Overview

- Light and Illumination Models
  - Phong Model: Ambient, Diffuse and Specular Reflection
  - Shading: Flat, Gouraud, Phong
  - Lighting in OpenGL

Illumination

Illumination Models:
- Light is a very complex physical phenomenon.
- Most illumination models in graphics are based on simple geometric optics, as opposed to more complex (but realistic) wave optics.

Local Illumination Models:
- Point light sources and direct interactions with light.

Global Illumination Models:
- Point-, area-, or volumetric-light sources.
- Models object interactions with and other objects, indirect reflections.

Shading

Shading:
- Determining the intensity of illumination (and its color) incident at a surface point.
- Can be computationally intensive, so it is common to compute accurately at a few points (e.g., vertices) and interpolate in between.

Interpolation techniques:
- Incremental: Updating intensity from pixel to pixel.
- Order of interpolation: How accurate? (zero, linear, quadratic)
- What is interpolated: light intensity? surface normals?
- Surface Type: Curved or flat.
Local Illumination

Local Illumination Models:
- Easy to implement and very efficient.
- Provide an acceptable level of realism.
- Often augmented to enhance realism (e.g., draw your own shadows).

Elements of Local Illumination Models:
- Point light sources.
- Local surface geometry.
- No shadows.
- No inter-object reflections or refractions.
- No diffraction or polarization.

Light/Surface Interaction

Light/Surface Interaction:
- Light contacting a surface interacts with it in various ways.
- Light leaving the object (e.g., by reflection) is what we see.

Illumination Model:
- The job of an illumination model is to describe these interactions.
- Tradeoff: Greater realism vs. greater computational complexity.

Basic Interactions:
- Emission: The surface generates its own light, e.g., phosphorescence.
- Reflection: Light strikes the surface and bounces off.
- Absorption: Light strikes the surface and dissipates (as heat).
- Transmission: Light strikes the surface and passes through.

Reflection:
- Ambient: A background glow that illuminates all objects, irrespective of light source location. (A common cheat used in light local models.)
- Diffuse: A uniform scattering of light, characterized by matte (non-shiny) objects, like cloth or foam rubber.
- Specular: Shiny (metallic-like) reflection, like a polished wood table.
- Reflective (Pure): No light scattering, like a mirror.

Transmission:
- Transparent: No scattering, like smooth colored glass.
- Translucent: Some scattering, like tissue paper.

Perspective View

Image courtesy, Foley, van Dam, Feiner, Hughes

Depth Cueing

Image courtesy, Foley, van Dam, Feiner, Hughes

Colored Wireframe

Image courtesy, Foley, van Dam, Feiner, Hughes
Overview

- Light and Illumination Models
- Phong Model: Ambient, Diffuse and Specular Reflection
- Shading: Flat, Gouraud, Phong.
- Lighting in OpenGL

Ambient Illumination

Ambient Intensity: (OpenGL model) OpenGL combines \( C \) with \( k_a \), allowing the user to specify an ambient object color:

- \( L_a \) is the intensity of ambient light source (RGB).
- \( C_o \) is object's ambient color (RGB).
- \( I_a \) is the intensity of ambient reflection (RGB).

\[ I_a = k_a \cdot L_a \cdot C_o \]

Example: \( L_a = (1, 0, 1) \), \( C_o = (0.7, 1, 0) \), then \( I_a = (0.7, 0, 0) \).

We will present the OpenGL model throughout.
Diffuse Illumination

- Also known as Lambertian Illumination.
- Simulates non-shiny surface reflection.
- Based on surface normal and light direction.
- Independent of viewer direction.
- Light is scattered uniformly in all directions.

Lambert’s Cosine Law:
- Reflected luminous intensity varies as the cosine of the angle between the direction of incidence and the local surface normal.
- For a differential area dA, the amount of incident radiation is dA \( \cos \theta \).

Diffuse Intensity: Repeated for each light source.
- \( L_d \) is the intensity of diffuse light emission (RGB).
- \( C_d \) is object’s diffuse color (RGB).
- \( n \) is the normalized (unit length) surface normal vector.
- \( \ell \) (letter “ell”) is the normalized vector pointing to the light source. \( (n \cdot \ell) \) is the cosine of the light incidence angle.
- \( I_d \) is the intensity of diffuse reflection (RGB).

\[
I_d = \max(0, n \cdot \ell) \cdot L_d \cdot C_d.
\]

Specular Illumination

- Also known as Phong Illumination.
- Used to simulate metallic, shiny surfaces.
- Needs to take into account relative positions of light and viewer.

Specularity brightest when viewer is aligned with reflection vector.

