Shading

Reading

Required:

- Angel 6.1-6.3, 6.5, 6.7-6.8

Optional:

- Angel 6.4
- OpenGL red book, chapter 5.

Introduction

So far, we've talked exclusively about geometry.

- What is the shape of an object?
- How do I place it in a virtual 3D space?
- How do I know which pixels it covers?
- How do I know which of the pixels I should actually draw?

Once we've answered all those, we have to ask one more important question:

- To what value do I set each pixel?

Answering this question is the job of the **shading model**.

Other names:

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

An abundance of photons

Properly determining the right color is *really hard*.

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

These photons can:

- interact with the atmosphere, or with things in the atmosphere
- strike a surface and
  - be absorbed
  - be reflected (scattered)
  - cause fluorescence or phosphorescence.
- interact in a wavelength-dependent manner
- generally bounce around and around

Our problem

We're going to build up to an *approximation* of reality called the **Phong illumination model**.

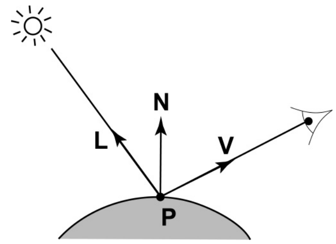
It has the following characteristics:

- *not* physically based
- 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.

Setup…



Given:

- a point **P** on a surface visible through pixel *p*
- The normal **N** at **P**
- The lighting direction, **L**, and intensity, *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:

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

"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.]

"Iteration one"

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

$$I = k_e + k_a L_a$$

- k_a is the **ambient reflection coefficient.**
  - really the reflectance of ambient light
  - "ambient" light is assumed to be equal in all directions
- L_a is the **ambient light intensity**.

Physically, what is "ambient" light?

Wavelength dependence

Really, k_e, k_a, and l_a are functions over all wavelengths λ.

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

$$I(\lambda) = k_a(\lambda) L_a(\lambda)$$

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

Traditionally, though, k_a and l_a are represented as RGB triples, and the computation is performed on each color channel separately:

$$I_R = k_{a,R}\ L_{a,R}$$
$$I_G = k_{a,G}\ L_{a,G}$$
$$I_B = k_{a,B}\ L_{a,B}$$

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

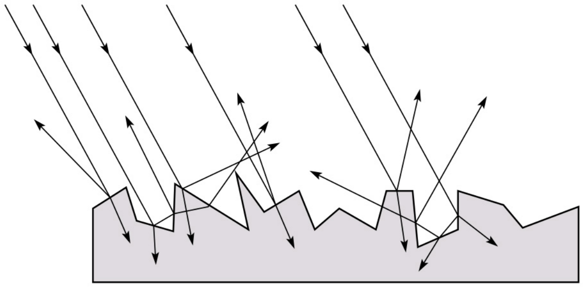
Diffuse, or **Lambertian** reflection will allow reflected intensity to vary with the direction of the light.

Diffuse reflectors

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

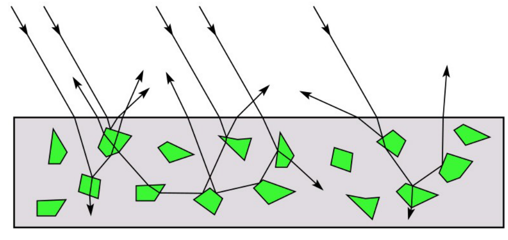
These **diffuse** or **Lambertian** reflectors reradiate light equally in all directions.

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



Diffuse reflectors

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



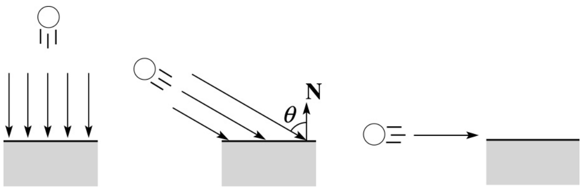
The microfacets and pigments distribute light rays in all directions.

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.

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:



"Iteration two"

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

$$I = k_e + k_a L_a + k_d L \underline{\qquad}$$

$$= k_e + k_a I_a + k_d L(\qquad)$$

where:

- k_d is the **diffuse reflection coefficient**
- L is the intensity of the light source
- **N** is the normal to the surface (unit vector)
- **L** is the direction to the light source (unit vector)
- $(x)_+$ means max $\{0,x\}$

[Note: Angel uses L_d instead of L.]

