Shading in OpenGL

Ed Angel
Professor of Computer Science, Electrical and Computer Engineering, and Media Arts
University of New Mexico
Objectives

• Introduce the OpenGL shading functions
• Discuss polygonal shading
  - Flat
  - Smooth
  - Gouraud
Steps in OpenGL shading

1. Enable shading and select model
2. Specify normals
3. Specify material properties
4. Specify lights
Normals

• In OpenGL the normal vector is part of the state
• Set by `glNormal*()`
  - `glNormal3f(x, y, z);`
  - `glNormal3fv(p);`
• Usually we want to set the normal to have unit length so cosine calculations are correct
  - Length can be affected by transformations
  - Note that scaling does not preserved length
  - `glEnable(GL_NORMALIZE)` allows for autonormalization at a performance penalty
Normal for Triangle

plane \quad \mathbf{n} \cdot (\mathbf{p} - \mathbf{p}_0) = 0

\mathbf{n} = (\mathbf{p}_2 - \mathbf{p}_0) \times (\mathbf{p}_1 - \mathbf{p}_0)

normalize \quad \mathbf{n} \leftarrow \mathbf{n} / |\mathbf{n}|

Note that right-hand rule determines outward face
Enabling Shading

• Shading calculations are enabled by
  - `glEnable(GL_LIGHTING)`
    - Once lighting is enabled, `glColor()` ignored
• Must enable each light source individually
  - `glEnable(GL_LIGHTi)` i=0,1…..
• Can choose light model parameters
  - `glLightModeli(parameter, GL_TRUE)`
    • `GL_LIGHT_MODEL_LOCAL_VIEWER` do not use simplifying distant viewer assumption in calculation
    • `GL_LIGHT_MODEL_TWO_SIDED` shades both sides of polygons independently
Defining a Point Light Source

• For each light source, we can set an RGBA for the diffuse, specular, and ambient components, and for the position

```c
GL float diffuse0[]={1.0, 0.0, 0.0, 1.0};
GL float ambient0[]={1.0, 0.0, 0.0, 1.0};
GL float specular0[]={1.0, 0.0, 0.0, 1.0};
GLfloat light0_pos[]={1.0, 2.0, 3.0, 1.0};
```

```c
 glEnable(GL_LIGHTING);
 glEnable(GL_LIGHT0);
 glLightv(GL_LIGHT0, GL_POSITION, light0_pos);
 glLightv(GL_LIGHT0, GL_AMBIENT, ambient0);
 glLightv(GL_LIGHT0, GL_DIFFUSE, diffuse0);
 glLightv(GL_LIGHT0, GL_SPECULAR, specular0);
```
Distance and Direction

- The source colors are specified in RGBA
- The position is given in homogeneous coordinates
  - If \( w = 1.0 \), we are specifying a finite location
  - If \( w = 0.0 \), we are specifying a parallel source with the given direction vector
- The coefficients in the distance terms are by default \( a = 1.0 \) (constant terms), \( b = c = 0.0 \) (linear and quadratic terms). Change by
  \[
  a = 0.80; \\
glLightf(GL_LIGHT0, GLCONSTANT_ATTENUATION, a);
  \]
Spotlights

- Use `glLightv` to set
  - Direction `GL_SPOT_DIRECTION`
  - Cutoff `GL_SPOT_CUTOFF`
  - Attenuation `GL_SPOT_EXPONENT`
    - Proportional to $\cos^\alpha \phi$
Global Ambient Light

- Ambient light depends on color of light sources
  - A red light in a white room will cause a red ambient term that disappears when the light is turned off
- OpenGL also allows a global ambient term that is often helpful for testing
  - `glLightModelfv(GL_LIGHT_MODEL_AMBIENT, global_ambient)`
Moving Light Sources

• Light sources are geometric objects whose positions or directions are affected by the model-view matrix