- Let \( h \) be the vector halfway between the view and light vectors.
- Specularity is high if the normal vector \( n \) and \( h \) are close.
- \( v \) is the normalized view vector from the surface to viewer.
- \( \alpha \) is a parameter that controls specular spreading:
  - Low \( \alpha \) produces a diffuse specular reflection.
  - High \( \alpha \) produces a concentrated specular reflection.
- \( I_s \) is the intensity of specular reflection (RGB).

\[
I_s = \max(0, n \cdot h) \cdot L_s \cdot C_s.
\]

Examples of parameter settings:
- AmbientStrength = \( C_a \)
- DiffuseStrength = \( C_d \)
- SpecularStrength = \( C_s \)
- SpecularExponent = \( \alpha \).
Attenuation

Attenuation:
- From physics we know that light intensity decreases as the inverse square of the distance to the light source.
- In practice, light sources are not pure points, and so this does not produce the most realistic results.
- OpenGL allows for attenuation to include constant, linear, and quadratic terms.

Attenuation Factor:
Let \( d \) be the distance from the surface point to the light source.
Let \( a, b, c \) be parameters set by the user.
Attenuation Factor: (multiplied by diffuse and specular terms)
\[
\frac{1}{a + bd + cd^2}
\]

Final Illumination Equation

Final Illumination Equation:
- Let \( C_e \) denote the color of emitted light from surface (RGB).
- Assume that there are \( m \) light sources.
- Let \( I_i \) denote the final intensity for a surface point.

\[
I_i = C_e + \sum_{j=1}^{m} \left( \frac{1}{a + bd + cd^2} \right) \left( \max(0, n \cdot l_i) \cdot I_{ij} C_i + \max(0, n \cdot h_i) \cdot I_{ih} C_i \right)
\]

Remember: This is really 3 equations, for each color component: red, green and blue.

Overview

- Light and Illumination Models.
- Phong Model: Ambient, Diffuse and Specular Reflection.
- Shading: Flat, Gouraud, Phong.
- Lighting in OpenGL.

Illumination vs. Shading

Illumination model: How light and surface properties affect illumination.
Shading: How pixels are colored (usually a function of lighting model).

- Flat shading: One shade for entire polygon.
- Gouraud shading: Interpolate colors.
- Phong shading: Interpolate surface normals.

Flat Shading

Flat Shading: Compute the color of a polygon, and use that same color on every pixel of the polygon.
- Fast and simple.
- One normal per polygon, constant color per polygon.
- Best for flat surfaces to give a faceted appearance.

In OpenGL:

\[glShadeModel(GL_FLAT)\]

Gouraud Shading

Gouraud Shading: (after Henri Gouraud, 1971)
- Compute the color of each vertex.
- Use linear interpolation to color each interior pixel.
- Produces tolerably good results for curved surfaces, provided that mesh elements are small.

In OpenGL:

\[glShadeModel(GL_SMOOTH)\]

Observations:
- Works quite well for diffuse shading where shades vary slowly and if polygon mesh elements are small.
- Problems with specular reflection: If no vertex hits the specular region, we see no specular contribution when we interpolate.
Diffuse Illumination & Flat Shading

Note variation in appearance of facets between flat shading and Gouraud.

Image courtesy: Foley, van Dam, Feiner, Hughes

Chapter 5, Slide 31

Copyright © D. M. Mount and A. Varshney

Image courtesy, Foley, van Dam, Feiner, Hughes

Diffuse Illumination and Gouraud Shading

Note variation in appearance of facets between flat shading and Gouraud.

Chapter 5, Slide 32

Copyright © D. M. Mount and A. Varshney

Gouraud Shading: Mach Bands

Named after Austrian-Czech physicist and philosopher Ernst Mach (1838-1916).

Mach bands:
- Optical illusion resulting from the human visual system's tendency to enhance variations in shade.
- Acts like a high-pass filter, accentuating discontinuities in the first derivative.
- Caused by interaction of neighboring retinal neurons.
- A well-known artifact with both flat and Gouraud shading. Linear interpolation causes first derivative discontinuities at polygon edges, and our visual system enhances them.

Image courtesy: Univ. of North Carolina

Chapter 5, Slide 33

Copyright © D. M. Mount and A. Varshney

Gouraud Shading: More Issues

Interpolation with Perspective:
- Linear interpolation among pixels ignores the fact that more distant pixels cover greater depth range.
- Can fix using perspective-sensitive nonlinear interpolation.

