Lecture 5:
Shading

Topics:

1. Light and matter
2. Light sources
3. The Phong reflection model
4. Polygonal shading
5. Light sources in OpenGL
6. Specification of materials in OpenGL

Chapter 6, Sections 6.1, 6.2, 6.3, 6.5, 6.7, 6.8.
Light and matter

The most general approach to rendering is based on physics, and uses conservation of energy to derive equations to describe multiple reflections. For example, in the figure, some light from the source is reflected from surface A. Some of this light reaches B and some of this is reflected back to A and so on. This recursive reflection of light leads to the so-called rendering equation. Unfortunately, this equation cannot be solved, even numerically. One can use instead an approximative approach: two of which are radiosity and ray tracing, each of which give an excellent approximation for particular types of surface. However, even these methods are often too slow in real-time applications. The solution is to use an even simpler rendering model, the Phong reflection model.
The approach is to follow rays of light from light-emitting surfaces, called light sources. We model what happens to these rays as they interact with reflecting surfaces in the scene. We consider only single interactions between light sources and surfaces (unlike in ray tracing). We consider three kinds of interaction between light and materials:

(a) **Specular surfaces** appear shiny because most light that is reflected is scattered in an angle close to the angle of reflection. Mirrors are perfectly specular surfaces.

(b) **Diffuse surfaces** scatter light in all directions and appear dull, e.g., walls painted in matte. Perfectly diffuse surfaces scatter light in all directions equally.

(c) **Translucent surfaces** allow some light to penetrate the surface, e.g., glass or water.
Light sources

A real light source, such as a light bulb, is a geometric object, and at each point \((x, y, z)\) on its surface, emits light in a certain direction and at a certain intensity with respect to each wavelength \(\lambda\). In order to keep the model simple, however, we assume that each light source is a single point: a point source. We will also assume that each point source has red, green, and blue components. Each source will thus have a three-component intensity or luminance

\[
I = \begin{pmatrix} I_r \\ I_g \\ I_b \end{pmatrix}.
\]

All colour-light computations are applied to the three components independently.

Ambient light. This is the uniform light, for example, in a room, resulting from the placement of many (typically large) light sources. We can model ambient light by a constant intensity at every point in the scene,

\[
I_a = \begin{pmatrix} I_{ar} \\ I_{ag} \\ I_{ab} \end{pmatrix}.
\]
Point sources. An ideal point source emits light equally in all directions. We can characterize a point source at point $p_0$ by a vector

$$I(p_0) = \begin{pmatrix} I_r(p_0) \\ I_g(p_0) \\ I_b(p_0) \end{pmatrix}.$$ 

The intensity of illumination is proportional to the inverse square of its distance to the surface, so at a point $p$, the intensity is

$$i(p, p_0) = \frac{1}{|p - p_0|^2} I(p_0).$$
This simple point source model tends to give harsh renderings with high contrast between light and dark. In real life the size of light sources leads to softer scenes: some areas are fully in shadow, the \textit{umbra}, while others are in partial shadow, the \textit{penumbra}.

The high-contrast effect can be softened by adding ambient light. Moreover, though the inverse square law is technically correct, it is usually better to replace the distance term $d^{-2}$ by $(a + bd + cd^2)^{-1}$ to soften the lighting. If a source is far away, the distance term is approximately constant anyway.
**Spotlights.** Spotlights are characterized by a narrow range of angles through which light is emitted. A simple model is to use a cone with apex $p_s$ and direction $l_s$ and angular width $\theta > 0$. The maximum width of $\theta = 180$ gives a point source.

![Diagram of a spotlight](image)

A more realistic model takes into account that light from a spotlight is more intense, the closer the object is to the direction $l_s$. Light intensity is a function of the angle $\phi$ between the object direction $d$ and the vector $l_s$. We approximate this function by the function $\cos^c \phi$. This cosine can be computed quickly by the dot product, $\cos \phi = d \cdot l_s$. 

![Graph of intensity vs. angle](image)
Distant light sources. Most shading calculations require the direction from the point on the surface to the light source. This direction will have to be recomputed repeatedly. If the source is far away, however, the light rays can be approximated by parallel ones, an effect similar to replacing a perspective projection by a parallel one.

