COMP3271 Computer Graphics

# Illumination & shading

2019-20

# Objectives

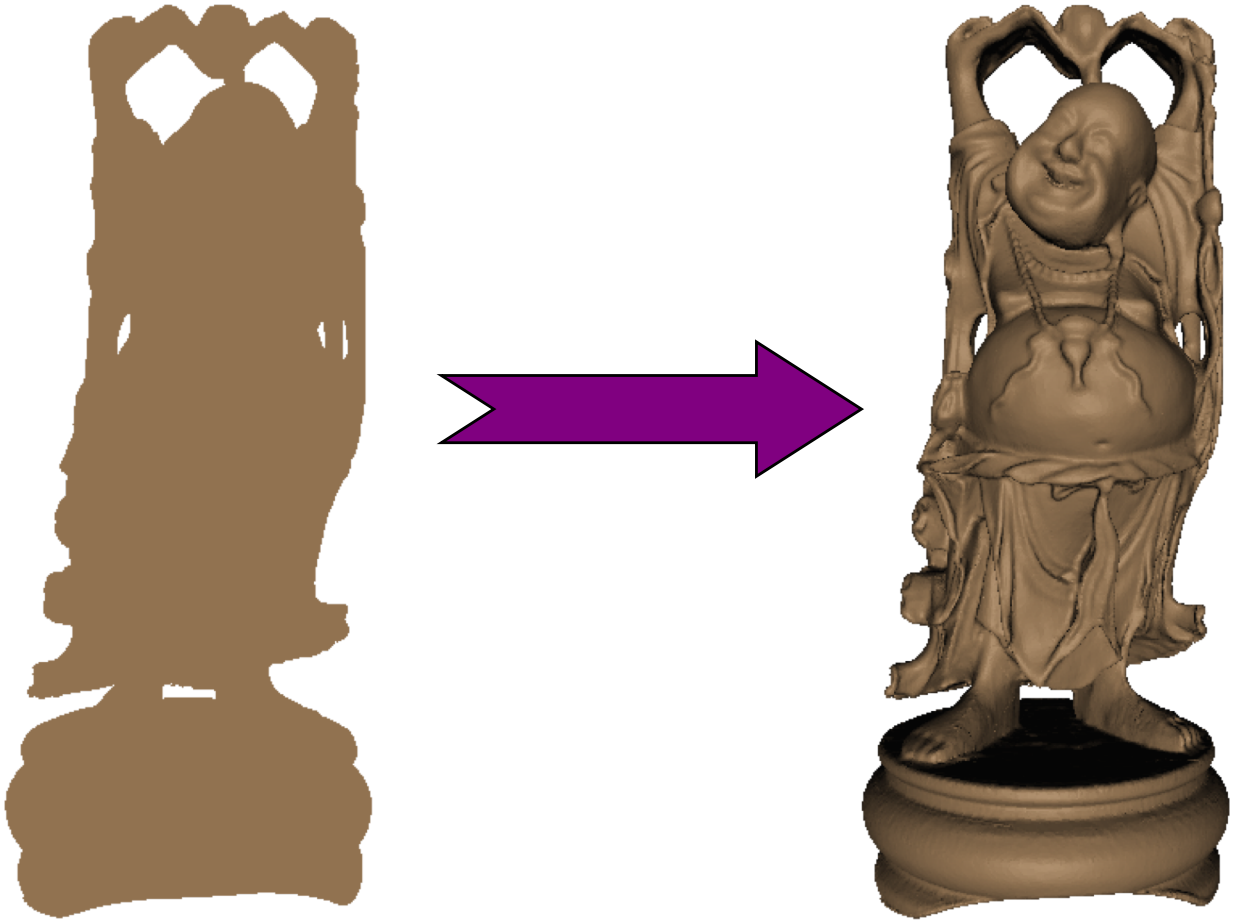
Phong illumination model

- Diffuse reflection
- Specular reflection
- Ambient

Polygonal shading

- Flat shading
- Gouraud shading
- Phong shading

# Illumination & Shading



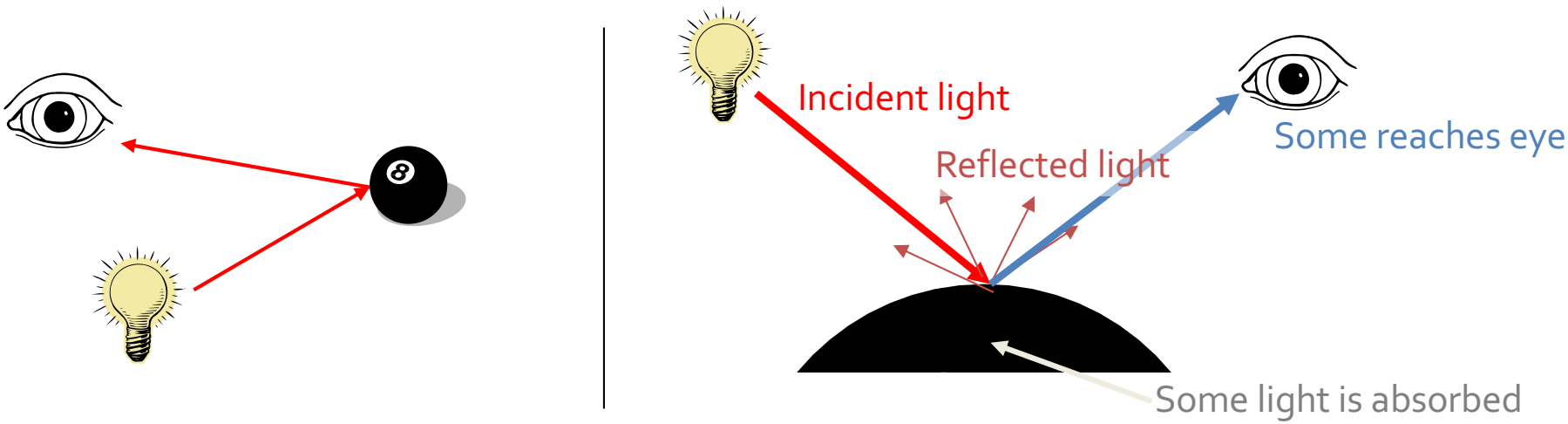
(1 million triangles drawn with 1 color per triangle)

# Determining an Object's Appearance

Ultimately, we're interested in modeling light transport in scene

- Light is emitted from light sources and interacts with surfaces
- on impact with an object, some is reflected and some is absorbed
- distribution of reflected light determines "finish" (matte, glossy, …)
- composition of light arriving at eye determines what we see

Let's focus on the local interaction of light with single surface point



# Modeling Light Sources

In general, light sources have a very complex structure

- incandescent light bulbs, the sun, CRT monitors, …

To simplify things, we'll focus on point light sources for now

- light source is a single infinitesimal point
- emits light equally in all directions (isotropic illumination)
- outgoing light is set of rays originating at light point

