EECS 487: Interactive Computer Graphics

Lecture 16:
- Phong Illumination Model
- Shading

Phong Illumination Model

A local illumination model
• one bounce: light → surface → viewer

Lighting a single point

At the point:
- \( n \): surface normal (orientation of surface)
- \( l \): light vector (surface to light)
- \( v \): viewing vector (surface to eye)
- \( \theta \): light angle of incidence
- \( \phi \): viewing angle

Phong Illumination Model

Approximate surface color as sum of three components:
- an ideal **diffuse** component
- a glossy/blurred **specular** component
- an **ambient** term

\[
\text{Light sources:} \quad s_d \quad \text{diffuse light intensity} \\
\quad s_s \quad \text{specular light intensity} \\
\quad s_a \quad \text{ambient light intensity}
\]

Surface Reflection Coefficients

**Approximate BRDFs:** not physically based

Modify surface color by changing reflection coefficients based on:
- material type
- surface finish
- texture maps
- what looks good
  - artistic license
  - trial and error
  - personal library

\[
\begin{array}{c|c}
\hline
m_a & \text{ambient color reflection} \\
m_d & \text{diffuse color reflection} \\
m_s & \text{specular color reflection} \\
m_{sh} & \text{shininess (blurriness)} \\
m_e & \text{emissive color intensity} \\
\hline
\end{array}
\]
**Diffuse Example**

Where is the light?

Where's the normal direction at the brightest point?

---

**Ideal Diffuse Reflectance**

Ideal diffuse surface reflects light **equally** in all directions, according to Lambert's cosine law:

1. Amount of light energy that falls on surface and gets reflected is proportional to the incidence angle, $\theta$.
2. Perceived brightness (reflectance) is view independent.

At the microscopic level, an ideal diffuse surface is a very rough surface:

- Microfacets are oriented in any which way.
- Examples: chalk, clay, surface of moon.

---

**Ideal Diffuse Model**

Amount of light energy that falls on surface is proportional to incidence angle, $\theta$:

$$ c_d = s_d \odot m_d \cos \theta $$

For normalized $\mathbf{n}$ and $\mathbf{l}$:

$$ c_d = s_d \odot m_d \max((\mathbf{n} \cdot \mathbf{l}), 0) $$

- $(\mathbf{n} \cdot \mathbf{l}) < 0 \Rightarrow \cos \theta < 0 \Rightarrow \theta > 90^\circ$, light source is behind surface.
- Why must $\mathbf{n}$ and $\mathbf{l}$ be normalized?

$$ \mathbf{u} \cdot \mathbf{v} = ||\mathbf{u}|| ||\mathbf{v}|| \cos \theta \Rightarrow \cos \theta = \frac{\mathbf{u} \cdot \mathbf{v}}{||\mathbf{u}|| ||\mathbf{v}||} $$
**Diffuse Reflectance and Viewing Angle**

Lambert’s Cosine Law: amount of light energy that falls on surface and gets reflected is proportional to incidence angle, $\theta_i$
- $\cos \theta_i = \text{dot product of light vector with normal (both normalized)}$

Shouldn’t the amount of energy reflected also be proportional to viewing angle, $\theta_v$?
- No, whereas larger $\theta_i$ means energy arriving over area $dA \cos \theta_i$ is spread across the larger area $dA$, larger $\theta_i$ simply means collecting energy from the same area $dA$, but channelling it through the smaller area $dA \cos \theta_i$.

**Ideal Specular/Mirror Reflectance**

Accounts for highlight seen on objects with smooth, shiny surfaces, such as:
- metal
- polished stone
- plastics
- apples
- skin

**Ideal Specular/Mirror Reflectance**

Reflection only at mirror angle
- highlight intensity depends on viewing direction

Model: all microfacets of mirror surface are oriented in the same direction as the surface itself
- examples: mirrors, highly polished metals

**Phong Specular Reflection**

Simulates a highlight
Reflection angle = incidence angle = $\theta$
Most intense specular reflection when $v = r$

How to compute $r$?
$$r = -l + 2(n \cdot l)n$$
(in OpenGL, specular term is 0 if $l \cdot n = 0$)
Glossy Reflectors

Real materials tend to deviate significantly from ideal mirror reflectors ⇒ highlight and reflections are blurry

Also known as: “rough specular”, “directional diffuse”, or “glossy” reflection

(there are no ideal diffuse surfaces either …)

Simple empirical model:
• we expect most of the reflected light to travel in the direction of the ideal reflection ray
• but due to variations in microfacet orientations, some of the light will be reflected just slightly off from the ideal reflected direction
• as we angle away from the reflected ray, we expect to see less light reflected

