Why do we need Lighting & shading?

- Has **visual cues** for humans (shape, light position, viewer position, surface orientation, material properties, etc).

Sphere without lighting & shading

Sphere with lighting & shading
What Causes Shading?

- Shading caused by different angles with light, camera at different points
Lighting?

- **Problem:** Calculate surface color based on angle of surface with light, viewer
- Programmer writes vertex shader code to calculate lighting at vertices!
- Equation for lighting calculation = *lighting model*

1. **Light attributes:** intensity, color, position, direction, shape
2. **Surface attributes** color, reflectivity, transparency, etc
3. **Interaction** between lights and objects
Shading?

- After triangle is rasterized (converted to pixels)
  - Per-vertex lighting calculation means color at vertices is accurate, known (red dots)
- Shading: Graphics hardware figures out color of interior pixels
- How? Assume linear change => interpolate

Lighting (calc at vertices in vertex shader)

Rasterization
Find pixels belonging to each object

Shading (done in hardware during rasterization)
Global Illumination (Lighting) Model

- Global illumination: model interaction of light from all surfaces in scene (track multiple bounces)

- translucent surface
- shadow
- multiple reflection
- translucent surface
The infinite reflection, scattering and absorption of light is described by the rendering equation:

\[ L_o = L_e(x, \bar{\omega}) + \int_{\Omega} fr(x, \bar{\omega}', \bar{\omega}) Li(x, \bar{\omega}')(\bar{\omega}' \cdot \bar{n}) d\bar{\omega}' \]

- **\( L_o \)** is outgoing radiance
- **\( L_i \)** incident radiance
- **\( L_e \)** emitted radiance,
- **\( fr \)** is bidirectional reflectance distribution function (BRDF)
  - Fraction of incident light reflected by a surface
Local Illumination (Lighting) Model

- One bounce!
  - Doesn’t track inter-reflections, transmissions

- Global Illumination (GI) is accurate, looks real
  - But raster graphics pipeline (e.g. OpenGL) renders each polygon independently (local rendering), no GI
Light Sources

- General light sources are difficult to model (e.g. light bulb)
- Why? We must compute effect of light coming from all points on light source
Light Sources Abstractions

- We generally use simpler light sources
- Abstractions that are easier to model

- Point light
- Directional light
- Spot light
- Area light

Light intensity can be **independent** or **dependent** of the distance between object and the light source.
Light-Material Interaction

- Light strikes object, some absorbed, some reflected
- Fraction reflected determines object color and brightness
  - **Example:** A surface looks red under white light because the red component of light is reflected, other wavelengths absorbed
Phong Model

- Simple lighting model that can be computed quickly
- 3 components
  - Diffuse
  - Specular
  - Ambient
- Compute each component separately
- Vertex Illumination =
  \[
  \text{ambient} + \text{diffuse} + \text{specular}
  \]
- Materials reflect each component differently
Phong Model

- Compute lighting (components) at each vertex (P)
- Uses 4 vectors, from vertex
  - To light source (l)
  - To viewer (v)
  - Normal (n)
  - Mirror direction (r)
Mirror Direction?

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

\[ r = 2 \ (l \cdot n) \ n - l \]
Surface Roughness

- **Smooth surfaces**: more reflected light concentrated in mirror direction
- **Rough surfaces**: reflects light in all directions
Diffuse Lighting Example
Diffuse Light Calculation

- How much light received from light source?
- Based on Lambert’s Law

Receive more light

Receive less light
Diffuse Light Reflected

- Illumination surface received from light source, reflected equally in all directions

Eye position does not matter
Lambert’s law: radiant energy \( D \) a small surface patch receives from a light source is:

\[ D = I \times k_D \cos(\theta) \]

- \( I \): light intensity
- \( \theta \): angle between light vector and surface normal
- \( k_D \): Diffuse reflection coefficient.

Controls how much diffuse light surface reflects.
Specular light example
Specular light contribution

- Incoming light reflected out in small surface area
- Specular depends on viewer position relative to mirror direction
- Specular bright in \textbf{mirror direction}
- Drops off away from mirror direction
Specular light calculation

- $\phi$ is deviation of view angle from mirror direction
- Small $\phi$ = more specular

\[ I_s = k_s I \cos^\alpha \phi \]

- incoming intensity
- reflected intensity
- shininess coef
- Absorption coef
The Shininess Coefficient, $\alpha$

