

## Shading

Brian Curless  
CSE 457  
Spring 2014

1

## Reading

Required:

- Angel chapter 5.

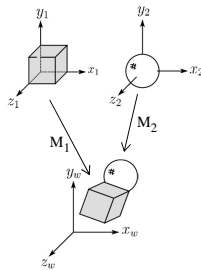
Optional:

- OpenGL red book, chapter 5.

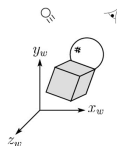
2

## Basic 3D graphics

With affine matrices, we can now transform virtual 3D objects in their local coordinate systems into a global (world) coordinate system:



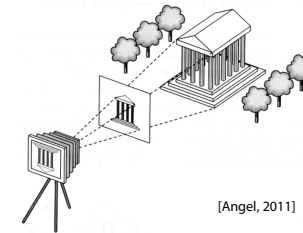
To synthesize an image of the scene, we also need to add light sources and a viewer/camera:



3

## Pinhole camera

To create an image of a virtual scene, we need to define a camera, and we need to model lighting and shading. For the camera, we use a **pinhole camera**.



[Angel, 2011]

The image is rendered onto an **image plane** (usually in front of the camera).

Viewing rays emanate from the **center of projection** (COP) at the center of the pinhole.

The image of an object point **P** is at the intersection of the viewing ray through **P** and the image plane.

But is P visible? This is the problem of **hidden surface removal** (a.k.a., **visible surface determination**). We'll consider this problem later.

4

## Shading

Next, we'll need a model to describe how light interacts with surfaces.

Such a model is called a **shading model**.

Other names:

- ♦ Lighting model
- ♦ Light reflection model
- ♦ Local illumination model
- ♦ Reflectance model
- ♦ BRDF

5

## An abundance of photons

Given the camera and shading model, properly determining the right color at each pixel is *extremely hard*.

Look around the room. Each light source has different characteristics. Trillions of photons are pouring out every second.

These photons can:

- ♦ interact with molecules and particles in the air ("participating media")
- ♦ strike a surface and
  - be absorbed
  - be reflected (scattered)
  - cause fluorescence or phosphorescence.
- ♦ interact in a wavelength-dependent manner
- ♦ generally bounce around and around

6

## Our problem

We're going to build up to a *approximations* of reality called the **Phong and Blinn-Phong illumination models**.

They have the following characteristics:

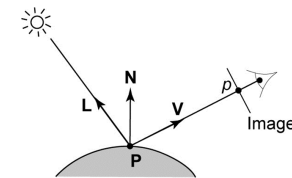
- ♦ *not* physically correct
- ♦ gives a "first-order" *approximation* to physical light reflection
- ♦ very fast
- ♦ widely used

In addition, we will assume **local illumination**, i.e., light goes: light source -> surface -> viewer.

No interreflections, no shadows.

7

## Setup...



Given:

- ♦ a point **P** on a surface visible through pixel *p*
- ♦ The normal **N** at **P**
- ♦ The lighting direction, **L**, and (color) intensity,  $I_L$ , at **P**
- ♦ The viewing direction, **V**, at **P**
- ♦ The shading coefficients at **P**

Compute the color, *I*, of pixel *p*.

Assume that the direction vectors are normalized:

$$\|\mathbf{N}\| = \|\mathbf{L}\| = \|\mathbf{V}\| = 1$$

8

## “Iteration zero”

The simplest thing you can do is...

Assign each polygon a single color:

$$I = k_e$$

where

- ♦  $I$  is the resulting intensity
- ♦  $k_e$  is the **emissivity** or intrinsic shade associated with the object

This has some special-purpose uses, but not really good for drawing a scene.

[Note:  $k_e$  is omitted in Angel.]

