Objectives

• Learn to shade objects so their images appear three-dimensional
• Introduce the types of light-material interactions
• Build a simple reflection model
Rendering

• Shading is the means by which we transform colored polygons and other graphical elements into gradations of colors that represent meaning such as shape, material properties and more
• Rendering is the process of producing an image from a modeled scene using shading and other techniques

Why we need shading

• Suppose we build a model of a sphere using many polygons and color each the same color. 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 (normal)

Scattering

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

• The infinite scattering and absorption of light can be described by the rendering equation

• Rendering equation is global and includes
  - Shadows
  - Multiple scattering from object to object

• Cannot be solved in general

• Ray-tracing is one local simplification appropriate for reflective surfaces

Global Effects

translucent surface

shadow

multiple reflection
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”
  - Use any of many techniques for approximating global effects

Light-Material Interaction

• Light that strikes an object is partially absorbed and partially scattered (reflected)

• The light 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

Modeling real-world light sources is difficult because we must consider light coming from all points on the source:

Simple Light Sources

- **Point source**
  - Model with position and color
- **Spotlight**
  - Restrict light from ideal point source (+more)
- **Directional source**
  - Point at infinite distance away (parallel)
- **Ambient light**
  - Same amount of light everywhere in scene
  - Can model contribution of many sources and reflecting surfaces
Ambient Light

- Ambient light is the result of multiple interactions between (large) light sources and the objects in the environment.
- Amount and color depend on both the color of the light(s) and the material properties of the object.

\[ L_v = k_a L_a \]

Coefficients, \( k_{r,b,g} \) for each color component.

Surface Types

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

- Perfectly diffuse reflector
- Light scattered equally in all directions
- Amount of light reflected is proportional to the vertical component of incoming light
  
  - reflected light is proportional to the angle of incidence or \( \sim \cos \theta_i \)
  
  - So, \( L_v = k_d \cos \theta_i L_d \)

  • Note, doesn’t matter where the viewer is (same true for ambient)

**Ideal Reflector (Mirror)**

- Normal is determined by local orientation
- Angle of incidence = angle of reflection

  ![Diagram](https://via.placeholder.com/150)

  - Here, viewer position matters. If viewer is not in the direction of the reflection, it is not seen by the viewer

  Note, vectors \( l, n, r \) are coplanar
Specular Surfaces

- Most surfaces are neither ideal diffusers nor perfectly reflective (ideal mirrors)
- But many surfaces show specular highlights where reflected light is concentrated in the direction close to that of the perfect reflection

- Here again, viewer position matters

Phong Model

- A simple model that can be computed rapidly
- Includes three components
  - Diffuse
  - Specular
  - Ambient
- Uses four vectors
  - To source, I
  - To viewer, v
  - Normal, n
  - Perfect reflector, r
Modeling Specular Reflections

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

\[ L_v = k_s L_s \cos^\alpha \phi \]

- Shininess coefficient (a.k.a. specular reflection coefficient) is used to model incoming intensity and specular intensity.

The Shininess Coefficient

- Values of \( \alpha \) between 100 and 200 correspond to metals.
- Values between 5 and 10 give surfaces that look like plastic.
Adding up the Components

The Phong model can be written as
\[ L = k_d L_d \cos \theta + k_s L_s \cos \alpha + k_a L_a \]

Replacing angles with dot product of vectors:
\[ L = k_d L_d \mathbf{l} \cdot \mathbf{n} + k_s L_s (\mathbf{v} \cdot \mathbf{r})^\alpha + k_a L_a \]

Note, I is sometimes used in place of L for illumination

Light Sources

• In the Phong Model, we add the results from each light source
• Each light source has separate diffuse, specular, and ambient terms to allow for maximum flexibility even though this form does not have a physical justification
• Separate red, green and blue components
• Hence, 9 coefficients for each light

- \( L_{vd} r^g b^t \), \( L_{vs} r^g b^t \), \( L_{va} r^g b^t \)
Material Properties