Thus while a close point source $p_0$ is represented in homogeneous coordinates as a point,

$$p_0 = \begin{pmatrix} x \\ y \\ z \\ 1 \end{pmatrix},$$

a distant light source can be replaced by a direction vector

$$p_0 = \begin{pmatrix} x \\ y \\ z \\ 0 \end{pmatrix}.$$

Scenes lit only by distant light sources can be rendered faster.
The Phong Reflection Model

The model uses four vectors to calculate the colour at an arbitrary point $p$ on a surface: the normal $n$; the direction of the viewer $v$; the direction of the light source $\ell$; and the direction of reflection $r$.

Each point source has ambient, diffuse, and specular components for each of the primary colours red, green, and blue. This source $i$ has a matrix of nine (fixed) intensity coefficients.

$$L_i = \begin{pmatrix} I_{ira} & I_{iga} & I_{iba} \\ I_{ird} & I_{igd} & I_{ibd} \\ I_{irs} & I_{igs} & I_{ibs} \end{pmatrix},$$

At each point $p$ on a surface we compute nine corresponding reflection terms

$$R_i = \begin{pmatrix} R_{ira} & R_{iga} & R_{iba} \\ R_{ird} & R_{igd} & R_{ibd} \\ R_{irs} & R_{igs} & R_{ibs} \end{pmatrix},$$

with respect to the $i$-th light source. The coefficients of $R_i$ will depend on the material properties, orientation of the surface, the direction of the light source $i$, and the viewer.
To compute, for example, the red intensity at point $p$ from source $i$, we combine the coefficients in the following way,

$$I_{ir} = R_{ira}L_{ira} + R_{ird}L_{ird} + R_{irs}L_{irs}$$

$$= I_{ira} + I_{ird} + I_{irs},$$

using the first column of each matrix. We then sum up all these contributions over all the light sources, and optionally add an ambient term to get the final red intensity

$$I_r = \sum_i (I_{ira} + I_{ird} + I_{irs}) + I_{ar},$$

where $I_{ar}$ is the ambient red intensity.

The computation is the same for each light source and for red, green, and blue and so we simplify notation by writing

$$I = I_a + I_d + I_s = R_aL_a + R_dL_d + R_sL_s,$$

for the intensity for any light source and any primary.

**Ambient reflection.** The amount of ambient light $L_a$ is the same at every point on the surface. Some is absorbed and some reflected. The proportion reflected is given by the ambient reflection coefficient $R_a = k_a$, with $0 \leq k_a \leq 1$, and so

$$I_a = k_aL_a.$$  

Here, $L_a$ can either be an individual light source or a global ambient term.
**Diffuse reflection.** Diffuse surfaces scatter light in all directions equally, though the amount reflected follows Lambert’s law: the diffuse reflection is proportional to the cosine of the angle \( \theta \) between the normal \( n \) and the direction of the source \( \ell \).

![Diffuse reflection diagram](image)

Since

\[
\cos \theta = \ell \cdot n,
\]

and adding a reflection coefficient \( k_d, 0 \leq k_d \leq 1 \), we have

\[
R_d = k_d (\ell \cdot n).
\]

If we also want to account for light attenuation as it travels a distance \( d \), we can use the quadratic attenuation term,

\[
R_d = \frac{k_d}{a + bd + cd^2} (\ell \cdot n),
\]

and so

\[
I_d = \frac{k_d}{a + bd + cd^2} (\ell \cdot n) L_d.
\]

11
**Specular reflection.** This is the kind that produces highlights.