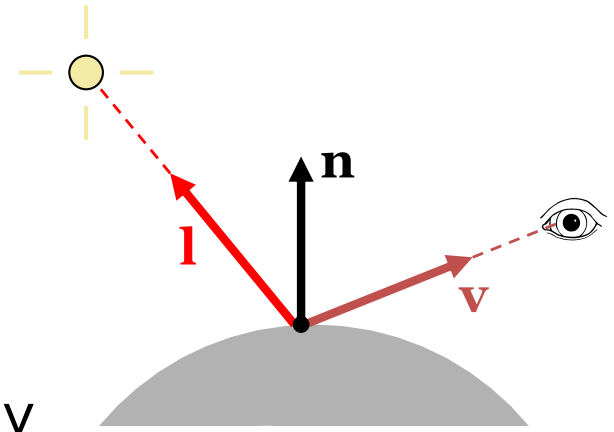
Creating lights in OpenGL

- `glEnable(GL_LIGHTING)` — turn on lighting of objects
- `glEnable(GL_LIGHT0)` — turn on specific light
- `glLight(…)` — specify position, emitted light intensity, …

# Basic Local Illumination Model

We're only interested in light that finally arrives at view point

- a function of the light & viewing positions
- and local surface reflectance




Characterize light using RGB triples

- can operate on each channel separately

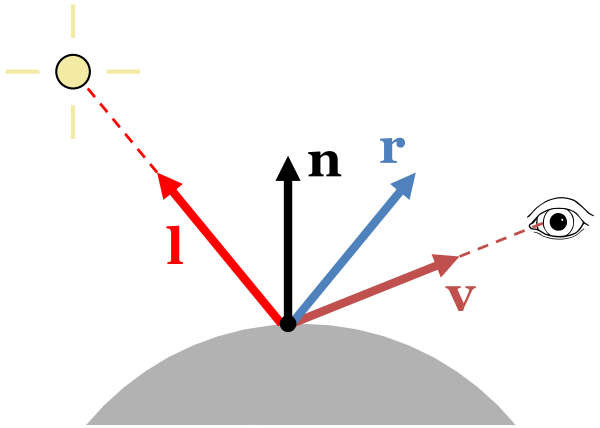
Given a point, compute intensity of reflected light

# Phong Illumination Model

A simple model that can be computed rapidly

Has three components

- Diffuse
- Specular
- Ambient

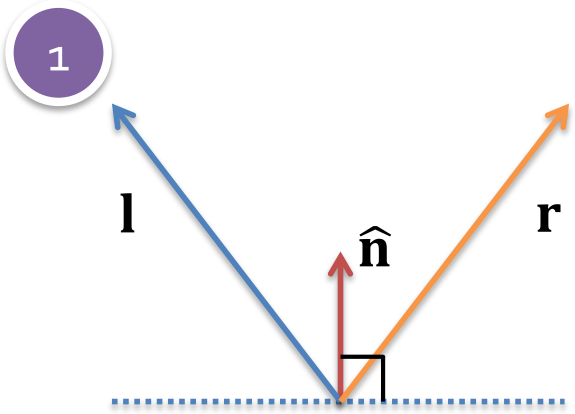


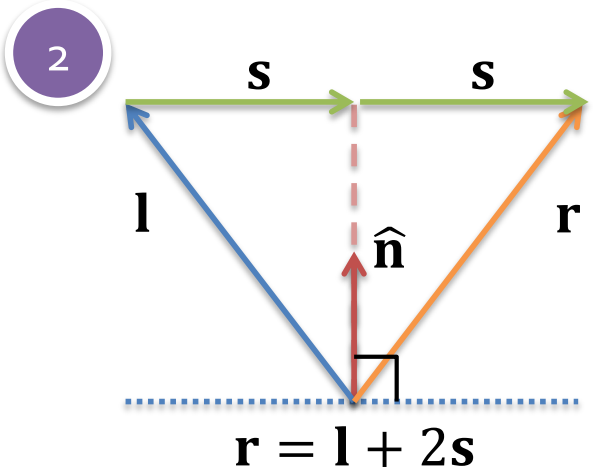

Uses four vectors

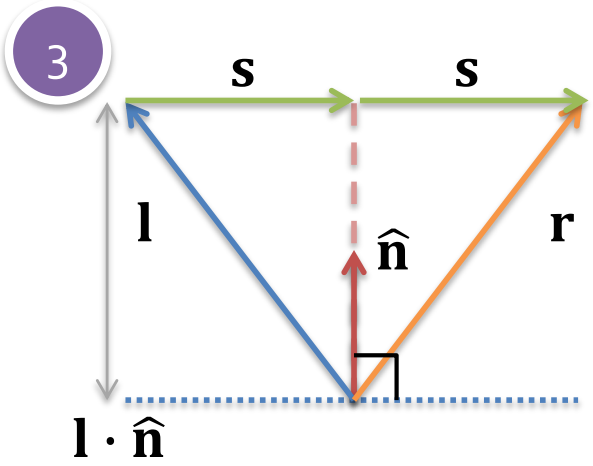
- To light source (**l**)
- To viewer (**v**)
- Surface normal (**n**)
- Perfect reflector (**r**)

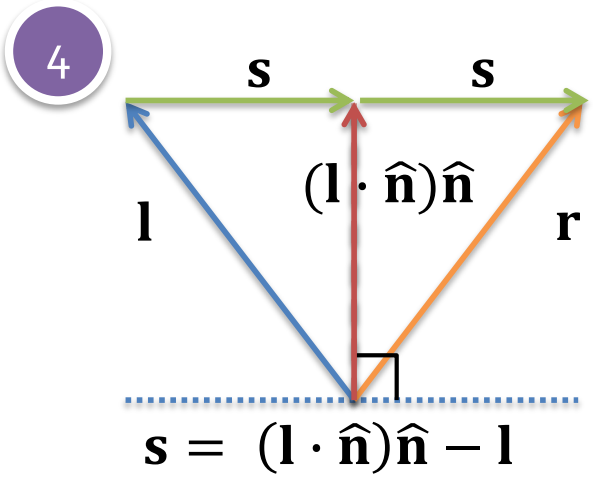
# Computing the Perfect Reflector

How to obtain the perfect reflector $\mathbf{r}$?




$$\begin{aligned}\mathbf{r} &= \mathbf{l} + 2\mathbf{s} \\ &= \mathbf{l} + 2((\mathbf{l}\cdot\hat{\mathbf{n}})\hat{\mathbf{n}} - \mathbf{l}) \\ &= 2(\mathbf{l}\cdot\hat{\mathbf{n}})\hat{\mathbf{n}} - \mathbf{l}\end{aligned}$$

