Lighting simulates how objects reflect light. Reflected light is based on a number of elements:
- Material composition of object
- Light's color and position
- Global lighting parameters
  - Ambient light
  - Two-sided lighting

Things to consider:
- Light sources
- Surface effects
- Lighting models

Light Sources
- Simplest form is a point source
- Energy is emitted from a single point
  - Typically in a single color, specified with RGB components
- Rays emitted in all directions
- For infinitely distant point sources, rays striking objects are essentially parallel
- Can treat non-point sources as point sources if they are sufficiently distant

Light Sources
- Can constrain the output of a light source so that it only travels in one direction
- Called a directional or spotlight source
  - Constraint determined by the angle $\theta$
- Differs from non-directional light in that it may not strike all objects in a scene

Types of Light
- Ambient
  - Light from no direction (due to scattering)
  - Surfaces illuminated by ambient light reflect in all directions
- Diffuse
  - Light from a certain direction
  - Reflected equally in all directions from the surface (causing the surface to look equally bright)
- Specular
  - Directional light which reflects off a surface in a particular direction
  - Causes the surface to have a shiny highlight
- Emissive
  - Light originating within an object
  - Does not affect other objects in the scene
  - We won't cover this in any detail

Ambient Light
- No apparent source - simply present (this is a real HACK for a complex phenomenon)
- Non-directional, uniform illumination
- Illumination equation:
  $$ A = I_a k_a $$
- $I_a$ is the ambient illumination
- $k_a$ is the ambient reflection coefficient
  - The amount of ambient light reflected by an object
  - Property of the object's material
  - Ranges between 0 and 1

Here is the effect of varying the ambient reflection coefficient from 0.0 to 0.1, 0.3, 0.5, and 0.7:
Diffuse Reflection

• Reflection from dull, matte surfaces (e.g., chalk)
  - Also called Lambertian reflection
• Light comes from a point source
• Light is reflected with equal intensity in all directions
• For a given surface, brightness depends only on the angle between the direction to the light source and the surface normal
  \[ D = I_p k_d \cos \theta \]
• \( k_d \) is the diffuse reflection coefficient

The angle \( \theta \) must be between 0° and 90° for the light source to have any effect on the object
• This means that a light source behind an object doesn’t illuminate it
• We say this object is self-occluding
• Technically, we should have a \( \max(\cos \theta, 0) \) term in the equation
• It’s simpler to just assume the angle is “legal”

The effect of varying the diffuse reflection coefficient from 0.0 to 0.2, 0.4, 0.6, 0.8, and 1.0:

If the vectors have been normalized, we can replace \( \cos \theta \) with their dot product:
\[ D = I_p k_d (\text{Normal} \cdot \text{Light}) \]

If the point source is far enough away, it makes essentially the same angle with all surfaces sharing the same surface normal
• In this case, it’s called a directional light source

Attenuation – Part I

• Does distance from a light matter?
• Radiant energy from a light traveling through space is naturally attenuated at a rate of \( 1/d^2 \) where \( d \) is the distance from the light source
  - In practice, this doesn’t look good in our pictures when we use a point source as it tends to produce too much variation of intensities!

Attenuation – Part II

• Distance from a light source affects the amount of illumination seen on an object
• We use an attenuation factor, \( f_{att} \), to account for this
  - Result is limited (clamped) to a maximum of 1
  \[ f_{att} = \min \left( \frac{1}{k_q + k_r d_s^2 + k_i d_i^2}, 1 \right) \]
• \( d_s \) is the distance light travels from the source to the surface
• The clamping is for a source at infinity
• \( k_q, k_r, \text{ and } k_i \) are attenuation constants
  - Constant, linear, and quadratic
  - \( k_i \) is there to prevent division by zero
Colored Lights and Surfaces

- Commonly, we deal with colored lights and surfaces by writing separate equations for each component of the color model.
- We represent an object's diffuse color with $O_d$.
  - One term for each component: $(O_dR, O_dG, O dB)$
- The light's components $I_pR$, $I_pG$, and $I_pB$ are reflected in proportion to $k_d O_dR$, $k_d O_dG$, and $k_d O dB$.
- Thus, for the red component,
  
  $$ I_R = I_a R k_d O_dR + f_{att} I_pR k_d O_dR (N \cdot L) $$

To account for other color models, we replace the color indicator with the symbol $\lambda$:

$$ I_{\lambda} = I_a \lambda k_{d\lambda} O_{d\lambda} + f_{att} I_{p\lambda} k_{d\lambda} O_{d\lambda} (N \cdot L) $$

Specular Reflection

- Observable on any shiny surface.
  - "Highlights" are caused by specular reflection.
- On a perfectly reflective surface, light is reflected only in one direction.
  - Angle of reflection = angle of incidence.
- The angle between the reflection and the viewpoint, $\alpha$, determines the amount of reflection seen.
  - Also affected by the specular reflection exponent of the material.
  - For a perfect mirror, can only see reflection when the angle between the Reflection and Viewer ($\alpha$) is zero.