Specular reflection

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

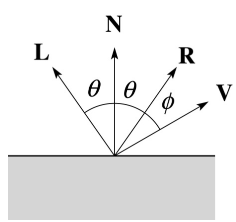
It is particularly important for *smooth, shiny* surfaces, such as:

- metal
- polished stone
- plastics
- apples
- skin

Properties:

- Specular reflection depends on the viewing direction ***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)

Specular reflection "derivation"



For a perfect mirror reflector, light is reflected about ***N***, so

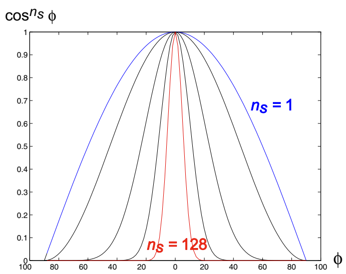
$$I = \begin{cases} L & \text{if } \mathbf{V} = \mathbf{R} \\ 0 & \text{otherwise} \end{cases}$$

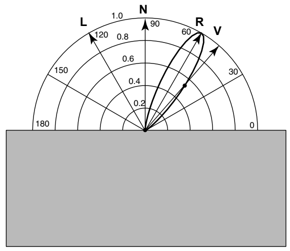
For a near-perfect reflector, you might expect the highlight to fall off quickly with increasing angle ϕ.

Also known as:

- **"rough specular" reflection**
- **"directional diffuse" reflection**
- **"glossy" reflection**

Derivation, cont.





One way to get this effect is to take (***R·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

"Iteration three"

The next update to the Phong shading model is then:

$$I = k_e + k_a I_a + k_d L(\mathbf{N}\cdot\mathbf{L})_+ + k_s L(\mathbf{V}\cdot\mathbf{R})_+^{n_s}$$

where:

- k_s is the **specular reflection coefficient**
- n_s is the **specular exponent** or **shininess**
- ***R*** is the reflection of the light about the normal (unit vector)
- ***V*** is viewing direction (unit vector)

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

Intensity drop-off with distance

OpenGL supports different kinds of lights: point, directional, and spot.

For point light sources, the laws of physics state that the intensity of a point light source must drop off inversely with the square of the distance.

We can incorporate this effect by multiplying I_l by $1/d^2$.

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

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

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

"Iteration four"

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

Our equation is now:

$$I = k_e + k_a L_a + \sum_j f_{atten}(d_j) L_j \left[k_d(\mathbf{N}\cdot\mathbf{L}_j)_+ + k_s(\mathbf{V}\cdot\mathbf{R}_j)_+^{n_s} \right]$$

This is the Phong illumination model.

Which quantities are spatial vectors?

Which are RGB triples?

Which are scalars?

Choosing the parameters

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

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

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 specifiying 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.

Shading in OpenGL

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

Thus, our original shading equation becomes:

$$I = k_e + k_a L_a + \sum_j f_{atten}(d_j)\left[k_a L_{a\,j} + k_d L_{d\,j} (\mathbf{N}\cdot\mathbf{L}_j)_+ + k_s L_{s\,j} (\mathbf{V}\cdot\mathbf{R}_j)_+^{n_s} \right]$$

where you can have a global ambient light with intensity L_a in addition to have an ambient light intensity $L_{a\,j}$ associated with each individual light.

Shading in OpenGL

Repeating from the previous slide…

$$I = k_e + k_a L_a + \sum_j f_{atten}(d_j)\left[k_a L_{a\,j} + k_d L_{d\,j} (\mathbf{N}\cdot\mathbf{L}_j)_+ + k_s L_{s\,j} (\mathbf{V}\cdot\mathbf{R}_j)_+^{n_s} \right]$$

In OpenGL this equation 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 a[] = { 1.0 };
GLfloat b[] = { 0.5 };
GLfloat c[] = { 0.25 };