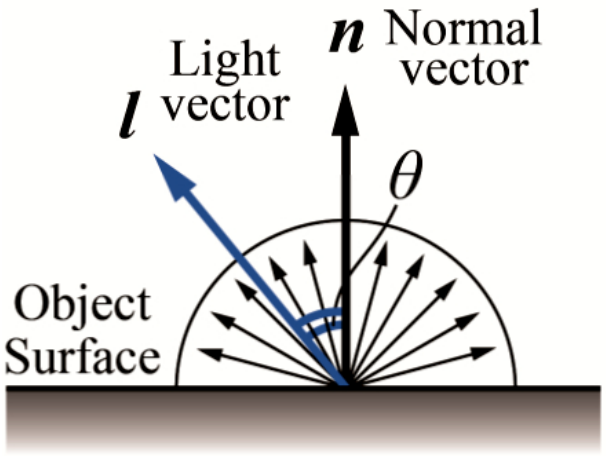
# Diffuse Reflection

This is the simplest kind of reflection

- also called Lambertian reflection
- models dull, matte surfaces — materials like chalk

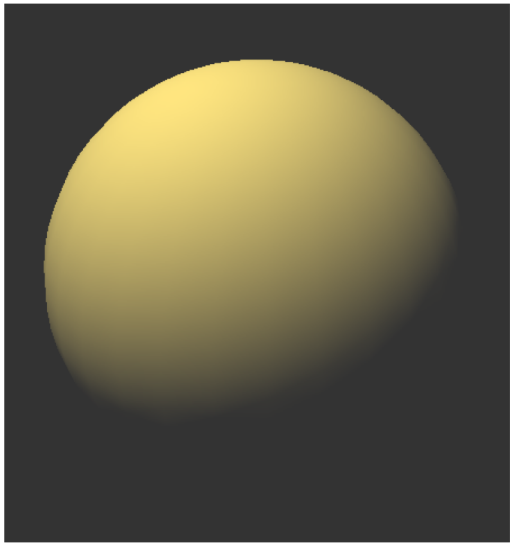
Ideal diffuse reflection

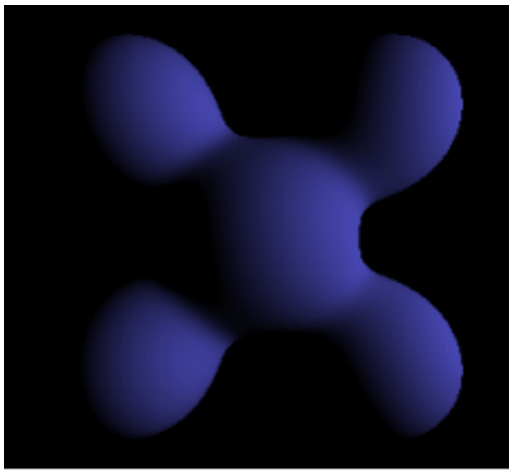
- scatters incoming light equally in all directions
- identical appearance from all viewing directions
- reflected intensity depends only on direction of light source



# Lambert's Law for Diffuse Reflection

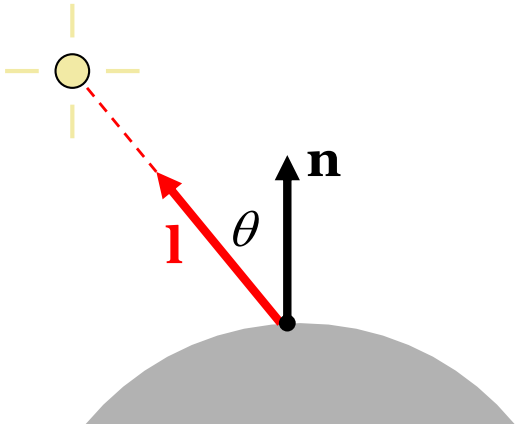
Purely diffuse objects





Intensity due to diffuse reflection

$$I_L k_d \cos\theta = I_L k_d (\mathbf{n} \cdot \mathbf{l})$$




$I_L$ : light source intensity

$k_d$ : diffuse surface reflectance coefficient, $k_d \in [0,1]$

$\theta$ : angle between normal & light direction

In practice, we use $\max(I_L k_d (\mathbf{n} \cdot \mathbf{l}), 0)$ to avoid lighting backfacing surfaces

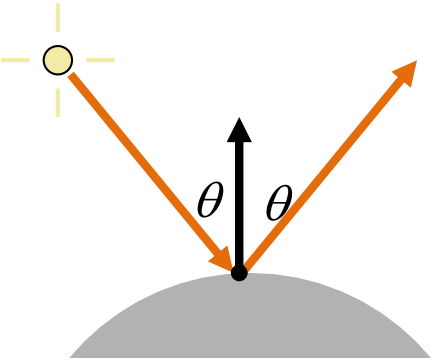
# Specular Reflection

Diffuse reflection is nice, but many surfaces are shiny

- their appearance changes as the viewpoint moves
- they have glossy specular highlights (or specularities)
- because they reflect light coherently, in a preferred direction

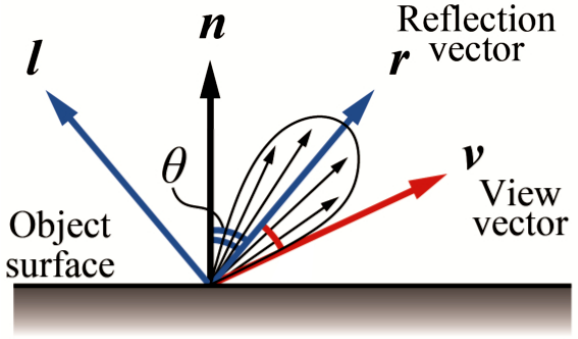
A mirror is a perfect specular reflector

- incoming ray reflected about normal direction
- nothing reflected in any other direction




Most surfaces are imperfect specular reflectors

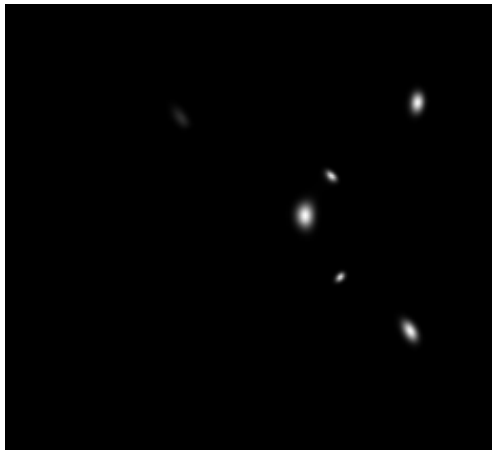
- reflect rays in cone about perfect reflection direction



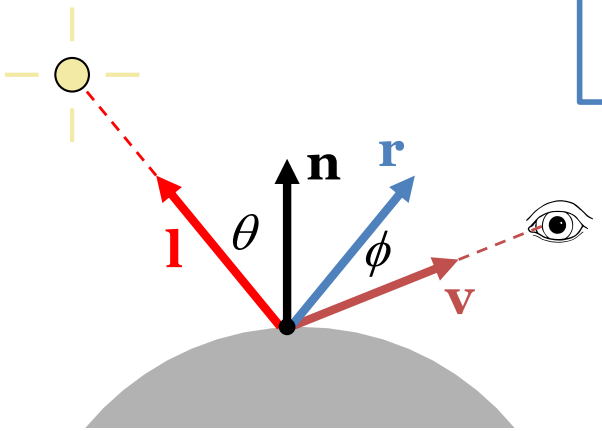
# Phong Specular Model

One particular specular reflection model

- quite common in practice
- it is purely empirical
- there's no physical basis for it



