Shading

- Assigning colour to pixels or fragments.
- Modelling Illumination
- We shall see how it is done in a rasterization model.

Illumination Model: The Phong Model

For a single light source total illumination at any point is given by:

\[ I = k_a I_a + k_d I_d + k_s I_s \]

where

- \( k_a \) is the contribution due to ambient reflection
- \( k_d \) is the contribution due to diffuse reflection
- \( k_s \) is the contribution due to specular reflection

Components of the Phong Model

- Ambient Illumination: 
  - Represents the reflection of all indirect illumination.
  - Has the same value everywhere.
  - Is an approximation to computing Global Illumination.

- Diffuse Illumination: 
  - Assumes ideal Diffuse Surface - that reflects light equally in all direction.
  - Surface is very rough at microscopic level. For e.g., Chalk and Clay.

\[ I_d = I_L \cos \theta_L \]

\[ \theta_L \equiv \cos \theta_L \cdot \hat{N} \]

- vector to the light source
- intensity of the light source
- surface normal
Components of the Phong Model

Diffuse Illumination: \( I_d = I_L \cos \theta_L \)
- Reflects light according to Lambert’s Cosine Law

Specular Illumination: \( I_s = I_L \cos \theta_s \)
- Ideal specular surface reflects only along one direction:
  - Reflected intensity is view dependent:
    - Mostly it is along the reflected ray but as we move away some of the reflection is slightly offset from the reflected ray due to microscopic surface irregularities.

Shading

The Phong Illumination Model

\[
I = k_a I_a + k_d I_d + k_s I_s
\]
- \( k_a, k_d, k_s \) are material constants defining the amount of light that is reflected as ambient, diffuse and specular. They may be defined in as three values with R, G, B components.

The Blinn-Phong Illumination Model

\[
I = I_a + \sum_{i=1}^{m} (k_i I_i + k_s I_s)
\]

Local Illumination Model

\[
I_{local} = I_a + \sum_{i=1}^{m} (k_i I_i + k_s I_s)
\]

Global Illumination Model

\[
I_{global} = I_{local} + k_r I_{reflected} + k_t I_{transmitted}
\]

Distance-based attenuation can be modeled by an inverse square falloff, i.e., \( I = \frac{I_t}{r^2} \)

If \( \vec{L} \) and \( \vec{N} \) are in opposite directions then the dot product is negative. Use \( I = \max(I, 0) \) to get the correct value.

http://en.wikipedia.org/wiki/Phong_shading
Shading

- Surface Material Properties
- Colour - For each object there can be a
  - Diffuse colour, Specular colour, Reflected colour and Transmitted
colour
- Remember differently coloured light is at different wavelength so:
  - Accounting for shadows:

\[
I = k_a I_a + \sum_{1 \leq i \leq m} (k_d I_d + k_s I_s + k_r I_r + k_t I_t)
\]

OpenGL uses the local Phong Illumination Model.

Where and how is colour of objects computed?

Material properties can be specified using

\[
gMaterial(v\text{ GLenum face, GLenum pname, const GLfloat(* params)};
\]

- face can be GL_FRONT, GL_BACK or GL_FRONT_AND_BACK
- pname can be GL_AMBIENT, GL_DIFFUSE, GL_SPECULAR, GL_EMISSION, GL_SHININESS
  GL_AMBIENT_AND_DIFFUSE
- Then colour is computed at:

\[
I = k_a I_a + \sum_{1 \leq i \leq m} (k_d I_d + k_s I_s + k_r I_r + k_t I_t)
\]

Enabling lighting and individual lights

- glEnable(GL_LIGHTING);
- glEnable(GL_LIGHT0);

Every GL implementation has at least 8 lights.

Property for the lights is defined using:

- glLightf{v}(GLenum light, GLenum pname, GLfloat {* param})

- light is the light enum like GL_LIGHT1
- pname can be
  - GL_AMBIENT, GL_DIFFUSE, GL_SPECULAR, GL_POSITION, GL_SPOT_CUTOFF, GL_SPOT_DIRECTION,
    GL_SPOT_EXPONENT, GL_CONSTANT_ATTENUATION, GL_LINEAR_ATTENUATION, and GL_QUADRATIC_ATTENUATION

Constant Shading - no interpolation of intensity, one intensity for whole object. No depth cues.
Shading

- Faceted Shading – One intensity per polygon computed from the surface normal and light vector. (GL_FLAT)

Pixar Shutterbug images from: http://www.siggraph.org/education/materials/HyperGraph/scanline/shade_models/constant.htm

Shading

- Gouraud Shading – Linear interpolation of intensity across triangles to eliminate edge discontinuity. (GL_SMOOTH)

Pixar Shutterbug images from: http://www.siggraph.org/education/materials/HyperGraph/scanline/shade_models/constant.htm

Shading

- Phong Shading – Interpolation of surface normals. Still local illumination – No GI.