9

## “Iteration one”

Let’s make the color at least dependent on the overall quantity of light available in the scene:

$$I = k_e + k_a I_{La}$$

- ♦  $k_a$  is the **ambient reflection coefficient**.
  - really the reflectance of ambient light
  - “ambient” light is assumed to be equal in all directions
- ♦  $I_{La}$  is the **ambient light intensity**.

Physically, what is “ambient” light?

[Note: Angel uses  $L_a$  instead of  $I_{La}$ .]

10

## Wavelength dependence

Really,  $k_e$ ,  $k_a$ , and  $I_{La}$  are functions over all wavelengths  $\lambda$ .

Ideally, we would do the calculation on these functions. For the ambient shading equation, we would start with:

$$I(\lambda) = k_a(\lambda) I_{La}(\lambda)$$

then we would find good RGB values to represent the spectrum  $I(\lambda)$ .

Traditionally, though,  $k_a$  and  $I_{La}$  are represented as RGB triples, and the computation is performed on each color channel separately:

$$\begin{aligned} I^R &= k_a^R I_{La}^R \\ I^G &= k_a^G I_{La}^G \\ I^B &= k_a^B I_{La}^B \end{aligned}$$

11

## Diffuse reflection

Let’s examine the ambient shading model:

- ♦ objects have different colors
- ♦ we can control the overall light intensity
  - what happens when we turn off the lights?
  - what happens as the light intensity increases?
  - what happens if we change the color of the lights?

So far, objects are uniformly lit.

- ♦ not the way things really appear
- ♦ in reality, light sources are localized in position or direction

**Diffuse**, or **Lambertian** reflection will allow reflected intensity to vary with the direction of the light.

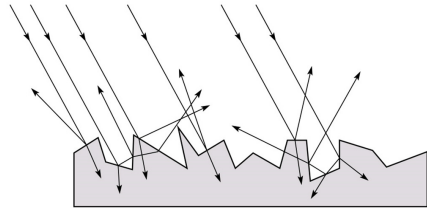
12

## Diffuse reflectors

Diffuse reflection occurs from dull, matte surfaces, like latex paint, or chalk.

These **diffuse** or **Lambertian** reflectors reradiate light equally in all directions.

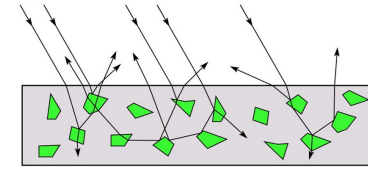
Picture a rough surface with lots of tiny **microfacets**.



13

## Diffuse reflectors

...or picture a surface with little pigment particles embedded beneath the surface (neglect reflection at the surface for the moment):



The microfacets and pigments distribute light rays in all directions.

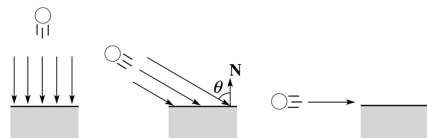
Embedded pigments are responsible for the coloration of diffusely reflected light in plastics and paints.

Note: the figures above are intuitive, but not strictly (physically) correct.

14

## Diffuse reflectors, cont.

The reflected intensity from a diffuse surface does not depend on the direction of the viewer. The incoming light, though, does depend on the direction of the light source:



15

## "Iteration two"

The incoming energy is proportional to \_\_\_\_\_, giving the diffuse reflection equations:

$$I = k_e + k_a I_{La} + k_d I_L B \quad \text{_____}$$

$$= k_e + k_a I_{La} + k_d I_L B( \quad )$$

where:

- ♦  $k_d$  is the **diffuse reflection coefficient**
- ♦  $I_L$  is the (color) intensity of the light source
- ♦  $\mathbf{N}$  is the normal to the surface (unit vector)
- ♦  $\mathbf{L}$  is the direction to the light source (unit vector)
- ♦  $B$  prevents contribution of light from below the surface:

$$B = \begin{cases} 1 & \text{if } \mathbf{N} \cdot \mathbf{L} > 0 \\ 0 & \text{if } \mathbf{N} \cdot \mathbf{L} \leq 0 \end{cases}$$

[Note: Angel uses  $L_d$  instead of  $I_L$  and  $f$  instead of  $B$ .]

