

## 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 14, 2011  
Jernej Barbic  
University of Southern California  
<http://www.bcf.usc.edu/~jbarbic/cs480-s11/>

1

## Outline

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

2

## Defining and Maintaining Normals

- Define unit normal before each vertex

<code>glNormal3f(nx, ny, nz);</code>	<code>glNormal3f(nx<sub>1</sub>, ny<sub>1</sub>, nz<sub>1</sub>);</code>
<code>glVertex3f(x<sub>1</sub>, y<sub>1</sub>, z<sub>1</sub>);</code>	<code>glVertex3f(x<sub>1</sub>, y<sub>1</sub>, z<sub>1</sub>);</code>
<code>glVertex3f(x<sub>2</sub>, y<sub>2</sub>, z<sub>2</sub>);</code>	<code>glNormal3f(nx<sub>2</sub>, ny<sub>2</sub>, nz<sub>2</sub>);</code>
<code>glVertex3f(x<sub>3</sub>, y<sub>3</sub>, z<sub>3</sub>);</code>	<code>glVertex3f(x<sub>2</sub>, y<sub>2</sub>, z<sub>2</sub>);</code>
	<code>glNormal3f(nx<sub>3</sub>, ny<sub>3</sub>, nz<sub>3</sub>);</code>
	<code>glVertex3f(x<sub>3</sub>, y<sub>3</sub>, z<sub>3</sub>);</code>

same normal  
for all vertices

different normals

3

## 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);
```

4

## Outline

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

5

## Enabling Lighting and Lights

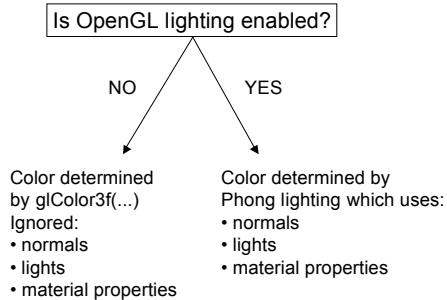
- Lighting “master switch” must be enabled:

```
glEnable(GL_LIGHTING);
```
- Each individual light must be enabled:

```
glEnable(GL_LIGHT0);
```
- OpenGL supports at least 8 light sources

6

## What Determines Vertex Color in OpenGL



See also: [http://www.sjbaker.org/steve/omniv/opengl\\_lighting.html](http://www.sjbaker.org/steve/omniv/opengl_lighting.html)

7

## 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

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

$l$  = unit vector to light       $r = l$  reflected about  $n$   
 $n$  = surface normal           $v$  = vector to viewer

8

## 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)

9

## Defining a Light Source

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

```
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);
```

10

## Point Source vs Directional Source

- Directional light given by "position" vector
 

```
GLfloat light_position[] = {-1.0, 1.0, -1.0, 0.0};
glLightfv(GL_LIGHT0, GL_POSITION, light_position);
```
- Point source given by "position" point
 

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

11

## 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);
```

12

## Outline

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

13

## 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_AMBIENT, 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.

14

## Color Material Mode

- Can shortcut material properties using glColor
- Must be explicitly enabled and disabled

```
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);
```

15

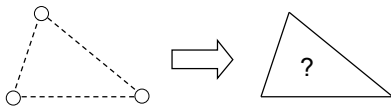
## Outline

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

16

## 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 ?



17

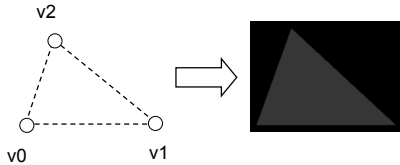
## Polygonal Shading

- Curved surfaces are approximated by polygons
- How do we shade?
  - Flat shading
  - Interpolative shading
  - Gouraud shading
  - Phong shading (different from Phong illumination!)

18

## Flat Shading

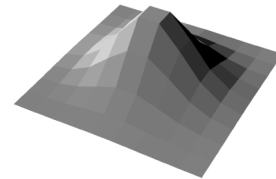
- Enable with `glShadeModel(GL_FLAT);`
- Shading constant across polygon
- Color of last vertex determines interior color
- Only suitable for *very* small polygons



19

## Flat Shading Assessment

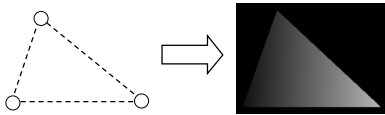
- Inexpensive to compute
- Appropriate for objects with flat faces
- Less pleasant for smooth surfaces



20

## 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)



21

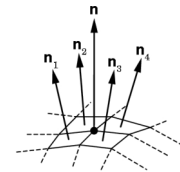
## Gouraud Shading

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

$$n = \frac{n_1 + n_2 + n_3 + n_4}{|n_1 + n_2 + n_3 + n_4|}$$

2. use  $n$  for Phong lighting
3. interpolate vertex colors into the interior



- Requires knowledge about which faces share a vertex

22

## 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

23

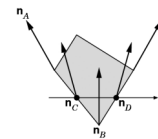
## Phong Shading (“per-pixel lighting”)