Final Light

- Maximum reflection occurs when $\alpha$ is zero.
- Falloff is approximated by $\cos^n \alpha$.
  - $n$ is the specular reflection exponent.
  - Low values of $n$ give gentle falloff; high values give sharp, focused highlights.

Math Reminder (Again)

- For a vector $w$:
  - Magnitude: $|w| = \sqrt{w_1^2 + w_2^2 + \ldots + w_n^2}$
  - Normalized unit vector: $\hat{w} = \frac{w}{|w|}$
  - $|\hat{w}| = 1$
  - Dot product of vectors $A$ and $B$: $A \cdot B = a_1 b_1 + a_2 b_2 + \ldots + a_n b_n = |A||B| \cos \theta$
  - Cross product of vectors $A$ and $B$: $A \times B = [\hat{x} \times \hat{y} \times \hat{z}] = |A||B| \sin \theta$

Math Reminder

- The sign of the dot product:
  - $A \cdot B > 0$ for $A$ and $B$ pointing in the same direction.
  - $A \cdot B = 0$ for $A$ and $B$ pointing in perpendicular directions.
  - $A \cdot B < 0$ for $A$ and $B$ pointing in opposite directions.

- The sign of the cross product uses the right hand rule:
  - Fingers align with $A$, palm faces direction of $B$.
  - Positive non-zero values point in direction of thumb.
Phong Illumination Model

- Developed in 1975 by Phong Bui-Tuong
- Illumination model for non-perfect reflectors
- Phong model was the first to account for viewers and lights at arbitrary positions
- If $W(\theta)$ is the fraction of specularly reflected light, Phong's model is
  \[ I_a = I_a k_a + f_{att} I_p [k_d (N \cdot L) + k_s (R \cdot V)^n] \]

Phong Illumination Model

- If we assume that all vectors are normalized, $\cos \alpha$ is the dot-product $R \cdot V$
- $W(\theta)$ is typically set to $k_s$, the specular reflection coefficient
- Resulting model:
  \[ I_a = I_a k_a + f_{att} I_p [k_d (N \cdot L) + k_s (R \cdot V)^n] \]

Phong Lighting

- Lighting with different values for $L$ and $n$

Halfway Vector

- An alternative to Phong's model
- $H$ is the halfway vector
- Also known as the direction of maximum highlights
- This term can be expressed as $(N \cdot H)$, where
  \[ H = \frac{L + V}{|L + V|} \]

- When the viewer and the light source are both at infinity, this offers computational simplicity

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
- OpenGL maintains a current normal, which is used to compute color at vertices
- Setting the current normal:
  \[
  \text{glNormal3[bdfis]}( x, y, z );
  \text{glNormal3[bdfis]}v( *v );
  \]
- Use unit normals for proper lighting
  - Scaling affects a normal's length
    \[
    \text{glEnable( GL_NORMALIZE ) ;}
    \text{glEnable( GL_RESCALE_NORMAL ) ;}
    \]
Determining the Normal

- Given a plane with points \( p_0, p_1, p_2 \)
- Points may not be collinear
- Recall: \( U \times V \) orthogonal to \( U \) and \( V \)
- \( n_0 = (p_1 - p_0) \times (p_2 - p_0) \)
- Remember that \( U \times V \) is NOT equal to \( V \times U \)!
  - Order of cross product determines orientation
- Normalize to \( n = n_0/|n_0| \)

Normal of a Sphere

- Implicit Equation \( f(x, y, z) = x^2 + y^2 + z^2 - 1 = 0 \)
- Vector form: \( f(p) = p \cdot p - 1 = 0 \)
- Normal given by gradient vector:
  - Take partial derivative with respect to \( x, y, \) and \( z \) and what do you get?
  - Normalize \( n_0/|n_0| = 2p/2 = p \)