16

## Specular reflection

**Specular reflection** accounts for the highlight that you see on some objects.

It is particularly important for *smooth, shiny* surfaces, such as:

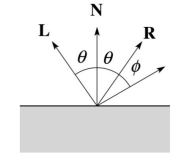
- ♦ metal
- ♦ polished stone
- ♦ plastics
- ♦ apples
- ♦ skin

Properties:

- ♦ Specular reflection depends on the viewing direction  $\mathbf{V}$ .
- ♦ For non-metals, the color is determined solely by the color of the light.
- ♦ For metals, the color may be altered (e.g., brass)

17

## Specular reflection “derivation”



For a perfect mirror reflector, light is reflected about  $\mathbf{N}$ , so

$$I = \begin{cases} I_L & \text{if } \mathbf{V} = \mathbf{R} \\ 0 & \text{otherwise} \end{cases}$$

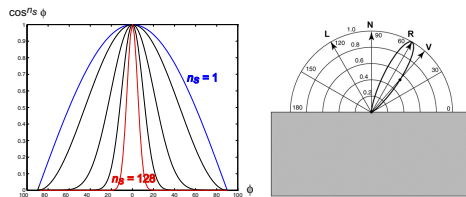
For a near-perfect reflector, you might expect the highlight to fall off quickly with increasing angle  $\phi$ .

Also known as:

- ♦ “**rough specular**” reflection
- ♦ “**directional diffuse**” reflection
- ♦ “**glossy**” reflection

18

## Phong specular reflection



One way to get this effect is to take  $(\mathbf{R} \cdot \mathbf{V})$ , raised to a power  $n_s$ .

As  $n_s$  gets larger,

- ♦ the dropoff becomes {more,less} gradual
- ♦ gives a {larger,smaller} highlight
- ♦ simulates a {more,less} mirror-like surface

Phong specular reflection is proportional to:

$$I_{\text{specular}} \sim B(\mathbf{R} \cdot \mathbf{V})_+^{n_s}$$

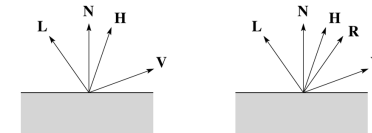
where  $(x)_+ \equiv \max(0, x)$ .

19

## Blinn-Phong specular reflection

A common alternative to specular reflection is the **Blinn-Phong model** (sometimes called the **modified Phong model**.)

We compute the vector halfway between  $\mathbf{L}$  and  $\mathbf{V}$  as:



Analogous to Phong specular reflection, we can compute the specular contribution in terms of  $(\mathbf{N} \cdot \mathbf{H})$ , raised to a power  $n_s$ :

$$I_{\text{specular}} = B(\mathbf{N} \cdot \mathbf{H})_+^{n_s}$$

where, again,  $(x)_+ \equiv \max(0, x)$ .

20

## “Iteration three”

The next update to the Blinn-Phong shading model is then:

$$I = k_e + k_a I_{La} + k_d I_L B(\mathbf{N} \cdot \mathbf{L}) + k_s I_L B(\mathbf{N} \cdot \mathbf{H})^{n_s}$$

$$= k_e + k_a I_{La} + I_L B \left[ k_d (\mathbf{N} \cdot \mathbf{L}) + k_s (\mathbf{N} \cdot \mathbf{H})^{n_s} \right]$$

where:

- $k_s$  is the **specular reflection coefficient**
- $n_s$  is the **specular exponent** or **shininess**
- $\mathbf{H}$  is the unit halfway vector between  $\mathbf{L}$  and  $\mathbf{V}$ , where  $\mathbf{V}$  is the viewing direction.

