CSCI 480 Computer Graphics
Lecture 10

Shading in OpenGL

Normal Vectors in OpenGL
Polygonal Shading
Light Source in OpenGL
Material Properties in OpenGL
Approximating a Sphere

[Angel Ch. 6.5-6.9]

February 20, 2013
Jernej Barbic
University of Southern California

http://www-bcf.usc.edu/~jbarbic/cs480-s13/
Outline

• Normal Vectors in OpenGL
• Polygonal Shading
• Light Sources in OpenGL
• Material Properties in OpenGL
• Example: Approximating a Sphere
Defining and Maintaining Normals

• Define **unit normal** before each vertex

```c
glNormal3f(nx, ny, nz);
glVertex3f(x_1, y_1, z_1);
glVertex3f(x_2, y_2, z_2);
glVertex3f(x_3, y_3, z_3);
```

**same normal for all vertices**

```c
glNormal3f(nx_1, ny_1, nz_1);
glVertex3f(x_1, y_1, z_1);
glNormal3f(nx_2, ny_2, nz_2);
glVertex3f(x_2, y_2, z_2);
glNormal3f(nx_3, ny_3, nz_3);
glVertex3f(x_3, y_3, z_3);
```

**different normals**
Normalization

- Length of normals changes under some modelview transformations (but not under translations and rotations)
- Ask OpenGL to automatically re-normalize
  
  ```
  glEnable(GL_NORMALIZE);
  ```

- Faster alternative (works only with translate, rotate and uniform scaling)
  
  ```
  glEnable(GL_RESCALE_NORMAL);
  ```
Outline

• Normal Vectors in OpenGL
• Light Sources in OpenGL
• Material Properties in OpenGL
• Polygonal Shading
• Example: Approximating a Sphere
Enabling Lighting and Lights

• Lighting “master switch” must be enabled:
  
  ```c
  glEnable(GL_LIGHTING);
  ```

• Each individual light must be enabled:
  
  ```c
  glEnable(GL_LIGHT0);
  ```

• OpenGL supports at least 8 light sources
What Determines Vertex Color in OpenGL

Is OpenGL lighting enabled?

NO

Color determined by glColor3f(...) Ignored:
• normals
• lights
• material properties

YES

Color determined by Phong lighting which uses:
• normals
• lights
• material properties

See also: http://www.sjbaker.org/steve/omniv/opengl_lighting.html
Reminder: Phong Lighting

• Light components for each color:
  – Ambient ($L_a$), diffuse ($L_d$), specular ($L_s$)

• Material coefficients for each color:
  – Ambient ($k_a$), diffuse ($k_d$), specular ($k_s$)

• Distance $q$ for surface point from light source

\[
I = \frac{1}{a + bq + cq^2} \left( k_d L_d (l \cdot n) + k_s L_s (r \cdot v) \alpha \right) + k_a L_a
\]

$I = \text{unit vector to light}$  \quad r \text{ = } I \text{ reflected about } n

$n = \text{surface normal}$  \quad v = \text{vector to viewer}$
Global Ambient Light

• Set ambient intensity for entire scene
  
  ```
  GLfloat al[] = {0.2, 0.2, 0.2, 1.0};
  glLightModelfv(GL_LIGHT_MODEL_AMBIENT, al);
  ```

• The above is default

• Also: local vs infinite viewer

  ```
  glLightModeli(GL_LIGHT_MODEL_LOCAL_VIEWER, GL_TRUE);
  ```

  – Local viewer: Correct specular highlights
    • More expensive, but sometimes more accurate

  – Non-local viewer: Assumes camera is far from object
    • Approximate, but faster (this is default)
Defining a Light Source

• Use vectors \{r, g, b, a\} for light properties
• Beware: light positions will be transformed by the modelview matrix

```c
GLfloat light_ambient[] = {0.2, 0.2, 0.2, 1.0};
GLfloat light_diffuse[] = {1.0, 1.0, 1.0, 1.0};
GLfloat light_specular[] = {1.0, 1.0, 1.0, 1.0};
GLfloat light_position[] = {-1.0, 1.0, -1.0, 0.0};
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);
```
Point Source vs Directional Source

