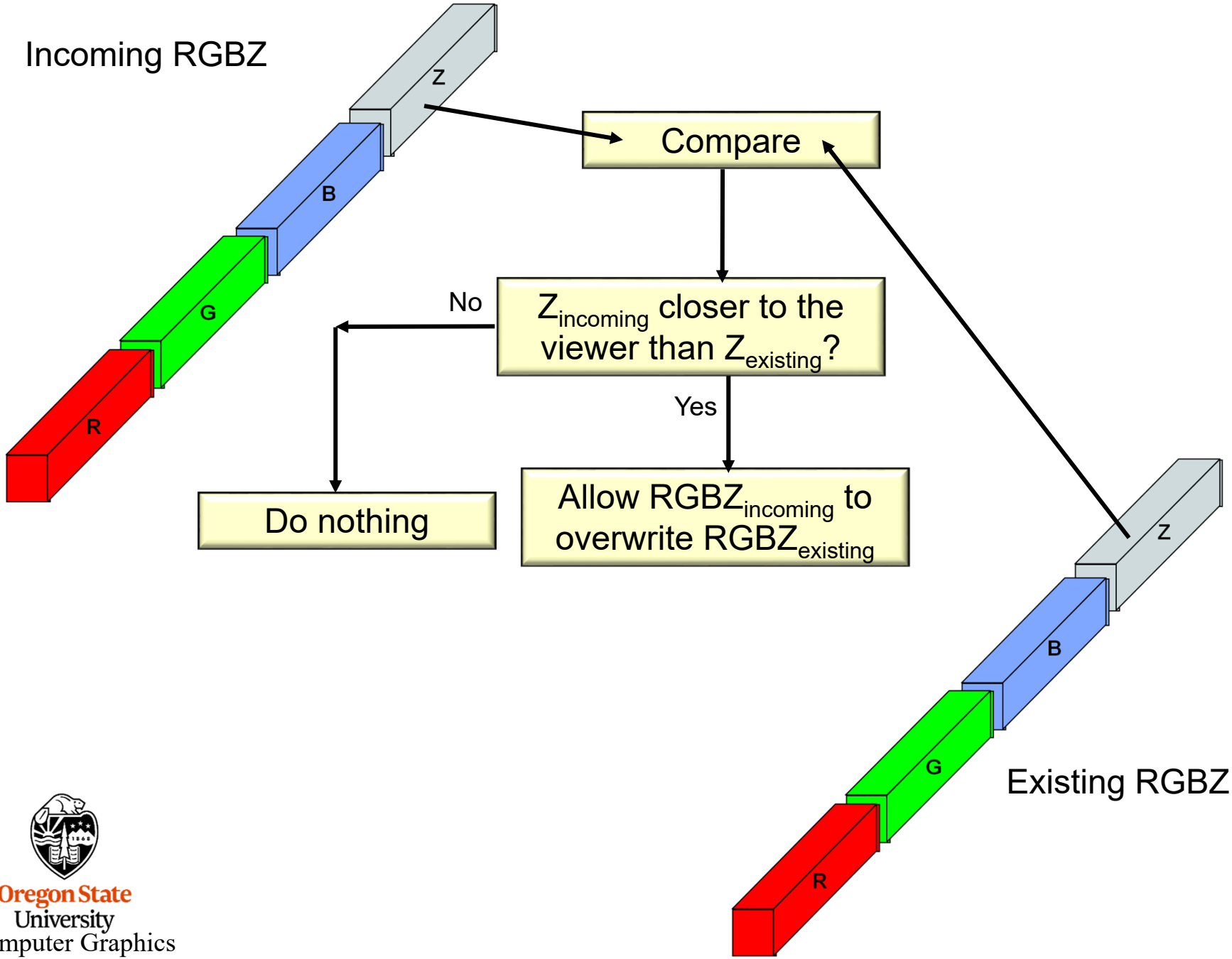
How do things in front look like they are *really* in front?

Your application might draw this cube's polygons in 1-2-3-4-5-6 order, but 1, 3, and 4 still need to look like they were drawn last:



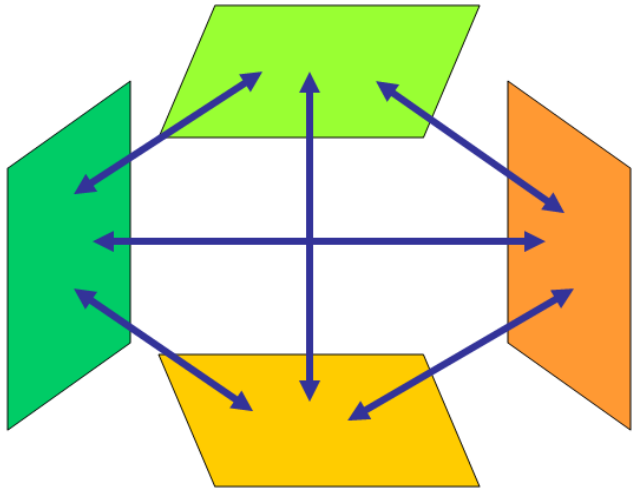


Solution #2: Add an extension to the framebuffer to store the depth of each pixel. This is called a **Depth-buffer** or **Z-buffer.** Only allow pixel stores when the depth of the incoming pixel is closer to the viewer than the pixel that is already there.