[Note: Angel uses  $\alpha$  instead of  $n_s$ , and maintains a separate  $L_d$  and  $L_s$ , instead of a single  $I_L$ . This choice reflects the flexibility available in OpenGL.]

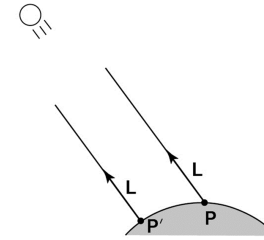
21

## Directional lights

The simplest form of lights supported by renderers are ambient, directional, and point. Spotlights are also supported often as a special form of point light.

We’ve seen ambient light sources, which are not really geometric.

**Directional light** sources have a single direction and intensity associated with them.

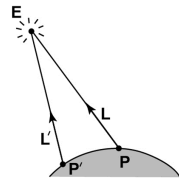


Using affine notation, what is the homogeneous coordinate for a directional light?

22

## Point lights

The direction of a **point light** sources is determined by the vector from the light position to the surface point.



Physics tells us the intensity must drop off inversely with the square of the distance:

$$f_{\text{atten}} = \frac{1}{r^2}$$

Sometimes, this distance-squared dropoff is considered too “harsh.” A common alternative is:

$$f_{\text{atten}} = \frac{1}{a + br + cr^2}$$

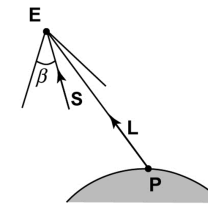
with user-supplied constants for  $a$ ,  $b$ , and  $c$ .

Using affine notation, what is the homogeneous coordinate for a point light?

23

## Spotlights

We can also apply a *directional attenuation* of a point light source, giving a **spotlight** effect.



A common choice for the spotlight intensity is:

$$f_{\text{spot}} = \begin{cases} \frac{(\mathbf{L} \cdot \mathbf{S})^e}{a + br + cr^2} & \text{if } \mathbf{L} \cdot \mathbf{S} \leq \cos \beta \\ 0 & \text{otherwise} \end{cases}$$

where

- $\mathbf{L}$  is the direction to the point light.
- $\mathbf{S}$  is the center direction of the spotlight.
- $\beta$  is the cutoff angle for the spotlight
- $e$  is the angular falloff coefficient

24

## “Iteration four”

Since light is additive, we can handle multiple lights by taking the sum over every light.

Our equation is now:

$$I = k_e + k_a I_{La} + \sum_j \frac{(\mathbf{L}_j \cdot \mathbf{S}_j)^{e_j} \beta_j}{a_j + b_j r_j + c_j r_j^2} I_{L,j} B_j \left[ k_d (\mathbf{N} \cdot \mathbf{L}_j) + k_s (\mathbf{N} \cdot \mathbf{H}_j)^{n_s} \right]$$

This is the Phong illumination model.

Which quantities are spatial vectors?

Which are RGB triples?

Which are scalars?

25

## Shading in OpenGL

The OpenGL lighting model allows you to associate different lighting colors according to material properties they will influence.

Thus, our original shading equation:

$$I = k_e + k_a I_{La} + \sum_j \frac{1}{a_j + b_j r_j + c_j r_j^2} I_{L,j} B_j \left[ k_d (\mathbf{N} \cdot \mathbf{L}_j)_+ + k_s (\mathbf{N} \cdot \mathbf{H}_j)_+^{n_s} \right]$$

becomes:

$$I = k_e + k_a I_{La} + \sum_j \frac{1}{a_j + b_j r_j + c_j r_j^2} \left[ k_a I_{La,j} + B_j \left\{ k_d I_{Ld,j} (\mathbf{N} \cdot \mathbf{L}_j)_+ + k_s I_{Ls,j} (\mathbf{N} \cdot \mathbf{H}_j)_+^{n_s} \right\} \right]$$

where you can have a global ambient light with intensity  $I_{La}$  in addition to having an ambient light intensity  $I_{La,j}$  associated with each individual light, as well as separate diffuse and specular intensities,  $I_{Ld,j}$  and  $I_{Ls,j}$ , respectively.

