Objectives

- Learn to shade objects so their images appear three-dimensional
- Introduce the types of light-material interactions
- Build a simple reflection model—the Phong model—that can be used with real-time graphics hardware

Lighting

Objectives

- Learn to shade objects so their images appear three-dimensional
- Introduce the types of light-material interactions
- Build a simple reflection model—the Phong model—that can be used with real-time graphics hardware

Lighting Principles

- Lighting simulates how objects reflect light
  - material composition of object
  - light’s color and position
  - global lighting parameters
    - ambient light
    - two-sided lighting
  - available in both color index and RGBA mode

Why we need shading

- Suppose we build a model of a sphere using many polygons and color it with glColor. We get something like
- But we want

Shading

- Why does the image of a real sphere look like
- Light-material interactions cause each point to have a different color or shade
- Need to consider
  - Light sources
  - Material properties
  - Location of viewer
  - Surface orientation

Scattering

- Light strikes A
  - Some scattered
  - Some absorbed
- Some of scattered light strikes B
  - Some scattered
  - Some absorbed
- Some of this scattered light strikes A and so on
Rendering Equation

• The infinite scattering and absorption of light can be described by the rendering equation
  – Cannot be solved analytically in general
  – Ray tracing is a special case for perfectly reflecting surfaces
• Rendering equation is global and includes
  – Shadows
  – Multiple scattering from object to object

Global Effects

• Rendering equation is global and includes
  – Shadows
  – Multiple reflections from object to object

Local vs Global Rendering

• Correct shading requires a global calculation involving all objects and light sources
  – Incompatible with pipeline model which shades each polygon independently (local rendering)
• However, in computer graphics, especially real-time graphics, we are happy if things “look right”
  – Exist many techniques for approximating global effects

Light-Material Interaction

• Light that strikes an object is partially absorbed and partially scattered (reflected)
• The amount reflected determines the color and brightness of the object
  – A surface appears red under white light because the red component of the light is reflected and the rest is absorbed
• The reflected light is scattered in a manner that depends on the smoothness and orientation of the surface

Light Sources

General light sources are difficult to work with because we must integrate light coming from all points on the source

Simple Light Sources

• Point source
  – Model with position and color
  – Distant source = infinite distance away (parallel)
• Spotlight
  – Restrict light from ideal point source
• Ambient light
  – Same amount of light everywhere in scene
  – Can model contribution of many sources and reflecting surfaces
Shading Schemes

*Flat Shading*: same shade to entire polygon

Mach Band Illusion

Surface Types

- The smoother a surface, the more reflected light is concentrated in the direction a perfect mirror would reflect the light
- A very rough surface scatters light in all directions

Phong Model

- A simple model that can be computed rapidly
- Has three components
  - Diffuse
  - Specular
  - Ambient
- Uses four vectors
  - To source
  - To viewer
  - Normal
  - Perfect reflector

Ideal Reflector

- Normal is determined by local orientation
- Angle of incidence = angle of reflection
- The three vectors must be coplanar

Lambertian Surface

- Perfectly diffuse reflector
- Light scattered equally in all directions
- Amount of light reflected is proportional to the vertical component of incoming light
- \[ r = \frac{1}{2} (l \cdot n) n \cdot l \]
- \[ \cos \theta = l \cdot n \] if vectors normalized
- There are also three coefficients, \( k_d, k_s, k_a \), that show how much of each color component is reflected
Specular Surfaces

- Most surfaces are neither ideal diffusers nor perfectly specular (ideal reflectors)
- Smooth surfaces show specular highlights due to incoming light being reflected in directions concentrated close to the direction of a perfect reflection

Modeling Specular Relections

- Phong proposed using a term that dropped off as the angle between the viewer and the ideal reflection increased

$$L_f = k_s I_r \cos^n \alpha$$

Shading Schemes

Gouraud Shading: smoothly blended intensity across each polygon

Phong Shading: interpolated normals to compute intensity at each point

Scan Convert Polygon $P$

Intensity Interpolation

Compute by direction evaluation of illumination expression, whichever formula is being used
Surface Normals

- Normals define how a surface reflects light
  \texttt{glNormal3f(x, y, z)}
  - Current normal is used to compute vertex's color
  - Use unit normals for proper lighting
- Scaling affects a normal's length
  \texttt{glEnable(GL_NORMALIZE)}
  or
  \texttt{glEnable(GL_RESCALE_NORMAL)}

Using Average Normals

\[ \bar{N} = \text{true (geometric) normal} \]

\[ \bar{N} = \frac{1}{2}(N_1 + N_2) \]

\[ N_u = \frac{N_1 + N_2 + N_3 + N_4}{\| N_1 + N_2 + N_3 + N_4 \|} \]

More generally,

\[ N_u = \frac{\sum_{i=1}^{n} N_i}{\sum_{i=1}^{n} |N_i|} \]
An Interactive Introduction to OpenGL Programming

Relevant Light (unit) Vectors
- Surface Normal
- Point light source direction
- Reflection direction
- Viewpoint direction

Flat (Cosine) Shading
- Compute constant shading function, over each polygon, based on simple cosine term
- Same normal and light vector across whole polygon
- Constant shading for polygon
\[ I = I_p k_d \cos(\theta) \]
\[ = I_p k_d N \cdot L \text{, for unit } N, L \]

