Previous Class

- OpenGL basics
- Drawing geometric objects
- Viewing
- Color

- This Class:
  - Lighting
  - Texture Mapping
Lighting
Why Lighting?

- What light source is used and how the object response to the light makes difference
  
  Ocean looks bright bluish green in sunny day but dim gray green in cloudy day

- Lighting gives you a 3D view to an object
  
  A unit sphere looks no difference from a 2D disk

- To get realistic pictures, the color computation of pixels must include lighting calculations
Types of Light

- **Ambient:**
  Light that’s been scattered so much by the environment that its direction is impossible to determine – it seems to come from all directions

- **Diffuse**
  Light that comes from one direction, but it gets scattered equally in all directions

- **Specular**
  Light comes from a particular direction, and its tends to bounce off the surface in a preferred direction
Materials Colors

- A material’s color depends on the percentage of the incoming different lights it reflects

- Materials have different ambient, diffuse and specular reflectances

- Material can also have an emissive color which simulates light originating from an object
  
  Headlights on a automobile
OpenGL Lighting Model

- Lighting has four independent components that are computed independently:
  - Emission, Ambient, Diffuse, and Specular

- OpenGL approximates lighting as if light can be broken into red, green, and blue components:
  - The RGB values for lights mean different than for materials
    - For light, the numbers correspond to a percentage of full intensity for each color
    - For materials, the numbers correspond to the reflected proportions of those colors

- Total effect is a combination of corresponding components of incoming light and illuminated material surface:
  - \((LR \times MR, LG \times MG, LB \times MB)\)
Theory of Illumination

- Not only knowledge about light but also about what happens when light is reflected from an object into our eyes is important to obtain realistic images.
- The general problem is to compute the apparent color at each pixel that corresponds to part of the object on the screen.
- The color produced by lighting a vertex (or a object) has several contributions:
  - Emission
  - Global ambient light
  - Contributions from light sources
Material Emission

- Emissive brightness of the material = $M_e$

- There is no attempt to model properties of the light or its effects on the objects

- The emissive color adds intensity to the object

\[ I_e = M_e \]
Global Ambient Light

• Light from all directions but not from any specific sources

• Ambient light intensity $= G_a$

• Ambient reflection coefficient of material $= M_a$

$$I_G = G_a M_a$$
A Point Source of Light

- Contribution from a point source of light include three terms
  - Light has ambient ($I_a$), diffuse ($I_d$) and specular ($I_s$) components
  - Material has ambient ($M_a$), diffuse ($M_d$) and specular reflection ($M_s$) properties
Point Light’s Contribution

\[ I_L^1 = I_a M_a + I_d M_d \text{max}\{N \cdot L, 0\} + I_s M_s \text{max}\{R \cdot V, 0\}\]

- First term = ambient
- Second term = diffuse
- Third term = specular
Point Light’s Contribution

• Ambient term

  The ambient component of each incoming light source is combined with a material’s ambient reflectance

• Diffuse term

  Brightness is inversely proportional to the area of the object illuminated (dot produce of light vector and surface normal)

  greatest when N and L are parallel

  smallest when N and L are orthogonal

  In calculations, max \{N.L, 0\} is used to avoid negative values

• Specular term

  Brightness depends on the angle between reflection vector (R) and viewer vector (V), i.e., on direction of viewer

  The specular reflection exponent \( n \) is 1 for a slightly glossy surface and infinity for a perfect mirror
Attenuation

- Attenuation factor

  Light attenuates with distance from the source

  \[ f = \frac{1}{k_c + k_l d + k_q d^2} \]

  where \( d = \) distance between the light and object

  \( k_c = \) constant attenuation

  A light source does not give an infinite amount of light

  \( k_l = \) linear term

  The light source is not a point

  \( k_q = \) quadratic term

  Models the theoretical attenuation from a point source

  The intensity becomes

  \[ I_L^2 = f [I_a M_a + I_d M_d (\max\{N \cdot L, 0\}) + I_s M_s (\max\{R \cdot V, 0\})^n] \]
Spotlight Effect

- When the vertex lies inside the cone of illumination produced by spotlight, its contribution to the light intensity is

\[ s = (\max\{D \cdot L, 0\})^m \]

Where \( D \) gives the spotlight’s direction. The intensity is maximum in the center of cone and is attenuated toward the edge of the cone