26

## Materials in OpenGL

The OpenGL code to specify the surface shading properties is fairly straightforward. For example:

```
GLfloat ke[] = { 0.1, 0.15, 0.05, 1.0 };
GLfloat ka[] = { 0.1, 0.15, 0.1, 1.0 };
GLfloat kd[] = { 0.3, 0.3, 0.2, 1.0 };
GLfloat ks[] = { 0.2, 0.2, 0.2, 1.0 };
GLfloat ns[] = { 50.0 };
glMaterialfv(GL_FRONT, GL_EMISSION, ke);
glMaterialfv(GL_FRONT, GL_AMBIENT, ka);
glMaterialfv(GL_FRONT, GL_DIFFUSE, kd);
glMaterialfv(GL_FRONT, GL_SPECULAR, ks);
glMaterialfv(GL_FRONT, GL_SHININESS, ns);
```

Notes:

- The `GL_FRONT` parameter tells OpenGL that we are specifying the materials for the front of the surface.
- Only the alpha value of the diffuse color is used for blending. It's usually set to 1.

27

## Shading in OpenGL, cont'd

In OpenGL this equation, for one light source (the 0<sup>th</sup>) is specified something like:

```
GLfloat La[] = { 0.2, 0.2, 0.2, 1.0 };
GLfloat La0[] = { 0.1, 0.1, 0.1, 1.0 };
GLfloat Ld0[] = { 1.0, 1.0, 1.0, 1.0 };
GLfloat Ls0[] = { 1.0, 1.0, 1.0, 1.0 };
GLfloat pos0[] = { 1.0, 1.0, 1.0, 0.0 };
GLfloat a0[] = { 1.0 };
GLfloat b0[] = { 0.5 };
GLfloat c0[] = { 0.25 };
GLfloat S0[] = { -1.0, -1.0, 0.0 };
GLfloat beta0[] = { 45 };
GLfloat e0[] = { 2 };

glLightModelfv(GL_LIGHT_MODEL_AMBIENT, La);
glLightfv(GL_LIGHT0, GL_AMBIENT, La0);
glLightfv(GL_LIGHT0, GL_DIFFUSE, Ld0);
glLightfv(GL_LIGHT0, GL_SPECULAR, Ls0);
glLightfv(GL_LIGHT0, GL_POSITION, pos0);
glLightfv(GL_LIGHT0, GL_CONSTANT_ATTENUATION, a0);
glLightfv(GL_LIGHT0, GL_LINEAR_ATTENUATION, b0);
glLightfv(GL_LIGHT0, GL_QUADRATIC_ATTENUATION, c0);
glLightfv(GL_LIGHT0, GL_SPOT_DIRECTION, S0);
glLightf(GL_LIGHT0, GL_SPOT_CUTOFF, beta0);
glLightf(GL_LIGHT0, GL_SPOT_EXPONENT, e0);
```

28

## Shading in OpenGL, cont'd

Notes:

You can have as many as `GL_MAX_LIGHTS` lights in a scene. This number is system-dependent.

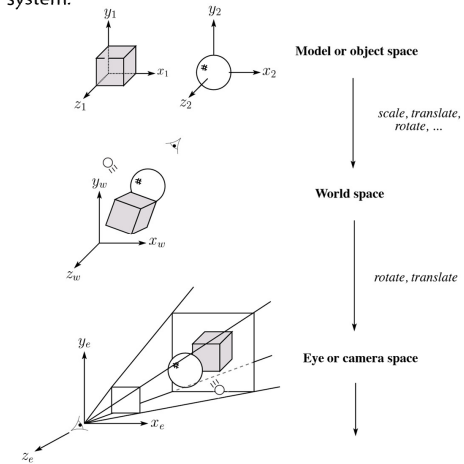
For directional lights, you specify a light direction, not position, and the attenuation and spotlight terms are ignored.