The amount of specular light the viewer sees depends on the angle $\phi$ between the reflection direction $r$ and the direction $v$ of the viewer. The Phong model sets

$$R_s = k_s \cos^\alpha \phi,$$

for some coefficient $k_s$, $0 \leq k_s \leq 1$. The exponent $\alpha$ is a **shininess** coefficient. As $\alpha$ increases, the reflected light is concentrated in a narrower region, centred around $r$. Values of $\alpha$ between 100 and 500 correspond to metallic surfaces, smaller values correspond to broader highlights. Assuming $v$ and $r$ have unit length, we have

$$R_s = k_s (r \cdot v)^\alpha,$$

and we can also multiply a distance term, similar to diffuse reflections.
Combining the three types of reflection gives the full **Phong model**, 

\[
I = \frac{1}{a + bd + cd^2} \left( k_d (\ell \cdot n) L_d + k_s (r \cdot v) L_s \right) + k_a L_a.
\]

Note that the direction of reflection \( r \) is easy to compute from \( n \) and \( \ell \), assuming \( n \) has unit length. Simply noticing that 

\[
\frac{\ell + r}{2} = (\ell \cdot n)n,
\]

implies that 

\[
r = 2(\ell \cdot n)n - \ell.
\]
Polygonal shading

We usually model surfaces using polygons. This can help to speed up the lighting calculations. We consider three ways to shade polygons: flat shading, Gouraud shading, and Phong shading.

Flat shading. The Phong shading model requires the three vectors $\ell$, $n$, and $v$. For a flat polygon, the normal vector $n$ is constant. Moreover, if the viewer is distant, $v$ is approximately constant, and the same is true of $\ell$ if the light source is distant. If we assume that all three vectors are constant, the shading calculation needs to be carried out only once for each polygon, and the polygon will be assigned a constant shade. This is flat shading, and is specified in OpenGL by

```c
glShadeModel(GL_FLAT);
```

If flat shading is in effect, OpenGL uses the normal associated with the first vertex of the polygon.
**Gouraud shading.** Flat shading does not look realistic: Gouraud shading is better. OpenGL uses **scan-line interpolation** to assign a colour at each point of a projected polygon as a weighted average of given vertex colours. Consider, for example, the projected polygon below, with four vertices $c_0$, $c_1$, $c_2$, $c_3$,

![Projected Polygon](image)

A given scan-line crosses the two edges $[c_0, c_1]$ and $[c_2, c_3]$ at the points

$$c_4 = (1 - \alpha)c_0 + \alpha c_1, \quad c_5 = (1 - \beta)c_2 + \beta c_3,$$

for some $\alpha, \beta$, with $0 \leq \alpha, \beta \leq 1$. If $C_i$ denotes the colour at the point $c_i$, then the colours at the two new points are set to

$$C_4 = (1 - \alpha)C_0 + \alpha C_1, \quad C_5 = (1 - \beta)C_2 + \beta C_3.$$

The same interpolation is applied to an arbitrary point $c_6$ along the scan-line. If $\gamma$ is such that

$$c_6 = (1 - \gamma)c_4 + \gamma c_5,$$

then we set

$$C_6 = (1 - \gamma)C_4 + \gamma C_5.$$
Similar to the blending of given vertex colours, we can also blend the vertex colours resulting from the Phong lighting calculation. We assign a different normal to each vertex of the polygon, and so achieve shading that smoothly interpolates across polygon edges and vertices. This ‘smooth’ shading is specified in OpenGL by

\[
\text{glShadeModel(GL\_SMOOTH)};
\]

Gouraud’s idea was to define the normal at each vertex to be the average of all the neighbouring polygon normals. In the figure below we set

\[
n = \frac{n_1 + n_2 + n_3 + n_4}{|n_1 + n_2 + n_3 + n_4|}.
\]

A good data structure is needed to make these calculations efficient! If the underlying smooth surface is known, use the true normals instead.
Phong shading. An even more realistic shading algorithm is to blend the vertex normals at each point of the polygon to give a local normal, and then to apply the full lighting calculation there. This is Phong shading. The vertex normals can be blended in the same way as vertex colours: scan-line by scan-line.

For example, if the scan-line in the figure crosses the edge \([c_i, c_{i+1}]\) at the point

\[
c_A = (1 - \alpha)c_i + \alpha c_{i+1},
\]

then we define the normal there to be

\[
n_A = \frac{(1 - \alpha)n_i + \alpha n_{i+1}}{(1 - \alpha)n_i + \alpha n_{i+1}},
\]

and similarly for \(n_B\). Then \(n_A\) and \(n_B\) are linearly interpolated and normalized to find a surface normal at each point along the scan-line.

