Chapter 6
Shading

Objectives

• Use shading to make objects appear real and three-dimensional
• In order to do so, we must understand
  – Light-Material Interactions
  – Simple Reflection Model

Light-Material Interactions

• A surface can emit light because
  – self-emission, or
  – it reflects light from other surfaces
• When two surfaces are in view of each other, lights striking them are reflected and absorbed back and forth
  – This mutual reflection is recursive and can be modeled mathematically by a rendering equation

Surface – Surface Interaction

Light-Surface Interaction

Rendering Equation

• Models the infinite reflection and absorption of light by surfaces
• Considered as a global model
  – taking into account all intra-surface interactions
  – include the effect of shadows
• The equation may not be solvable
• Approximate with a simpler local model

Light Source

• [Ideal] Point Source
  – Intensity falls on a surface is inversely proportional to the squared distance between the surface and the light source
• Ambient Source (“Bounced Light”)
  – Same intensity of light everywhere
  – Scattered by the environment
• [Distributed Source]
Light Source: Spotlight

- Can be approximated by an ideal point source
  - Ideal point source located at $p_s$
  - Cone with apex at $p_s$, direction of cone axis $I_s$
  - Half width of cone $\theta$
  - Attenuation function $(s \cdot I)^s$, $e$ controls the rate of drop off.
    - $s =$ vector to the illuminated point

Local Illumination Model

- Considers the interaction of a light ray with only one surface
  - Inter-surface interaction is ignored
- A light ray that hits a surface is
  - partially absorbed and
  - partially reflected
The amount of reflected light determines the color and brightness of the surface

Light Reflection by a Surface: Diffuse & Specular

- The reflected light is scattered in a manner that depends on smoothness and orientation of the surface

Types of Illumination

- Ambient: comes from unknown direction
- Diffuse: comes from one direction, scattered in all direction by the surface
- Specular: comes from one direction, tends to bounce off a particular direction
- Emissive: comes from the surface itself

Local Reflection: Phong Model

- Four (unit) vectors:
  - to light source ($l$)
  - normal ($n$)
  - to the viewer ($v$)
  - ideal reflection ($r$)
- Three shading / reflection components
  - ambient reflection
  - diffuse reflection: function of $l$ and $n$
  - specular reflection: a function of $r$ and $v$

Light-Material Interaction in Phong Model

Brightness / darkness of a surface is determined by
- Intensity of the light source
  - Upper case $L$ in Phong equation
- The amount of light falls on the surface
- The amount of light reflected by the surface
  - Lowercase $k$ in Phong equation
Ideal Reflection Vector

- $r$, $l$, and $n$ are coplanar and unit vectors
- Angle of incident = angle of reflection
- Given $l$ and $n$, how to calculate $r$?

$$l + r = zn$$

dot product of above equation by $n$

$$l \cdot n + r \cdot n = zn \cdot n$$

but $l \cdot n = r \cdot n$

hence $2l \cdot n = zn$

$$r = 2(l \cdot n)n - l$$

Ambient Reflection

- Intensity of ambient light is the same everywhere in space
- The amount of ambient reflection depends solely on the ambient reflection coefficient
- Ambient reflection is:

$$I_a = k_a L_a$$

$k_a$ = ambient reflection coefficient

$L_a$ = intensity of ambient light source

Diffuse Reflection

- Incident light rays are scattered equally in all directions by the surface
- Amount of reflected light is proportional to the “density” of incoming light
  - Analogy: density of flashlight on a flat surface
  - Density depends on the angle of incident:

$$I_d = k_d L_a (l \cdot n)$$

Specular Reflection

- Reflected light rays are concentrated around the direction of the ideal reflection vector ($r$)
- Specular reflection produces highlight
  - Different viewer may see different highlight
  - Shinier = narrow area of highlight
- Specular reflection depends on $r$ and $v$:

$$I_s = k_s L_s (r \cdot v)^h$$

$h$ is shininess coefficient

OpenGL: 0 $<$ h $<$ 128

Phong Reflection Model

- Reflected light = ambient + diffuse + specular

$$I = I_a + k_d L_d (l \cdot n) + k_s L_s (r \cdot v)^h$$

- Attenuation by distance

$$I = I_a + \frac{1}{k_d + k_s d + k_s d^2} \left( \frac{1}{k_d} \frac{L_d (l \cdot n)}{d} + \frac{1}{k_s} \frac{L_s (r \cdot v)^h}{d^2} \right)$$