- $\alpha$ controls falloff sharpness
- High $\alpha = \text{sharper falloff} = \text{small, bright highlight}$
- Low $\alpha = \text{slow falloff} = \text{large, dull highlight}$
  - $\alpha$ between 100 and 200 = metals
  - $\alpha$ between 5 and 10 = plastic look
Specular light: Effect of ‘$\alpha$’

$$I_s = k_s I \cos^\alpha \phi$$

$\alpha = 10$

$\alpha = 90$

$\alpha = 30$

$\alpha = 270$
Ambient Light Contribution

- Very simple approximation of global illumination (Lump 2\textsuperscript{nd}, 3\textsuperscript{rd}, 4\textsuperscript{th}, .... etc bounce into single term)
- Assume to be a constant
- No direction!
  - Independent of light position, object orientation, observer’s position or orientation

\[
\text{Ambient} = I_a \times K_a
\]

constant
Ambient Light Example

**Ambient**: background light, scattered by environment
Light Attenuation with Distance

- Light reaching a surface \textit{inversely proportional} to square of distance \( d \)
- We can multiply by factor of form \( 1/(ad + bd + cd^2) \) to \textit{diffuse} and \textit{specular} terms
Adding up the Components

- Adding all components (no attenuation term), phong model for each light source can be written as:

  \[ I = k_d I_d \cos\theta + k_s I_s \cos\phi^\alpha + k_a I_a \]
  \[ = k_d I_d (l \cdot n) + k_s I_s (v \cdot r)^\alpha + k_a I_a \]

- Note:
  - \( \cos\theta = l \cdot n \)
  - \( \cos\phi = v \cdot r \)
Separate RGB Components

- Can separate red, green and blue components. Instead of:

\[
I = k_d I_d (l \cdot n) + k_s I_s (v \cdot r)\alpha + k_a I_a
\]

- We computing lighting for RGB colors separately

\[
I_r = k_{dr} I_{dr} l \cdot n + k_{sr} I_{sr} (v \cdot r)^\alpha + k_{ar} I_{ar} \quad \text{Red}
\]
\[
I_g = k_{dg} I_{dg} l \cdot n + k_{sg} I_{sg} (v \cdot r)^\alpha + k_{ag} I_{ag} \quad \text{Green}
\]
\[
I_b = k_{db} I_{db} l \cdot n + k_{sb} I_{sb} (v \cdot r)^\alpha + k_{ab} I_{ab} \quad \text{Blue}
\]

- Above equation is just for one light source!!

- For N lights, repeat calculation for each light

Total illumination for a point \( P \) = \( \Sigma \) (Lighting for all lights)
## Coefficients for Real Materials

<table>
<thead>
<tr>
<th>Material</th>
<th>Ambient Kar, Kag, kab</th>
<th>Diffuse Kdr, Kdg, kdb</th>
<th>Specular Ksr, Ksg, ksb</th>
<th>Exponent, $\alpha$</th>
</tr>
</thead>
<tbody>
<tr>
<td>Black plastic</td>
<td>0.0</td>
<td>0.01</td>
<td>0.5</td>
<td>32</td>
</tr>
<tr>
<td></td>
<td>0.0</td>
<td>0.01</td>
<td>0.5</td>
<td></td>
</tr>
<tr>
<td></td>
<td>0.0</td>
<td>0.01</td>
<td>0.5</td>
<td></td>
</tr>
<tr>
<td>Brass</td>
<td>0.329412</td>
<td>0.780392</td>
<td>0.992157</td>
<td>27.8974</td>
</tr>
<tr>
<td></td>
<td>0.223529</td>
<td>0.568627</td>
<td>0.941176</td>
<td></td>
</tr>
<tr>
<td></td>
<td>0.027451</td>
<td>0.113725</td>
<td>0.807843</td>
<td></td>
</tr>
<tr>
<td>Polished Silver</td>
<td>0.23125</td>
<td>0.2775</td>
<td>0.773911</td>
<td>89.6</td>
</tr>
<tr>
<td></td>
<td>0.23125</td>
<td>0.2775</td>
<td>0.773911</td>
<td></td>
</tr>
<tr>
<td></td>
<td>0.23125</td>
<td>0.2775</td>
<td>0.773911</td>
<td></td>
</tr>
</tbody>
</table>

Figure 8.17, Hill, courtesy of McReynolds and Blythe
**Modified Phong Model**

\[
I = k_d I_d \ l \cdot n + k_s I_s (v \cdot r)^\alpha + k_a I_a \\
I = k_d I_d \ l \cdot n + k_s I_s (n \cdot h)^\beta + k_a I_a
\]

- Blinn proposed using **halfway vector**, more efficient
- \( h \) is normalized vector halfway between \( l \) and \( v \)
- Similar results as original Phong

\[
h = (l + v)/|l + v|
\]
References