Today’s class

- Simple shadows
- Shading
- Lighting in OpenGL
Simple shadows

- Simple shadows can be gotten by using projection matrices
- Consider a light source and the shadows it will cause on the floor (e.g. the $xz$-plane, $y=0$)
- Here the shadow will be a flat polygon
- The shadow polygon is the projection of the original object onto the surface
Shadow setup

- Let the light source be located at $(x_l, y_l, z_l)$, where $y_l$ is positive.
- Let the projection plane be $y = 0$.
- Translate the light source to the origin.
- Now have a perspective projection through the origin (like we did previously for $z = d$), but the projection plane is now $y = -y_l$.
- Translate back.
Shadow projection matrix

The three transformation matrices are:

\[
\begin{bmatrix}
1 & 0 & 0 & x_l \\
0 & 1 & 0 & y_l \\
0 & 0 & 1 & z_l \\
0 & 0 & 0 & 1
\end{bmatrix}
\begin{bmatrix}
1 & 0 & 0 & 0 \\
0 & 1 & 0 & 0 \\
0 & 0 & 1 & 0 \\
0 & 0 & 0 & 1
\end{bmatrix}
\begin{bmatrix}
1 & 0 & 0 & -x_l \\
0 & 1 & 0 & -y_l \\
0 & 0 & 1 & -z_l \\
0 & 0 & 0 & 1
\end{bmatrix}
\]

It will be easy to implement this as 3 transformations in OpenGL
Example program

- RotatingCubeShadow.cpp shows an example of generating a simple shadow
Shading models

- Shading models combine information about light sources with data about roughness and color of a face’s surface to approximate how the actual light reflects from the surface of the face
Two approaches

- Flat shading
  - Model is applied to only one point on the face
  - The shade determined there fills the entire face

- Smooth shading
  - Model applied to several points on the face
  - Shades blended to yield shade at each point
A simple model

- Total light = point light sources + ambient light
  \[ I = I_{\text{src}} + I_{\text{amb}} \]
- \( I_{\text{src}} \) consists of two components - diffuse and specular
  \[ I_{\text{src}} = I_d + I_{sp} \]
Geometry of light hitting an object

\[ r \text{ is reflected light ray } = -s + 2(s \cdot u_n)u_n \]
Diffuse light

- Uniformly reemitted from a surface after absorbing incident light
- Brightness is proportional to surface orientation (Lambert’s law; depends on $\cos \theta$)

$$I_d = I_s r_d (\mathbf{u}_s \cdot \mathbf{u}_n)$$

- $I_s$ is the intensity of the light source
- $r_d$ is the diffuse reflection coefficient, usually chosen by trial and error
Specular light

- Light reflected from a surface
- Intensity eye sees is proportional to angle between reflected ray and eye ($\phi$)
  \[ I_{sp} = I_s r_s (u_r \cdot u_v)^f \]
- $r_s$ is the specular reflection coefficient, usually chosen by trial and error
- $f$ is usually between 1 and 200
**Ambient light**

- Soft, non-directional light
- Assigned an intensity \((I_a)\), experimentally chosen
- Assigned a reflection coefficient \((r_a)\), usually the same as \(r_d\)
- Total ambient light for a surface is
  \[ I_{amb} = I_a r_a \]
Total light

- Combining the terms just developed gives the total light for a point:

\[ I = I_{\text{src}} + I_{\text{amb}} = I_d + I_{sp} + I_{amb} = \]

\[ I_s[r_d(u_s \cdot u_n) + r_s(u_r \cdot u_v)^f] + I_a r_a \]
Flat shading

- Total intensity is an expensive number (computationally) to compute
- Want to do it as infrequently as possible
- Calculate it once per face and use this value to color the face
Flat shading algorithm

- For each visible face:
  - Choose a point \( p \) on the face (easiest choice is a vertex, but a better approximation would be the center - just average the vertices)
  - Find the normal to the face
  - Compute \( I \) at \( p \) using the equation
  - Fill the projected polygon with the computed shade
Flat shading pros/cons

- Advantages to flat shading
  - Fast
  - Simple

- Disadvantages
  - Image is not very realistic
  - Highlights are rendered poorly
  - A large specular component at a point is spread over the entire face
Flat shading in OpenGL