Good for modeling metallic surfaces




Intensity due to specular reflection

$$I_L k_s (\mathbf{r} \cdot \mathbf{v})^{\alpha} = I_L k_s \cos^{\alpha} \phi$$

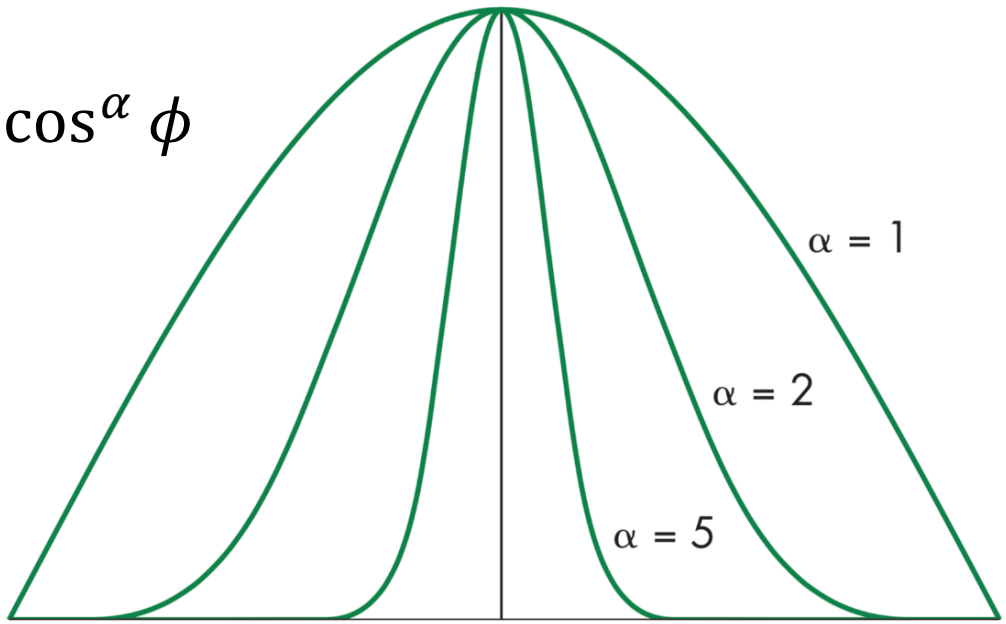
$I_L$ : light source intensity
$k_s$: specular surface reflectance coefficient, $k_s \in [0,1]$
$\phi$ : angle between viewing and reflection direction
$\alpha$ : shininess factor

# The Shininess Coefficient

$\alpha$ approaching to infinity corresponds to perfect mirror

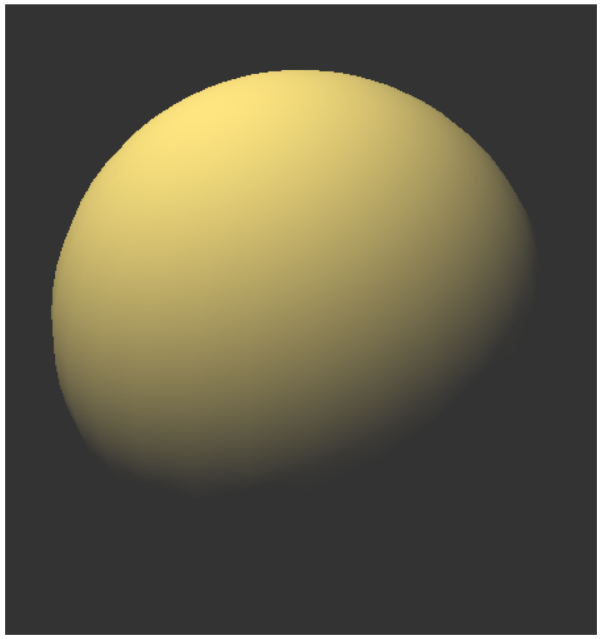
Values of $\alpha$ between 100 and 500 correspond to metals

Values between 5 and 10 give surface that look like plastic

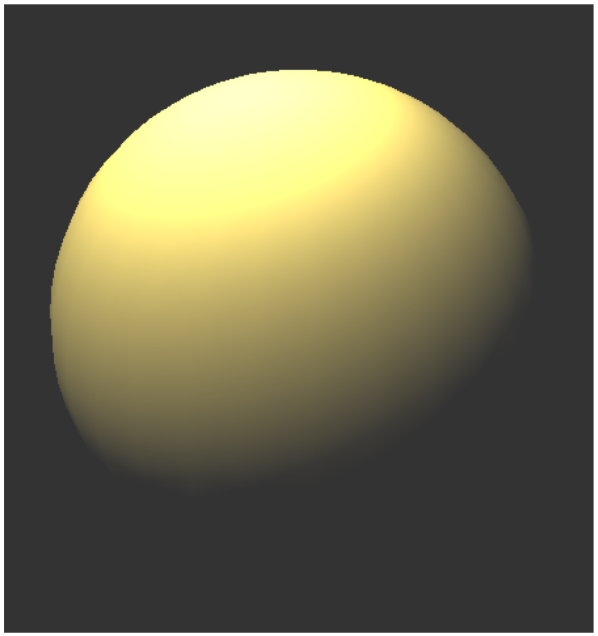


# Examples of Phong Specular Model

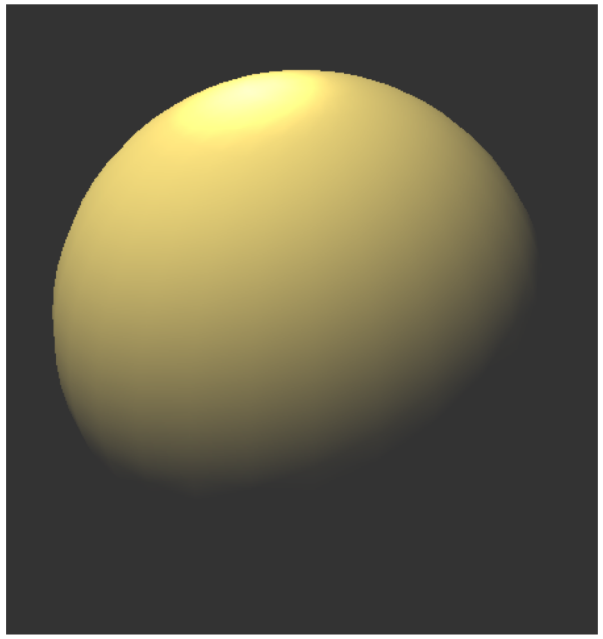
Diffuse only



Diffuse + Specular
(shininess 5)