$d$ = distance between the point on the surface and the light source

Guidelines for “Realistic” Lighting

- $L_a$ should be the same for all light sources
- $L_d$ and $L_s$ of each light source should be the same
  - These values determine the color of the light source
- $k_d$, $k_s$: specify the color of the surface
  - These two parameters typically have similar tone
  - $k_d$ determines the perceived color
Guidelines for “Realistic” Lighting

- \( k_s \) is a brighter color than \( k_a \) or \( k_d \)
  - Use color wheels
- **Beware of possibility of washout!** Shading color values are clamped at 1.0

Guidelines for “Realistic” Lighting

- Set attenuation factors (constant, linear, quadratic)
  
  ```
  glLightf (GL_LIGHTX, parm, value);
  ```
  
  **parm:**
  - GL_CONSTANT_ATTENUATION \( (k_c) \)
  - GL_LINEAR_ATTENUATION \( (k_l) \)
  - GL_QUADRATIC_ATTENUATION \( (k_q) \)

Using Phong Reflection Model (1)

The following parameters (13 values) must be specified for each surface:

- \( k_a, k_d, k_s \): ambient, diffuse, specular reflectance coefficient (for R-, G-, B-channel)
- shininess coefficient \( (h) \)
- normal vector \( (n) \)

Using Phong Reflection Model (2)

The following parameters (≥ 15 values) must be specified for each point light source:

- \( L_a, L_d, L_s \): ambient, diffuse, specular intensity of light source (for R-, G-, B-channel)
- \( k_c, k_l, k_q \): attenuation coefficients
- Position of light source
- Additional parameters for spotlight

Modified Phong Model: Blinn-Phong Model

- Phong model has to recalculate \( r \cdot v \) as we move to a different point on the same surface
- To avoid expensive calculation of \( r \) (and \( v \)), Blinn uses the halfway vector \( h \) (halfway between \( I \) and \( v \))
  - \( r \cdot v \) in Phong model is replaced by \( h \cdot n \)
  - recalculation of \( h \) is still needed
- Blinn-Phong model is used by OpenGL

Utah Teapots
5-Step OpenGL Lighting

• Define normal vector for each surface of every object
• Create and position one or more light sources
• Select a lighting model
• Define material properties (reflection coefficients) for the objects in the scene
• Enable Lighting

Step 1: Defining Normal in OpenGL

• Set the normal vector
  – glNormal3f(b, d, f, i, s) (val1, val2, val3);
  – glNormal3f(b, d, f, i, s) (array);
• Normal vectors should be unit vectors, but glScale() may not preserve normal vectors. For correct reflection calculation:
  – glEnable(GL_NORMALIZE); /* tell GL to renormalize normal vectors */

Step 1: Example

/* perform some calculation of nx, ny, nz */
nx = ...; ny = ...; nz = ...;
glNormal3f(nx, ny, nz);
glVertex3f(1., 2., 3.); /* Point A */
glVertex3f(1., 2., 3.); /* Point B */

The normal vector is assigned to both A and B.

Step 2: Create & Position Light

• glLightf(GL_LIGHTn, paramname, val);
• glLightfv(GL_LIGHTn, paramname, array);
• To set various properties of light source
  – Ambient, diffuse, specular color of light source (L_a, L_d, and L_s in the equation)
  – Position, spot light direction, attenuation factors (k, k_i, k_s in the equation)
  – See detail of parameters in manual page!

