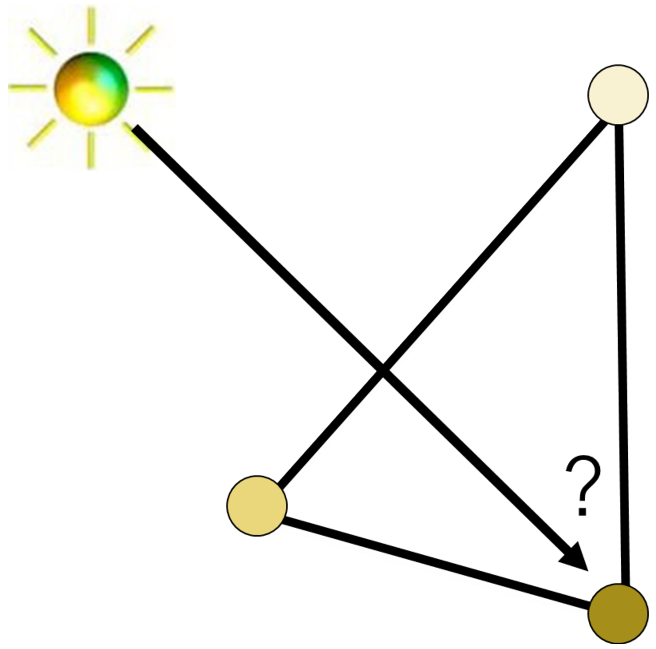
UNIVERSITY OF MINNESOTA
Driven to Discover®

Lighting and Shading

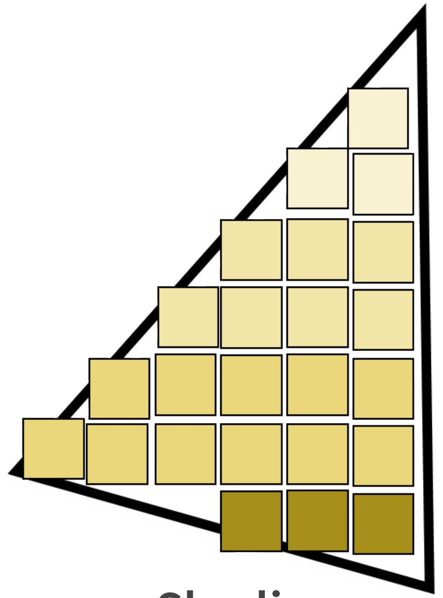
CSCI 4611: Programming Interactive Computer Graphics and Games

Evan Suma Rosenberg | CSCI 4611 | Fall 2022

Lighting and Shading



Lighting



Shading

Lighting models the light/surface interactions to determine the final color and brightness for a specific point on a surface.

Shading applies the lighting model to the entire surface, "filling in" each triangle.

Illumination Models

An illumination model describes inputs, assumptions, and outputs used to calculate the illumination of a specific point on a surface.

Today we will learn an equation for calculating the lighting at each vertex.

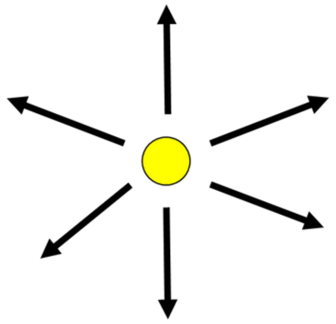
Technically, this is different than a shading model, which tells us how to shade every spot on the entire object.

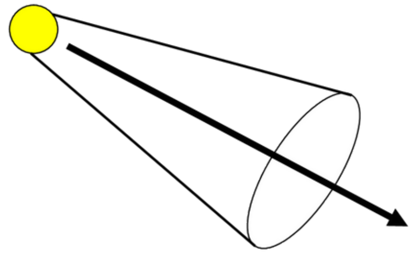
This involves calculate the light also inside each triangle.

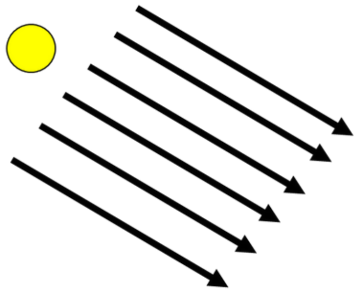
For a specific point *p* on a surface, how bright should it be given:

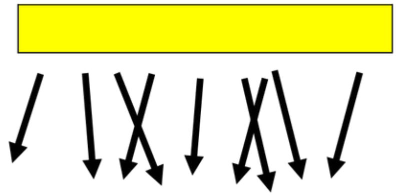
- Light attributes (intensity, color, position, direction, shape).
- Object surface attributes (color, reflectivity, transparency, etc).

Basic Light Sources


Point Light

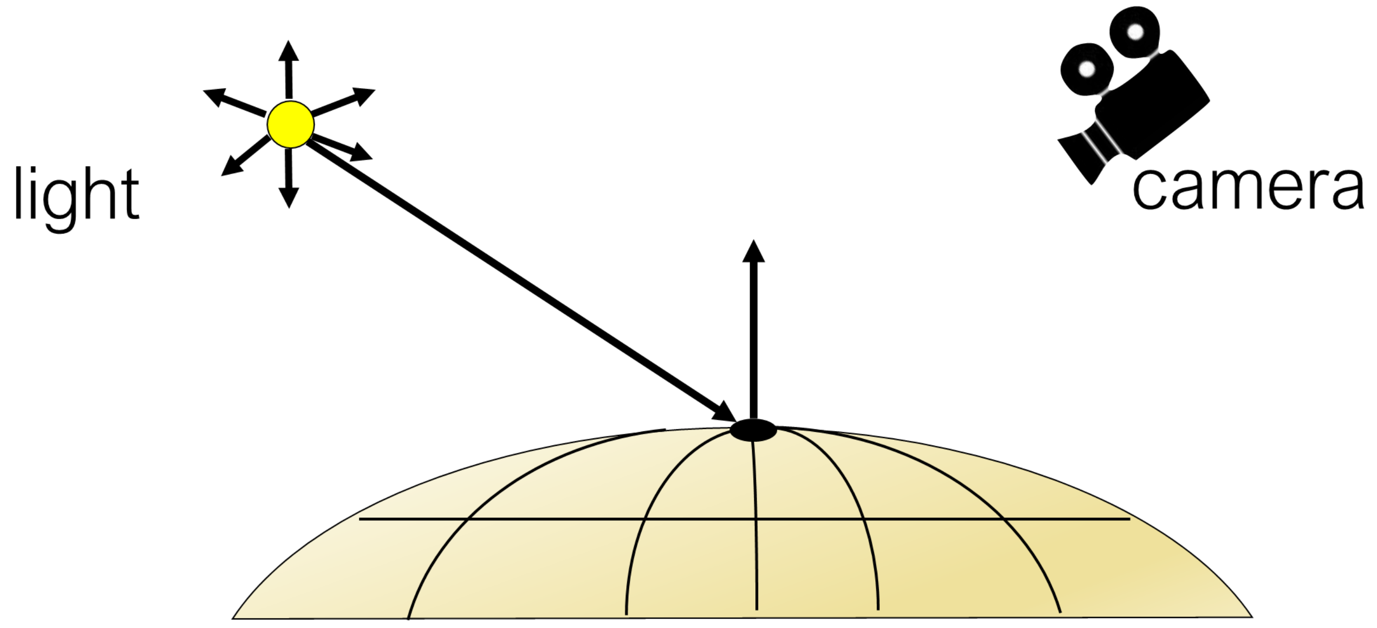

Spot Light


Directional Light


Area Light

As part of some illumination models, you can specify whether the light intensity varies with the distance between the object and the light source.