Solutions?
- Finer mesh size minimizes the problem.
- But suffers from higher performance cost.
- Gouraud suggested higher-order interpolation:
  - But suffers from higher performance cost.
  - Probably not worth it.
- Use a better shading model:
  - Phong shading (we'll present next).
  - Perform shading computations on a per-pixel basis (modern GPUs).

Chapter 5, Slide 34

Copyright © D. M. Mount and A. Varshney

Phong Shading

Phong Shading:
- Compute the normal for each vertex. (Either supplied by the user or computed automatically from normals of incident faces.)
- For each pixel, interpolate normal vector from the normals at the vertices. (Note: Need to normalize vectors to unit length, because linear interpolation of vectors does not preserve lengths.)
- Compute the color of the pixel (e.g., by Phong illumination model).

Interpolation Methods:
- Linear Interpolation (Lerp): Given two vectors $v_1$ and $v_2$.
  $$\text{Lerp}(v_1, v_2, t) = (1-t)v_1 + tv_2 \text{ for } 0 \leq t \leq 1.$$  
- Spherical Linear Interpolation (Slerp): Interpolates cyclically between two vectors $v_1$ and $v_2$, where $\theta$ is the angle between them:
  $$\text{Slerp}(v_1, v_2, t) = \frac{\sin(t \theta)}{\sin(\theta)} v_1 + \frac{\sin(t \theta)}{\sin(\theta)} v_2 \text{ for } 0 \leq t \leq 1.$$  

Performance Issues:
- Computationally more intensive than Gouraud shading.
- Produces much better results, especially for curved surfaces when specular reflection is involved.
- Reduces Mach banding effects, which are evident in Gouraud.

Note:
- Do not confuse Phong illumination with Phong shading.

Chapter 5, Slide 35

Copyright © D. M. Mount and A. Varshney

Chapter 5, Slide 36

Copyright © D. M. Mount and A. Varshney

6
Specular Illumination and Gouraud Shading

Note variation in specular variations between Gouraud and Phong.

Phong + Specular + Curved Surfaces

Note smoothness of object boundaries.

Specular Illumination and Phong Shading

Note variation in specular variations between Gouraud and Phong.

Overview

- Light and Illumination Models.
- Phong Model: Ambient, Diffuse and Specular Reflection.
- Shading: Flat, Gouraud, Phong.
- Lighting in OpenGL

Lighting in OpenGL

There is a bewildering number of options and parameters that need to be set up in OpenGL in order to use lighting:

- Lighting/Shading model: Global parameters that affect how illumination and shading are computed.
- Light properties: Options that define the location, colors, and intensities of the lights.
- Object material properties: The color of the object and degree of ambient, diffuse, specular reflection.
- Enabling: Lighting can be enabled or disabled.

Lighting in OpenGL:

- Ambient, diffuse, specular illuminations are supported.
- Attenuation is supported.
- Users define light sources: position, type, color.
- The concept of color (for unlit scenes) is replaced with object material for each object drawn.
- Front and back sides of polygons may be given different colors.
- OpenGL support for Phong Shading?
  - Not OpenGL does not do Phong shading.
  - Most graphics APIs (D3D, OGL) employ some variation of Phong illumination and Gouraud shading.
To determine the supported OpenGL needs

The \([x, y, z, w]\) vector is given in

When to set light position

– Lights at infinity

– Light positions should be set (turn on) each light that you plan to use.

– You need to enable (turn on) each light that you plan to use.

– Other lighting settings need be done only once.

The light position is given as a homogenous \([x, y, z, w]\) vector.

– OpenGL supports at least \(8\) light sources: \(GL\_LIGHT0, ..., GL\_LIGHT7\).

– \(gl\_Light\{if\}v\) \((\text{GLenum \(light\), \(GLenum \(pname\), \(TYPE\) \(param\)}\))

where:

\((\text{light})\) can be: \(GL\_LIGHT0, ..., GL\_LIGHT7\).

\((\text{pname})\) can be:

\(GL\_POSITION\): Light position.

\(GL\_AMBIENT, GL\_DIFFUSE, GL\_SPECULAR\): Light colors (RGBA vector)

\(GL\_SPOT\_DIRECTION, GL\_SPOT\_EXPOSURE, GL\_SPOT\_CUTOFF\): Spotlight parameters.

