This Week

- We will examine:
  - Visual Realism
  - Lighting
  - Shading
  - Texturing; and
  - Shadowing

Visual Realism

- When rendering (painting) an image we need to consider the factors that will make the scene appear real.
- Provides a "suspension of disbelief" for the viewer. e.g. Shrek

Visual Realism

- Lowest Level is Wireframe
  - Outline of Solids
  - Quick to render
  - OpenGL takes care of hidden lines.

Visual Realism

- Flat Shading
  - Each polygon is shaded in the same tone.
  - Amount of light is calculated from a single point on the surface.
  - Same normal applies for whole surface.

Visual Realism

- Smooth Shading
  - Gouraud Shading
  - Different grey levels are used across a polygon by interpolating the difference between vertices.
Shading Models

- **Shading Model**
  - how light is scattered or reflected from a surface
  - frequently presupposes that two types of light sources illuminate the objects in a scene: point light and ambient light
  - light can be absorbed, reflected and refracted.

Reflection
- **diffuse scattering**
  - some light penetrates the surface and is re-radiated uniformly in all directions
- **specular reflection**
  - mirrorlike and highly directional

Reflected Light
- **m** is a vector normal to the surface at P
- **s** is the vector from P to the light
- **v** is the vector from P to the viewer

The amount of reflected light can be calculated using the angles between these vectors.

A mesh has two sides. We need to calculate the light for the side we are viewing from.

This is why it is important that we have the right normal vector.

We determine if a side is visible by looking at the normal, **m**, and the vector to the eye, **v**.
- If v.m > 0 (less than 90°) the side is visible.
- E.g. the eye must be on the same side of the mesh as the light.

Calculating diffuse reflected light using **m**, **v** and **s**.
- As diffuse light is scattered uniformly in all directions, the location of the eye, **v**, is not important unless v.m < 0 where we want the light intensity I = 0
- The relationship between the brightness of a surface and its orientation toward the light is based on \( \cos(\theta) \).
- \( I_d = I_d \cdot \cos(\theta) \) or \( I_d = I_d \cdot \frac{(s.m)(s|m|)}{|s||m|} \)
  - \( I_d \) is the diffuse reflection coefficient
  - \( \theta \) is how reflective the surface is
Reflected Light

- Calculating diffuse reflected light using m, v and s.
  - If the eye is on the other side of the surface the dot product will be negative.
  - In this case we want \( I_d \) to be 0, therefore:
    \[ I_d = I_p \max (s \cdot m |s| |m|, 0) \]
  - This relationship is called Lambert’s Law

Specular Light

- Real objects do not scatter light uniformly.
- A specular component accounts for this.
- We will examine the Phong model.
- In the Phong model, the amount of light reflected is greatest in the direction of the mirror reflection.

Specular Light

- This is the direction in which all light would travel if the surface was a perfect mirror.

Specular Light

- From module 4, we remember that a perfect reflection is:
  \[ r = s + 2 (s \cdot m |m|^2) m \]
  - Light, not reflected from the true reflection angle falls off determined by the angle, \( \Phi \), between \( r \) and \( v \).
  - In the Phong model, the actual fall off is calculated using a power, \( f \), of \( \cos(\Phi) \), or
    \[ I_s = I_p p_s \cos(f \cdot r / |r| |v|) \] with \( 1 \leq f \leq 400 \) (or whatever looks good!!)
  - \( p_s \) specular reflection coefficient

Ambient Light

- Light lands on objects from multiple angles of reflection from other objects and the environment.
- Computationally expensive to calculate.
- Ambient light has no particular origin.
Combining Light Sources

\[ I = I_p + I_g + I_s \]

Colour

- As we know, colour can be constructed from amounts of red, green and blue.
- The intensity of reflected colour can be calculated by computing the intensity for red, green and blue and adding them.

\[ I_r = I_{p,r} + I_{g,r} + I_{s,r} \]
\[ I_g = I_{p,g} + I_{g,g} + I_{s,g} \]
\[ I_b = I_{p,b} + I_{g,b} + I_{s,b} \]

Textures

- Textures enhance realism by pasting images onto surfaces of a mesh.
- We can create a surface texture with:
  - bitmaps
  - computed functions

Bitmap Textures

- Take a bitmap and paste onto a mesh surface.

An image is specified as W=1 and H=1 regardless of the aspect ratio.
- At W=1 we are all (100%) of the way across the width.
- Vice versa for the Height.
Bitmap Textures

```c
void texcoord2f(float s, float t)
{
    glTexCoord2f(s, t);
}
void glVertex3f(float x, float y, float z)
{
    glVertex3f(x, y, z);
}
void glBegin(GL_QUADS)
{
    glBegin(GL_QUADS);
}
void glEnd()
{
    glEnd();
}
```

Coded Textures

- A texture created on the fly using programming code.
- The texture can be modified as the program executes.

Coded Textures

```
Step One: Specify the Texture

static char *texture[] = {
    "................",
    "................",
    "................",
    "...xx.....xx....",
    "...xx.....xx....",
    "................",
    "......xx........",
    "................",
    "..xx.......xx...",
    "....xx...xxxx....",
    "......xxx.......",
    "................",
    "................",
    "................",
    "................",
    "................",
};
```

Coded Textures

```
Step Two: Create an RGB image

GLubyte floorTexture[16][16][3];
GLubyte *loc;
loc = (GLubyte*) floorTexture;
for (int i = 0; i < 16; i++) {
    for (int j = 0; j < 16; j++) {
        if (texture[i][j] == 'x') {
            /* Nice green. */
            loc[0] = 0x9f;
            loc[1] = 0x2f;
            loc[2] = 0x1f;
        } else {
            /* Light gray. */
            loc[0] = 0xaa;
            loc[1] = 0xaa;
            loc[2] = 0xaa;
        }
        loc += 3;
    }
}
```

Coded Textures

```
Step Three: Build Texture Map

gluBuild2DMipmaps(GL_TEXTURE_2D, 3, 16, 16,
    GL_RGB, GL_UNSIGNED_BYTE, floorTexture);
```
Bump Mapping

- Modifying the surface shading by manipulating the normal vectors.

Environment Mapping

- An alternative to calculating reflections.
- Take a snapshot of the environment and paste it onto the object.

Shadows

- Shadows provide realism
- Shadows are the result of light being blocked by an object and fall on another object.
- Shadows can be:
  - 1. 'hand-drawn' using textures
  - however they are not dynamic and no good for an animated environment.
  - 2. calculated

Case Study: Making Shadows.
- \texttt{tuteshadows.cpp}

Reflections

- Case Study: Creating Reflections
- \texttt{reflections.cpp}