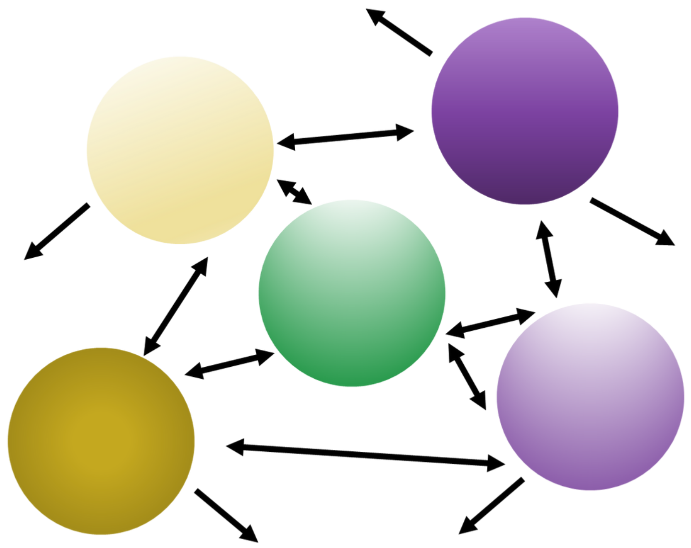
Local Illumination

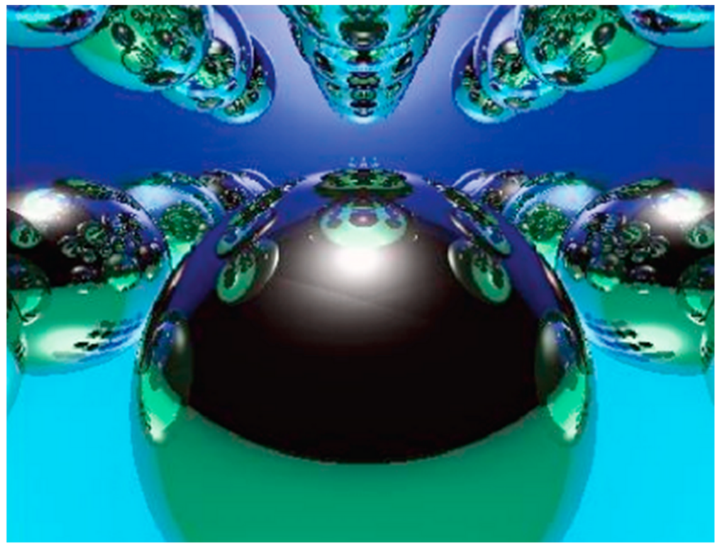


- **Local illumination** only considers the light, the camera position, and the object material properties. This is commonly used in games because it's fast!

  However, it leaves out things realistic effects like shadows, reflections, etc.

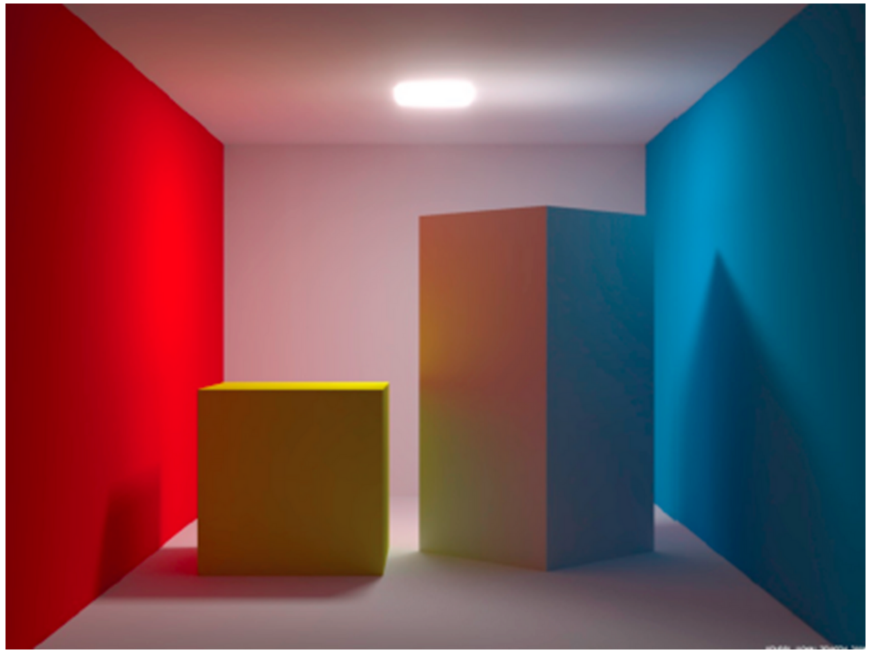
Global Illumination





- On the other hand, we have **global illumination**, which takes into account the interaction of light from all the surfaces in the scene.
- Ray tracing is the most famous example of global illumination, and a really fun algorithm to write – you will do this in CSCI 5607. It models light bouncing around.

Global Illumination



- **Radiosity** is another example that models energy moving from emitters (lights) into the scene.

  This illumination model is view independent.

Simple Local Illumination

The model used by OpenGL and nearly all games.

Real-time ray tracing is a fairly recent development.

We reduce the complex workings of light to three components:

- Ambient
- Diffuse
- Specular

Then, we calculate the final illumination at a point.

ambient + diffuse + specular

To model a variety of different materials, we include reflection coefficients that can reflect each light component differently.

These constants are typically represented as ***Ka***, ***Kd***, ***Ks***.

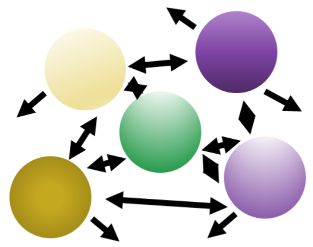
Ambient Light Contribution

$$\text{Ambient} = K_a \times I$$

where:

K_a = ambient reflection coefficient.

I = light intensity



Ambient light is background illumination.

Diffuse surfaces reflect light. Some light goes to eye, some goes to scene.

Light bounces off of other objects and eventually reaches this surface element. This is expensive to keep track of accurately, so we use a bit of a hack instead.