Pixar Shutterbug images from: http://www.siggraph.org/education/materials/HyperGraph/scanline/shade_models/constant.htm

Shading

- Shadows, texture mapping, reflection mapping – simulating GI.

Pixar Shutterbug images from: http://www.siggraph.org/education/materials/HyperGraph/scanline/shade_models/constant.htm

Shading

- Faceted Shading
  - Fast
  - Surface does not look smooth if a piece wise linear approximation to a flat surface is being done
  - Mach Band Effect accentuate the facets.

Pixar Shutterbug images from: http://www.skidmore.edu/~hfoley/Perc4.htm

Shading

- Faceted Shading
Shading

- Gouraud Shading
  - Linearly interpolate intensity along scan lines: eliminates intensity discontinuities at polygon edges; still have gradient discontinuities, mach banding is largely ameliorated, not eliminated.
  - Must differentiate desired creases from tesselation artifacts (edges of cube vs. edges on tessellated sphere).
  - Calculate approximate vertex normals as an average of normals of polygons meeting at that vertex.
  - Neighboring polygons sharing vertices and edges approximate smoothly curved surfaces and will not have greatly differing surface normals hence this approximation is reasonable.
  - Calculate intensity at vertices.
  - Interpolate intensity along polygon edges.
  - Interpolate along scan lines

```
#version 430
in vec3 VertexPosition;
in vec3 VertexNormal;
in vec2 VertexTex;
out Data {
  vec3 FrontColor;
  vec3 BackColor;
  vec2 TexCoord;
} data;

Gouraud Shading: Vertex Shader
```

```cpp
struct LightInfo {
  vec3 Position; //Light Position in eye-coords
  vec3 La;           //Ambient light intensity
  vec3 Ld;           //Diffuse light intensity
  vec3 Ls;           //Specular light intensity
};

struct MaterialInfo {
  vec3 Ka;           //Ambient reflectivity
  vec3 Kd;           //Diffuse reflectivity
  vec3 Ks;           //Specular reflectivity
  float Shininess;   //Specular shininess factor
};

uniform LightInfo Light[LIGHTCOUNT];
uniform MaterialInfo Material;
uniform mat4 ModelViewMatrix;
uniform mat3 NormalMatrix;
uniform mat4 MVP;

void getEyeSpace( out vec3 norm, out vec3 position )
{
  norm = normalize( NormalMatrix * VertexNormal );
  position = vec3( ModelViewMatrix * vec4( VertexPosition, 1 ) );
}

vec3 light( int lightIndex, vec3 position, vec3 norm )
{
  vec3 s = normalize( vec3( Light[lightIndex].Position - position ) );
  vec3 v = normalize( -position.xyz );
  vec3 r = reflect( -s, norm );
  vec3 ambient = Light[lightIndex].La * Material.Ka;
  float sDotN = max( dot( s, norm ), 0.0 );
  vec3 diffuse = Light[lightIndex].Ld * Material.Kd * sDotN;
  vec3 spec = vec3( 0.0 );
  if ( sDotN > 0.0 )
    spec = Light[lightIndex].Ls * Material.Ks * pow( max( dot(r,v) , 0.0 ), Material.Shininess );
  return ambient + diffuse + spec;
}
```
void main()
{
    vec3 eyeNorm;
    vec3 eyePosition;
    getEyeSpace( eyeNorm, eyePosition );
    data.FrontColor = vec3(0);
    data.BackColor = vec3(0);
    for( int i=0; i<LIGHTCOUNT; ++i )
    {
        data.FrontColor += light( i, eyePosition, eyeNorm );
        data.BackColor += light( i, eyePosition, -eyeNorm );
    }
    data.TexCoord = VertexTex;
    gl_Position = MVP * vec4( VertexPosition, 1 );
}

Shading

- **Gouraud Shading**
  - Integrates well with scanline rasterization. On an edge $\Delta S/A$ is constant.
  - vs. Faceted Shading

- **Phong Shading**
  - Interpolate normals along scan lines.
  - Normalize after interpolating (expensive!).
  - Not available in plain OpenGL - done as per pixel lighting on hardware.
  - Still no Global Illumination - most of the effects of Ray Tracing still missing.