- `glShadeModel (GL_FLAT);`
- **Color of object is determined by color of:**
  - Last vertex for `GL_LINES`, `GL_TRIANGLES`, `GL_QUADS`
  - First vertex for `GL_POLYGON`
Ray tracing

- At the other extreme, we can evaluate $I$ at every point on the face.
- This gives rise to very realistic images at the expense of a huge computational time.
Smooth shading

- Two approaches to smooth shading
  - Gouraud shading
  - Phong shading
Gouraud shading

- Interpolates intensities over the face
- Assumes polygon face is not planar
- Computes true intensities at each vertex
- Takes a scan line approach
  - Interpolates intensities at each end of scan line
  - Interpolates across scan line
Gouraud shading (cont.)

- At each vertex, several faces meet
- To find normal at each vertex, average normal for each face that meets at that vertex
- Due to interpolation, highlights are poorly handled here as well
Phong shading

- Instead of interpolating intensities, the normals are interpolated.
- The interpolated normals are used to compute intensities at each pixel.
- Since better normals are used we get a better effect from specular components.
Color and the lighting model

- Each light source can be considered to be the sum of red, green and blue components.
- Specular reflection is independent of the color of the object; thus, specularly reflected light is the same color as the incident light.
- Diffuse and ambient components depend upon the color of the surface.
**Color and the lighting model (cont.)**

- Define \( r_{dR}, r_{dG}, \) and \( r_{dB} \) as the diffuse reflection coefficients, which measure the percentage of their respective colors that are reflected.

- The amount of red light that reaches the eye is therefore

\[
I_R = I_{SR}[r_{dR}(u_s \cdot u_n) + r_s(u_r \cdot u_v)^f] + I_{aR}r_{aR}
\]

- \( r_{aR} \) is usually set equal to \( r_{dR} \).
Lighting in OpenGL

- Light in a scene comes from several light sources
- Lighting is divided into four components which are independently computed and added together
  - emitted
  - ambient
  - diffuse
  - specular
Lighting in OpenGL (cont.)

- Emitted light originates from an object and is unaffected by any light sources.
- Ambient light is from a source that has been scattered so much by the environment that its direction is impossible to determine.
Lighting in OpenGL (cont.)

- Diffuse light comes from one direction, but once it hits a surface it is scattered equally in all directions, so it appears equally bright no matter where the eye is located.

- Specular light comes from one direction and bounces off a surface in a preferred direction.
Lighting in OpenGL (cont.)

- A light source delivers a single distribution of frequencies
- However, the ambient, diffuse, and specular components might be different
- You can specify the RGB values for each component of light independently
- RGB values are in [0.0, 1.0]. For multiple lights the RGB values are summed and clamped to 1.0
Enabling light in OpenGL

- Enable lighting with `glEnable (GL_LIGHTING);`
- Enable a particular light with `glEnable (GL_LIGHT0); glEnable (GL_LIGHT1);` etc.
- Each OpenGL implementation must provide a minimum of 8 light sources
- Do the enabling **after** setting the lights’ properties
Light properties in OpenGL

- `glLight* (light, parameter, value);`
  - `GL_AMBIENT` specifies RGBA values for ambient portion of light
  - `GL_DIFFUSE`
  - `GL_SPECULAR`
  - `GL_POSITION` specifies $x, y, z, w$ for position of light
    - $w = 0$ is a directional light
    - $w \neq 0$ is a positional light
Global ambient light

- `glLightModel* (GL_LIGHT_MODEL_AMBIENT, value);` provides global ambient light that is not from a particular light source
Material properties

- RGB values correspond to the reflected proportions of the colors
- `glMaterial* (face, parameter, value);` specifies properties of objects in the scene
- `face` is one of
  - `GL_FRONT`
  - `GL_BACK`
  - `GL_FRONT_AND_BACK`
**glMaterial** parameters

- **GL_AMBIENT** specifies RGBA values for ambient color
- **GL_DIFFUSE**
- **GL_AMBIENT_AND_DIFFUSE**
- **GL_SPECULAR**
- **GL_SHININESS** specifies specular exponent
- **GL_EMISSION** specifies RGBA values for emissive color
What reaches the eye?

- For a particular lighting component, the light that reaches the eye is the product of the light value and the material value.
Example program

- *sphere.cpp* is an example program showing different lighting models
- The program generates a sphere from a tetrahedron using a technique known as *recursive subdivision*