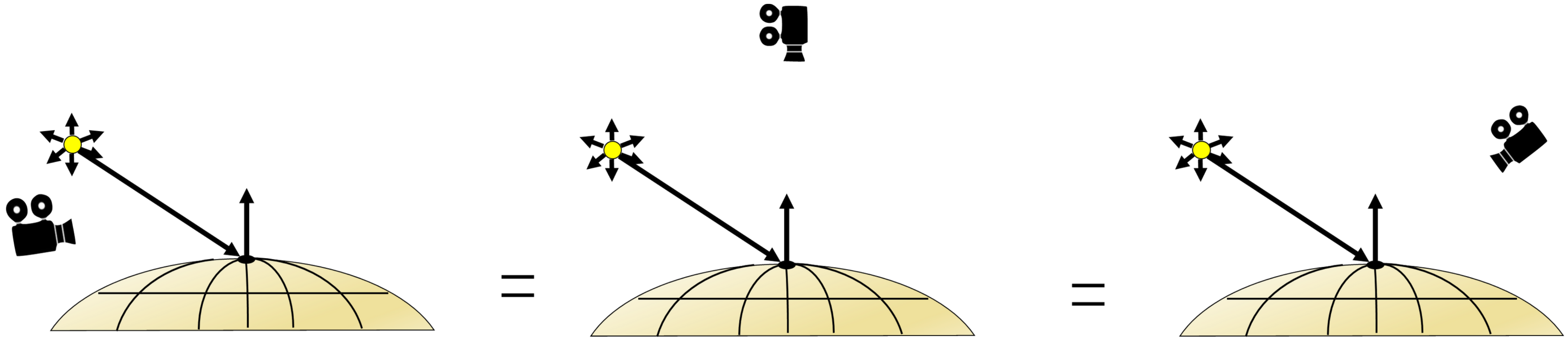
Ambient component

- constant
- independent of object position and viewer position
- exists in most environments

  some light hits surface from all directions (a crude approximation of indirect lighting/global illumination)

  images without some form of ambient lighting look stark, they have too much contrast

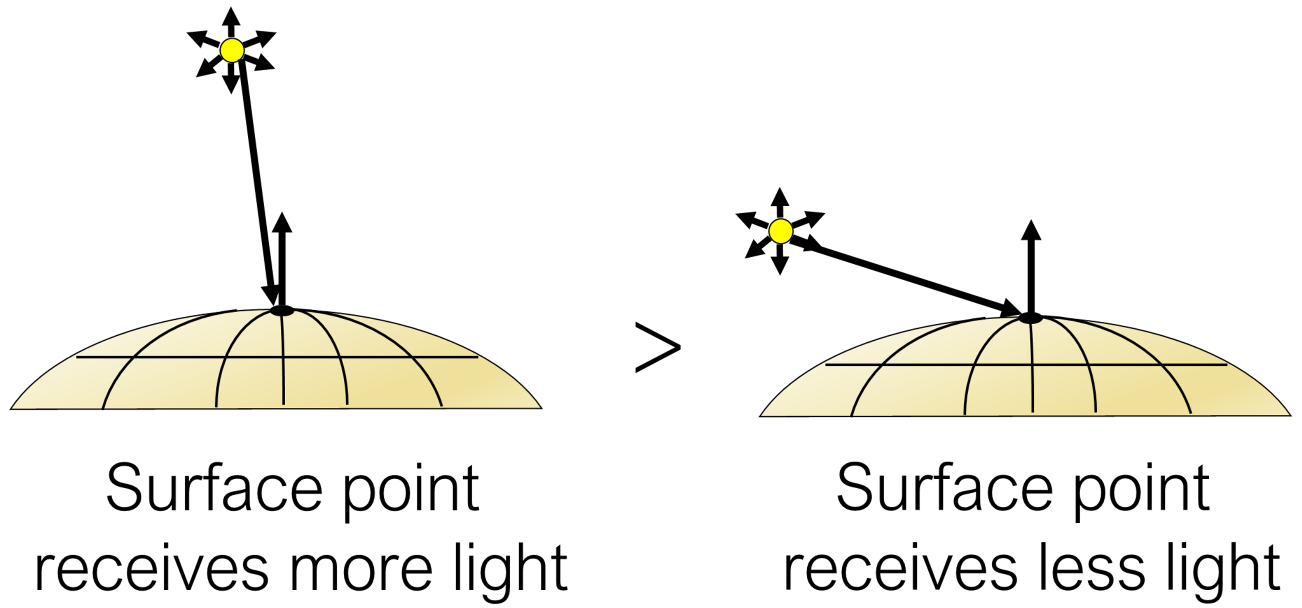
Diffuse Light Contribution



Diffuse light is the illumination that a surface receives from a light source that reflects equally in all directions.

The camera’s position or "eye point" does not matter.

Diffuse Light Calculation


>
Surface point receives more light
Surface point receives less light

Lambert's Law tells us the amount of light received at a given point *p* on the surface is proportional to the angle between the direction to the light and the surface normal.

Diffuse Light Calculation

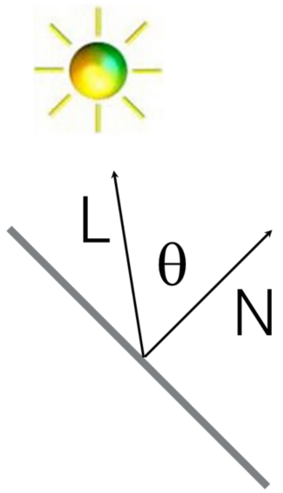
Lambert's Law: The radiant energy D that a small surface patch receives from a light sources is:

Diffuse = K_d x I x cos(theta)

K_d = diffuse reflection coefficient

I = light intensity

theta = angle between the light vector and the surface normal



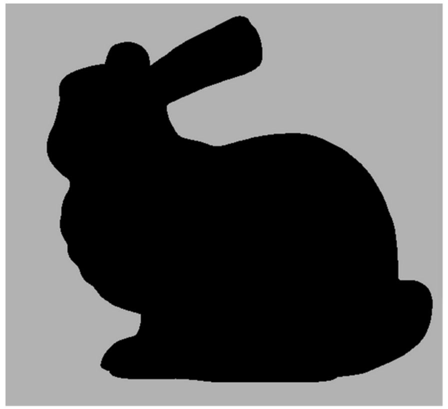
How do we calculate cos(θ)?

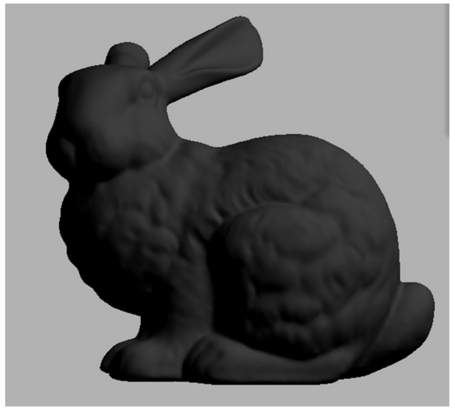
N dot L
(this is a familiar saying in the gfx community)

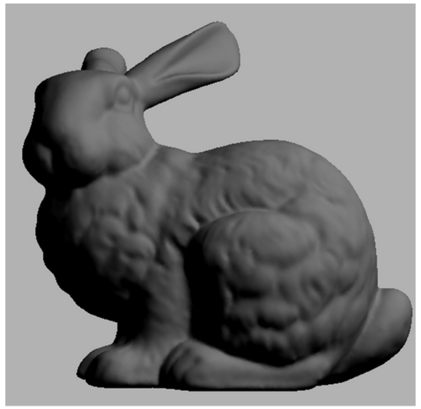
Diffuse Light Examples

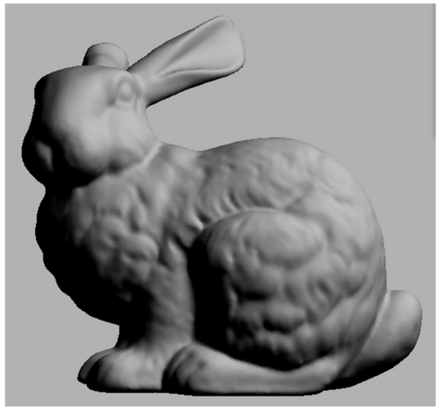
Diffuse = K_d x I x cos(theta)