Incoming RGBZ
Z
B
G
R
Compare
No
$Z_{incoming}$ closer to the viewer than $Z_{existing}$?
Yes
Do nothing
Allow $RGBZ_{incoming}$ to overwrite $RGBZ_{existing}$
Z
B
G
R
Existing RGBZ

Another From-the-Object Method -- Radiosity

Based on the idea that all surfaces gather light intensity from all other surfaces



The fundamental radiosity equation is an energy balance that says:

"The light energy leaving surface *i* equals the amount of light energy generated by surface *i* plus surface *i*'s reflectivity times the amount of light energy arriving from all other surfaces"

$$B_i A_i = E_i A_i + \rho_i \sum_j B_j A_j F_{j \to i}$$

Oregon State University
Computer Graphics

This is a very good approximation to the **Rendering Equation**

The Radiosity Equation

$$B_i A_i = E_i A_i + \rho_i \sum_j B_j A_j F_{j \to i}$$

B_i is the light energy intensity shining from surface element *i*

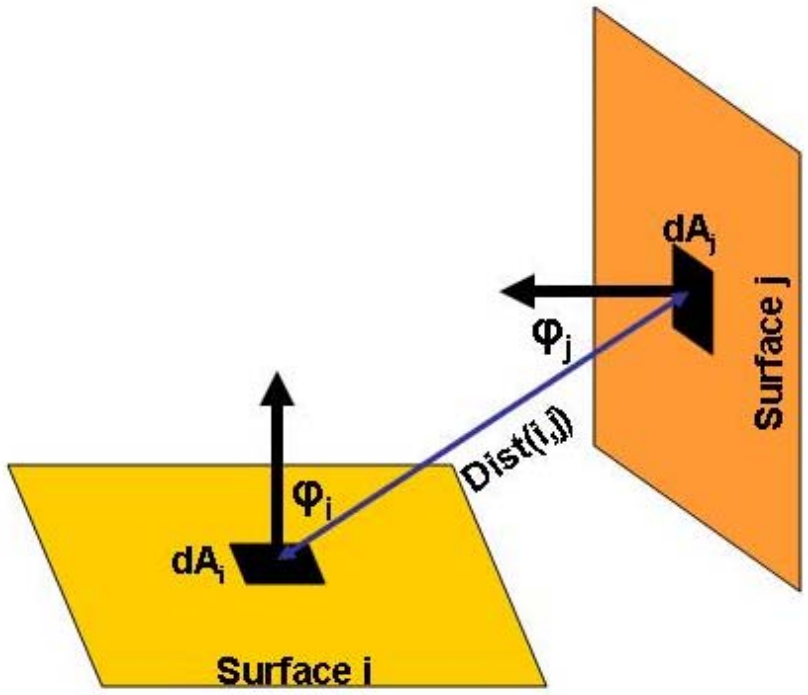
A_i is the area of surface element *i*

E_i is the internally-generated light energy intensity for surface element *i*

ρ_i is surface element *i*'s reflectivity

$F_{j \to i}$ is referred to as the **Shape Factor**, and describes what percent of the energy leaving surface element *j* arrives at surface element *i*