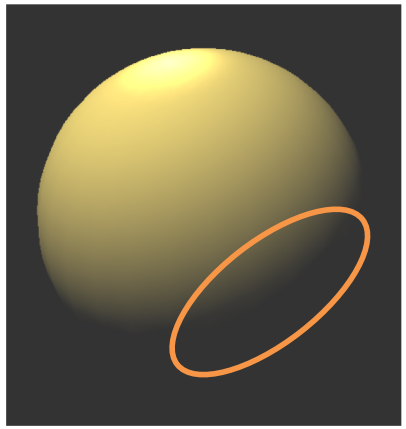
Diffuse + Specular
(shininess 50)



# Ambient Light

So far, areas not directly illuminated by any light appear black

- this tends to look rather unnatural
- in the real world, there's lots of ambient light



To compensate, we invent new light source

- assume there is a constant ambient "glow"
- this ambient glow is purely fictitious

Just add in another term to our illumination equation

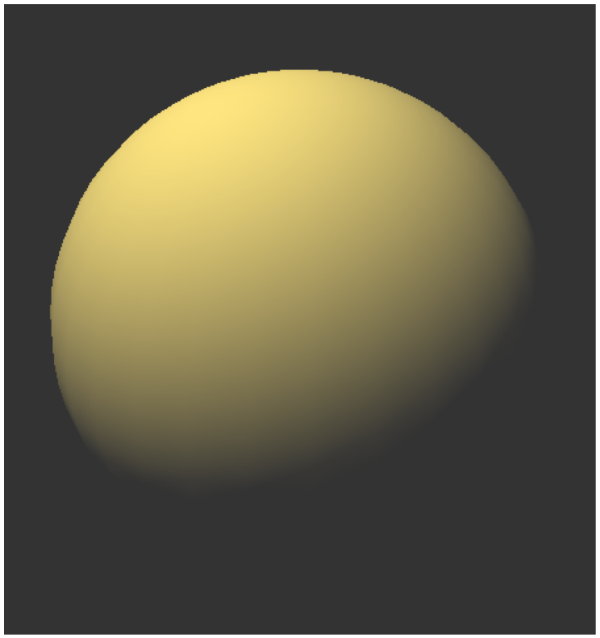
Intensity due to ambient light

$$I_a k_a$$

$I_a$ : ambient light intensity
$k_a$ : ambient surface reflectance coefficient

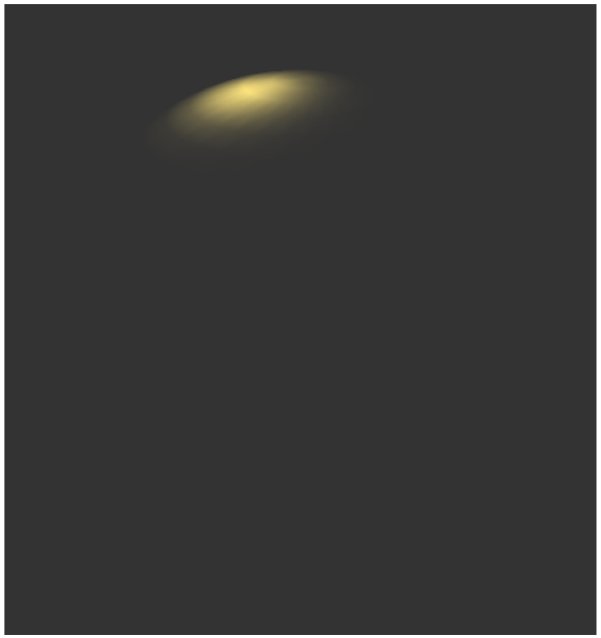
# Three Basic Components of Illumination