Step 2: Setting Light Color

• Using local variables for light pos / color may create unwanted effect. Use global or static local

  static GLfloat amb_color[] = {0.4, 0.4, 0.4, 1.0};
  static GLfloat light_color[] = {1.0, 1.0, 1.0, 1.0};

  glLightf(GL_LIGHT0, GL_AMBIENT, amb_color);
  /* keep diffuse and specular color the same */
  glLightf(GL_LIGHT0, GL_DIFFUSE, light_color);
  glLightf(GL_LIGHT0, GL_SPECULAR, light_color);

Step 2: Setting Light Position

• Positional Light
  static GLfloat light_pos[] = {20, 1.0, 2.0, 1.0};
  glLightfv(GL_LIGHT0, GL_POSITION, light_pos);

• Directional Light
  static GLfloat light_dir[] = {20, 1.0, 2.0, 0.0};
  glLightfv(GL_LIGHT0, GL_DIRECTION, light_dir);

  Directional light ≠ spot light direction!!
Step 3: Select Lighting Model

- `glLightModel(i,f) (paramname, val)`
- `glLightModel(i,f) v (paramname, array)`
- To specify:
  - Global ambient light
  - Local view (more computation) vs. infinite viewer (less computation)
  - One- or two-side lighting
  - See detail of parameters in manual page

Step 3: Example

// Assume local viewer
glLightModeli (GL_LIGHT_MODEL_LOCAL_VIEWER, GL_TRUE);

// Do lighting for front and back side of polygon
glLightModeli (GL_LIGHT_MODEL_TWO_SIDE, GL_TRUE);

Step 4: Define Material Property

- `glMateriali(f) (face, paramname, val)`
- `glMateriali(f) v (face, paramname, array)`
- For specifying the current material property:
  - ambient, diffuse, specular reflection coefficient of material \( k_a, k_d, k_s \) in the equation
  - shininess /specular exponent \( h \) in the equation
  - emissive color of material
  - See detail of parameters in manual page

Step 4: Example

/* keep ambient and diffuse color “the same” */
static GLfloat[] mat_color = {.3, .8, .13, 1.0};
static GLfloat[] mat_shine = {.6, .6, .6, 1.0};
glMaterialfv (GL_FRONT, GL_AMBIENT, mat_color);
glMaterialfv (GL_FRONT, GL_DIFFUSE, mat_color);
glMaterialfv (GL_FRONT, GL_SPECULAR, mat_shine);
glMaterialfv (GL_FRONT, GL_SHININESS, 15);

Step 5: Enable Lighting

- Remember to “switch the light on”
  - `glEnable (GL_LIGHT[i]);` /* n = 0-7 */
- Remember to “plug the cord to the outlet”
  - `glEnable (GL_LIGHTING);`

`glNormal()` and `glMaterial()`

- Like `glColor()`, `glNormal()` and `glMaterial()` are modal
  - Their values remain the same until changed in the program
  - They affect the property of surfaces specified after the changes
- See example on the web (SVN repo)
Controlling Light Position

- **Position** (and **direction**) of a light source is transformed by the current modelview matrix
  - Projection matrix has no effect
- `glRotate()`, `glTranslate()` affects light position
- Effect of transformations on light source
  - Stationary light (fixed in the world coordinate)
  - Light source attached to viewer

Light In World Coordinate

- Set the light position **after** the camera is placed
  - `glMatrixMode(GL_MODELVIEW);`  
  - `glLoadIdentity();`  
  - // **viewing transform here**
  - `gluLookat(...) or user defined transform`
  - // Lpos is in **WORLD** coordinate frame
  - `glLightfv(GL_LIGHT0, GL_POSITION, Lpos);`
  - // **modeling transform here**
  - `glCallList(object-to-show)`

Light Source Attached to Viewer

- Set the light position **before** the camera is placed
  - `glMatrixMode(GL_MODELVIEW);`  
  - `glLoadIdentity();`
  - // Lpos is in **CAMERA** coordinate frame
  - `glLightfv(GL_LIGHT0, GL_POSITION, Lpos);`
  - // **viewing transform here**
  - `gluLookat(...) or user-defined transform`
  - // **modeling transform**
  - `glCallList(object-to-show)`

