Shading in OpenGL
Objectives

- Introduce 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 normal vector is part of state
- Set by `glNormal*()`
  - `glNormal3f(x, y, z);`
  - `glNormal3fv(p);`
- Usually want to set normal to have unit length
  - `=> cosine calculations are correct`
    - Length can be affected by transformations
    - Note: scaling does not preserved length
  - `glEnable(GL_NORMALIZE) allows for autonormalization at 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 enabled by
  - `glEnable(GL_LIGHTING)`
  - Once lighting 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, set
- RGBA for diffuse, specular, ambient components
- 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};
GL float light0_pos[] = {1.0, 2.0, 3.0, 1.0};

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

- Source colors specified in RGBA
- Position in homogeneous coordinates
  - If $w = 1.0$, ==> finite location
  - If $w = 0.0$, ==> parallel source with given direction vector
- Coefficients in distance terms by default
  - $a = 1.0$ (constant term)
  - $b = c = 0.0$ (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
  - R light in white room ==> R ambient term disappears when light turned off
- OpenGL also allows global ambient term
- Often helpful for testing
  - `glLightModelfv(GL_LIGHT_MODEL_AMBIENT, global_ambient)`
Moving Light Sources

- Light sources = geometric objects
- positions & directions affected by model-view matrix
- Depending on where place position (direction) setting function, can
  - Move light source(s) with object(s)
  - Fix object(s) and move light source(s)
  - Fix light source(s) and move object(s)
  - Move light source(s) and object(s) independently
Material Properties

- Material properties also part of OpenGL state
- Match terms in 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

- Default: shade only front faces
  ==> works correctly for convex objects
- If set two sided lighting, OpenGL will shade both sides of surface
- Each side can have own properties
  set by GL_FRONT, GL_BACK, or GL_FRONT_AND_BACK in glMaterialf

back faces not visible
back faces visible
Emissive Term

• Can simulate light source in OpenGL by giving material emissive component
• This component unaffected by any sources or transformations

GLfloat emission[] = 0.0, 0.3, 0.3, 1.0);
glMateriall(GL_FRONT, GL_EMISSION, emission);
Transparency

- Material properties specified as RGBA values
- A value can make surface translucent
- Default: all surfaces opaque regardless of A
- Later will enable blending and use this feature
Efficiency

- Material properties are part of state
- ==> if change materials for many surfaces
  ==> can affect performance
- Can make code cleaner by defining material
  structure and setting all materials during
  initialization

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

- Can then select material by pointer
Polygonal Shading

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

- Polygons have single normal
  - Shades at vertices as computed by Phong model can be almost same
  - Identical for distant viewer (default) or if there is no specular component

- Consider model of sphere
- Want different normals at each vertex even though not quite correct mathematically
Smooth Shading

- Can set new normal at each vertex
- Easy for sphere model
  - If centered at origin $\mathbf{n} = \mathbf{p}$
- Now smooth shading works
- Note *silhouette edge*
Mesh Shading

• Previous example not general
• because we knew normal at each vertex analytically
• For polygonal models, Gouraud proposed we use average of normals around mesh vertex

\[ n = \frac{(n_1 + n_2 + n_3 + n_4)}{|n_1 + n_2 + n_3 + n_4|} \]
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 polygon mesh approximates surfaces with 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 can obtain vertex normals