Surface Roughness

Phong “Glossy” Reflection Model

As \( v \) angles away from \( r \), specular reflection falls off, simulating “glossy” reflection:
Phong “Glossy” Reflection Model

Phong specular model:
\[ c_s = s_s \odot m_s \cos \phi = s_s \odot m_s \max((r \cdot v), 0) \]

Material Glossiness

To account for the shininess of different material:
\[ c_s = s_s \odot m_s \max((r \cdot v), 0)^{m_{shi}} \]
larger \( m_{shi} \) gives tighter and shinier highlight, with sudden dropoff, simulating a more mirror-like surface

Phong Specular Reflection

larger \( m_{shi} \) tighter highlight →

Blinn-Phong (Blinn-Torrance) Model

Back to micro facets:
- model surface by a collection of tiny mirrors
- specular reflectance comes from mirrors oriented halfway between \( I \) and \( v \), in direction of \( h \)
**Blinn-Phong**

Model specular reflection with “halfway” vector ($h$) instead of $r$

- consider the microfacet through point $p$ that reflects light perfectly to the viewer
- $h$ is the normal of this plane, it is halfway between $l$ and $v$ (by definition), $h$ normalized
- angle between $h$ and $n$ is $\beta$; $v = r$ when $h = n$ ($\beta$ is not the angle between $v$ and $r$)
- specular reflection modeled as:
  $$c_s = s_s \odot m_s (n \cdot h)^{\text{diff}} = s_s \odot m_s (\cos \beta)^{\text{diff}}$$

**Illumination Models**

A rendering process can be modeled as an integral equation representing the transport of light through the environment ⇒ the rendering equation

Local illumination: an approx. of the rendering eqn.
- assumes light is scattered only once: light from light source is reflected by a surface and modulated on its way towards the eye

**Global illumination:**
- light rays traveling from light to surface may be
  - blocked by intervening surfaces (shadows) or
  - bent or scattered by intervening material (refraction and atmospheric effects) or
  - light arriving at a surface may come indirectly via another surface (reflection and color bleed)

**Phong Ambient Term**

“Approximates” the contribution of all indirect illumination

Surface uniformly lit
- areas with no direct illumination are not completely dark
- independent of:
  - light direction
  - surface normal
  - viewing angle

Spheres rendered with ambient reflection only

**Putting It All Together**

Phong Illumination Model

$$c_f = m_s \cdot s_a + (m_d (n \cdot l) + m_l (r \cdot v)^{\text{diff}}) f(d)s$$

<table>
<thead>
<tr>
<th>Phong</th>
<th>$P_a$</th>
<th>$P_d$</th>
<th>$P_l$</th>
<th>$P_{\text{total}}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\theta = 60^\circ$</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$\theta = 25^\circ$</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$\theta = 0^\circ$</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
OpenGL Light Sources

### glLightfv (lightname, param, value)
- **parameters**
  - GL_AMBIENT
  - GL_DIFFUSE
  - GL_SPECULAR
  - GL_POSITION
  - GL_SPOT_DIRECTION
  - GL_SPOT_CUTOFF
  - GL_SPOT_EXPONENT
  - GL_CONSTANT_ATTENUATION
  - GL_LINEAR_ATTENUATION
  - GL_QUADRATIC_ATTENUATION

### Turning on the lights:
- glEnable (GL_LIGHTING)
- glEnable (GL_LIGHT0)

OpenGL Lighting and Reflectance

```c
/* Initialize material property, light source, lighting model, and depth buffer. */
void init(void)
{
    GLfloat mat_specular[] = {1.0, 1.0, 1.0, 1.0};
    GLfloat mat_shininess[] = {50.0};
    GLfloat light_position[] = {1.0, 1.0, 1.0, 0.0};

    glClearColor(0.0, 0.0, 0.0, 0.0);
    glShadeModel(GL_SMOOTH);
    glMaterialfv(GL_FRONT, GL_SPECULAR, mat_specular);
    glMaterialfv(GL_FRONT, GL_SHININESS, mat_shininess);
    glLightfv(GL_LIGHT0, GL_POSITION, light_position);
    glEnable(GL_LIGHTING);
    glEnable(GL_LIGHT0);
    glEnable(GL_DEPTH_TEST);
}
```

OpenGL Material Properties

### glMaterial (face, param, value)
- **face**
  - GL_FRONT
  - GL_BACK
  - GL_FRONT_AND_BACK
- **parameters**
  - GL_AMBIENT
  - GL_DIFFUSE
  - GL_AMBIENT_AND_DIFFUSE
  - GL_SPECULAR
  - GL_SHININESS
  - GL_EMISSION

Choosing the Parameters

Experiment with different parameter settings:
- $m_a + m_d + m_s < 1$
- use a small $m_a$ (~0.1)
- try $m_{shi} \in [0, 100]$

See Redbook Chapter 5 for techniques to minimize performance costs associated with changing material properties.