\( s = 1 \) if the source is not spotlight

\( m \) is exponent determining the concentration of the light

The intensity of light source is

\[ I_L = f_s[I_a M_a + I_d M_d \max\{N \cdot L, 0\} + I_s M_s \max\{R \cdot V, 0\}]^n \]
Putting All Together

- Entire lighting calculation in RGB mode gives

\[
\text{Vertex color} = M_e + G_a M_a + \sum_{i=1}^{n-1} f_i s_i [I_a M_a + I_d M_d (\max\{N \cdot L, 0\}) + I_s M_s (\max\{R \cdot V, 0\})^n]_i
\]
Adding Lighting to the Scene

- Define normal vectors for each vertex of each object
- Create, select, and position one or more light sources
- Create and select a lighting model
- Define material properties for the objects in the scene
Creating Light Sources

- Properties of Light sources are color, position, and direction

```c
void glLight{if}(Glenum light, Glenum pname, TYPE param);
void glLight{if}v(Glenum light, Glenum pname, TYPE *param);
```

Creates the light specified by light that can be GL_LIGHT0, GL_LIGHT1,…or GL_LIGHT7

`pname` specifies the characteristics of the light being set

`param` indicates the values to which the `pname` characteristic

- `glEnable (GL_LIGHT0)`
Color for a Light Source

\[
\text{GLfloat }\text{light\_ambient}[] = \{0.0,0.0,0.0,1.0\};
\]
\[
\text{GLfloat }\text{light\_diffuse}[] = \{1.0,1.0,1.0,1.0\};
\]
\[
\text{GLfloat }\text{light\_specular}[] = \{1.0,1.0,1.0,1.0\};
\]

\[
\text{glLightfv}(\text{GL\_LIGHT0}, \text{GL\_AMBIENT}, \text{light\_ambient});
\]
\[
\text{glLightfv}(\text{GL\_LIGHT0}, \text{GL\_DIFFUSE}, \text{light\_diffuse});
\]
\[
\text{glLightfv}(\text{GL\_LIGHT0}, \text{GL\_SPECULAR}, \text{light\_specular});
\]
Position of Light Source

- **Positional light source**
  
  \((x, y, z)\) values specify the location of the light

  ```
  GLfloat light_position[] = {x, y, z, w};
  glLightfv(GL_LIGHT0, GL_POSITION, light_position);
  ```

- **Directional light source**
  
  \((x, y, z)\) values specify the direction of the light located at the infinity

  **No attenuation**

  ```
  GLfloat light_position[] = {x, y, z, 0};
  glLightfv(GL_LIGHT0, GL_POSITION, light_position);
  ```
Attenuation

- Attenuation factor for a positional light

\[ f = \frac{1}{k_c + k_l d + k_q d^2} \]

Needs to specify three coefficients

```c
    glLightf(GL_LIGHT0, GL_CONSTANT_ATTENUATION, 2.0);
    glLightf(GL_LIGHT0, GL_LINEAR_ATTENUATION, 1.0);
    glLightf(GL_LIGHT0, GL_QUADRATIC_ATTENUATION, 1.0);
```

- Ambient, diffuse, and specular contributions are all attenuated
Spotlights

- The shape of the light emitted is restricted to a cone

- `glLightf(GL_LIGHT0, GL_SPOT_CUTOFF, 45.0);`  
  The cutoff parameter is set to 45 degrees

- `GLfloat spot_direction[] = {-1.0, -1.0, 0.0};`  
  `glLightfv(GL_LIGHT0, GL_SPOT_DIRECTION, spot_direction);`  
  specifies the spotlight’s direction which determines the axis of the cone of light

- `glLightf(GL_LIGHT0, GL_SPOT_EXPONENT, 2.0);`  
  Controls how concentrated the light is
Multiple Lights

- You can define up to eight light sources
  Need to specify all the parameters defining the position and characteristics of the light

- OpenGL performs calculations to determine how much light each vertex gets from each source

- Increasing number of lights affects performance
Controlling a Light’s Position and Direction

- A light source is subject to the same matrix transformations as a geometric model.
  - Position or direction is transformed by the current modelview matrix and stored in eye coordinates.
- Keeping the light stationary.
  - Specify the light position after modelview transformations.
- Independently moving the light.
  - Set the light position after the modeling transformation that you want to apply for light.
- Moving the light together with the viewpoint.
  - Set the light position before the viewing transformation.
Setting a Lighting Model