Diffuse

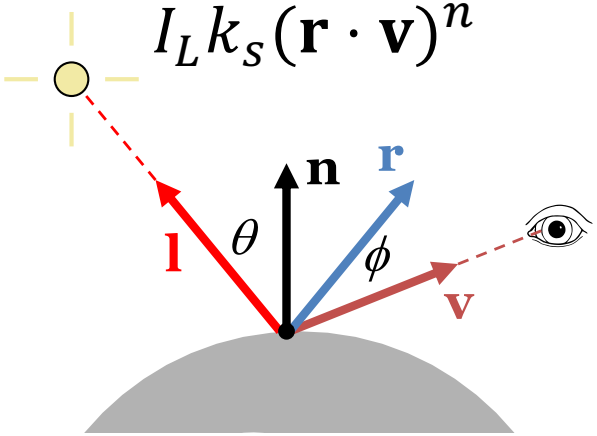


$I_L k_d (\mathbf{n} \cdot \mathbf{l})$

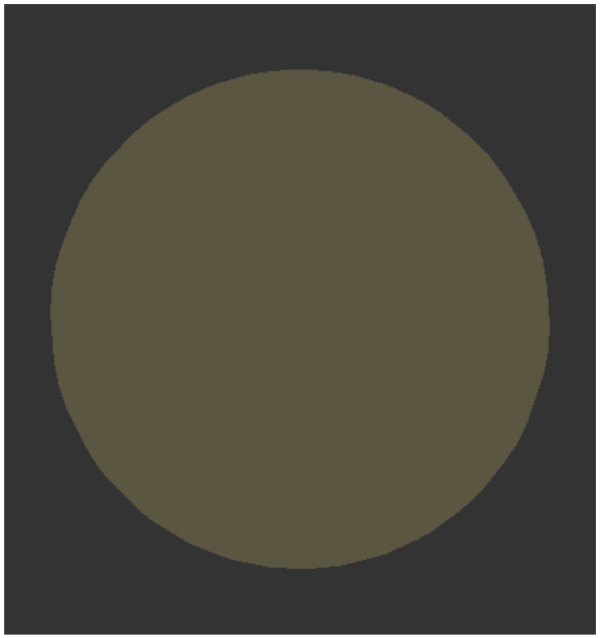
Specular



$I_L k_s (\mathbf{r} \cdot \mathbf{v})^n$

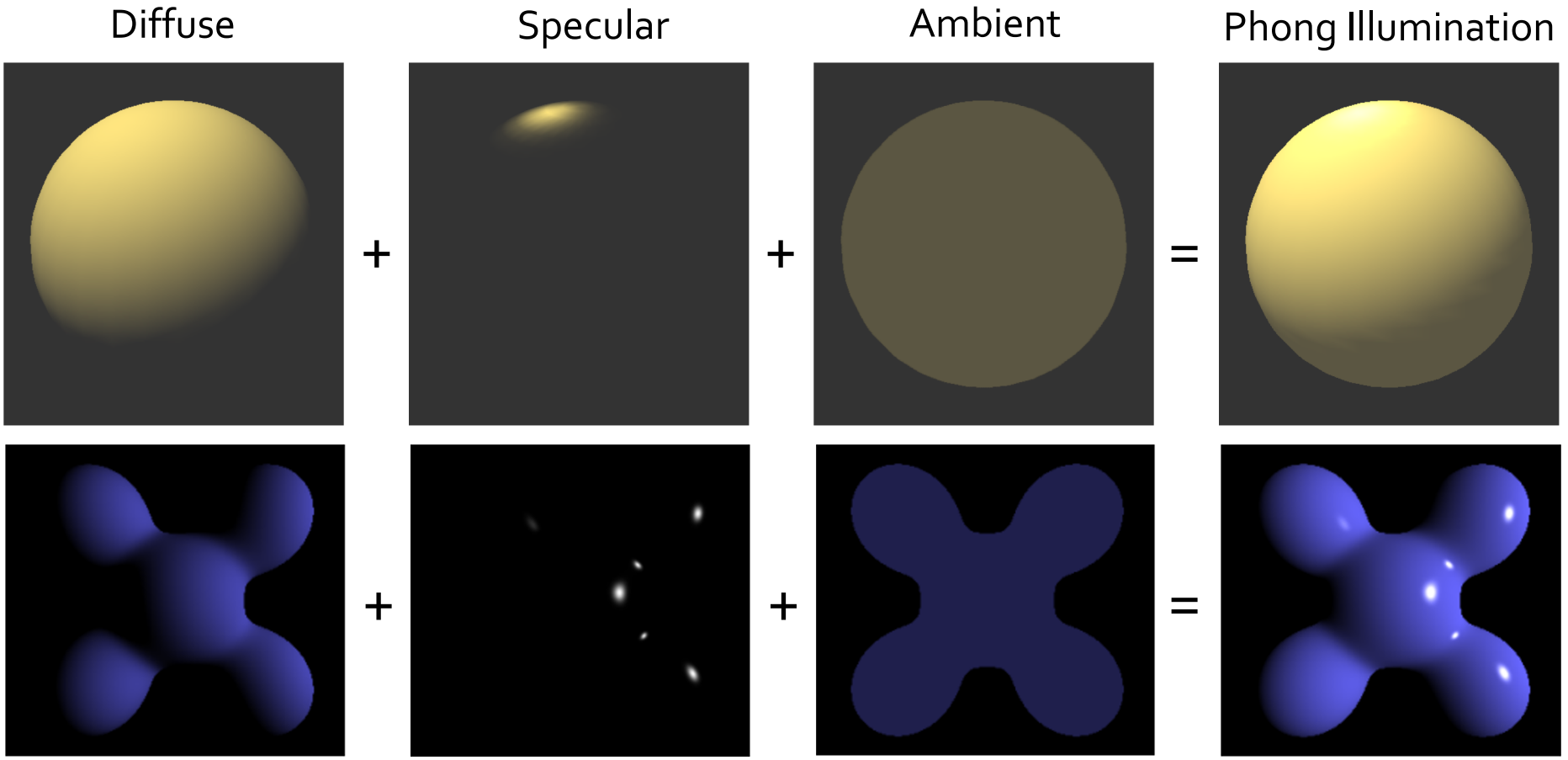



Ambient



$I_a k_a$

# Combined for the Final Result




The Phong illumination equation

$$I = I_L k_d(\mathbf{n} \cdot \mathbf{l}) + I_L k_s(\mathbf{r} \cdot \mathbf{v})^n + I_a k_a$$

# Attenuation

The light from a point source that reaches a surface is inversely proportional to the square of the distance between them

We can add a factor of the form

$$f(d) = \frac{1}{a+bd+cd^2}$$

where $a$, $b$, $c$ are some constants, to the diffuse and specular terms; and $d$ is the distance

# Multiple Light Sources

For single light source:

$$I = I_L k_d(\mathbf{n} \cdot \mathbf{l}) + I_L k_s(\mathbf{r} \cdot \mathbf{v})^n + I_a k_a$$

For multiple light sources:

$$I = I_a k_a + \sum_i I_{L_i}(k_d(\mathbf{n} \cdot \mathbf{l}_i) + k_s(\mathbf{r}_i \cdot \mathbf{v})^n)$$

With attenuation:

$$I = I_a k_a + \sum_i \mathrm{f}(d_{L_i})[I_{L_i}(k_d(\mathbf{n} \cdot \mathbf{l}_i) + k_s(\mathbf{r}_i \cdot \mathbf{v})^n)]$$

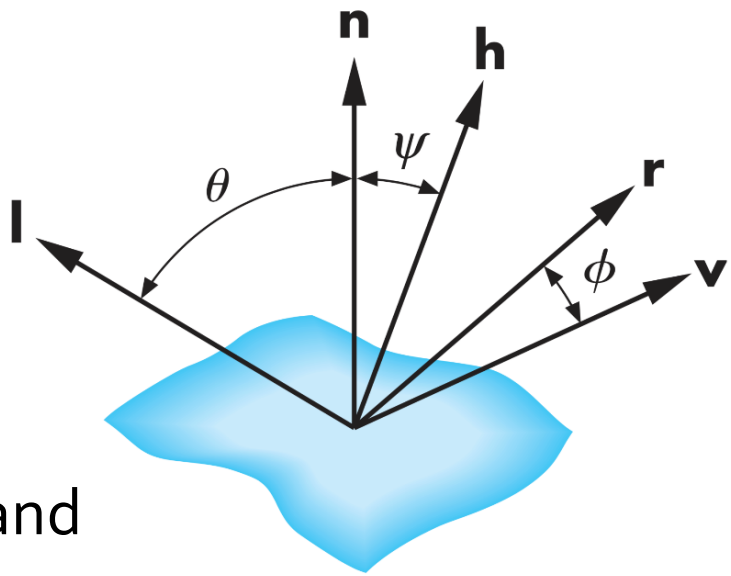
# Modified Phong Lighting Model

To avoid computing **r** for each vertex in the specular term, Blinn suggested an approximation using the halfway vector that is more efficient