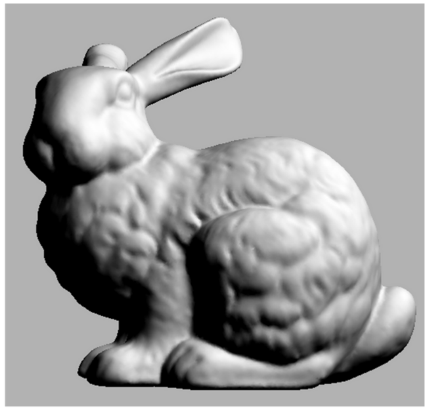
for I = 1.0


Kd = 0.0


Kd = 0.25


Kd = 0.5

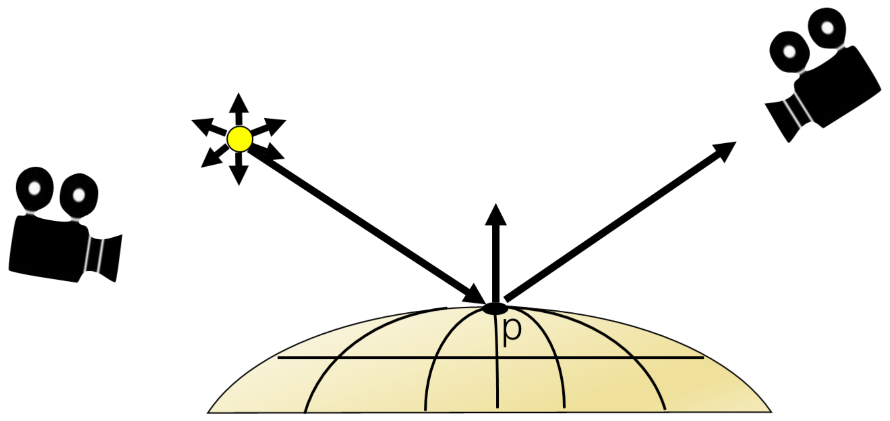

Kd = 0.75


Kd = 1.0

Specular Light Contribution

2. But when the camera is over here, point p will not have any specular reflection on it.

3. Where would a specular reflection show up on the surface when the camera is here?



1. When the camera is here, point p will have a lot of specular reflection on it

Specular light is directed reflection from shiny surfaces.

Dependent on light source position and viewer position.

Typical of bright, shiny surfaces, e.g. mirrors.

Color depends on material and how it scatters light energy

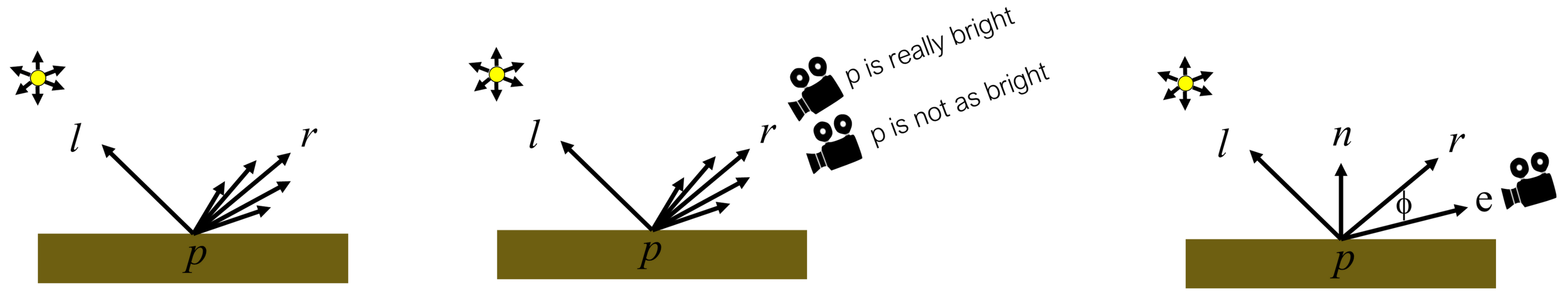
- in plastics: color of point light source
- in metal: color of metal

  in others: combine color of light and material

Specular Light Calculation

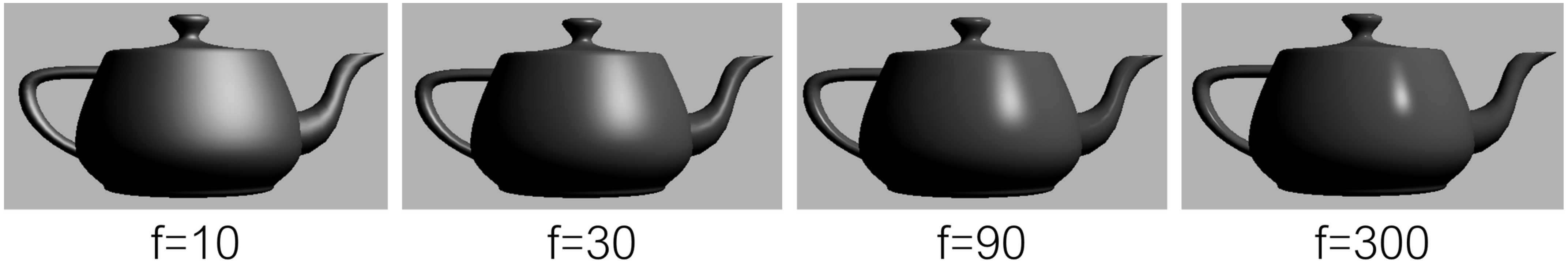
How much reflection you see depends on where you are.

But.. for non-perfect surfaces, you will still see a specular highlight when you move a little bit away from the ideal reflection direction.



ϕ is the angle between the view direction and a perfect mirror reflection.
When ϕ is small, you see more specular highlight.

Specular Light Calculation



The Phong lighting model:

Specular = K_s x I x cos(theta)^f

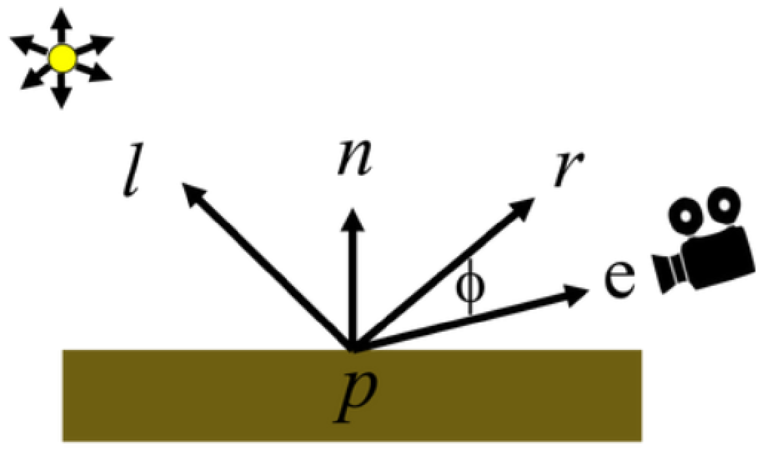
As *f* increases, the highlight is more concentrated and the surface appears glossier.