<table>
<thead>
<tr>
<th>material</th>
<th>$m_d$</th>
<th>$m_s$</th>
<th>$m_{shi}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>metal</td>
<td>small, color of metal</td>
<td>large, color of metal</td>
<td>large</td>
</tr>
<tr>
<td>plastic</td>
<td>medium, color of object</td>
<td>medium, color of light (or white)</td>
<td>medium</td>
</tr>
<tr>
<td>planet</td>
<td>varying</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>
Highlight Color

For non-metals, e.g., plastics, highlight color is color of light (plastics has a transparent/white coating)
For metals, e.g., brass, highlight depends on surface color

Attenuation Model

Whereas the ambient term simulates indirect lighting, attenuation model and fog simulate scattering effect

Intensity attenuation: light falls off the further away one gets from the source
• distinguishes overlapping surfaces having the same reflection parameters
• radiant energy disperses as $1/d^2$
• $d$ is the distance from the light source to surface
• reason for the inverse square falloff?
• attenuation function $f(d) = 1/(a_0 + a_1 d + a_2 d^2)$
  • user defined constants $a_0$, $a_1$, $a_2$
  • since we’re not modeling atmospheric scattering, $1/d^2$ often appears too harsh
• instead, we use $f(d) = 1/(a_0 + a_1 d)$ or no attenuation
Lighting with attenuation: $c' = f(d)c$

Emissive Term

Material’s emissive parameter $m_e$
• assumed not lighting other objects

$c_i = m_e + m_s a + (m_d(n \cdot l) + m_r(n \cdot r)^{m_w})f(d)s$

Global Ambient and Multiple Lights

Global ambient light source: $c_g = s_g \otimes m_a$
• $s_g$ set with:
  `glLightModel(GL_LIGHT_MODEL_AMBIENT,ambient_light)`

Multiple lights:

$c_i = c_g + m_e + \sum_{k=1}^{n} c_a^{(k)} + f(d^{(k)}) (c_d^{(k)} + c_r^{(k)})$,

$k$: light number, not exponentiation
Fog

Simple atmospheric effect
• a little better realism
• help to convey distances
• 2 forms of depth cueing (ignoring scattering)
  • light-to-object distance conveyed by light attenuation
  • object-to-viewer atmospheric scattering simulated by fog

Fog in OpenGL

```c
#include <GL/gl.h>

void drawFog() {
    glEnable(GL_FOG);
    glHint(GL_FOG_HINT, GL_NICEST);
    glEnable(GL_DEPTH_TEST);
    glFogf(GL_FOG_COLOR, 1.0f);
    glFogf(GL_FOG_DENSITY, 0.5f);
    glFogf(GL_FOG_END, 1.0f);
    glFogf(GL_FOG_START, 0.5f);
    glFogf(GL_FOG_MODE, GL_LINEAR);
    glFogfv(GL_FOG_COLOR, &colorFog);
    glFogfv(GL_FOG_DENSITY, &densityFog);
    glFogfv(GL_FOG_END, &endFog);
    glFogfv(GL_FOG_START, &startFog);
    glFogfv(GL_FOG_COORD, &coordFog);
    glEnd();
}
```

Fog

- color of fog: \( c_f \)
- color of surface: \( c_s \)
- fog effect:
  \[
  c_p = f c_s + (1 - f) c_f, f \in [0,1]
  \]

How to compute \( f \)?
• linear: \( f_p = \frac{z_{\text{end}} - z_p}{z_{\text{end}} - z_{\text{start}}} \)
• exponential: \( f_p = e^{-d_p} \)
• exponential-squared: \( f_p = e^{-d_p^2} \)

can also be a function of height off the ground

Lighting and Shading

Lighting: computing interaction between light and surfaces of different materials, and interaction with the geometry of objects to determine the luminous intensity reflected from a specified 3D point

Shading: performing the lighting computations on polygonal objects and coloring the pixels

- flat shading
- Gouraud shading
- Phong shading
Flat Shading

Entire polygon displayed with the same intensity
- calculate intensity from the reflection model
- use the surface normal (for triangles)
- compute an average normal (for >3 vertex polygons)