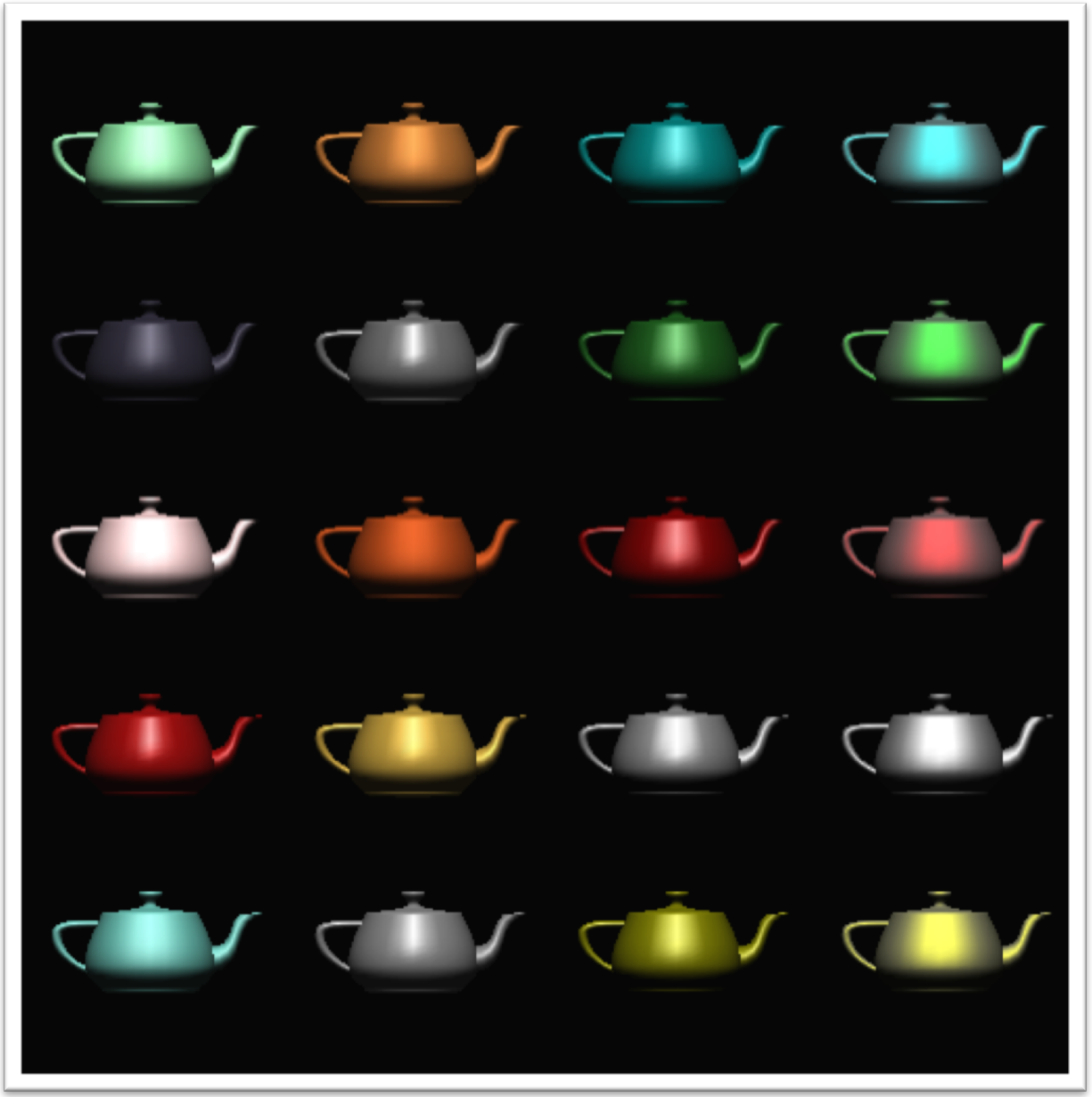
- **h** is normalized vector halfway between **l** and **v**

$$\mathbf{h} = (\mathbf{l} + \mathbf{v})/|\mathbf{l} + \mathbf{v}|$$

- Replace $(\mathbf{r} \cdot \mathbf{v})^{\alpha}$ by $(\mathbf{n} \cdot \mathbf{h})^{\beta}$
- If **v**, **l**, **n** are coplanar, the halfway angle $\psi = \phi/2$
- Also known as Blinn lighting model and is used in OpenGL



# OpenGL Example



Using different parameters in the Phong model

# Shading Polygons: Flat Shading

Illumination equations are evaluated at surface locations

- so where do we apply them?

## Flat Shading

- do it once per polygon
- fill every pixel covered by polygon with the resulting color

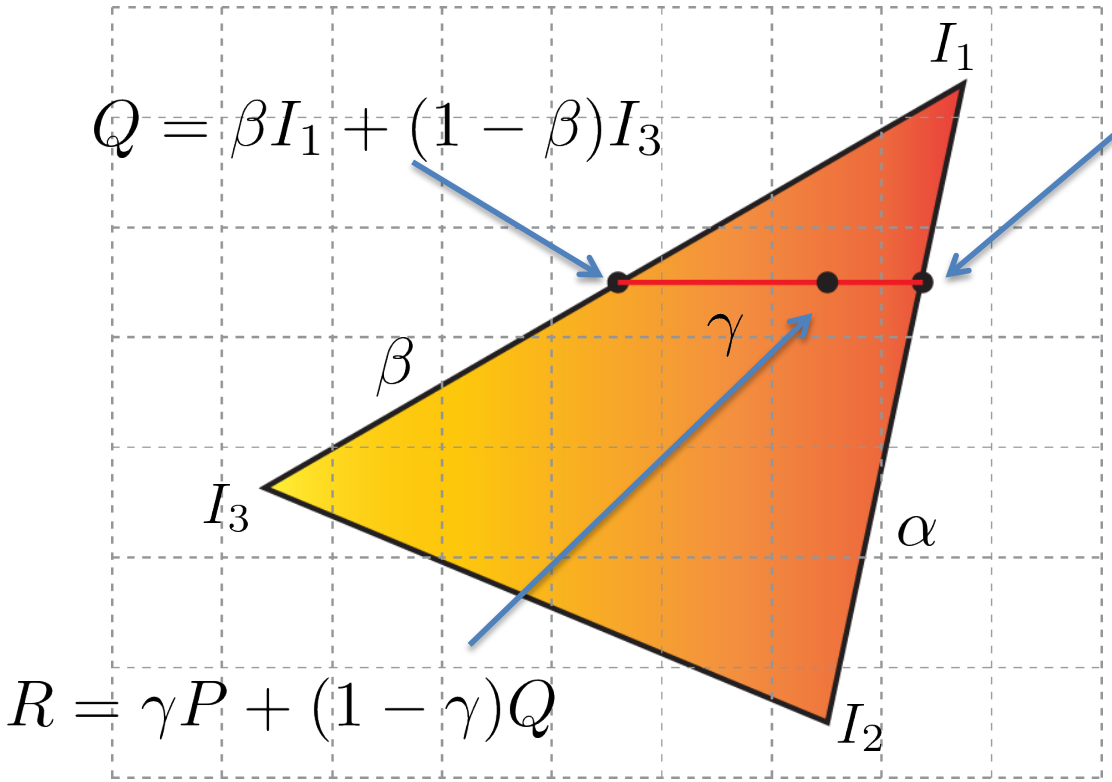
[www.dma.ufg.ac.at/app/link/Grundlagen%3A3D-Grafik/module/9728]

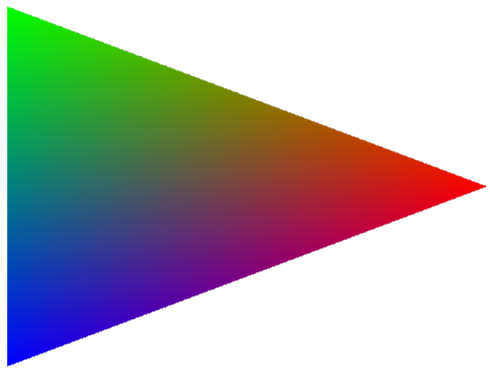
OpenGL — `glShadeModel(GL_FLAT)`

# Shading Polygons: Gouraud Shading

Alternatively, we do lighting calculation once for each vertex

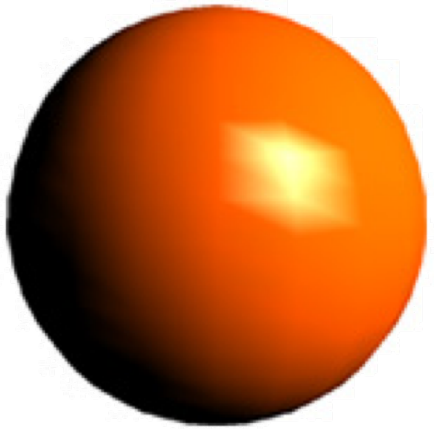
- compute color for each covered pixel
- linearly interpolate colors over polygon