To Get the Normal of a Shared Vertex

- Average the normal vectors of all polygons in the surface mesh that share that vertex
  - \( n_0 = \frac{\sum n_j}{|\sum n_j|} \)
- For shared vertices:
  \[ N = \frac{\sum a_i N_j}{\sum a_i} \]
  where \( a_i \) is the inverse \( \cos \) of the dot product of two edges that meet at the vertex

Lights in OpenGL

- Two possible light sources:
  - Local (point) light sources
  - Infinite (directional) light sources
- Type of light controlled by \( w \) coordinate in \( (x, y, z, w) \):
  - \( w = 0 \): infinite light, directed along \( (x, y, z) \)
  - \( w \neq 0 \): local light, positioned at \( (x/w, y/w, z/w) \)
- A light positioned at the origin of the world:
  \[ \text{float positionalLight[]} = \{ 0.0, 0.0, 0.0, 1.0 \}; \]
- A directional light moving towards the \( +z \) axis:
  \[ \text{float directionalLight[]} = \{ 0.0, 0.0, 1.0, 0.0 \}; \]

Lighting Variables

- Lights values are specified in terms of R, G, B and A components
  - Floating point values
  - Range is 0.0 (no intensity) to 1.0 (full intensity)
- Global light reflecting off everything will be blue-ish:
  \[ \text{float ambientLight[]} = \{ 0.3, 0.5, 0.8, 1.0 \}; \]
- The surface that diffuse light hits will be lighter than areas which it barely hits (which will be dark)
  \[ \text{float diffuseLight[]} = \{ 0.25, 0.25, 0.25, 1.0 \}; \]
Light Properties in OpenGL

- Position or direction
- Color
- How it is attenuated (diminished) over distance
- Omni-directional (default) or spotlight
  - Direction (3D vector)
  - Cutoff (0 to 90)
  - Dropoff exponent

Setting Light Properties

gllight[fi]( light, property, value );
gllight[fi]v( light, property, *value );

- light specifies which light
  - GL_LIGHT0, etc.
- property is a light source parameter
  - Color
  - Position and type
  - Attenuation
- value is the desired setting
  - May be a single value (e.g., attenuation factor) or a vector
    (e.g., intensities for RGBA)

Parameter Settings for Lights

- Some parameter names and defaults:

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
<th>Default(s)/</th>
</tr>
</thead>
<tbody>
<tr>
<td>GL_AMBIENT</td>
<td>ambient intensity, RGBA</td>
<td>(0.0, 0.1)</td>
</tr>
<tr>
<td>GL_DIFFUSE</td>
<td>diffuse intensity, RGBA</td>
<td>(1.1, 1.1)</td>
</tr>
<tr>
<td>GL_SPECULAR</td>
<td>specular intensity, RGBA</td>
<td>(1.1, 1.1)</td>
</tr>
<tr>
<td>GL_POSITION</td>
<td>light position, world coordinates</td>
<td>(0.0, 1.0)</td>
</tr>
<tr>
<td>GL_SPOT_DIRECTION</td>
<td>eye coordinates</td>
<td>(0.0, 1.0)</td>
</tr>
<tr>
<td>GL_SPOT_EXPONENT</td>
<td>intensity distribution (0.128)</td>
<td>0</td>
</tr>
<tr>
<td>GL_SPOT_CUTOFF</td>
<td>maximum spread angle (0.90)</td>
<td>180</td>
</tr>
<tr>
<td>GL_CONSTANT_ATTENUATION</td>
<td></td>
<td>1</td>
</tr>
<tr>
<td>GL_LINEAR_ATTENUATION</td>
<td></td>
<td>0</td>
</tr>
<tr>
<td>GL_QUADRATIC_ATTENUATION</td>
<td></td>
<td>0</td>
</tr>
</tbody>
</table>

Controlling a Light’s Position

- 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

Lighting Models

- OpenGL’s lighting model lets you define 4 components
  - The ambient light intensity of the scene
  - The location of the viewport (local or infinite)
  - Affects specular reflection angle calculation
  - One-sided or two-sided lighting
  - Whether specular color is separate from ambient and diffuse

Light Model Properties

gllightModel[fi]( property, value );
gllightModel[fi]v( property, *value );