- Directional light given by “position” vector

```c
GLfloat light_position[] = {-1.0, 1.0, -1.0, 0.0};
glLightfv(GL_LIGHT0, GL_POSITION, light_position);
```

- Point source given by “position” point

```c
GLfloat light_position[] = {-1.0, 1.0, -1.0, 1.0};
glLightfv(GL_LIGHT0, GL_POSITION, light_position);
```
Spotlights

• Create point source as before
• Specify additional properties to create spotlight

GLfloat sd[] = {-1.0, -1.0, 0.0};
glLightfv(GL_LIGHT0, GL_SPOT_DIRECTION, sd);
glLightf(GL_LIGHT0, GL_SPOT_CUTOFF, 45.0);
glLightf(GL_LIGHT0, GL_SPOT_EXPONENT, 2.0);
Outline

• Normal Vectors in OpenGL
• Light Sources in OpenGL
• Material Properties in OpenGL
• Polygonal Shading
• Example: Approximating a Sphere
Defining Material Properties

GLfloat mat_a[] = {0.1, 0.5, 0.8, 1.0};
GLfloat mat_d[] = {0.1, 0.5, 0.8, 1.0};
GLfloat mat_s[] = {1.0, 1.0, 1.0, 1.0};
GLfloat low_sh[] = {5.0};
glMaterialfv(GL_FRONT, GL_AMBIENT, mat_a);
glMaterialfv(GL_FRONT, GL_DIFFUSE, mat_d);
glMaterialfv(GL_FRONT, GL_SPECULAR, mat_s);
glMaterialfv(GL_FRONT, GL_SHININESS, low_sh);

OpenGL is a state machine:
material properties stay in effect until changed.
Color Material Mode

- Alternative way to specify material properties
- Uses glColor
- Must be explicitly enabled and disabled

```c
glEnable(GL_COLOR_MATERIAL);
/* affect all faces, diffuse reflection properties */
glColorMaterial(GL_FRONT_AND_BACK, GL_DIFFUSE);
glColor3f(0.0, 0.0, 0.8);
/* draw some objects here in blue */
glColor3f(1.0, 0.0, 0.0);
/* draw some objects here in red */
glDisable(GL_COLOR_MATERIAL);
```
Outline

• Normal Vectors in OpenGL
• Light Sources in OpenGL
• Material Properties in OpenGL
• Polygonal Shading
• Example: Approximating a Sphere
Polygonal Shading

• Now we know vertex colors
  – either via OpenGL lighting,
  – or by setting directly via glColor3f if lighting disabled

• How do we shade the interior of the triangle?
Polygonal Shading

• Curved surfaces are approximated by polygons

• How do we shade?
  – Flat shading
  – Interpolative shading
  – Gouraud shading
  – Phong shading (different from Phong illumination!)
Flat Shading

- Enable with `glShadeModel(GL_FLAT);`
- Shading constant across polygon
- Color of last vertex determines interior color
- Only suitable for very small polygons
Flat Shading Assessment

- Inexpensive to compute
- Appropriate for objects with flat faces
- Less pleasant for smooth surfaces
Interpolative Shading

• Enable with `glShadeModel(GL_SMOOTH);`
• Interpolate color in interior
• Computed during scan conversion (rasterization)
• Much better than flat shading
• More expensive to calculate (but not a problem for modern graphics cards)
Gouraud Shading
Invented by Henri Gouraud, Univ. of Utah, 1971

- Special case of interpolative shading
- **How do we calculate vertex normals for a polygonal surface?** Gouraud:
  1. average all adjacent face normals

\[
\mathbf{n} = \frac{\mathbf{n}_1 + \mathbf{n}_2 + \mathbf{n}_3 + \mathbf{n}_4}{|\mathbf{n}_1 + \mathbf{n}_2 + \mathbf{n}_3 + \mathbf{n}_4|}
\]