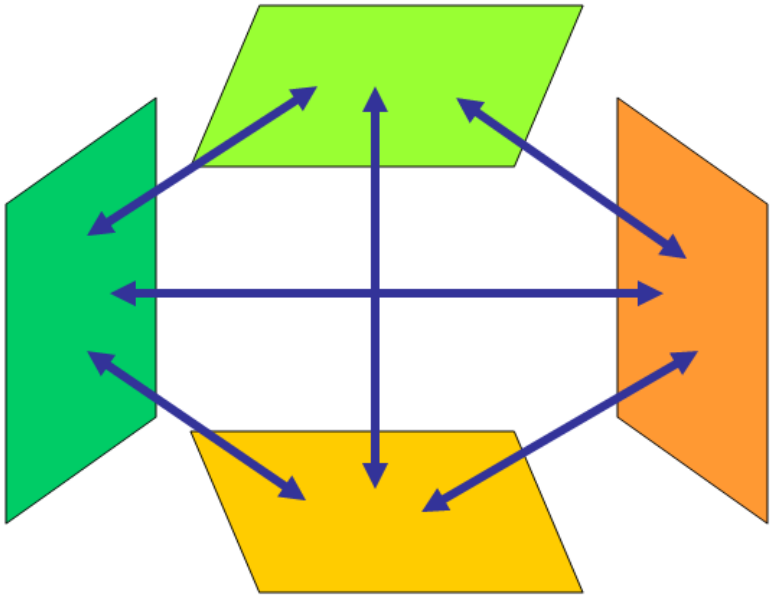
The Radiosity Shape Factor



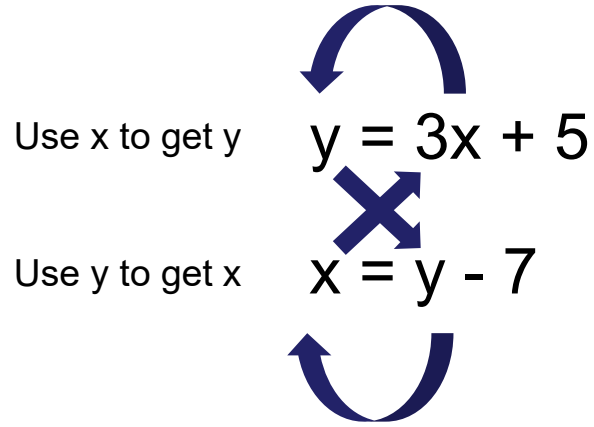
$$F_{j\to i} = \int_{Ai}\int_{A_j} visibility(di,dj)\frac{\cos\Theta_i \cos\Theta_j}{\pi Dist(di,dj)^2} dA_j dA_i$$

Oregon State
University
Computer Graphics

Does it seem to you that the light just keeps propagating and you never get an answer?



To many people, radiosity seems like this:



"x produces y, then y produces x,
then x produces y, then …"

Not really – it is simply N equations, N unknowns – you solve for the unique solution

$$-3x + y = 5$$
$$x - y = -7$$

x = 1
y = 8

The Radiosity Matrix Equation

Expand $B_i A_i = E_i A_i + \rho_i \sum_j B_j A_j F_{j \to i}$

For each surface element, and re-arrange to **solve for the surface intensities, the *B*'s**:

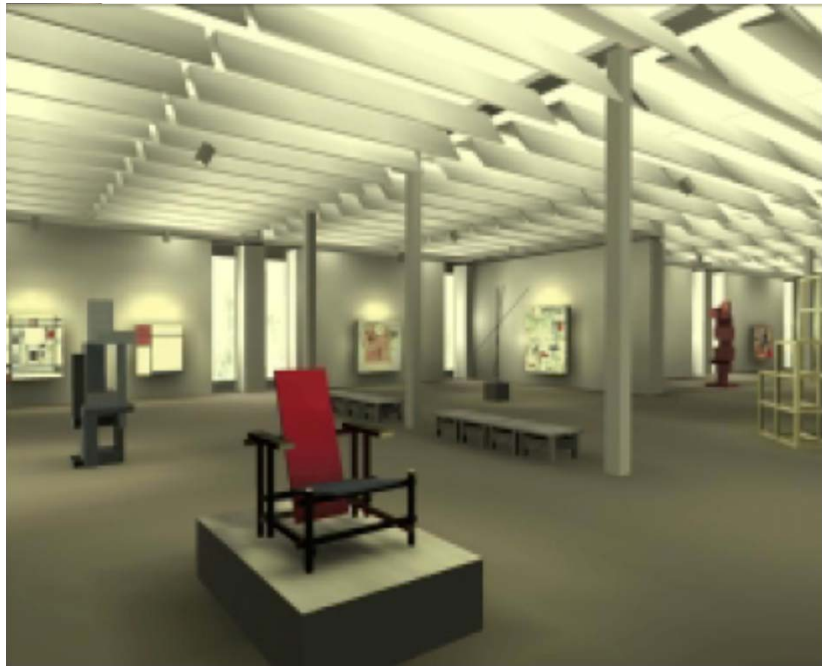
$$\begin{bmatrix} 1-\rho_1 F_{1\to1} & -\rho_1 F_{1\to2} & \bullet\bullet\bullet & -\rho_1 F_{1\to N} \\ -\rho_2 F_{2\to1} & 1-\rho_2 F_{2\to2} & \bullet\bullet\bullet & -\rho_2 F_{2\to N} \\ \bullet\bullet\bullet & \bullet\bullet\bullet & \bullet\bullet\bullet & \bullet\bullet\bullet \\ -\rho_N F_{N\to1} & -\rho_N F_{N\to2} & \bullet\bullet\bullet & 1-\rho_N F_{N\to N} \end{bmatrix} \begin{Bmatrix} B_1 \\ B_2 \\ \bullet\bullet\bullet \\ B_N \end{Bmatrix} = \begin{Bmatrix} E_1 \\ E_2 \\ \bullet\bullet\bullet \\ E_N \end{Bmatrix}$$

This is a lot of equations!