- GL_LIGHT_MODEL_TWO_SIDE
  - Specifies whether one- (0, front only, default) or two-sided lighting (non-zero) calculations are done for polygons
- GL_LIGHT_MODEL_AMBIENT
  - Ambient RGBA intensity of the entire scene
- GL_LIGHT_MODEL_LOCAL_VIEWER
  - How specular reflection angles are computed
  - 0 (default): view direction parallel to & in the direction of the -z axis
  - Other: from the origin of the eye coordinate system
- GL_LIGHT_MODEL_COLOR_CONTROL
  - Specular color calculated separately from ambient and diffuse
Lighting Models

• To specify the amount of global ambient light:
  ```c
  // medium light
  float ambientLightModel[] = { 0.5, 0.5, 0.5, 1.0 };
  glLightModelfv( GL_LIGHT_MODEL_AMBIENT, ambientLightModel );
  ```

• To specify the location of the viewpoint:
  ```c
  // local viewport
  glLightModeli( GL_LIGHT_LOCAL_VIEWER, GL_TRUE );
  ```

Putting it All Together

```c
glEnable(GL_LIGHTING);

// set up materials
glLightfv( GL_LIGHT0, GL_AMBIENT, ambientLight );
glLightfv( GL_LIGHT0, GL_DIFFUSE, diffuseLight );
glLightfv( GL_LIGHT0, GL_POSITION, lightPosition );

.glEnable(GL_LIGHT0);

// draw scene (specify normals in step using
  // glNormal3f() and its corresponding vertices)
```

• By default there is no ambient light
• Default RGBA for GL_DIFFUSE for GL_LIGHT0:
  `(1.0, 1.0, 1.0, 1.0)`
• Default RGBA for GL_DIFFUSE for all other lights:
  `(0.0, 0.0, 0.0, 0.0)`

Specular Highlight

• The bright reflection seen on an object under a specially-directed light
• GL_SPECULAR determines the color of this highlight
  - Often this is specified with the same values as GL_DIFFUSE
• Defaults for GL_SPECULAR
  - GL_LIGHT0: `(1.0, 1.0, 1.0, 1.0)`
  - Others: `(0.0, 0.0, 0.0, 0.0)`

```c
float specularLight[] = { 1.0, 1.0, 1.0, 1.0 };
float lightPosition[] = { 0.0, 0.0, 0.0, 0.0 };

glLightFv( GL_LIGHT0, GL_SPECULAR, specularLight );
glLightFv( GL_LIGHT0, GL_POSITION, lightPosition );

// define specular material properties
```

Attenuation

• Intensity of light decreases as you get further away from the origin of the light
  - e.g., a street lamp at night in the fog
• Only affects positional light sources
• OpenGL supports three attenuation factors:

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>GL_CONSTANT_ATTENUATION</td>
<td>1.0</td>
</tr>
<tr>
<td>GL_LINEAR_ATTENUATION</td>
<td>0.0</td>
</tr>
<tr>
<td>GL_QUADRATIC_ATTENUATION</td>
<td>0.0</td>
</tr>
</tbody>
</table>

Spotlights

• Reduce the radiance of the positional light from all directions to a specific direction
• Must specify 3 additional parameters to a normal positional light:
  - Spotlight cutoff - how wide is the cone of light in the direction it is pointing?

```c
  // a 30 degree light cone
  glLightf( GL_LIGHT0, GL_SPOT_CUTOFF, 15.0 )
```
Spotlights

- **Spotlight direction** - direction the spotlight is facing, as a vector \((x, y, z)\)
  - The default is \((0.0, 0.0, -1.0)\), down the \(-z\) axis
  ```c
  // point light down the -y axis
  float spotLightDirection[] = { 0.0, -1.0, 0.0 };
  glLightfv(GL_LIGHT0, GL_SPOT_DIRECTION, spotLightDirection);
  ```

- **Spotlight focus** - concentration of the spotlight in the center of the light cone
  - As you move away from the center of the cone, the light is attenuated at a certain ratio
  - GL_SPOT_EXPONENT sets the amount of concentration
  - Larger values \(\rightarrow\) more focused light source
  ```c
  glLightfv(GL_LIGHT0, GL_SPOT_EXPONENT, 10.0);
  ```

Controlling a Light’s Position

- The light’s position specified by `glLight*()` is manipulated by the current Modelview matrix
- Static lights are positioned before any transformation occurs
  - In OpenGL, this would be after the camera is specified
- Can apply a series of transformations on the light source position (vertices) to change the lights dynamically

Tips for Better Lighting

- Recall that lighting/illumination is computed only at vertices due to Gouraud shading model
  - 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