2. use \( \mathbf{n} \) for Phong lighting
3. interpolate vertex colors into the interior

- Requires knowledge about which faces share a vertex
Data Structures for Gouraud Shading

- Sometimes vertex normals can be computed directly (e.g. height field with uniform mesh)
- More generally, need data structure for mesh
- Key: which polygons meet at each vertex
Phong Shading ("per-pixel lighting")
Invented by Bui Tuong Phong, Univ. of Utah, 1973

- **At each pixel** (as opposed to at each vertex):
  1. Interpolate *normals* (rather than colors)
  2. Apply Phong lighting to the interpolated normal

- Significantly more expensive

- Done off-line or in GPU shaders (not supported in OpenGL directly)
Phong Shading Results

Michael Gold, Nvidia

- Single light
  - Phong Lighting
  - Gouraud Shading

- Two lights
  - Phong Lighting
  - Gouraud Shading

- Two lights
  - Phong Lighting
  - Phong Shading
Polygonal Shading Summary

• Gouraud shading
  – Set vertex normals
  – Calculate colors at vertices
  – Interpolate colors across polygon
• Must calculate vertex normals!
• Must normalize vertex normals to unit length!
Outline

- Normal Vectors in OpenGL
- Light Sources in OpenGL
- Material Properties in OpenGL
- Polygonal Shading
- Example: Approximating a Sphere
Example: Icosahedron

• Define the vertices

```c
#define X .525731112119133606
#define Z .850650808352039932

static GLfloat vdata[12][3] = {
    {-X, 0.0, Z}, {X, 0.0, Z}, {-X, 0.0, -Z}, {X, 0.0, -Z},
    {0.0, Z, X}, {0.0, Z, -X}, {0.0, -Z, X}, {0.0, -Z, -X},
    {Z, X, 0.0}, {-Z, X, 0.0}, {Z, -X, 0.0}, {-Z, -X, 0.0}
};
```

• For simplicity, this example avoids the use of vertex arrays
Defining the Faces

• Index into vertex data array

```cpp
static GLuint tindices[20][3] = {
    {1,4,0}, {4,9,0}, {4,9,5}, {8,5,4}, {1,8,4},
    {1,10,8}, {10,3,8}, {8,3,5}, {3,2,5}, {3,7,2},
    {3,10,7}, {10,6,7}, {6,11,7}, {6,0,11}, {6,1,0},
    {10,1,6}, {11,0,9}, {2,11,9}, {5,2,9}, {11,2,7}
};
```

• Be careful about orientation!
Drawing the Icosahedron

• Normal vector calculation next

```c
glBegin(GL_TRIANGLES);
for (i = 0; i < 20; i++) {
    icoNormVec(i);
    glVertex3fv(&vdata[tindices[i][0]][0]);
    glVertex3fv(&vdata[tindices[i][1]][0]);
    glVertex3fv(&vdata[tindices[i][2]][0]);
}
glEnd();
```

• Should be encapsulated in display list
Calculating the Normal Vectors

• Normalized cross product of any two sides

```c
GLfloat d1[3], d2[3], n[3];

void icoNormVec (int i) {
    for (k = 0; k < 3; k++) {
        d1[k] = vdata[tindices[i][0]][k] – vdata[tindices[i][1]][k];
        d2[k] = vdata[tindices[i][1]][k] – vdata[tindices[i][2]][k];
    }
    normCrossProd(d1, d2, n);
    glNormal3fv(n);
}
```
The Normalized Cross Product

- Omit zero-check for brevity

```c
void normalize(float v[3]) {
    GLfloat d = sqrt(v[0]*v[0] + v[1]*v[1] + v[2]*v[2]);
}

void normCrossProd(float u[3], float v[3], float out[3]) {
    out[1] = u[2]*v[0] – u[0]*v[2];
    out[2] = u[0]*v[1] – u[1]*v[0];
    normalize(out);
}
```
The Icosahedron