Where,
\[ I_p = \text{intensity of point light source} \]
\[ k_d = \text{diffuse reflection coefficient} \]

Gouraud Shading
- Compute constant shading function, for each vertex, based on simple cosine term
- Different normal and light vector for each vertex
- Interpolated shading for polygon
\[ \sim N \cdot L \]

Intensity Interpolation (Gouraud)
\[ I_a = I_{p1} \frac{y_3 - y_2}{y_1 - y_2} + I_{p2} \frac{y_1 - y_3}{y_1 - y_2} \]
\[ I_b = I_{p1} \frac{y_3 - y_1}{y_1 - y_3} + I_{p2} \frac{y_1 - y_3}{y_1 - y_3} \]
\[ I_p = I_{p1} \frac{x_3 - x_p}{x_1 - x_p} + I_{p2} \frac{x_p - x_3}{x_1 - x_p} \]
Normal Interpolation (Phong)

\[ N_a = N_1 \frac{y_2 - y_3}{y_1 - y_2} + N_2 \frac{y_1 - y_3}{y_1 - y_2} \]
\[ N_b = N_1 \frac{y_2 - y_3}{y_1 - y_2} + N_3 \frac{y_1 - y_3}{y_1 - y_2} \]

\[ \tilde{N}_p = \frac{N_a}{||N_a||} \left[ \frac{x_b - x_p}{x_b - x_a} \right] + \frac{N_b}{||N_b||} \left[ \frac{x_p - x_a}{x_b - x_a} \right] \]

Normalizing makes this a unit vector

Phong Illumination Formula (1/2)

\[ I_p = I_a k_a + f_{at} I_d k_d (N \cdot L)^n + f_{sk} I_s k_s (R \cdot V)^n \]

\[ \cos^n(\alpha) \]

Effect of Exponent Parameter

As \( n \) increases, highlight is more concentrated, surface appears glossier

Illumination Formula (2/2)

Where,

\( a \) denotes ambient term
\( d \) denotes diffuse term
\( s \) denotes specular term
\( k \) denotes coefficient
\( I \) denotes intensity

How OpenGL Simulates Lights

- Phong lighting model
  - Computed at vertices
- Lighting contributors
  - Surface material properties
  - Light properties
  - Lighting model properties
**Surface Normals**

- Normals define how a surface reflects light

  ```glNormal3f(x, y, z);```

  - Current normal is used to compute vertex’s color
  - Use unit normals for proper lighting
    - scaling affects a normal’s length
    ```
    glEnable(GL_NORMALIZE)
    or
    glEnable(GL_RESCALE_NORMAL)
    ```

**Material Properties**

- Define the surface properties of a primitive

  ```glMaterialfv(face, property, value);```

  - separate materials for front and back

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>GL_DIFFUSE</td>
<td>Base color</td>
</tr>
<tr>
<td>GL_SPECULAR</td>
<td>Highlight Color</td>
</tr>
<tr>
<td>GL_AMBIENT</td>
<td>Low-light Color</td>
</tr>
<tr>
<td>GL_EMISSION</td>
<td>Glow Color</td>
</tr>
<tr>
<td>GL_SHININESS</td>
<td>Surface Smoothness</td>
</tr>
</tbody>
</table>

**Light Properties**

- Define the light properties

  ```glLightfv(light, property, value);```

  - `light` specifies which light
    - multiple lights, starting with GL_LIGHT0
      ```
      glGetIntegerv(GL_MAX_LIGHTS, &n);
      ```
  - `properties`
    - colors
    - position and type
    - attenuation

**Light Sources (cont’d.)**

- Light color properties

  ```
  - GL_AMBIENT
  - GL_DIFFUSE
  - GL_SPECULAR
  ```

**Types of Lights**

- OpenGL supports two types of Lights
  - Local (Point) light sources
  - Infinite (Directional) light sources

- Type of light controlled by w coordinate
  - `w = 0`  Infinite Light directed along `(x, y, z)`
  - `w ≠ 0`  Local Light positioned at `(x, y, z)`

**Turning on the Lights**

- Flip each light's switch
  ```
  glEnable(GL_LIGHTn);
  ```

- Turn on the power
  ```
  glEnable(GL_LIGHTING);
  ```
**Light Material Tutorial**

- Modelview matrix affects a light's position
  - Different effects based on when position is specified
    - eye coordinates
    - world coordinates
    - model coordinates
  - Push and pop matrices to uniquely control a light's position

**Light Position Tutorial**

- Recall lighting computed only at vertices
  - model tessellation heavily affects lighting results
    - better results but more geometry to process
  - Use a single infinite light for fastest lighting
    - minimal computation per vertex

**Tips for Better Lighting**

- Recalling lighting computed only at vertices
  - model tessellation heavily affects lighting results
    - better results but more geometry to process
  - Use a single infinite light for fastest lighting
    - minimal computation per vertex

<table>
<thead>
<tr>
<th>Lighting Models</th>
</tr>
</thead>
<tbody>
<tr>
<td>From</td>
</tr>
<tr>
<td>Somewhere</td>
</tr>
<tr>
<td>Everywhere</td>
</tr>
</tbody>
</table>