Positional vs. Directional Light Source

- Using homogeneous coordinate changing a local light source to a distant one (and vice versa) is very easy
  - `(8, 2, 3, 1)` is a positional light at (8,2,3)
  - `(8, 2, 3, 0)` is a directional light source in the direction
  - \[
  \begin{bmatrix}
  8 \\
  2 \\
  3 \\
  \end{bmatrix}
  \]

Shading a Single Polygon

- In a polygon: normal vector $n$ is the same everywhere
- In Phong Reflection Model
  - vectors $l$, $v$ vary with location on the polygon
- Flat shading simplifies the Phong Reflection by assuming distant viewer & light source
  - vectors $l$, $v$ are constant across the polygon

Smooth vs. Flat Shading

- `glShadeModel(GL_SMOOTH);`
- `glShadeModel(GL_FLAT);`
- Colors (and normals) are specified at vertices
- With smooth shading vertex colors are interpolated across polygon
- With flat shading the color of the **first** vertex determines the color of the polygon
Calculation of Normal Vectors $\mathbf{n}$: Planar Surfaces

- From implicit equation: $ax + by + cz + d = 0$
  \[ \mathbf{n} = \begin{vmatrix} a \\ b \\ c \end{vmatrix} \]

- From parametric equation
  \[ Z(\alpha,\beta) = P + \alpha(Q-P) + \beta(R-P) \]
  take the cross product: $(Q-P) \times (R-P)$

Calculation of Normal Vectors $\mathbf{n}$: Curved Surfaces

- For implicit equation $f(x,y,z) = 0$, calculate the partial derivative
  \[ \mathbf{n} = \frac{\partial f}{\partial x} \mathbf{i} + \frac{\partial f}{\partial y} \mathbf{j} + \frac{\partial f}{\partial z} \mathbf{k} \]

- From parametric equation $x(u,v)$, $y(u,v)$, $z(u,v)$
  take the cross product of the two partial derivatives:

Shading a Polygonal Mesh

- A smooth curved surface is generally decomposed into a mesh of (flat) polygons
- Discontinuity in normal vectors of adjacent polygons produces discontinuous shading
- Two approaches for handling this situation
  - Gouraud shading
  - Phong shading

Phong vs Gouraud Shading

- Gouraud Shading (used by OpenGL)
  - Calculate average normal at each vertex
  - Apply Phong Reflection model at each vertex
  - Interpolate vertex shade across polygon
- Phong Shading
  - Calculate average normal at each vertex
  - Interpolate the vertex normals across the edges
  - Interpolate the edge normals across interior vertices
  - Apply Phong Reflection model at each vertex

Phong Shading vs. Phong Reflection Model

- Phong Reflection Model is an equation that describes the relationship between the intensity of a point on a surface and vectors $\mathbf{n}$, $\mathbf{l}$, $\mathbf{v}$, and $\mathbf{r}$
- Phong Shading is a technique that interpolates the shade of a vertex from other vertices

Interpolation

- Color (or normal) of $P$, $Q$, $R$, $S$, are given
- Linear Interpolation
  - $C_A = (1 - u)C_P + uC_Q$
  - $C_B = (1 - v)C_R + vC_S$
- Bilinear (linearly interpolated from linear interpolation)
  - $C_M = (1 - w)C_A + wC_B$

\[ 0 \leq u, v, w \leq 1 \]
Scan-Line Bilinear Interpolation

Undesired Effect of Shading by Spotlights

- Consider a spotlight source directly above a polygon
- Due to spotlight cutoff angle, some vertices of polygon may not get illuminated
- Interior of polygon is not correctly shaded
- To avoid this problem, break up the polygon into smaller patches

u,v,w can be calculated from x, y, or z-coordinate (only one of them) of the endpoints