Oregon State University
Computer Graphics

Radiosity Examples

Cornell University



Cornell University

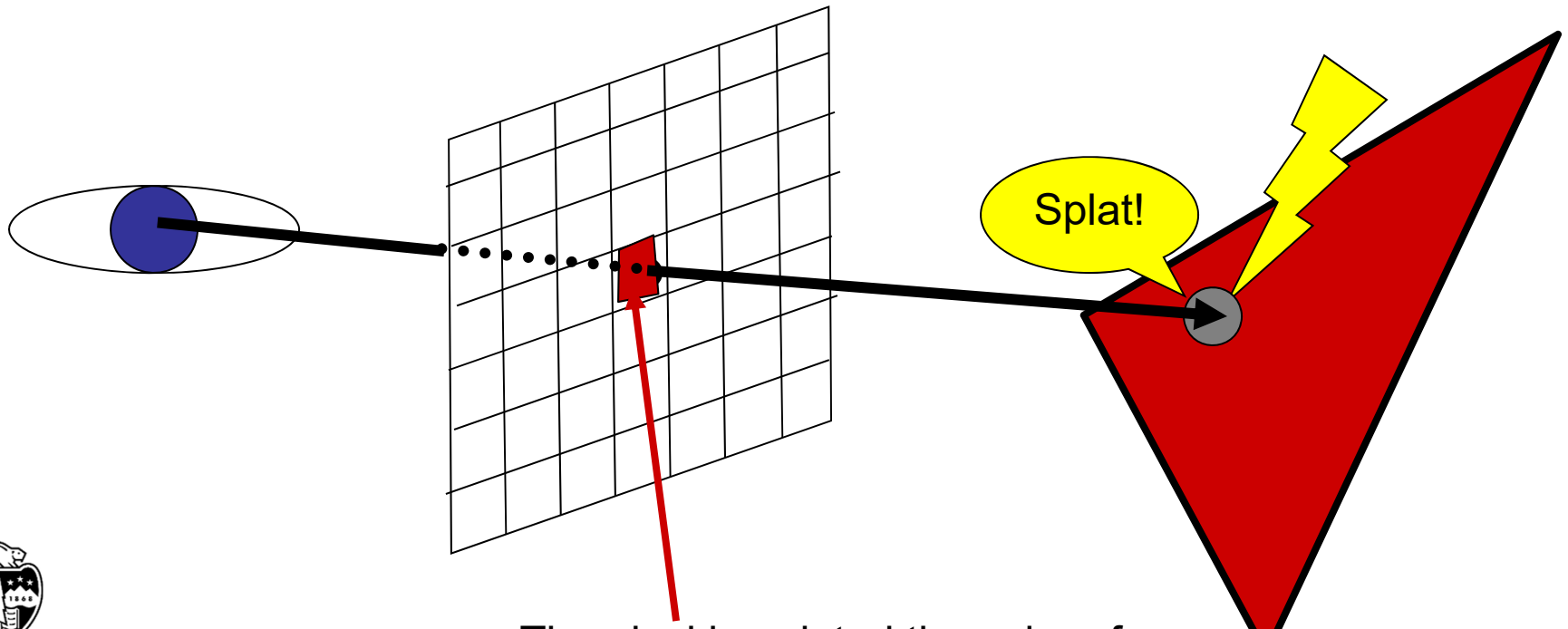
Radiosity Examples

AR Toolkit

Autodesk

Starts at the Eye, Works Towards the Objects

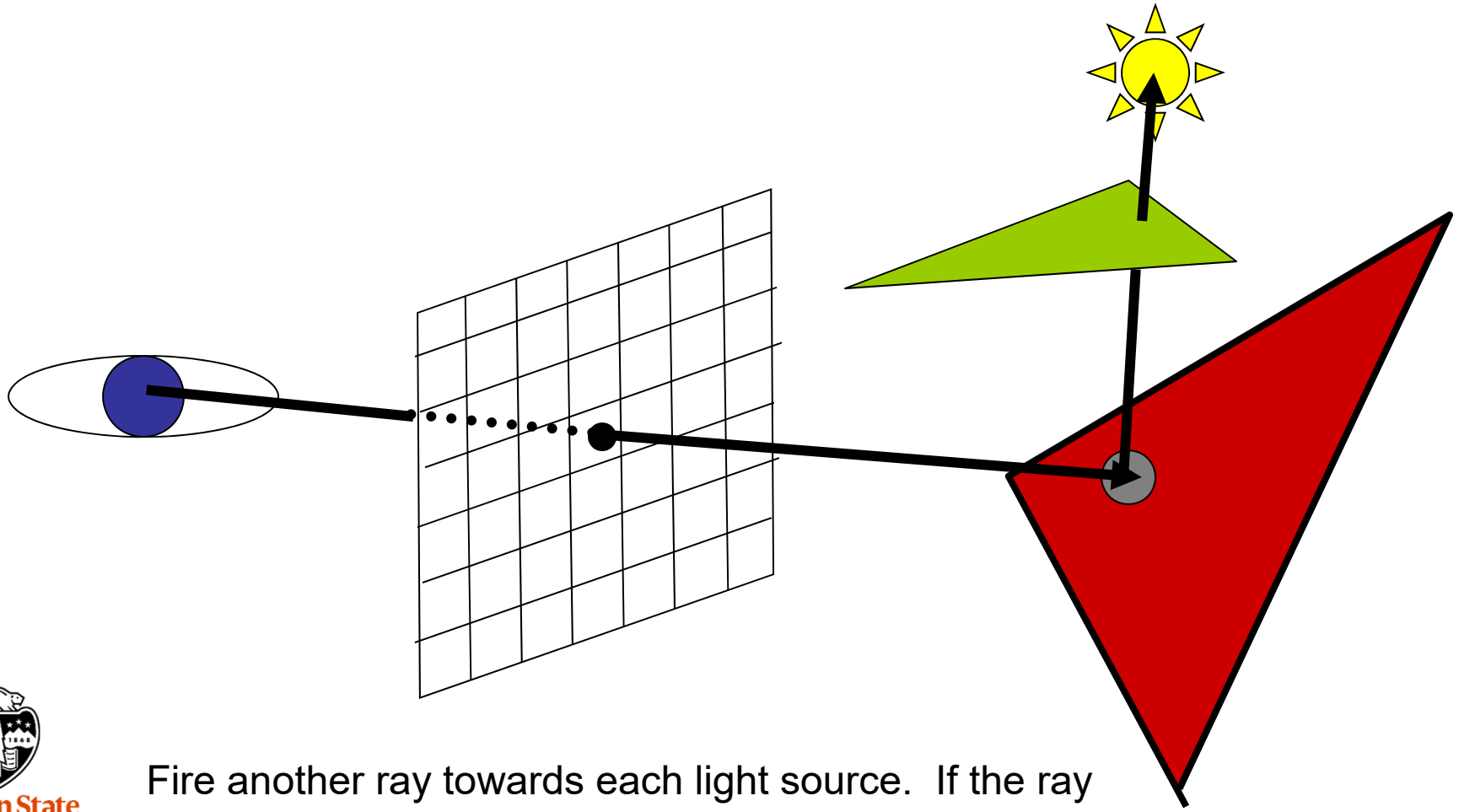
The most common approach in this category is **ray-tracing**:



The pixel is painted the color of the nearest object that is hit.

Starts at the Eye

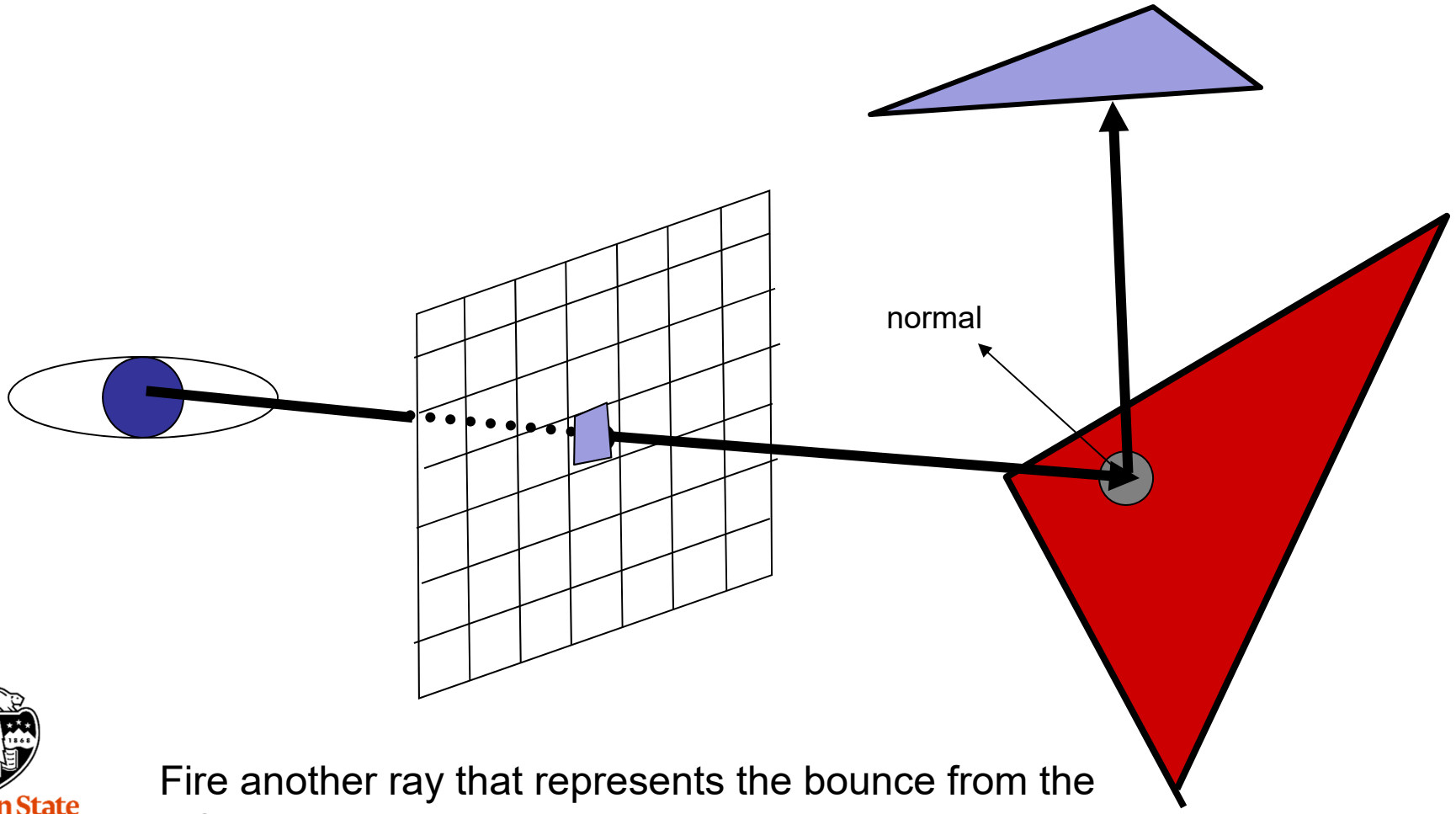
It's also straightforward to see if this point lies in a shadow:



Fire another ray towards each light source. If the ray hits anything, then the point does not receive that light.