• Depending on where we place the position (direction) setting function, we can
  - Move the light source(s) with the object(s)
  - Fix the object(s) and move the light source(s)
  - Fix the light source(s) and move the object(s)
  - Move the light source(s) and object(s) independently
Material Properties

- Material properties are also part of the OpenGL state and match the terms in the modified Phong model

- Set by `glMaterialv()`

```c
GLfloat ambient[] = {0.2, 0.2, 0.2, 1.0};
GLfloat diffuse[] = {1.0, 0.8, 0.0, 1.0};
GLfloat specular[] = {1.0, 1.0, 1.0, 1.0};
GLfloat shine = 100.0
glMaterialf(GL_FRONT, GL_AMBIENT, ambient);
glMaterialf(GL_FRONT, GL_DIFFUSE, diffuse);
glMaterialf(GL_FRONT, GL_SPECULAR, specular);
glMaterialf(GL_FRONT, GL_SHININESS, shine);
```
Front and Back Faces

- The default is shade only front faces which works correctly for convex objects.
- If we set two sided lighting, OpenGL will shade both sides of a surface.
- Each side can have its own properties which are set by using `GL_FRONT`, `GL_BACK`, or `GL_FRONT_AND_BACK` in `glMaterialf`.

back faces not visible  back faces visible
Emissive Term

- We can simulate a light source in OpenGL by giving a material an emissive component
- This component is unaffected by any sources or transformations

```c
GLfloat emission[] = 0.0, 0.3, 0.3, 1.0);
glMaterialf(GL_FRONT, GL_EMISSION, emission);
```
Transparency

• Material properties are specified as RGBA values
• The A value can be used to make the surface translucent
• The default is that all surfaces are opaque regardless of A
• Later we will enable blending and use this feature
Efficiency

• Because material properties are part of the state, if we change materials for many surfaces, we can affect performance
• We can make the code cleaner by defining a material structure and setting all materials during initialization

```c
typedef struct materialStruct {
    GLfloat ambient[4];
    GLfloat diffuse[4];
    GLfloat specular[4];
    GLfloat shininess;
} MaterialStruct;
```

• We can then select a material by a pointer
Polygonal Shading

• Shading calculations are done for each vertex
  - Vertex colors become vertex shades
• By default, vertex shades are interpolated across the polygon
  - `glShadeModel(GL_SMOOTH);`
• If we use `glShadeModel(GL_FLAT);` the color at the first vertex will determine the shade of the whole polygon
Polygon Normals

- Polygons have a single normal
  - Shades at the vertices as computed by the Phong model can be almost same
  - Identical for a distant viewer (default) or if there is no specular component
- Consider model of sphere
- Want different normals at each vertex even though this concept is not quite correct mathematically
Smooth Shading

- We can set a new normal at each vertex
- Easy for sphere model
  - If centered at origin $n = p$
- Now smooth shading works
- Note *silhouette edge*
Mesh Shading

The previous example is not general because we knew the normal at each vertex analytically.

For polygonal models, Gouraud proposed we use the average of the normals around a mesh vertex:

\[
\mathbf{n} = \frac{\mathbf{n}_1 + \mathbf{n}_2 + \mathbf{n}_3 + \mathbf{n}_4}{|\mathbf{n}_1 + \mathbf{n}_2 + \mathbf{n}_3 + \mathbf{n}_4|}
\]
Gouraud and Phong Shading

• Gouraud Shading
  - Find average normal at each vertex (vertex normals)
  - Apply modified Phong model at each vertex
  - Interpolate vertex shades across each polygon

• Phong shading
  - Find vertex normals
  - Interpolate vertex normals across edges
  - Interpolate edge normals across polygon
  - Apply modified Phong model at each fragment
Comparison

• If the polygon mesh approximates surfaces with a high curvatures, Phong shading may look smooth while Gouraud shading may show edges
• Phong shading requires much more work than Gouraud shading
  - Until recently not available in real time systems
  - Now can be done using fragment shaders (see Chapter 9)
• Both need data structures to represent meshes so we can obtain vertex normals