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

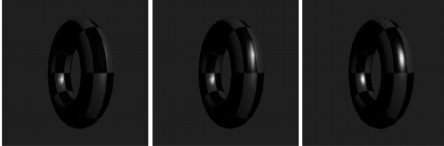
- Done off-line or in GPU shaders (not supported in OpenGL directly)



24

## Phong Shading Results

Michael Gold, Nvidia



Single light  
Phong Lighting  
Gouraud Shading

Two lights  
Phong Lighting  
Gouraud Shading

Two lights  
Phong Lighting  
Phong Shading

25

## 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!

26

## Outline

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

27

## Example: Icosahedron

- Define the vertices

```
#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

28

## Defining the Faces

- Index into vertex data array

```
static GLuint indices[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!

29

## Drawing the Icosahedron

- Normal vector calculation next

```
glBegin(GL_TRIANGLES);
for (i = 0; i < 20; i++) {
  icoNormVec(i);
  glVertex3fv(&vdata[indices[i][0]] [0]);
  glVertex3fv(&vdata[indices[i][1]] [0]);
  glVertex3fv(&vdata[indices[i][2]] [0]);
}
glEnd();
```
- Should be encapsulated in display list

30

## Calculating the Normal Vectors

- Normalized cross product of any two sides

```
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);
}
```

31

## The Normalized Cross Product

- Omit zero-check for brevity

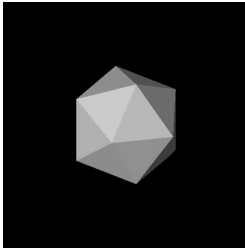
```
void normalize(float v[3]) {
    GLfloat d = sqrt(v[0]*v[0] + v[1]*v[1] + v[2]*v[2]);
    v[0] /= d; v[1] /= d; v[2] /= d;
}

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

32

## The Icosahedron

- Using simple lighting setup



33

## Sphere Normals

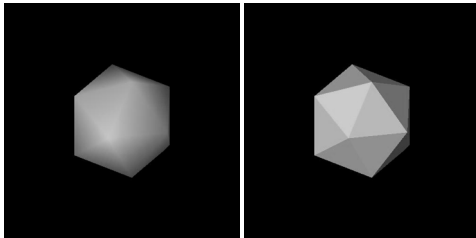
- Set up instead to use normals of sphere
- Unit sphere normal is exactly sphere point

```
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();
```

34

## Icosahedron with Sphere Normals

- Interpolation vs flat shading effect



35

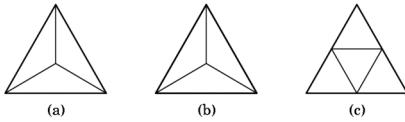
## Recursive Subdivision

- 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

36

## Methods of Subdivision

- Bisecting angles
- Computing center
- Bisecting sides



- Here: bisect sides to retain regularity

37

## Bisection of Sides

- Draw if no further subdivision requested

```
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;
  }
  ...
}
```

38

## Extrusion of Midpoints

- Re-normalize midpoints to lie on unit sphere

```
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);
}
```

39

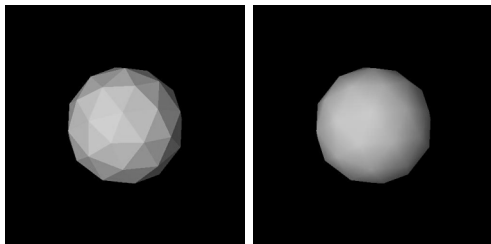
## Start with Icosahedron

- 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();
}
```

40

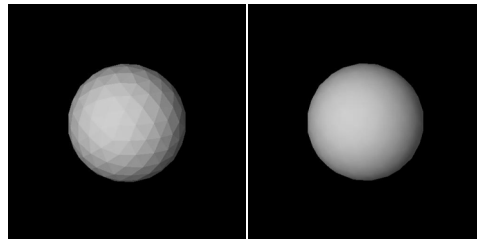
## One Subdivision



41

## Two Subdivisions

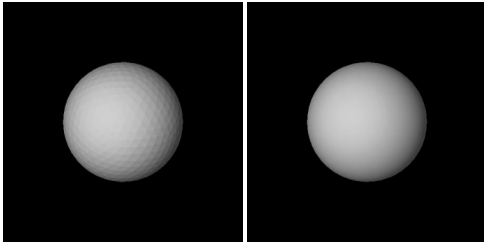
- Each time, multiply number of faces by 4



42

## Three Subdivisions

- Reasonable approximation to sphere



43

## 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);
```

44

## 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 */
```

45

## Summary

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

46