- Material properties match light source properties
  - Nine absorption coefficients
    - $k_{dr}$, $k_{dg}$, $k_{db}$, $k_{sr}$, $k_{sg}$, $k_{sb}$, $k_{ar}$, $k_{ag}$, $k_{ab}$
  - Shininess coefficient $\alpha$

Example

Only differences in these teapots are the parameters in the Phong model
Alternative to Phong Shading

The Phong model can be written as

\[ L = k_d \cdot L_d \cdot l \cdot n + k_s \cdot L_s \cdot (v \cdot r) + k_a \cdot L_a \]

Alternative to Phong Shading

Starting with the Phong model

\[ L = k_d \cdot L_d \cdot l \cdot n + k_s \cdot L_s \cdot (v \cdot r) + k_a \cdot L_a \]

Replace

\[(v \cdot r) \text{ with } (h \cdot n)\]

where \(h\) is "halfway" vector:

\[ h = \frac{l + v}{|l + v|} \]
Efficiency with "halfway" vector:

- $h$ is independent of position and surface curvature
  so calculate once for each light, use for entire frame (or longer)
- but $r$ depends on the normal
  and must be recalculated for each pixel of the image

*OpenGL computes shading in this fashion*
Attenuation Terms

• The light from a point source that reaches a surface is inversely proportional to the square of the distance between them.
• We can add a factor of the form $1/f(d)$ to the diffuse and specular terms.
• The constant and linear terms soften the effect of the point source.

Emissive Term

• We can simulate a glowing object (like a light source) by giving a material an emissive component.
• Affects only that material.
• This color is unaffected by any transformations.
• (Note, lights could always be negative as well.)
Transmission

• Let $\mu_i$ be the index of refraction of medium $i$
• Let $\mu_t$ be the index of refraction of $t$

Snell’s Law:
$$\frac{\sin(\theta_i)}{\sin(\theta_t)} = \frac{\mu_t}{\mu_i}$$

Transmission

• Refraction for different materials
  - Medium          Index of Refraction
    Water           1.33
    Ethyl Alcohol   1.36
    Air             1.0003
    Quartz          1.46
    Glass           1.5-1.6

Note, actually the index of refraction differs with each wavelength of light
Shading Polygons

- Polygons often approximate curve surfaces but are inherently flat
- Consider polygonal ‘sphere’
- Want to smooth the rough face of each surface facet
- How do we fix this?

Flat Shading

- Polygons have a single normal
- Shades at the vertices as computed by the Phong model lead to "flat" or uniform polygon
- Note, if we select flat shading in GL, the color at the first vertex will determine the color of the whole polygon
Smooth Shading

- We can simply find a new normal at each vertex for a sphere.
- Easy for sphere model.
  - If centered at origin $n = p$.
- Results in smoother shading.
- Note *silhouette edge*.

Vertex Normals

- The sphere example is not general because we knew the normal at each vertex analytically.
- For polygonal meshes, Gouraud in 70's proposed we use the average of normals around a mesh vertex.

$$n = \frac{n_1 + n_2 + n_3 + n_4}{\|n_1\| + \|n_2\| + \|n_3\| + \|n_4\|}$$
Gouraud Smoothing

- AKA: intensity interpolation shading
- Used in OpenGL
- Find vertex normals
- Apply Phong light model at each vertex
- Interpolate vertex shades across each polygon

Phong Smoothing

- Not the Phong lighting model!
- AKA: normal-vector interpolation shading
- Find vertex normals
- Interpolate vertex normals across edges and then across polygon
- Find shades using normals across polygons
Interpolating Over Polygons

- Given values at vertices of polygon, how do we interpolate data over interior?