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, a);
glLightfv(GL_LIGHT0, GL_LINEAR_ATTENUATION, b);
glLightfv(GL_LIGHT0, GL_QUADRATIC_ATTENUATION, c);
```

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

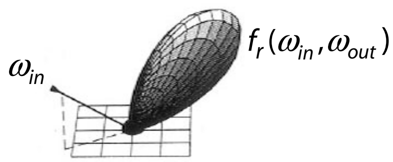
BRDF

The Phong illumination model is really a function that maps light from incoming (light) directions ω_{in} to outgoing (viewing) directions ω_{out}:

$$f_r(\omega_{in}, \omega_{out})$$

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

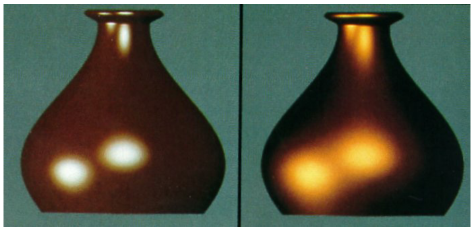
Here's a plot with ω_{in} held constant:

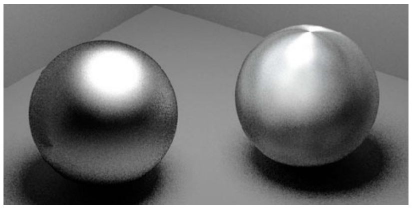


BRDF's can be quite sophisticated…

More sophisticated BRDF's

Cook and Torrance, 1982





Westin, Arvo, Torrance 1992

Gouraud vs. Phong interpolation

Now we know how to compute the color at a point on a surface using the Phong lighting model.

Does graphics hardware do this calculation at every point? Typically not (although this is changing)…

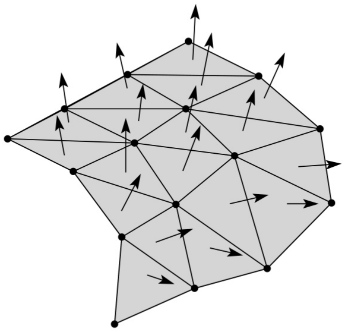
Smooth surfaces are often approximated by polygonal facets, because:

- Graphics hardware generally wants polygons (esp. triangles).
- Sometimes it easier to write ray-surface intersection algorithms for polygonal models.

How do we compute the shading for such a surface?

Faceted shading

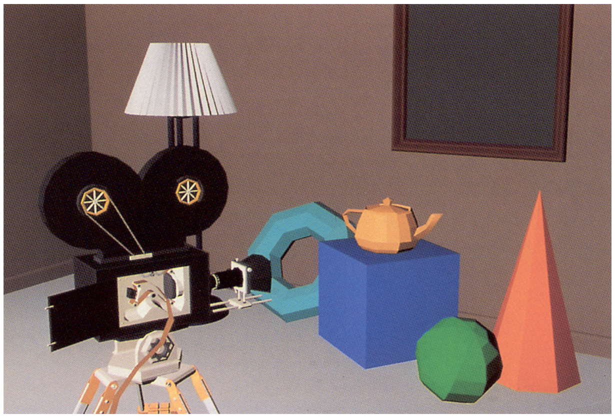