Starts at the Eye

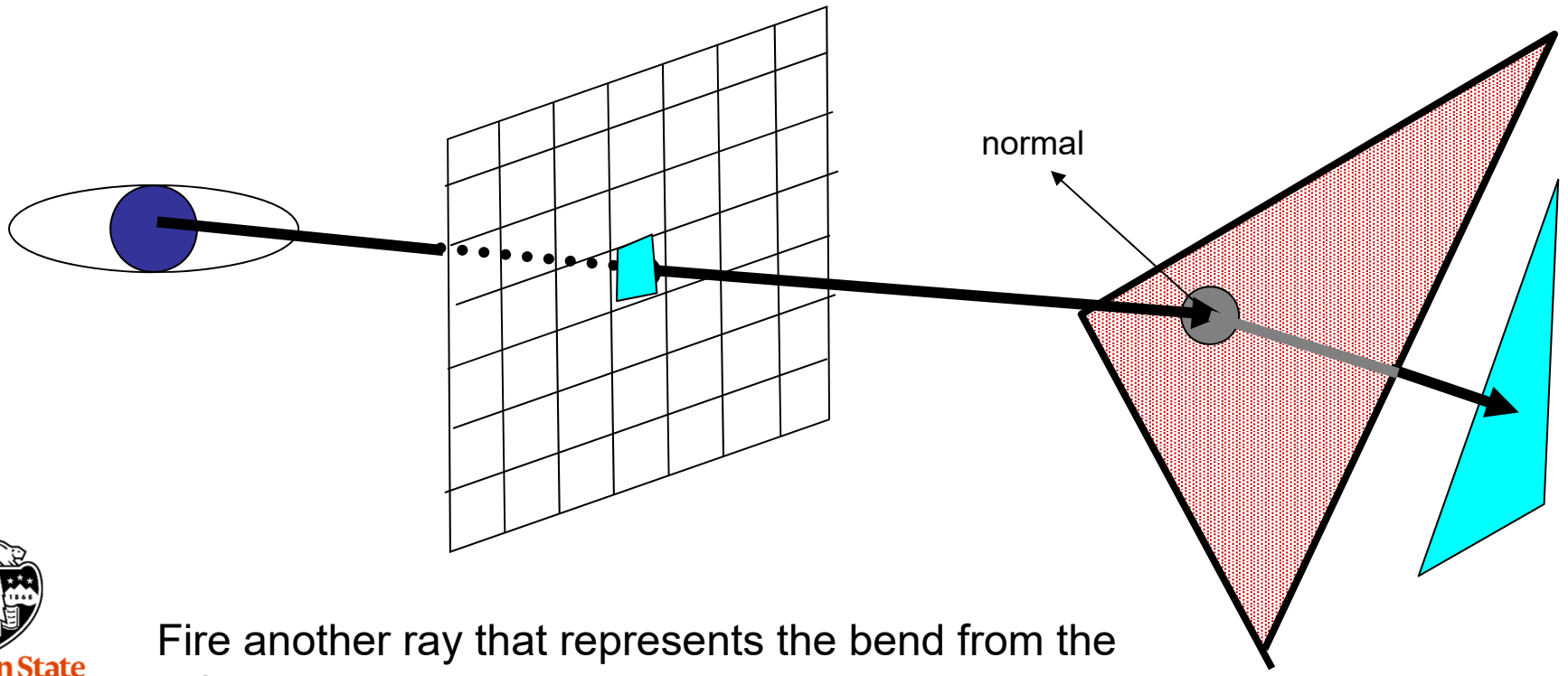
It's also straightforward to handle reflection



Fire another ray that represents the bounce from the reflection. Paint the pixel the color that this ray sees.

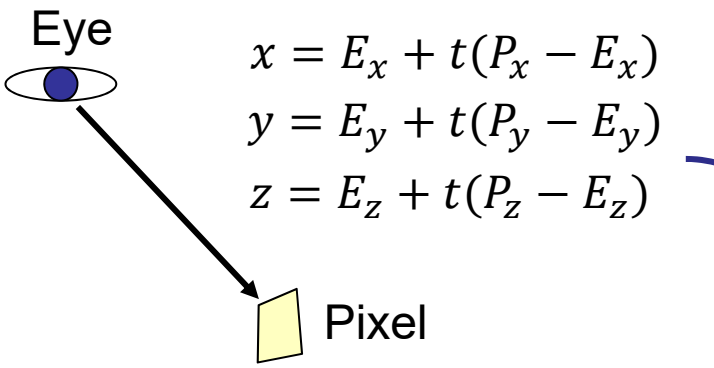
Starts at the Eye

It's also straightforward to handle refraction



Fire another ray that represents the bend from the refraction. Paint the pixel the color that this ray sees.