Is a valid shading model when:
- object is truly planar (not an approximation of a curved surface)
- all k light sources are far enough away that \( \mathbf{n} \cdot \mathbf{l}^{(k)} \) is constant over the polygon surface
- viewing position is far enough away that \( \mathbf{n} \cdot \mathbf{h} \) is constant over the polygon surface

Problem with Flat Shading: Mach Band Illusion

In 1865, Mach discovered that human eyes perceive a surface with flat shading of different hues as having a “fluted” aspect due to the abrupt change in shades of color (discontinuity in the first derivative of the shading function)

Mach Band Illusion is due to lateral inhibition of the human visual system:
- neighboring receptors are connected
- when one fires, it inhibits its neighbors from firing

Effect:
- eye sensitive to difference from surrounding area
- good for edge and motion detection

Lateral Inhibition

Gouraud Shading

Match intensity across polygon edges
Linear interpolation of color across polygon surface
Gouraud shading algorithm:
- determine average normal at each vertex (averaged over the normals of all polygons that share the vertex)
- compute color at each vertex using the average normal
- linearly interpolate color across a single polygon surface

\[
\mathbf{n}_\text{avg} = \frac{1}{n} \sum_{i=1}^{n} \mathbf{n}_i
\]

\( n = \text{all polygons that share the vertex} \)
Gouraud Shading Example

1. Compute average normal at each vertex \((1, 2, 3)\)
2. Compute colors \(c_1, c_2, c_3\)
3. Compute colors \(c_4, c_5\)
   
   \[ c_4 = \frac{y_2 - y_3}{y_1 - y_2} c_1 + \frac{y_1 - y_2}{y_1 - y_2} c_2 \]

4. Compute color \(c_p\)
   
   \[ c_p = \frac{x_5 - x_p}{x_5 - x_4} c_4 + \frac{x_5 - x_p}{x_5 - x_4} c_5 \]

Or use barycentric coordinates!
Steps 2–4 were cheap enough to be implemented in hardware even for fixed pipeline GPU

Mach Band not Eliminated
when there's an abrupt change in the orientation of two polygons and there's discontinuity in the 1\textsuperscript{st} derivative of the shading function

\[ c_4 = y_4 - y_2 \]
\[ c_2 = y_1 - y_2 \]

effect can be ameliorated with increased polygon count or Phong shading ...

Limitation: Wrong Highlights

Highlights depend on polygonal shape
Does not capture highlight in the middle of polygon
• \( c_1 = 0 \) because \((\mathbf{n} \cdot \mathbf{h}) < 0\)
• \( c_2 = 0 \) because \((\mathbf{n} \cdot \mathbf{l}) < 0\)
• any interpolation of \(c_1\) and \(c_2\) will be 0

Shows up in animation as flashing highlights between frames as orientation of polygons change

Limitation: Wrong Highlights

Why does Gouraud give inaccurate simulation of highlights?
• color at vertex may not be (highlight) color inside the polygon

What would give the most accurate simulation of highlights?
• compute normal of actual surface (not polygon-approximated surface) at each pixel and color accordingly \(\Rightarrow \) very expensive!

Phong shading: interpolate normal of vertices across polygon surface and color each pixel according to interpolated normals
Phong Shading

Algorithm:
• determine average normal at each vertex
• linearly interpolate per-pixel normal across surface
• compute color for each pixel using the value of the approximated per-pixel normal

Phong illumination model ≠ Phong specular reflection ≠ Phong shading

OpenGL Lighting and Reflectance

/* Initialize material property, light source, lighting model, and depth buffer. */
void init(void)
{
    GLfloat mat_specular[] = { 1.0, 1.0, 1.0, 1.0 };
    GLfloat mat_shininess[] = { 50.0 };
    GLfloat light_position[] = { 1.0, 1.0, 1.0, 0.0 };

    glClearColor(0.0, 0.0, 0.0, 0.0);
    glShadeModel(GL_SMOOTH); // or GL_FLAT
    // interpolate color of vertices, but does not average normals at the vertices
    glMaterialfv(GL_FRONT, GL_SPECULAR, mat_specular);
    glMaterialfv(GL_FRONT, GL_SHININESS, mat_shininess);
    glLightfv(GL_LIGHT0, GL_POSITION, light_position);
    glEnable(GL_LIGHTING);
    glEnable(GL_LIGHT0);
    glEnable(GL_DEPTH_TEST);
}

Where in the Pipeline?

Application
AAPL_Vertex Processing
AAPL_Primitive Processing
AAPL_Rasterization
AAPL_Fragment Processing
AAPL_Framebuffer
AAPL_Display

vertex colors computed

vertex colors interpolated across polygon (Gouraud shading)

per-pixel Phong shading (expensive)