Specular Light Calculation

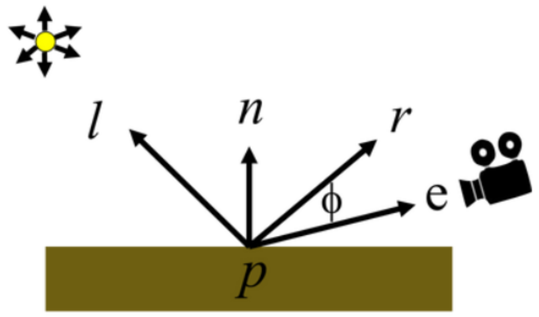
The Phong lighting model:

Specular = K_s x I x cos(theta)^f

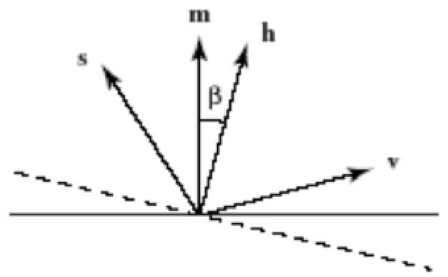
Specular = K_s x I x $(e \cdot r)^f$



Specular Light Calculation



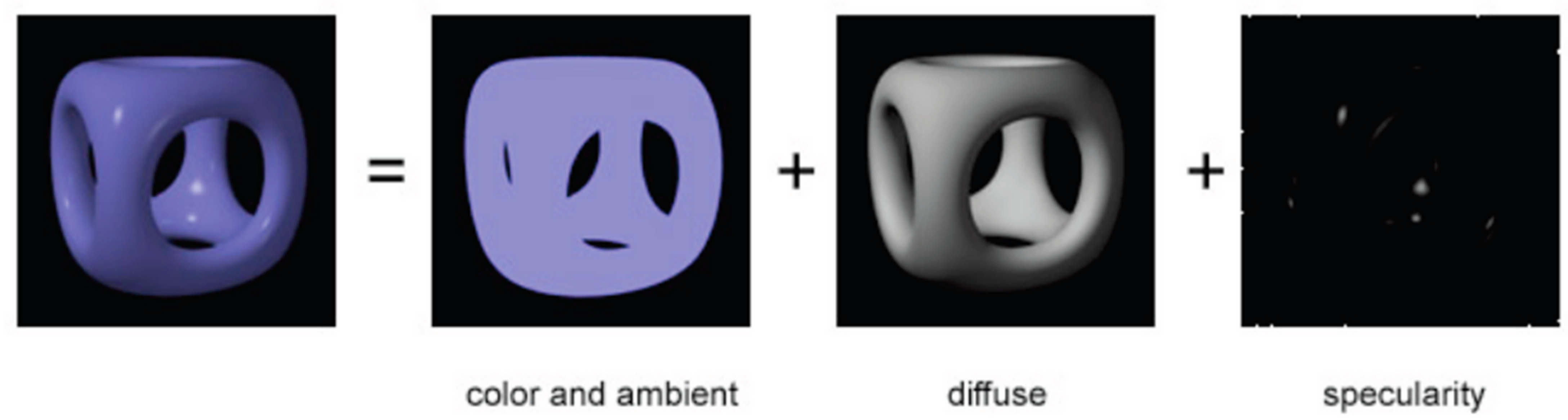
Using Halfway Vector



- Using the "halfway vector" is common practice because calculating the reflection vector is expensive.
- Rather than calculating the exact specular reflection vector **r**, we can also calculate the halfway vector:
  h = **l** + **e**
- Then the angle **β** between **h** and **n** (the surface normal) approximately measures the falloff of intensity.
- Note that if **h** is aligned with **n,** then the viewer will see the brightest specular highlight.

  Since this is an approximation, it's not exactly the same result as what we described previously, but if we pick a different value of the exponent, then we can achieve similar, physically plausible results.

Putting It Together



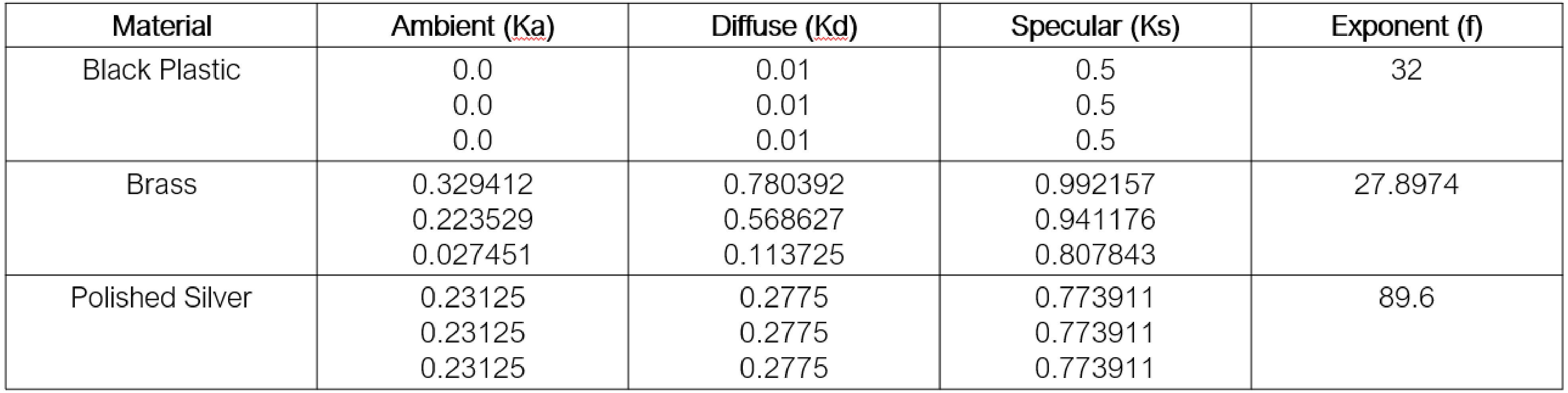
Adding Color

Sometimes the light or surfaces are colored.

The light intensity comes from the I_a, I_d, I_s constants in our equation. If we want colored light, this means that constants need to have different values for R, G, and B.

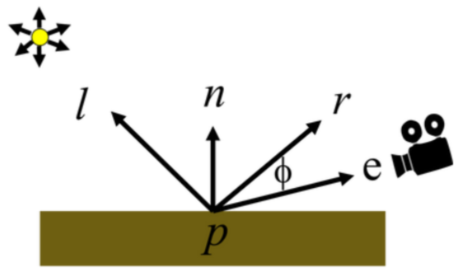
The intensity and reflection coefficients are really 3-element arrays, with separate floats for R, G, and B.