- Using simple lighting setup
Sphere Normals

- Set up instead to use normals of sphere
- Unit sphere normal is exactly sphere point

```c
glBegin(GL_TRIANGLES);
for (i = 0; i < 20; i++) {
    glNormal3fv(&vdata[tindices[i][0]][0]);
    glVertex3fv(&vdata[tindices[i][0]][0]);
    glNormal3fv(&vdata[tindices[i][1]][0]);
    glVertex3fv(&vdata[tindices[i][1]][0]);
    glNormal3fv(&vdata[tindices[i][2]][0]);
    glVertex3fv(&vdata[tindices[i][2]][0]);
}
glEnd();
```
Icosahedron with Sphere Normals

flat shading

interpolation
Recursive Subdivision

• General method for building approximations
• Research topic: construct a good mesh
  – Low curvature, fewer mesh points
  – High curvature, more mesh points
  – Stop subdivision based on resolution
  – Some advanced data structures for animation
  – Interaction with textures
• Here: simplest case
• Approximate sphere by subdividing icosahedron
Methods of Subdivision

- Bisecting angles
- Computing center
- Bisecting sides

Here: bisect sides to retain regularity
Bisection of Sides

• Draw if no further subdivision requested

```c
void subdivide(GLfloat v1[3], GLfloat v2[3],
               GLfloat v3[3], int depth)
{
    GLfloat v12[3], v23[3], v31[3]; int i;
    if (depth == 0) { drawTriangle(v1, v2, v3); }
    for (i = 0; i < 3; i++) {
        v12[i] = (v1[i]+v2[i])/2.0;
        v23[i] = (v2[i]+v3[i])/2.0;
        v31[i] = (v3[i]+v1[i])/2.0;
    }
    ...
```
Extrusion of Midpoints

- Re-normalize midpoints to lie on unit sphere

```c
void subdivide(GLfloat v1[3], GLfloat v2[3],
               GLfloat v3[3], int depth)
{
    ...  
    normalize(v12);
    normalize(v23);
    normalize(v31);
    subdivide(v1, v12, v31, depth-1);
    subdivide(v2, v23, v12, depth-1);
    subdivide(v3, v31, v23, depth-1);
    subdivide(v12, v23, v31, depth-1);
}
```
Start with Icosahedron

• In sample code: control depth with ‘+’ and ‘-’

    void display(void)
    {
        ...  
        for (i = 0; i < 20; i++) {
            subdivide(&vdata[tindices[i][0]][0],
                      &vdata[tindices[i][1]][0],
                      &vdata[tindices[i][2]][0],
                      depth);
        }
        glFlush();
    }
One Subdivision

flat shading

interpolation
Two Subdivisions

- Each time, multiply number of faces by 4

flat shading

interpolation
Three Subdivisions

- Reasonable approximation to sphere

flat shading  interpolation
Example Lighting Properties

GLfloat light_ambient[] = {0.2, 0.2, 0.2, 1.0};
GLfloat light_diffuse[] = {1.0, 1.0, 1.0, 1.0};
GLfloat light_specular[] = {0.0, 0.0, 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);
Example Material Properties

GLfloat mat_specular[]={0.0, 0.0, 0.0, 1.0};
GLfloat mat_diffuse[]={0.8, 0.6, 0.4, 1.0};
GLfloat mat_ambient[]={0.8, 0.6, 0.4, 1.0};
GLfloat mat_shininess={20.0};
glMaterialfv(GL_FRONT, GL_SPECULAR, mat_specular);
glMaterialfv(GL_FRONT, GL_AMBIENT, mat_ambient);
glMaterialfv(GL_FRONT, GL_DIFFUSE, mat_diffuse);
glMaterialf(GL_FRONT, GL_SHININESS, mat_shininess);

glShadeModel(GL_SMOOTH); /*enable smooth shading */
glEnable(GL_LIGHTING); /* enable lighting */
glEnable(GL_LIGHT0);  /* enable light 0 */
Summary

• Normal Vectors in OpenGL
• Polygonal Shading
• Light Sources in OpenGL
• Material Properties in OpenGL
• Example: Approximating a Sphere