- How to specify a lighting model

- `glLightModel{if}(GLenum pname, TYPE param);`  
  `glLightModel{if}v(GLenum pname, TYPE *param);`
  Sets properties of the lighting model
  
  *pname* defines the characteristic of the model being set
  
  *param* indicates the values to which the *pname* characteristic is set

- Needs to be enabled or disabled
  
  `glEnable(GL_LIGHTING);`
  `glDisable(GL_LIGHTING);`
Components of Lighting Model

- **Global ambient light**
  Ambient light from not any particular source
  ```
  GLfloat lmodel_ambient[] = {0.2, 0.2, 0.2, 1.0}
  glLightModelfv(GL_LIGHT_MODEL_AMBIENT, lmodel_ambient);
  ```

- **Local or Infinite viewpoint**
  Whether the viewpoint position is local to the scene or whether it should be considered to be an infinite distance away
  ```
  glLightModeli(GL_LIGHT_MODEL_LOCAL_VIEWER, GL_TRUE);
  ```
  Default is an infinite viewpoint

- **Two-sided lighting**
  Whether lighting calculations should be performed differently for both the front and back faces of objects
  ```
  glLightModeli(GL_LIGHT_MODEL_TWO_SIDE, GL_TRUE);
  ```
Defining Material Properties

- Specifying the ambient, diffuse, and specular colors, the shininess, and the color of any emitted light

- `void glMaterial{if}(GLenum face, GLenum pname, TYPE param);`
  - `void glMaterial{if}v(GLenum face, GLenum pname, TYPE *param);`

  Specifies a current material property for use in lighting calculations

  *Face* can be GL_FRONT, GL_BACK, or GL_FRONT_AND_BACK

  *Pname* identifies the particular material property being set

  *Param* defines the desired values for that property
Reflectances

- **Diffuse and ambient reflection**
  
  *Gives color*
  
  ```c
  GLfloat mat_amb_diff[] = {0.1, 0.5, 0.8, 1.0};
  glMaterialfv(GL_FRONT_AND_BACK, GL_AMBIENT_AND_DIFFUSE, mat_amb_diff);
  ```

- **Specular reflection**
  
  *Produces highlights*
  
  ```c
  GLfloat mat_specular[] = {1.0, 1.0, 1.0, 1.0};
  GLfloat low_shininess[] = {5.0};
  glMaterialfv(GL_FRONT, GL_SPECULAR, mat_specular);
  glMaterialfv(GL_FRONT, GL_SHININESS, low_shininess);
  ```

- **Emission**
  
  *Make an object glow (to simulate lamps and other light sources)*
  
  ```c
  GLfloat mat_emission[] = {0.3, 0.2, 0.2, 0.0};
  glMaterialfv(GL_FRONT, GL_EMISSION, mat_emission);
  ```
Changing Material Properties

- Different material properties for different vertices on the same object or different objects

- `glMaterialfv()` needs to be called repeatedly to set the material property that needs to be re-specified for each case

- `glColorMaterial(GLenum face, GLenum mode);`
  Specifies the property (properties) defined by `mode` of the selected material `face` (or faces) to track the value of the current color at all times
  Needs enabling
Example: A lit sphere

- 2D disk in the absence of lighting
- 3D sphere
- Shinning sphere
- Emissive sphere
- Moving light source
Texture Mapping
Drawing Pixel Data

Geometric Versus Pixel Data

- Rendering of geometric data (arrays of vertices) points, lines, polygons

- Rendering of pixel data (arrays of pixels)
  - Bitmaps
    - Characters in fonts
    - Array of 0s and 1s (1 = the pixel is affected)
    - Serves as drawing mask for overlying another image
  - Image data
    - A photograph that is scanned or an image calculated by some program in memory by pixels or an image first drawn and then read back pixel by pixel
    - Several pieces of data per pixel (R,G,B,A values)
    - Simply overwrites in the framebuffer
void **glRasterPos**(TYPE x, TYPE y, TYPE z, TYPE w);

Sets the current raster position where the next bitmap (or image) is to be drawn.