The directions of directional lights and spotlights are specified in the coordinate systems *of the lights*, not the surface points as we've been doing in lecture.

29

## 3D Geometry in the Graphics Hardware Pipeline

Graphics hardware applies transformations to bring the objects and lighting into the camera's coordinate system:

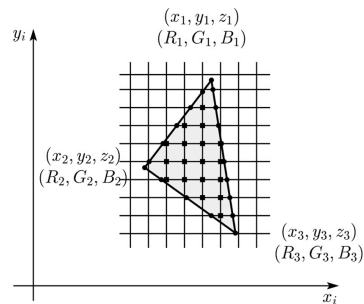


The geometry is assumed to be made of triangles, and the **vertices** are projected onto the image plane.

30

## Rasterization

After projecting the vertices, graphics hardware "smears" vertex properties across the interior of the triangle in a process called **rasterization**.



Smearing the z-values and using a Z-buffer will enable the graphics hardware to determine if a point inside a triangle is visible. (More on this in another lecture.)

If we have stored colors at the vertices, then we can smear these as well.

31

## Shading the interiors of triangles

We will be computing colors using the Blinn-Phong lighting model.

Let's assume (as graphics hardware does) that we are working with triangles.

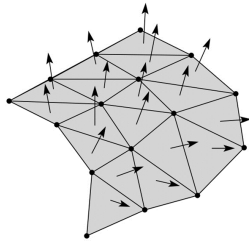
How should we shade the interiors of triangles?

32



## Shading with per-face normals

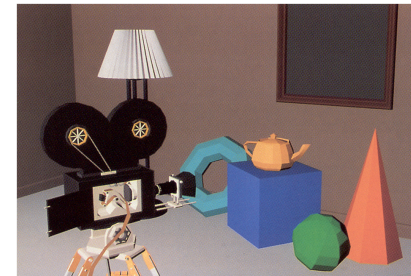
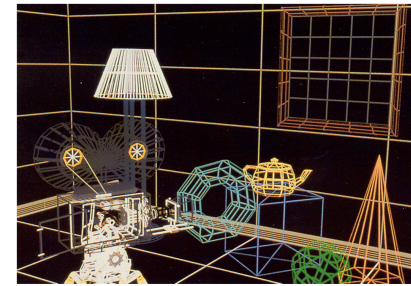
Assume each face has a constant normal:



For a distant viewer and a distant light source and constant material properties over the surface, how will the color of each triangle vary?

33

## Faceted shading (cont'd)



[Williams and Siegel 1990]

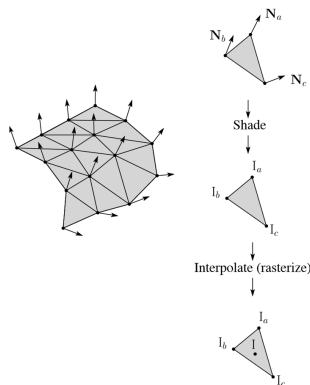
34

## Gouraud interpolation

To get a smoother result that is easily performed in hardware, we can do **Gouraud interpolation**.

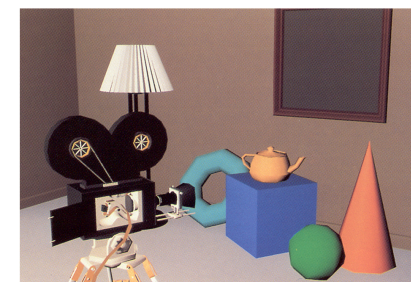
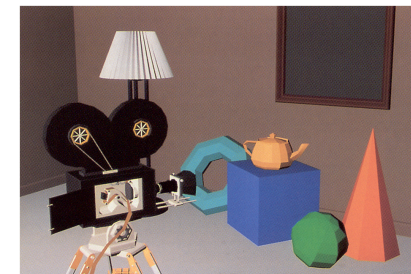
Here's how it works:

1. Compute normals at the vertices. (More on this in a moment...)
2. Shade only the vertices.
3. Interpolate the resulting vertex colors.