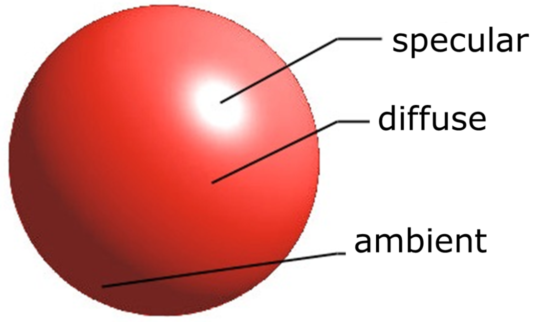
Example materials from Hill (Computer Graphics Using OpenGL):



Material	Ambient (Ka)	Diffuse (Kd)	Specular (Ks)	Exponent (f)
Black Plastic	0.0 0.0 0.0	0.01 0.01 0.01	0.5 0.5 0.5	32
Brass	0.329412 0.223529 0.027451	0.780392 0.568627 0.113725	0.992157 0.941176 0.807843	27.8974
Polished Silver	0.23125 0.23125 0.23125	0.2775 0.2775 0.2775	0.773911 0.773911 0.773911	89.6

The Lighting Equation





$$light\_intensity = \underbrace{k_a\, I_a}_{\text{ambient}} + \underbrace{k_d\, I_d\, (n \cdot l)}_{\text{diffuse}} + \underbrace{k_s\, I_s\, (h \cdot n)^s}_{\text{specular}}$$

🟥🟩🟦 (under $light\_intensity$, k_a, I_a, k_d, I_d, k_s, I_s)

or

$$(e \cdot r)^s$$

🟥🟩🟦 = because we can have colored lights or objects, these are 3D arrays (vec3) of floats for r, g, b

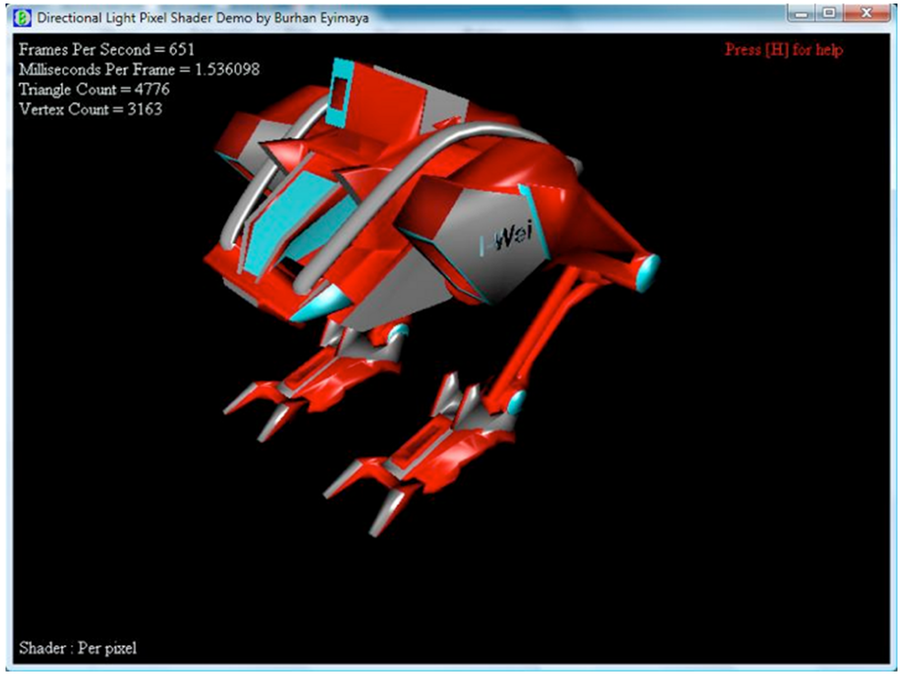
If there are N lights, then the **total illumination** for a point is the sum of the equation above for all lights.

Normals Are Critical for Lighting!

- All these calculations rely upon normals.
- If your surface normals are bad, your lighting will also be bad!
- Note that we've also been assuming that the normals are unit length. Otherwise the lighting calculation will have a scale factor in it.

  This is a common bug. Remember to normalize your vectors!

Shading Models



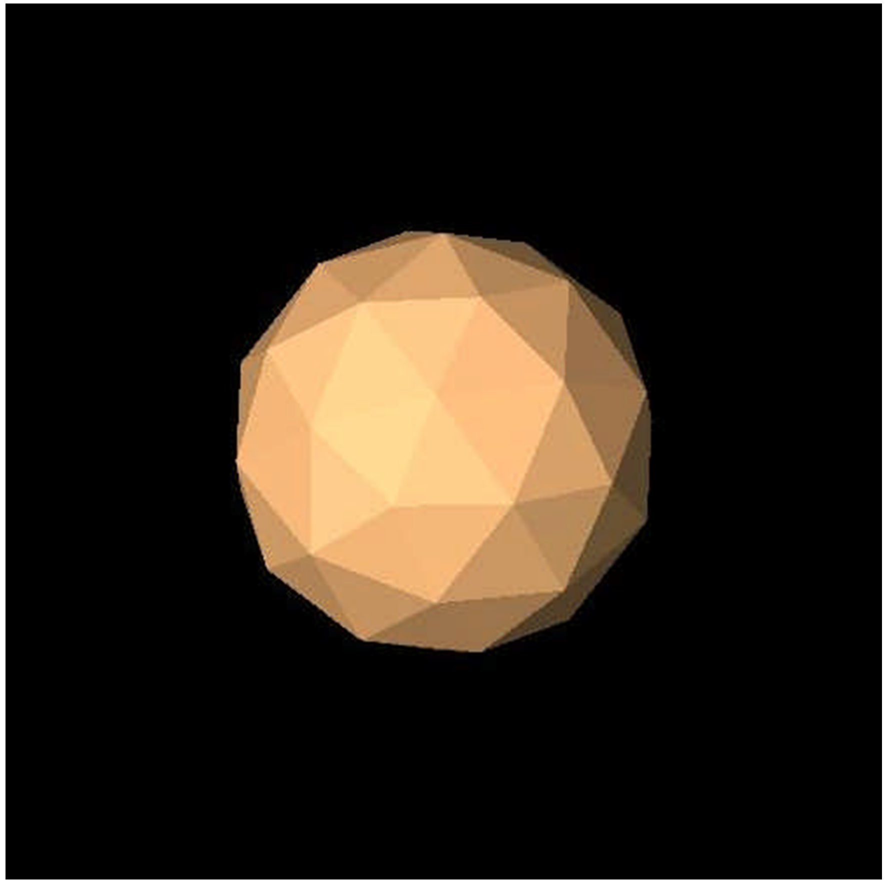
```
Frames Per Second = 651
Milliseconds Per Frame = 1.536098
Triangle Count = 4776
Vertex Count = 3163

Shader : Per pixel
```

Remember, everything we talked about today relates to calculating the lighting at just a **single point**.

We need to calculate the lighting at many points on the surface so that we can color the whole object.