Determining Ray-Shape Intersections



$$x = E_x + t(P_x - E_x) \qquad t > 1.$$
$$y = E_y + t(P_y - E_y)$$
$$z = E_z + t(P_z - E_z)$$

substitute

$$(x - x_c)^2 + (y - y_c)^2 + (z - z_c)^2 = R^2$$

collect terms

$$At^2 + Bt + C = 0$$

solve

$$t = \frac{-B \pm \sqrt{B^2 - 4AC}}{2A}$$

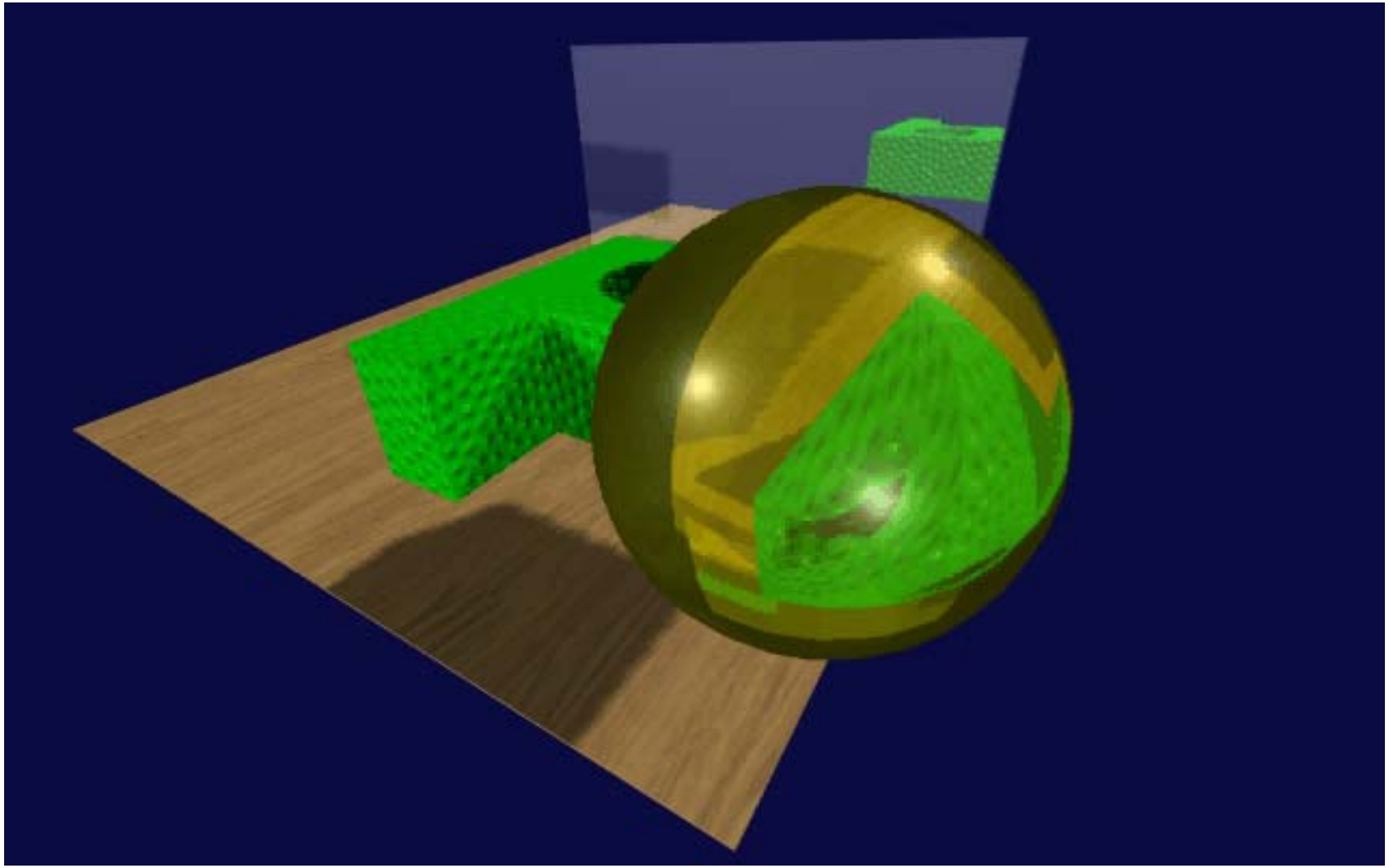
3 cases:

1.