35

## Faceted shading vs. Gouraud interpolation



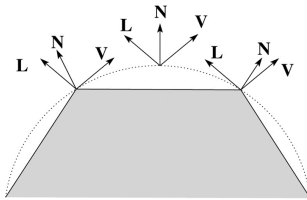
[Williams and Siegel 1990]

36

## Gouraud interpolation artifacts

Gouraud interpolation has significant limitations.

1. If the polygonal approximation is too coarse, we can miss specular highlights.



2. We will encounter **Mach banding** (derivative discontinuity enhanced by human eye).

This is what graphics hardware does by default.

A substantial improvement is to do...

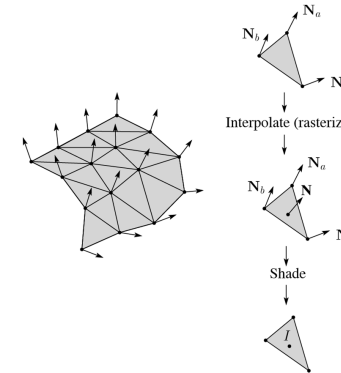
37

## Phong interpolation

To get an even smoother result with fewer artifacts, we can perform **Phong interpolation**.

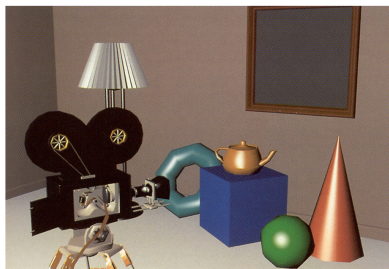
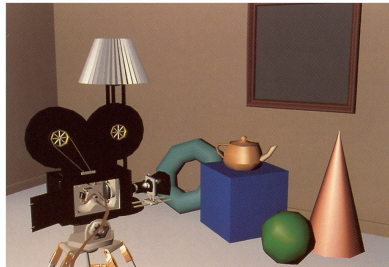
Here's how it works:

1. Compute normals at the vertices.
2. Interpolate normals and normalize.
3. Shade using the interpolated normals.



38

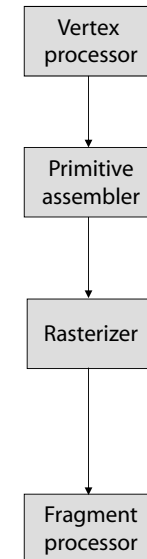
## Gouraud vs. Phong interpolation



[Williams and Siegel 1990]

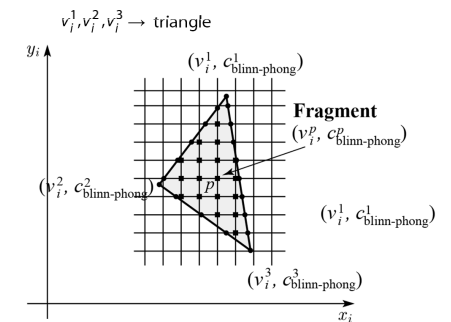
39

## Default pipeline: Gouraud interpolation



### Default vertex processing:

$L \leftarrow$  determine lighting direction  
 $V \leftarrow$  determine viewing direction  
 $N \leftarrow$  normalize( $n_g$ )  
 $c_{\text{blinn-phong}} \leftarrow$  shade with  $L, V, N, k_d, k_s, n_s$   
 attach  $c_{\text{blinn-phong}}$  to vertex as "varying"  
 $v_i \leftarrow$  project  $v$  to image