The raster position coordinates are subject to the modelview and projection transformations in the same way as the vertex Coordinates.

To specify the raster position directly in the screen coordinates, only 2D version of transformations need to be specified.
Drawing Bitmaps

- void **glBitmap**(GLsizei *width*, GLsizei *height*, GLfloat *xbo*, GLfloat *ybo*, GLfloat *xbi*, GLfloat *ybi*, const Glubyte **bitmap**);

  Draws the bitmap specified by *bitmap*
  *Width and height* define size of the bitmap
  Subscript *bo* means the origin of the bitmap
  Subscript *bi* means increment to the current raster position after the bitmap is rasterized
Manipulating Images

- `void glReadPixels(GLint x, GLint y, GLsizei width, GLsizei height, GLenum format, GLenum type, GLvoid *pixels)`
  
  Reads pixel data from the specified framebuffer rectangle and stores data in the array pointed by `pixels`
  
  `format` can be `GL_RGBA`, `GL_RED`, `GL_ALPHA`, `GL_DEPTH_COMPONENT`
  
  `type` can be `s, u, i, f, etc.`

- `void glDrawPixels(GLsizei width, GLsizei height, GLenum format, GLenum type, GLvoid *pixels)`
  
  Draws a rectangle of pixel data with dimensions `width` and `height`
  
  Pixel rectangle has its lower-left corner at the current raster position

- `void glCopyPixels(GLint x, GLint y, GLsizei width, GLsizei height, GLenum buffer)`
  
  Copies pixel data from the specified framebuffer rectangle
  
  Buffer can be `GL_COLOR`, `GL_DEPTH`, `GL_STENCIL`
Example: Drawing Image

- Make a checkerboard image
- Define raster position
- Draw an pixel rectangle of the image
What is Texture Mapping?

- Gluing an image (such as scanned photograph) to a polygon
  - Bricks on wall
  - Ground in flight simulation
  - Vegetation

- Textures are rectangular arrays of data (colors, luminance)
  - Individual values are called texels

- Textures can be manipulated with transformations
  - Repeat textures in different directions to cover the surface
  - Apply textures in different shapes and sizes
Steps in Texture Mapping

- Create a texture object and specify a texture for the object
- Indicate how the texture is to be applied to each pixel
- Enable texture mapping
- Draw the scene by supplying both texture and geometric coordinates
Sample Example

- Check board texture is generated
  ```c
  makeCheckImage()
  ```

- All texture mapping initialization occurs in `init()`
  ```c
  glGenTextures(1, &texName);
  glBindTexture(GL_TEXTURE_2D, texName);
  ```

- Single, full resolution texture map is specified
  ```c
  glTexImage2D(GL_TEXTURE_2D, 0, GL_RGBA, checkImageWidth, checkImageHeight, 0, GL_RGBA, GL_UNSIGNED_BYTE, checkImage);
  ```

- Specify how the texture to be wrapped and how the colors are to be filtered if there is not an exact match between texels and pixels
  ```c
  glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_S, GL_REPEAT);
  glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER, GL_NEAREST);
  ```
More on Example

- In `display(void)`, texture is turned on:
  ```
  glEnable(GL_TEXTURE_2D);
  ```

- Drawing mode is set so as to draw the textured polygons using the colors from the texture map:
  ```
  glTexEnvf(GL_TEXTURE_ENV, GL_TEXTURE_ENV_MODE, GL_REPLACE);
  ```

- Two polygons are drawn by specifying texture coordinates along with vertex coordinates:
  ```
  glTexCoord2f(0.0, 0.0); glVertex3f(-2.0, -1.0, 0.0);
  ```

- Texture is finally turned off:
  ```
  glDisable(GL_TEXTURE_2D);
  ```
3D Textures

- 3D textures are used in scientific visualization e.g. volume rendering
- Defining a 3D texture:
  ```
  glTexImage3D(GL_TEXTURE_3D, 0, GL_RGB, iWidth, iHeight, iDepth, 0, GL_RGB, GL_UNSIGNED_BYTE, image);
  ```
- Replace all or some of the texels of a 3D texture
- Using compressed texture images
- Mipmaps: Multiple levels of detail
- Filtering
- Texture objects
- Texturing functions
Example:

- Putting all together: cube.cpp

  Viewing
  Color
  Lighting
  Texture Mapping