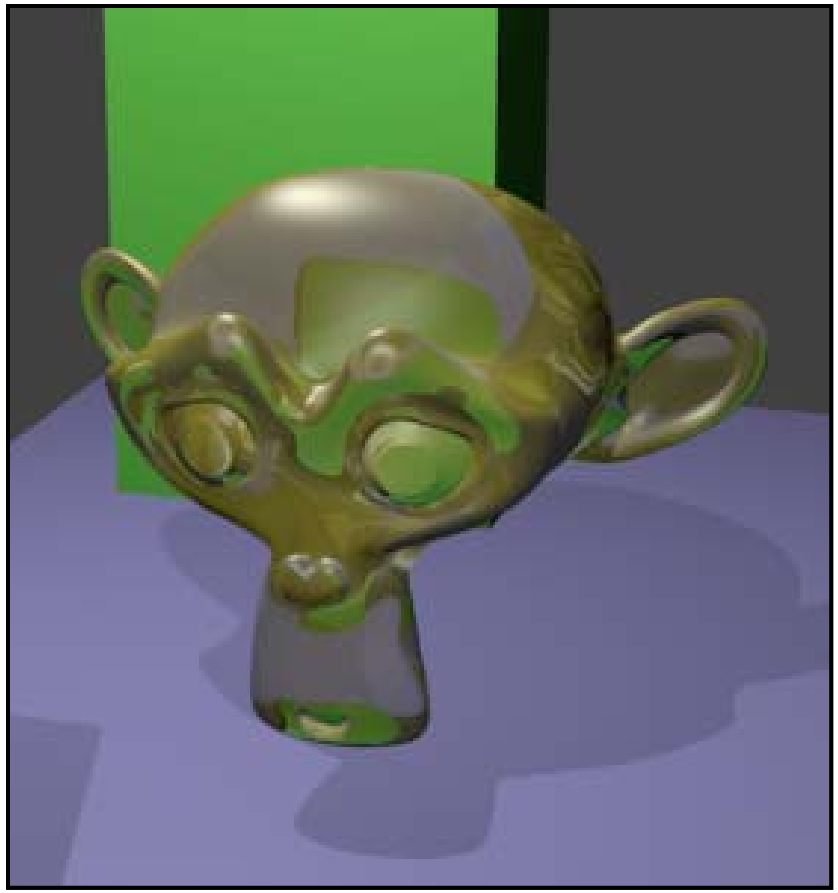
2.

3.

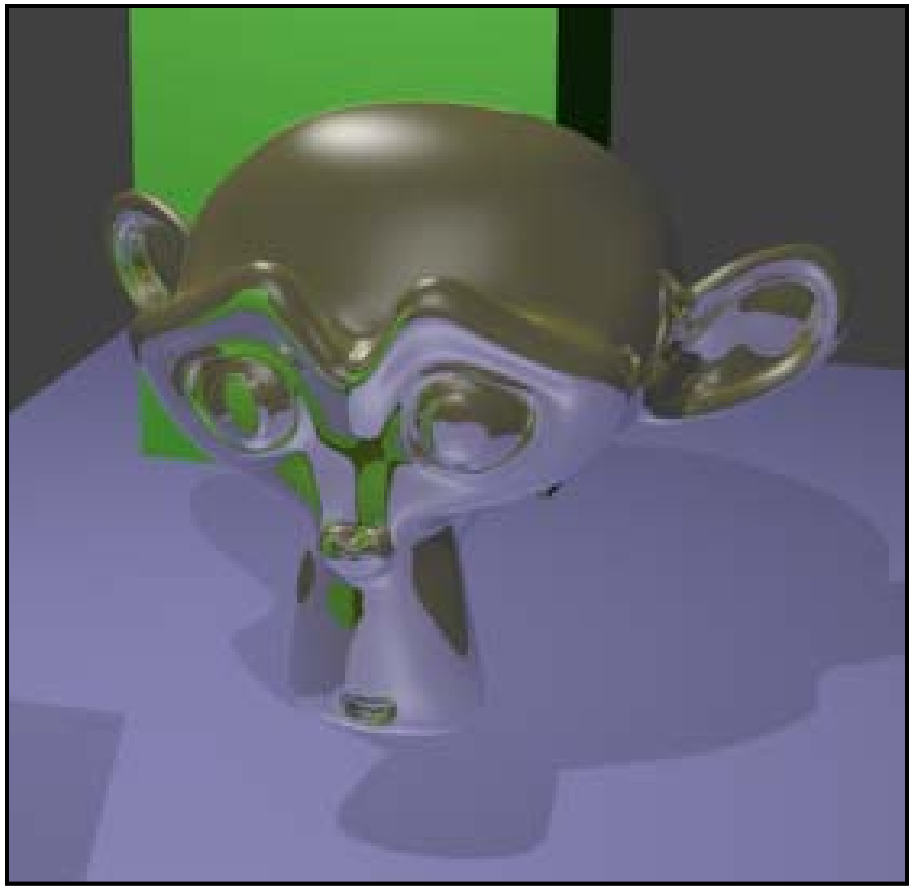
IronCAD Ray-tracing Example



Blender Ray-tracing Example



Refraction



Reflection

More Ray-tracing Examples

Quake 4 Ray-Tracing Project

More Ray-tracing Examples

IBM's Cell Interactive Ray-tracer

More Ray-tracing Examples

Bunkspeed

Subsurface Scattering

- Subsurface Scattering mathematically models light bouncing around within an object before coming back out.
- This is a good way to render skin, wax, milk, etc.

Original rendering

Subsurface scattering