This process is called **shading.**

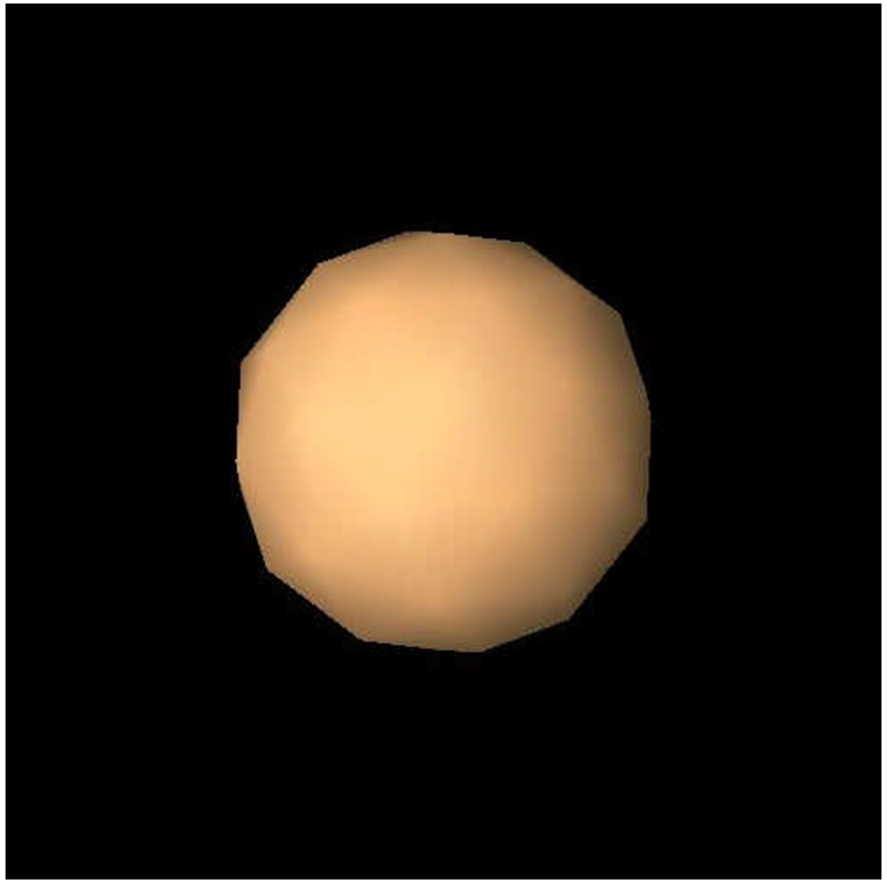
Flat vs. Smooth Shading



Flat Shading

Uses face normals.

Looks OK for a cube, but not for a sphere.

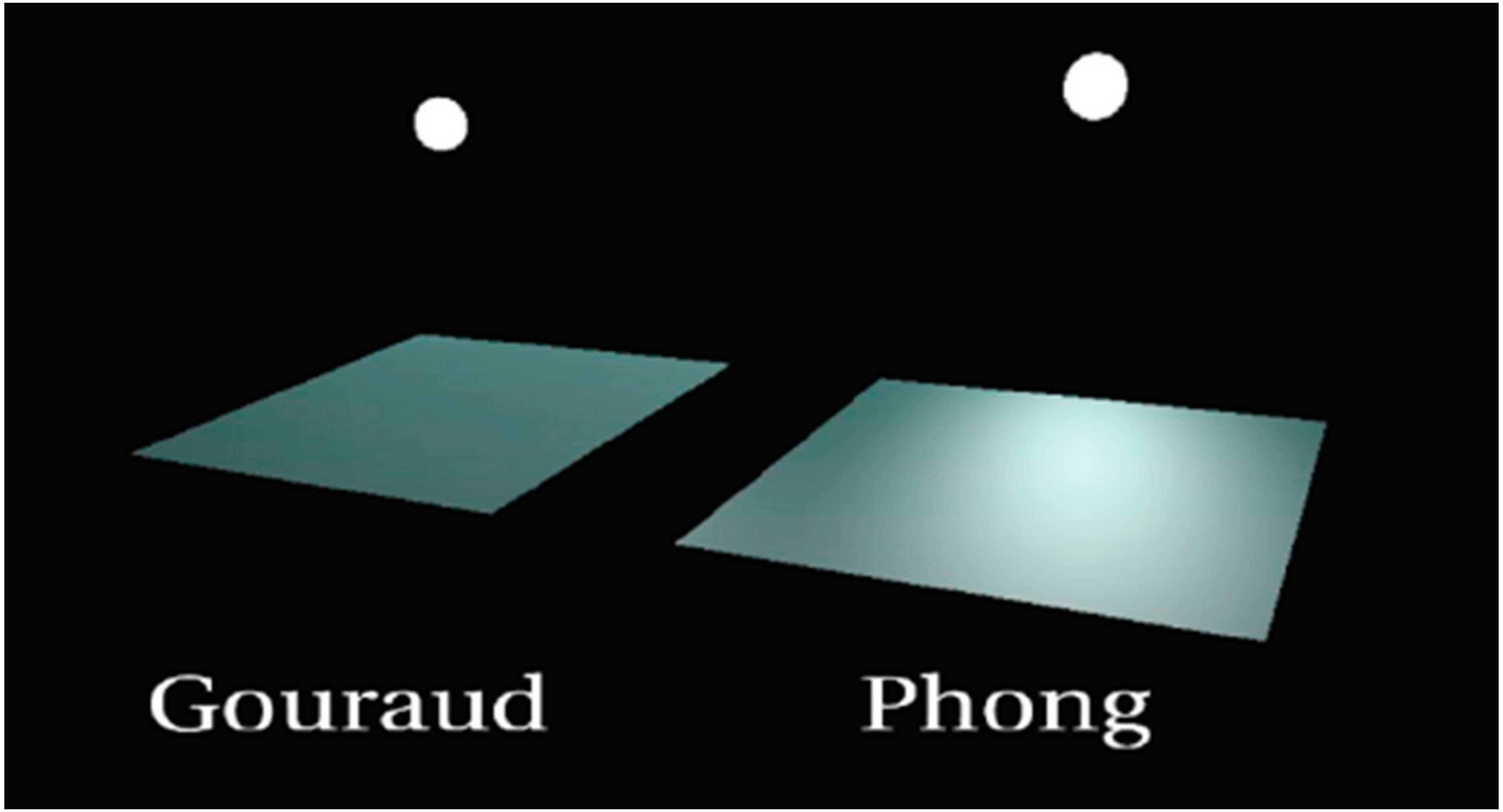


Smooth Shading

Per-vertex or per-pixel normals.

Best for smooth shapes.

Smooth Shading Models



Gourad vs. Phong Shading

Gouraud

Per-Vertex Lighting — this means lighting is calculated once for each vertex.

Shading a whole triangle requires **interpolating the colors** calculated at the vertices.

Faster, each triangle requires just **three lighting calculations**.

The only style of shading supported by “old-school” OpenGL.

Phong

Per-Pixel Lighting — this means lighting is calculated once for each pixel.

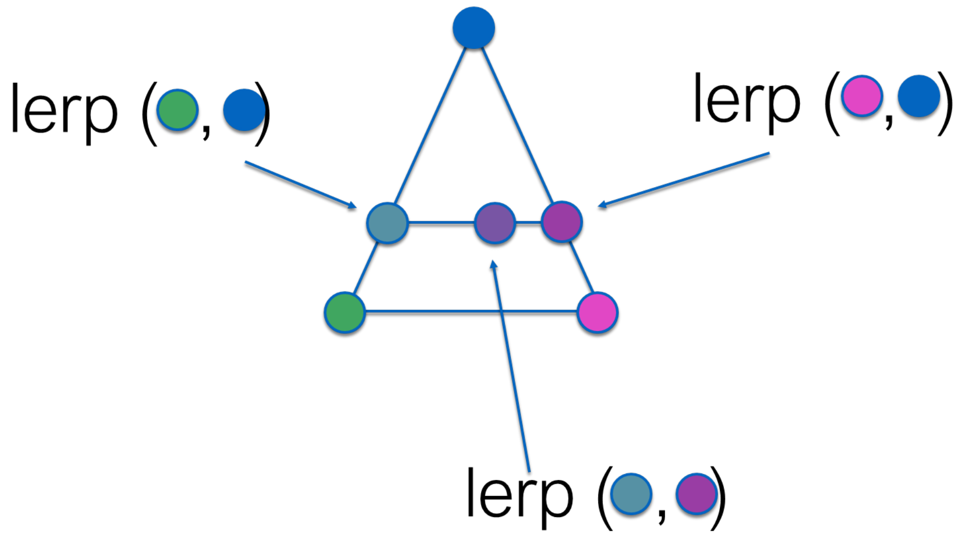
Shading a whole triangle requires **interpolating the normals** at each vertex for each pixel.