Phong shading is superior to Gouraud shading, but at significantly greater computational cost, and is not normally used for real-time rendering.
Light sources in OpenGL

OpenGL allows all four types of light source: ambient light; point source; spotlight, and distant light. The parameters correspond exactly to the Phong reflection model. The functions

```c
    glLightfv(source, parameter, pointer_to_array);
    glLightf(source, parameter, a);
```

are used to set vector or scalar parameters.

For example, suppose we locate the first light source at the point \((1,0,2,0,3,0)\). We use homogeneous coordinates

```c
    GLfloat light0_pos[] = {1.0, 2.0, 3.0, 1.0};
```

or for a distant light in direction \((1,0,2,0,3,0)\),

```c
    GLfloat light0_dir[] = {1.0, 2.0, 3.0, 0.0};
```

We might for example, want red ambient and diffuse light, and white specular light:

```c
    GLfloat ambient0[] = {1.0, 0.0, 0.0, 1.0};
    GLfloat diffuse0[] = {1.0, 0.0, 0.0, 1.0};
    GLfloat specular0[] = {1.0, 1.0, 1.0, 1.0};
```
Light source 0 is then specified by the code

```c
glEnable(GL_LIGHTING);
glEnable(GL_LIGHT0);

gllightfv(GL_LIGHT0, GL_POSITION, light0_pos);
gllightfv(GL_LIGHT0, GL_AMBIENT, ambient0);
gllightfv(GL_LIGHT0, GL_DIFFUSE, diffuse0);
gllightfv(GL_LIGHT0, GL_SPECULAR, specular0);
```

To specify an additional small amount of white ambient light, we use

```c
GLfloat global_ambient0[] = {0.1, 0.1, 0.1, 1.0};
gllightModelfv(GL_LIGHT_MODEL_AMBIENT, global_ambient);
```

The coefficients $a, b, c$ in the distance term $1/(a + bd + cd^2)$ are set, e.g., by

```c
gllightfv(GL_LIGHT0, GL_CONSTANT_ATTENUATION, a);
```

Spotlights can be created using `GL_SPOT_DIRECTION`, `GL_SPOT_EXPONENT`, `GL_SPOT_CUTOFF`.  

19
There are two more light parameters provided by OpenGL. To reduce calculations, the viewer is by default assumed to be an infinite distance from the object, so that the vector $v$ is constant. If the true vector $v$ is required, you can change the model through

$$\text{glLightModeli(GL\_LIGHT\_MODEL\_LOCAL\_VIEWER, GL\_TRUE);}$$

The other parameter concerns the question of whether only the front face of the polygon is to be shaded. If both the front and back faces are to be shaded, this can be invoked by the function call

$$\text{glLightModeli(GL\_LIGHT\_MODEL\_TWO\_SIDED, GL\_TRUE);}$$

Back face shading unnecessary

Back face shading necessary
Specification of materials in OpenGL

Similar to light sources, material properties in the Phong reflection model are all supported by OpenGL, through

\[
\text{glMaterialfv(face, type, pointer\_to\_array);} \\
\text{glMaterialf(face, value);} 
\]

We can set ambient, diffuse, and specular reflectivity coefficients \((k_a, k_d, k_s)\) using three arrays, e.g.,

\[
\text{GLfloat ambient[]} = \{1.0, 0.8, 0.0, 1.0\}; \\
\text{GLfloat diffuse[]} = \{0.2, 0.2, 0.2, 1.0\}; \\
\text{GLfloat specular[]} = \{1.0, 1.0, 1.0, 1.0\};
\]

followed by

\[
\text{glMaterialfv(GL\_FRONT\_AND\_BACK, GL\_AMBIENT, ambient);} \\
\text{glMaterialfv(GL\_FRONT\_AND\_BACK, GL\_DIFFUSE, diffuse);} \\
\text{glMaterialfv(GL\_FRONT\_AND\_BACK, GL\_SPECULAR, specular);} 
\]

The diffuse and specular components are often the same. They can be set in one go using GL\_DIFFUSE\_AND\_SPECULAR. If the front and back faces of the polygon have different material properties, we specify GL\_FRONT and GL\_BACK separately.