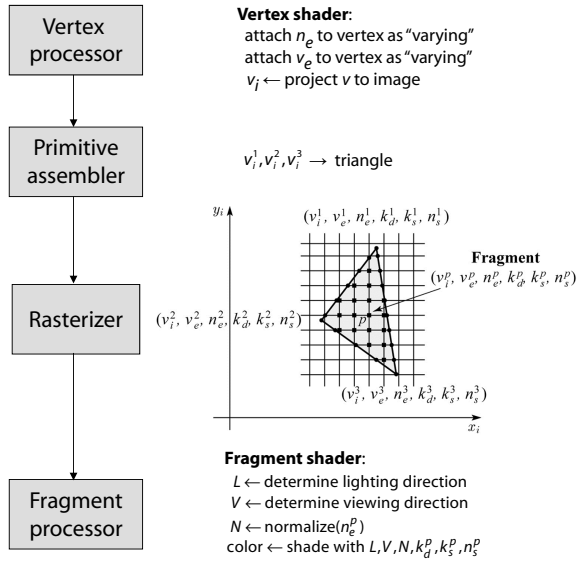


### Default fragment processing:

color  $\leftarrow c_{\text{blinn-phong}}^p$

40

## Programmable pipeline: Phong-interpolated normals!



41

## Choosing Blinn-Phong shading parameters

Experiment with different parameter settings. To get you started, here are a few suggestions:

- Try  $n_s$  in the range [0,100]
- Try  $k_d + k_d + k_s < 1$
- Use a small  $k_d$  (~0.1)

	$n_s$	$k_d$	$k_s$
Metal	large	Small, color of metal	Large, color of metal
Plastic	medium	Medium, color of plastic	Medium, white
Planet	0	varying	0

42

## BRDF

The diffuse+specular parts of the Blinn-Phong illumination model are a mapping from light to viewing directions:

$$I = I_L B \left[ k_d (\mathbf{N} \cdot \mathbf{L}) + k_s \mathbf{N} \cdot \left( \frac{\mathbf{L} + \mathbf{V}}{\|\mathbf{L} + \mathbf{V}\|} \right)^{n_s} \right]$$

$$= I_L f_r(\mathbf{L}, \mathbf{V})$$

The mapping function  $f_r$  is often written in terms of incoming (light) directions  $\omega_{in}$  and outgoing (viewing) directions  $\omega_{out}$ :

$$f_r(\omega_{in}, \omega_{out}) \text{ or } f_r(\omega_{in} \rightarrow \omega_{out})$$

This function is called the **Bi-directional Reflectance Distribution Function (BRDF)**.

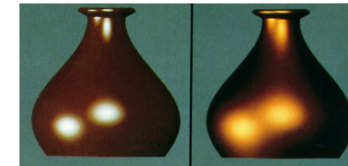
Here's a plot with  $\omega_{in}$  held constant:



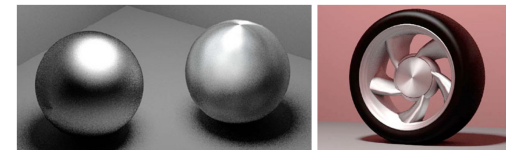
BRDF's can be quite sophisticated...

43

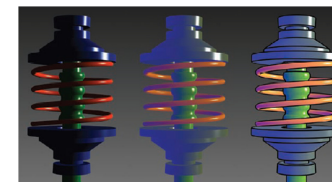
## More sophisticated BRDF's



[Cook and Torrance, 1982]



Anisotropic BRDFs [Westin, Arvo, Torrance 1992]



Artistics BRDFs [Gooch]

44

## Summary

You should understand the equation for the Blinn-Phong lighting model described in the "Iteration Four" slide:

- ♦ What is the physical meaning of each variable?
- ♦ How are the terms computed?
- ♦ What effect does each term contribute to the image?
- ♦ What does varying the parameters do?

You should also understand the differences between faceted, Gouraud, and Phong *interpolated* shading.

And you should understand how to compute the normal to a surface of revolution.