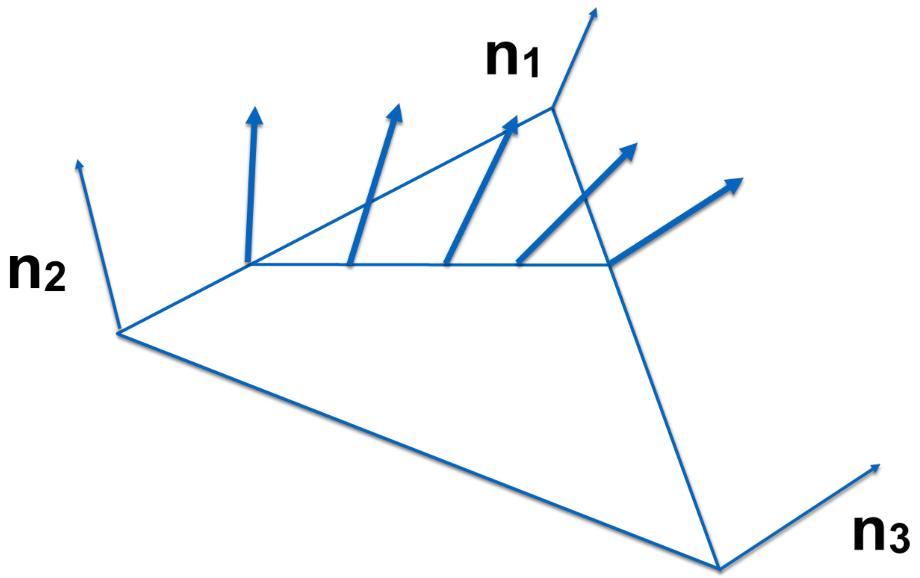
Slower, each triangle, requires **many, many lighting calculations**.

Shader programs in modern OpenGL make these calculations fast enough for games.

Gouraud

Phong





Here, the vertex colors are calculated first using a lighting model like:

$$C = K_a \times I_a + K_d \times I_d \times (n \bullet l) + K_s \times I_s \times (h \bullet n)^s$$

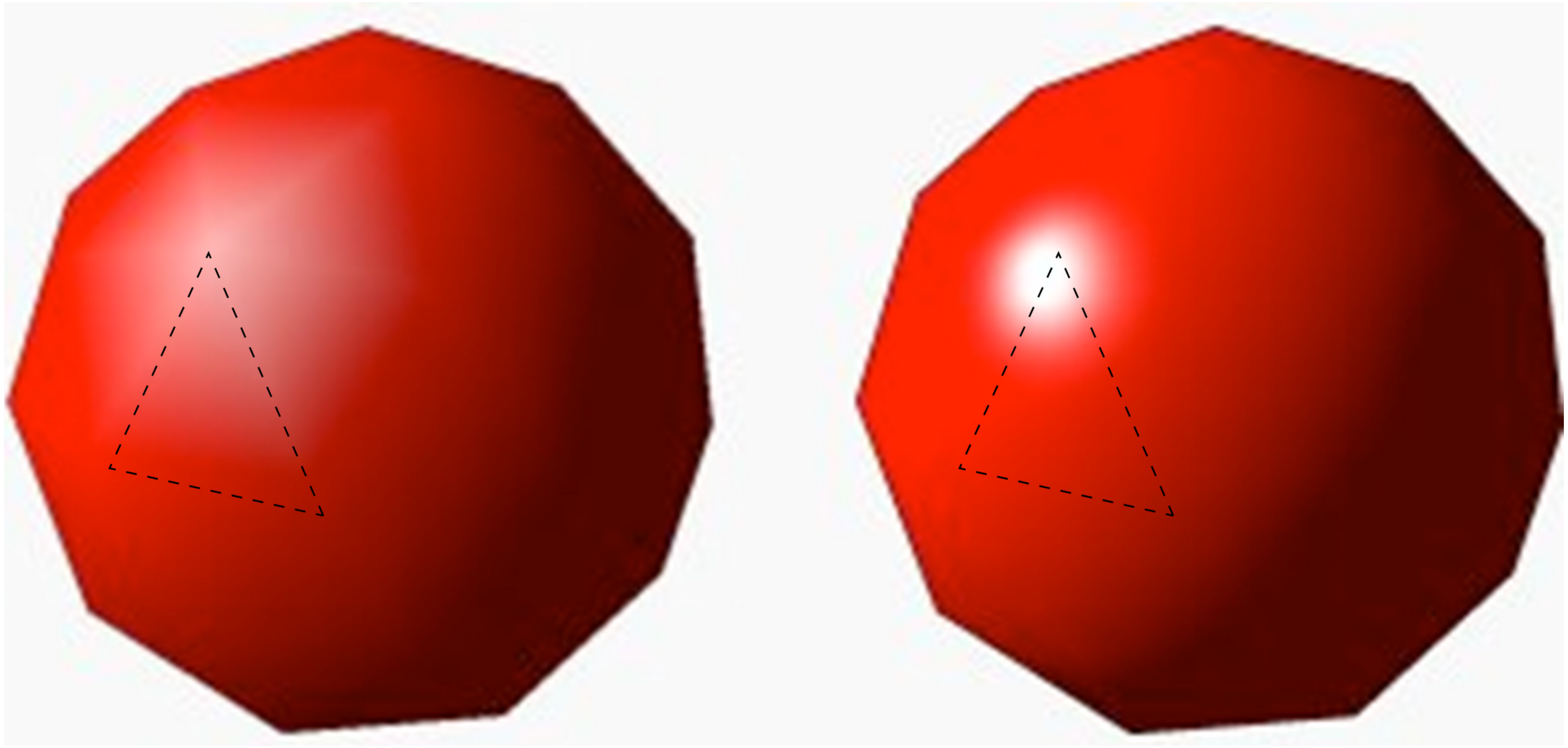
Then, the color for each interior pixel is set via trilinear interpolation.

Here, the vertex normals n1, n2, and n3 are interpolated to find a normal each individual pixel.

Then, using this pixel-specific normal, the lighting for each pixel is calculated using a lighting model like:

$$C = K_a \times I_a + K_d \times I_d \times (n \bullet l) + K_s \times I_s \times (h \bullet n)^s$$

Gouraud **Phong**



Assignment 5

Artistic Rendering

https://github.com/CSCI-4611-Fall-2022/Assignment-5