- Values could be either normal or color.
### Interpolating Over Polygons

- Coloring with Edge Tables

<table>
<thead>
<tr>
<th>Edge</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>maxY</td>
<td>highest y-value</td>
</tr>
<tr>
<td>currentX</td>
<td>x-value of end-point with lowest y-value</td>
</tr>
<tr>
<td>xIncr</td>
<td>1 / slope</td>
</tr>
<tr>
<td>currentF</td>
<td>data value at end-point with lowest y-value</td>
</tr>
<tr>
<td>fIncr</td>
<td>((f_0-f_1)/(y_0-y_1))</td>
</tr>
</tbody>
</table>

\((F\) for color Frequency)
### Interpolating Over Polygons

**Active Edge List**

<table>
<thead>
<tr>
<th>$\text{maxY}$</th>
<th>$\text{currentX}$</th>
<th>$x\text{Incr}$</th>
<th>$\text{currentF}$</th>
<th>$f\text{Incr}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>5</td>
<td>$\frac{3}{2}$</td>
<td>4</td>
<td>$\frac{3}{2}$</td>
<td>$\frac{1}{2}$</td>
</tr>
<tr>
<td>5</td>
<td>5</td>
<td>-1</td>
<td>(0 0 1)</td>
<td>(0 $\frac{1}{2}$ $\frac{1}{2}$)</td>
</tr>
</tbody>
</table>

**Edge Table**

<table>
<thead>
<tr>
<th>7</th>
<th>6</th>
<th>5</th>
<th>4</th>
<th>3</th>
<th>2</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td>CB</td>
<td>AB</td>
<td>AC</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

### Interpolating Normals

- Exactly the same as colors
- Must renormalize

$\mathbf{n}_0 \rightarrow \mathbf{n}_1$
Interpolating Normals

• Exactly the same as colors
• Must renormalize
Comparison

• If the polygon mesh approximates surfaces with a high curvatures, Phong smoothing may look smooth when Gouraud shows edges
• Phong smoothing requires much more work than Gouraud smoothing
• Both need data structures to represent meshes so we can obtain vertex normals
• Both leave the silhouette jagged
Smooth Shading in GL

- For general polygon meshes in GL, shading calculations are done per vertex
  - Vertex colors become vertex shades
- When smooth shading, vertex colors are interpolated across the polygon

Enabling Shading

- Shading calculations are enabled by
  - `glEnable(GL_LIGHTING)`
  - Once lighting is enabled, `glColor()` ignored
- Polygon shading is turned on as:
  - `glShadeModel(GL_SMOOTH)` or
  - `glShadeModel(GL_FLAT)`
- Must enable light sources individually
  - `glEnable(GL_LIGHTi) i=0,1…..`
- Choose lighting parameters
Defining a Light Source

• For each light source, we can set an RGB for the diffuse, specular, and ambient parts, and the position

```c
GLfloat diffuse0[] = {1.0, 0.0, 0.0, 1.0};
GLfloat ambient0[] = {1.0, 0.0, 0.0, 1.0};
GLfloat specular0[] = {1.0, 0.0, 0.0, 1.0};
GLfloat light0_pos[] = {1.0, 2.0, 3.0, 1.0};
```

```c
glEnable(GL_LIGHTING);
glEnable(GL_LIGHT0);
glLightv(GL_LIGHT0, GL_POSITION, light0_pos);
glLightv(GL_LIGHT0, GL_AMBIENT, ambient0);
glLightv(GL_LIGHT0, GL_DIFFUSE, diffuse0);
glLightv(GL_LIGHT0, GL_SPECULAR, specular0);
```

Material Properties

• Material properties are also part of the OpenGL state and match the terms in the Phong model

• Set by glMaterialv()

```c
GLfloat ambient[] = {0.2, 0.2, 0.2, 1.0};
GLfloat diffuse[] = {1.0, 0.8, 0.0, 1.0};
GLfloat specular[] = {1.0, 1.0, 1.0, 1.0};
```

```c
glMaterialf(GL_FRONT, GL_AMBIENT, ambient);
glMaterialf(GL_FRONT, GL_DIFFUSE, diffuse);
glMaterialf(GL_FRONT, GL_SPECULAR, specular);
```