Example of linear interpolation of three different colors at three vertices

# Shading Polygons: Gouraud Shading



If underlying geometry is too coarse, may lead to shading artifacts

Misses details that don't fall on vertex

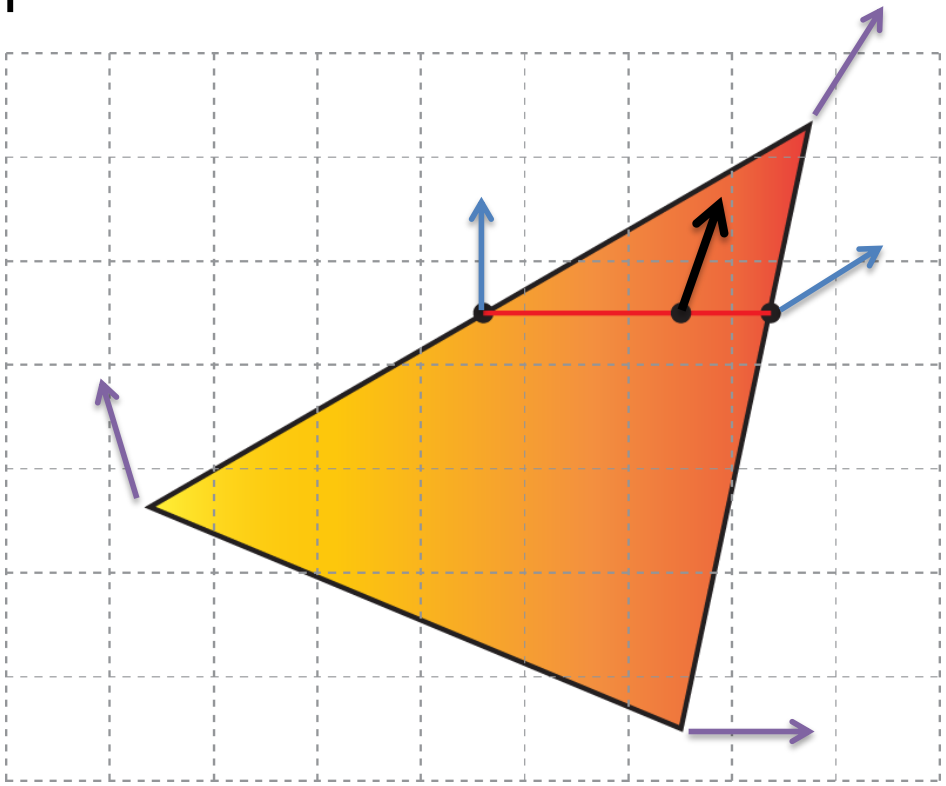
- specular highlights, for instance

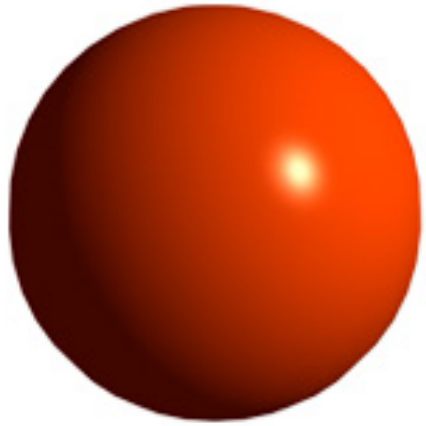
OpenGL — `glShadeModel(GL_SMOOTH)`

# Shading Polygons: Phong Shading

Lighting calculation is carried out for every pixel covered by a triangle.

Surface normal at a pixel is estimated using bilinear interpolation.





Best shading but computationally intensive

OpenGL — not directly supported

# Defining Materials in OpenGL

Just like everything else, there is a current material

- specifies the reflectances of the objects being drawn
- reflectances (e.g., $k_d$) are RGB triples

Set current values with `glMaterial(...)`

```
GLfloat tan1[] = {0.8, 0.7, 0.3, 1.0};
GLfloat tan2[] = {0.4, 0.35, 0.15, 1.0};

glMaterialfv(GL_FRONT_AND_BACK, GL_AMBIENT, tan1);
glMaterialfv(GL_FRONT_AND_BACK, GL_DIFFUSE, tan1);
glMaterialfv(GL_FRONT_AND_BACK, GL_SPECULAR, tan2);
glMaterialf(GL_FRONT_AND_BACK, GL_SHININESS, 50.0);
```

# Defining Lights in OpenGL

A fixed set of lights are available (at least 8)

- turn them on with `glEnable(GL_LIGHTx)`
- set their values with `glLight(...)`

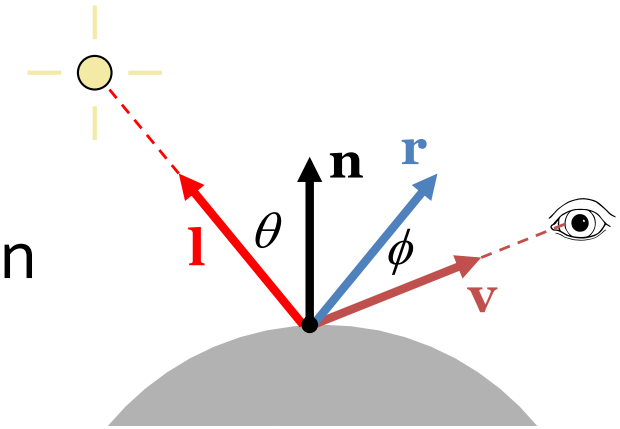
```
GLfloat white[] = {1.0, 1.0, 1.0, 1.0}
GLfloat p[] = {-2.0, -3.0, 10.0, 1.0}; // w=0 for directional light

glEnable(GL_LIGHTING);
glEnable(GL_LIGHT0);
glLightModeli(GL_LIGHT_MODEL_TWO_SIDE, GL_TRUE);

glLightfv(GL_LIGHT0, GL_POSITION, p);
glLightfv(GL_LIGHT0, GL_DIFFUSE, white);
glLightfv(GL_LIGHT0, GL_SPECULAR, white); // can be different

glEnable(GL_NORMALIZE); // guarantee unit normals
```

# Summarizing the Shading Model

We describe local appearance with illumination equations

- consists of a sum of set of components — light is additive
- treat each wavelength independently
- currently: diffuse, specular, and ambient terms

$$I = I_L k_d (\mathbf{n} \cdot \mathbf{l}) + I_L k_s (\mathbf{r} \cdot \mathbf{v})^n + I_a k_a$$




Must shade every pixel covered by polygon

- flat shading: constant color
- Gouraud shading: interpolate vertex colors
- Phong shading: interpolate vertex normals