Assume each face has a constant normal:



For a distant viewer and a distant light source, how will the color of each triangle vary?

Result: faceted, not smooth, appearance.

Faceted shading (cont'd)

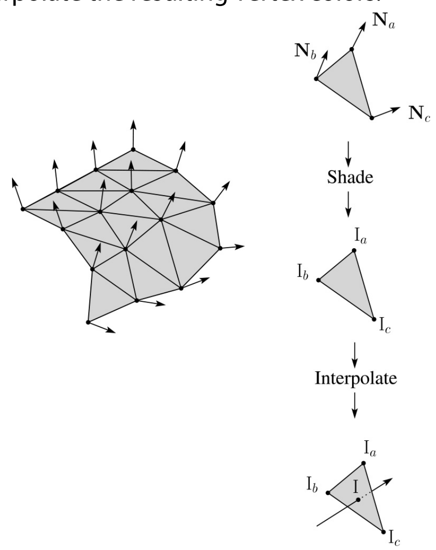


Gouraud interpolation

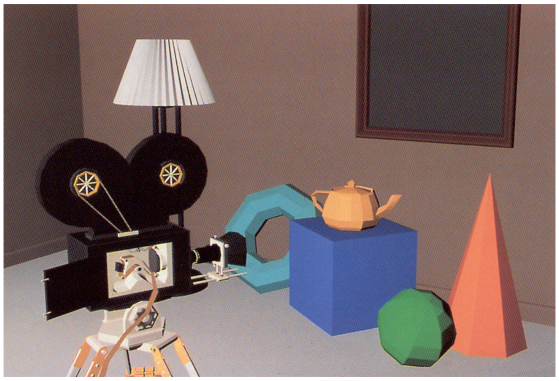
To get a smoother result that is easily performed in hardware, we can do **Gouraud interpolation**.

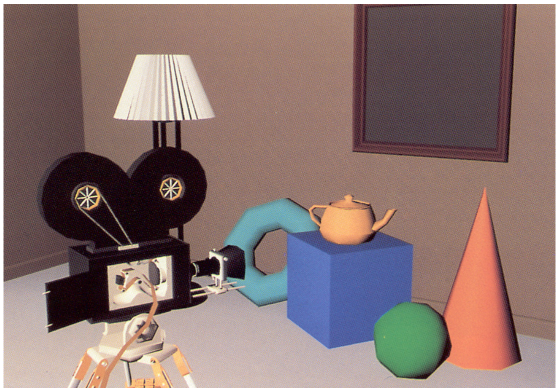
Here's how it works:

1. Compute normals at the vertices.
2. Shade only the vertices.
3. Interpolate the resulting vertex colors.



Faced shading vs. Gouraud interpolation

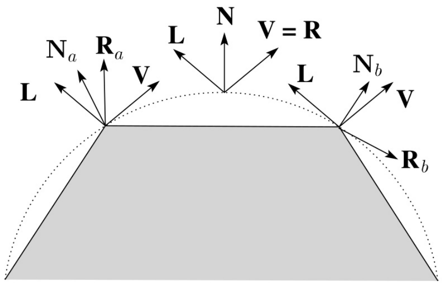




Gouraud interpolation artifacts

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).

Alas, this is usually what graphics hardware supports.

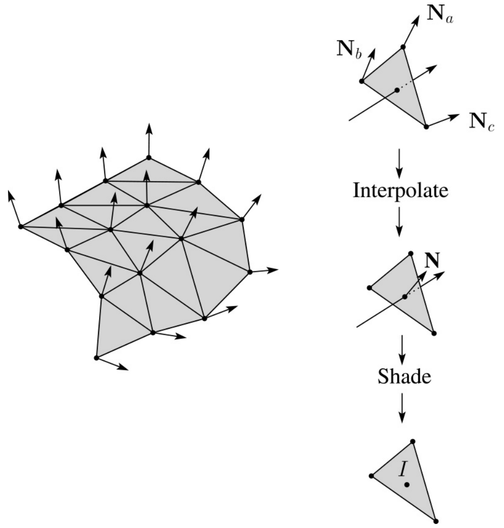
Maybe someday soon all graphics hardware will do…

Phong interpolation

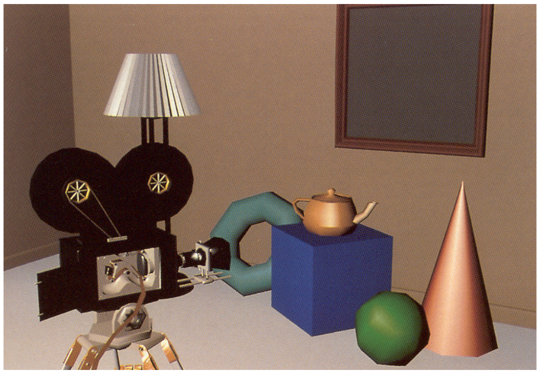
To get an even smoother result with fewer artifacts, we can perform **Phong *interpolation***.

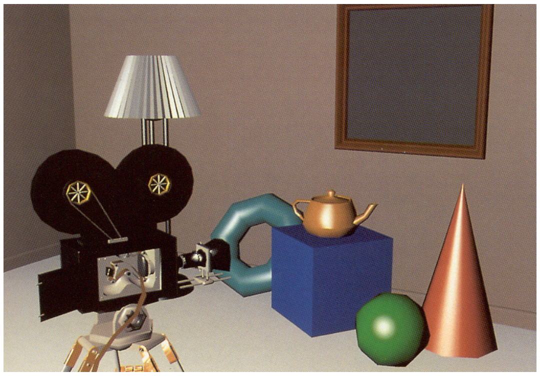
Here's how it works:

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



Gouraud vs. Phong interpolation





Summary

The most important thing to take away from this lecture is the equation for the Phong 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.