Shading

Reading

Required:
- Angel 6.1-6.5, 6.7-6.8

Optional:

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:

$$||N|| = ||L|| = ||V|| = 1$$

---

**“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, $k_x$, $k_y$, and $L_a$ are functions over all wavelengths $\lambda$.

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

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

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

Traditionally, though, $k_a$ and $I_a$ 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.

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:

\[ I = k_d + k_s L_n + k_a L \theta \]

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)
- \( \theta \) means max \{0,\}

(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

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

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
Derivation, cont.

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

“Iteration three”

The next update to the Phong shading model is then:

$$ I = I_0 + k_a L_a + k_s L_s (N \cdot \mathbf{L}) + k_v L_v (\mathbf{V} \cdot \mathbf{R})^{n_s} $$

where:
- $k_s$ is the **specular reflection coefficient**
- $n_s$ is the **specular exponent or shininess**
- $\mathbf{R}$ is the reflection of the light about the normal (unit vector)
- $\mathbf{V}$ is viewing direction (unit vector)

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

Lights

OpenGL supports three different kinds of lights: ambient, directional, and point. Spot lights are also supported 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.

Point lights

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

$$ \mathbf{L} = \frac{\mathbf{E} - \mathbf{P}}{||\mathbf{E} - \mathbf{P}||} $$

$$ d = ||\mathbf{E} - \mathbf{P}|| $$

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

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

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

$$ f_{\text{atten}} = \frac{1}{a + bd + cd^2} $$

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

OpenGL also allows one to apply a directional attenuation of a point light source, giving a spotlight effect.

The spotlight intensity factor is computed in OpenGL as:

\[ f_{\text{spot}} = (\mathbf{L} \cdot \mathbf{S})^\beta \]

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
- \( \epsilon \) is the angular falloff coefficient
- \( (x)^\beta = \max(\frac{\pi}{2} - \beta, 0) \)

"Iteration four"

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

Our equation is now:

\[ I = k_a + k_d \mathbf{L} \cdot \mathbf{a} + \sum_j \left( \alpha_j + b_j d_j + c_j d_j^2 \right) \mathbf{L} \cdot \mathbf{R}_j \]

This is the Phong illumination model.

Which quantities are spatial vectors?

Which are RGB triples?

Which are scalars?

Materials in OpenGL

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

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

\[
I = k_a + k_d L_a + \frac{\sum_j (\mathbf{L}_j \cdot \mathbf{S}_j)^e_j}{\sum_j a_j + b_j d_j + c_j d_j^2} \left[ k_d (\mathbf{N} \cdot \mathbf{L}_j)_z + k_s (\mathbf{V} \cdot \mathbf{R}_j)_z \right]
\]

becomes:

\[
I = k_a + k_d L_a + \frac{\sum_j (\mathbf{L}_j \cdot \mathbf{S}_j)^e_j}{\sum_j a_j + b_j d_j + c_j d_j^2} \left[ k_d L_a + k_d L_d + k_s L_s (\mathbf{V} \cdot \mathbf{R}_j)_z \right]
\]

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, cont’d

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

```gl
GLfloat La[] = { 0.2, 0.2, 0.2, 1.0 };  // ambient light intensity
GLfloat La0[] = { 0.1, 0.1, 0.1, 1.0 };  // individual light intensity
GLfloat Ld0[] = { 1.0, 1.0, 1.0, 1.0 };  // diffuse light intensity
GLfloat Ls0[] = { 1.0, 1.0, 1.0, 1.0 };  // specular light intensity
GLfloat pos0[] = { 1.0, 1.0, 1.0, 0.0 };  // light position
GLfloat a0[] = { 1.0 };            // constant attenuation
GLfloat b0[] = { 0.5 };            // linear attenuation
GLfloat c0[] = { 0.25 };           // quadratic attenuation
GLfloat S0[] = { -1.0, -1.0, 0.0 }; // spotlight direction
GLfloat beta0[] = { 45 };          // spotlight cut-off
GLfloat e0[] = { 2 };              // spotlight exponent

glLightModelfv(GL_LIGHT_MODEL_AMBIENT, La);  // set ambient light
glLightfv(GL_LIGHT0, GL_AMBIENT, La0);       // set individual light intensity
glLightfv(GL_LIGHT0, GL_DIFFUSE, Ld0);       // set diffuse light intensity
glLightfv(GL_LIGHT0, GL_SPECULAR, Ls0);      // set specular light intensity
glLightfv(GL_LIGHT0, GL_POSITION, pos0);     // set light position
glLightfv(GL_LIGHT0, GL_CONSTANT_ATTENUATION, a0); // set constant attenuation
glLightfv(GL_LIGHT0, GL_LINEAR_ATTENUATION, b0); // set linear attenuation
glLightfv(GL_LIGHT0, GL_QUADRATIC_ATTENUATION, c0); // set quadratic attenuation
glLightfv(GL_LIGHT0, GL_SPOT_DIRECTION, S0); // set spotlight direction
glLightf(GL_LIGHT0, GL_SPOT_CUTOFF, beta0);  // set spotlight cut-off
glLightf(GL_LIGHT0, GL_SPOT_EXPONENT, e0);   // set spotlight exponent
```

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.

BRDF

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

\[
f_r(\omega_{\text{in}}, \omega_{\text{out}})
\]

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

Here’s a plot with \( \omega_{\text{in}} \) held constant:

BRDF’s can be quite sophisticated…
More sophisticated BRDF’s

Cook and Torrance, 1982

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 and constant material properties over the surface, 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.

**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 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.