Shading

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

Shading

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

Shading

- Components of the Phong Model
- Diffuse Illumination: $I_d = I_L \cos \theta_L$
  - Assumes ideal Diffuse Surface - that reflects light equally in all direction.
  - Surface is very rough at microscopic level. For e.g., Chalk and Clay.

Shading

- Components of the Phong Model
- Diffuse Illumination: $I_d = I_L \cos \theta_L$
  - Reflects light according to Lambert’s Cosine Law
  - $\theta_L$ : Angle between the vector to the light source and the surface normal
  - $I_L$ : Intensity of the light source
  - $N$ : Surface normal
Shading

- Components of the Phong Model

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

- Specular Illumination: \( I_s = I_L \cos \omega \)
  - 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.

\[
I_d = I_L \cos \theta \\
I_s = I_L \cos \omega
\]

- Specular Illumination: \( I_s = I_L \cos \omega \)
  - \( \omega \) is called the coefficient of shininess and \( I_s = I_L \cos \omega \)

- If \( L \) and \( R \) are in opposite directions then the dot product is negative. Use \( \cos \theta = \frac{L \cdot R}{|L||R|} \) to get the correct value.

- If \( d \) is distance to the light source and \( I_t \) is its true intensity then a distance based attenuation can be modelled by an inverse square falloff, i.e., \( I = \frac{I_t}{d^2} \)

\[
I_d = I_L \cos \theta \\
I_s = I_L \cos \omega
\]

- The Phong Illumination Model

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

- Local Illumination Model

\[
I_{local} = k_a I_a + \sum_{i=1}^{m} (k_d I_{d,i} + k_s I_{s,i})
\]

- Global Illumination Model

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

- \( k_a, k_d, k_s \) 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.

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

\[
I = k_a C_d I_a + \sum_{1 \leq i \leq m} (k_d C_d I_{di} + k_s C_s I_{si} + k_r C_r I_{ri} + k_t C_t I_{ti})
\]

Shading

- OpenGL uses the local Phong Illumination Model.

\[
I = k_a C_d I_a + \sum_{1 \leq i \leq m} (k_d C_d I_{di} + k_s C_s I_{si} + k_r C_r I_{ri} + k_t C_t I_{ti})
\]

Shading

- 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 {*})`
  - `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_CONSTANT_ATTENUATION, GL_LINEAR_ATTENUATION, and GL_QUADRATIC_ATTENUATION

- For example:
  - `GLfloat light_ambient(0.0, 0.0, 0.0, 1.0);`
  - `GLfloat light_diffuse(1.0, 1.0, 1.0, 1.0);`
  - `GLfloat light_specular(0.0, 1.0, 0.0, 1.0);`
  - `GLfloat light_position(3.0, 4.0, 0.0, 1.0);`
  - `glLightfv(GL_LIGHT0, GL_AMBIENT, light_ambient);`
  - `glLightfv(GL_LIGHT0, GL_DIFFUSE, light_diffuse);`
  - `glLightfv(GL_LIGHT0, GL_SPECULAR, light_specular);`
  - `glLightfv(GL_LIGHT0, GL_POSITION, light_position);`
  - `glEnable(GL_LIGHT0);`

Deprecated OGL2.x content. See the shading tutorial instead.

Shading

- Material properties can be specified using
  - `glMaterialf{v}(GLenum face, GLenum pname, const GLfloat {*})`
  - `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:

\[
I = k_a C_d I_a + \sum_{1 \leq i \leq m} (k_d C_d I_{di} + k_s C_s I_{si})
\]

Deprecated OGL2.x content. See the shading tutorial instead.

Shading

- 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)

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

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

- **Faceted Shading** – Fast
  - Surface does not look smooth if a piece wise linear approximation to a flat surface is being done.

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

- **Faceted Shading** – `glShadeModel(GL_FLAT);`
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 tessellation artifacts (edges of cube vs. edges on tessellated sphere).

\[
\mathbf{N} = \sum_{i=1}^{n} \mathbf{N}_i \parallel \sum_{i=1}^{n} \mathbf{N}_i
\]

- 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.

\[
I_a = \frac{y_s - y_2}{y_1 - y_2} I_1 + \frac{y_1 - y_s}{y_1 - y_2} I_2
\]

- Interpolate intensity along polygon edges.
- Interpolate along scan lines.

Gouraud Shading: Vertex Shader

```glsl
#version 430

in vec3 VertexPosition;
in vec3 VertexNormal;
in vec2 VertexTex;

out Data {
vec3 FrontColor;
vec3 BackColor;
vec2 TexCoord;
} data;

struct LightInfo {
vec3 Position;
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
};

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 );
  for( int i=0; i<LIGHTCOUNT; ++i )
    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, $\Delta f/\Delta x$ is constant.
  - vs. Faceted Shading

- Faceted Gouraud

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.