\(GL\_CONSTANT\_ATTENUATION, GL\_LINEAR\_ATTENUATION, GL\_QUADRATIC\_ATTENUATION\): Parameters for attenuation (scalar)

Sample Lighting Setup

```cpp
void setUpMyLighting ( ) {
   // global lighting options
   glEnable ( GL_LIGHT0 );
   glEnable ( GL_LIGHT1 );
   glEnable ( GL_LIGHT2 );

   // light intensity and location
   GLfloat position [4] = { 2.0, 4.0, 5.0, 1.0 };
   glEnable ( GL_LIGHT0 );
   glLightfv ( GL_LIGHT0,   GL_POSITION,  position );
   glLightfv ( GL_LIGHT1,   GL_POSITION,  { 5.0, 3.0, 1.0, 1.0 } );
   glLightfv ( GL_LIGHT2,   GL_POSITION,  { 5.0, -3.0, 3.0, 1.0 } );

   // light colors
   GLfloat diffSpecIntensity [4] = { 1.2, 1.2, 1.2, 1.0 }; // white
   glEnable ( GL_LIGHT0 );
   glLightfv ( GL_LIGHT0,   GL_DIFFUSE,   { 1.0, 0.0, 0.0, 1.0 } );
   glLightfv ( GL_LIGHT0,   GL_SPECULAR,  { 1.0, 0.0, 0.0, 1.0 } );
   glLightfv ( GL_LIGHT0,   GL_AMBIENT,   ambientIntensity );

   // material colors
   GLfloat ambientIntensity [4] = { 0.9, 0.0, 0.0, 1.0 }; // red
   glEnable ( GL_LIGHT0 );
   glMaterialfv ( GL_FRONT, GL_AMBIENT_AND_DIFFUSE, ambientIntensity );
   glMaterialfv ( GL_FRONT, GL_SPECULAR, specularIntensity );
   glMaterialfv ( GL_FRONT, GL_SHININESS, shininess );

   // enable lighting
   glShadeModel ( GL_SMOOTH ); // (or GL_FLAT)
   glEnable ( GL_LIGHTING );

   // set up light 0 properties
   glLightfv ( GL_LIGHT0,   GL_AMBIENT,   { 0.5, 0.5, 0.5, 0.0 } );
   glLightfv ( GL_LIGHT0,   GL_DIFFUSE,   { 1.0, 0.0, 0.0, 1.0 } );
   glLightfv ( GL_LIGHT0,   GL_SPECULAR,  { 0.0, 0.0, 1.0, 1.0 } );
   glLightfv ( GL_LIGHT0,   GL_SHININESS, shininess );

   // set up light 1 properties
   glLightfv ( GL_LIGHT1,   GL_AMBIENT,   { 0.5, 0.5, 0.5, 0.0 } );
   glLightfv ( GL_LIGHT1,   GL_DIFFUSE,   { 1.0, 0.0, 0.0, 1.0 } );
   glLightfv ( GL_LIGHT1,   GL_SPECULAR,  { 0.0, 0.0, 1.0, 1.0 } );
   glLightfv ( GL_LIGHT1,   GL_SHININESS, shininess );

   // set up light 2 properties
   glLightfv ( GL_LIGHT2,   GL_AMBIENT,   { 0.5, 0.5, 0.5, 0.0 } );
   glLightfv ( GL_LIGHT2,   GL_DIFFUSE,   { 1.0, 0.0, 0.0, 1.0 } );
   glLightfv ( GL_LIGHT2,   GL_SPECULAR,  { 0.0, 0.0, 1.0, 1.0 } );
   glLightfv ( GL_LIGHT2,   GL_SHININESS, shininess );
}
```

Drawing Objects with Lighting

\(gl\_Material\{if\}v\) \((\text{GLenum \(face\), \(GLenum \(pname\), \(TYPE\) \(param\)}\))

where:

\((\text{face})\) can be: \(GL\_FRONT, GL\_BACK, GL\_FRONT\_AND\_BACK\)

Indicates which side of the polygon is being colored. (Recall that front face is the side from which vertices are listed in CCW order.)

\((\text{pname})\) can be:

\(GL\_AMBIENT, GL\_DIFFUSE, GL\_SPECULAR, GL\_EMISSION\): Material colors (RGBA vectors). (You can set both ambient and diffuse at once using \(GL\_AMBIENT\_AND\_DIFFUSE\).)

\(GL\_SHININESS\): Specular illumination exponent (called \(\alpha\) above).
void doMyDrawing ( ) {
    GLfloat red [4] = {1.0, 0.0, 0.0, 1.0};// RGBA object color (red)
    glMaterialfv ( GL_FRONT_AND_BACK,
                GL_AMBIENT_AND_DIFFUSE, red );
    glBegin ( GL_PLYGON );  // draw polygon
    glNormal3f ( … );  glVertex3f ( … );
    glNormal3f ( … );  glVertex3f ( … );
    glNormal3f ( … );  glVertex3f ( … );
    glEnd ( );
}