Lecture 10
Lighting and Shading in OpenGL

1

Light Sources and Material Properties

- Appearance depends on
  - Light sources, their locations and properties
  - Material (surface) properties:
  - Viewer positions

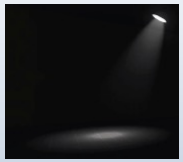


2

Types of Light Sources

These lighting types are sufficient for rendering most simple scenes:

- Ambient light
  - no identifiable source or direction
- Point source
  - given only by point
- Distant light
  - given only by direction
- Spotlight
  - from source in direction
  - Cut-off angle defines a cone of light
  - Attenuation function (brighter in center) (衰减)



3

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);
```

4

Defining Material Properties

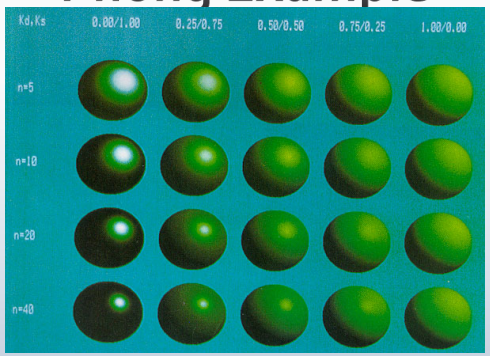
- OpenGL is a state machine:
- material properties stay in effect until changed

```
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);
```

5

Phong Example

α = n



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

6

Ambient Light

- Uniform lighting
- Lights entire scene
- Computationally inexpensive

$$\mathbf{I}_a = \begin{bmatrix} I_{ar} \\ I_{ag} \\ I_{ab} \end{bmatrix}$$

- Simply add $[I_{ar}\ I_{ag}\ I_{ab}]$ to every pixel on every object
- A cheap hack to make the scene brighter.

7

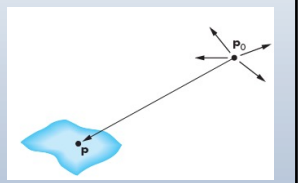
Point Source

- Given by a point p_o

$$\mathbf{I}(\mathbf{p}_0) = \begin{bmatrix} I_r(\mathbf{p}_0) \\ I_g(\mathbf{p}_0) \\ I_b(\mathbf{p}_0) \end{bmatrix}$$

- Light emitted equally in all directions
- Intensity decreases with square of distance
  - At a point p, the intensity of light received from the point source p_o:

$$\mathbf{i}(\mathbf{p}, \mathbf{p}_0) = \frac{1}{|\mathbf{p} - \mathbf{p}_0|^2}\mathbf{I}(\mathbf{p}_0)$$

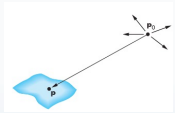


8

Point Sources

Limitations

- Shading and shadows inaccurate
  - objects appear either bright or dark



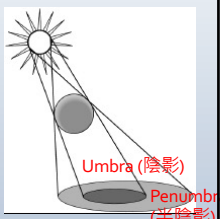
Example: penumbra (partial "soft" shadow)

- Compensate with attenuation

$$\frac{1}{a+bq+cq^2}$$

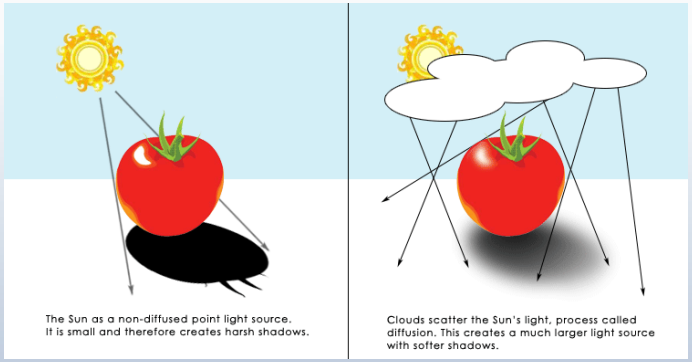
q: distance $|p - p_0|$
a, b, c: constants

- Softens lighting



Shadows created by finite-size light source

9

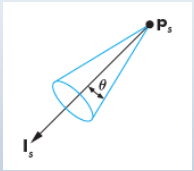


10

Spotlight

- Light still emits from point
  - Spotlight is constructed from a point source by limiting the angles at which light can be seen
- Cut-off by a cone
  - apex is at P_s
  - width is determined by an angle θ
  - points in the direction l_s

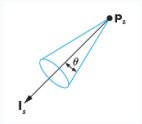
a narrow range of angles through which light is emitted



If θ = 180, the spotlight becomes a point source

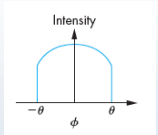
13

Spotlight



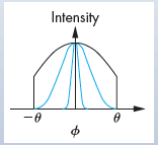
- Attenuation of a spotlight (More realistic)
  - most of the light concentrated in the center of the cone
    - Use the intensity function

$$\cos^e \phi$$



cos ϕ

  - Φ: the angle between the direction of the source l_s and a vector s to a point on the surface
  - Φ <θ
  - exponent e: determines how rapidly the light intensity drops off
  - Easy to compute: cos Φ = u•v , if u, v are unit vector



$\cos^e \phi$

14

Distant Light Source

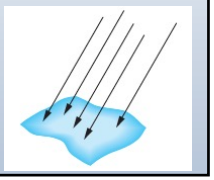
- Given by a direction vector
- Simplifies some calculations
  - most shading calculations require the direction
    - from the point on the surface to the light source position
  - if the light source is far from the surface, the vector does not change much as we move from point to point
    - E.g., the sun strikes all objects at proximity the same angle
- In OpenGL:

$$\mathbf{p}_0 = \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$$

Point source

$$\mathbf{p}_0 = \begin{bmatrix} x \\ y \\ z \\ 0 \end{bmatrix}$$

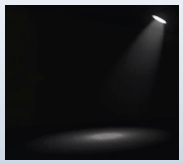
Distant source



15

Light Sources Summary

- Ambient light
  - no identifiable source or direction
- Point source
  - given only by point
- Distant light
  - given only by direction
- Spotlight
  - from source in direction
  - Cut-off angle defines a cone of light
  - Attenuation function (brighter in center) (衰减)



16

Enabling Lighting and Lights

- Lighting must be enabled:

```
glEnable(GL_LIGHTING);
```

- Each individual light must be enabled:

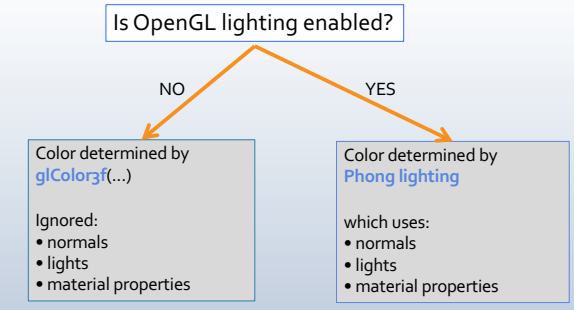
```
glEnable(GL_LIGHT0);
```

- OpenGL supports at least 8 light sources

GL_LIGHT0, GL_LIGHT1, GL_LIGHT2, ..., GL_LIGHT7

17

What Determines Vertex Color in OpenGL

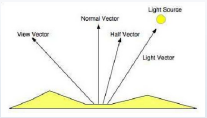


See also: $I = \frac{1}{a + bq + cq^2}(k_d L_d (l \cdot n) + k_s L_s (r \cdot v)^{\alpha}) + k_a L_a$

18

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

19

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

20

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);
```

22

Point Source v.s. Directional Source

- **Directional light:** given by "position" vector

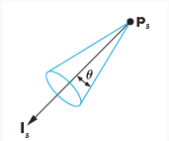
```
GLfloat light_position[] = {-1.0, 1.0, -1.0, 0.0};
glLightfv(GL_LIGHT0, GL_POSITION, light_position);
```

- **Point source:** given by "position" point

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

23

Spotlights



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

$\cos^e \phi$

```
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);
```

24

Outline

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

25

Defining Material Properties

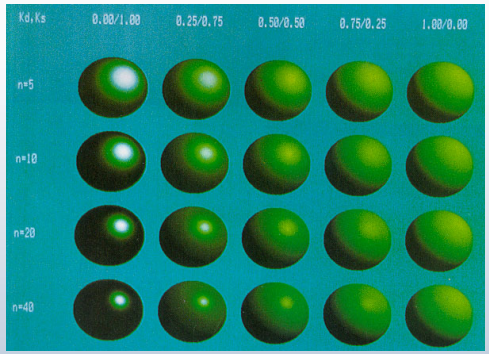
- OpenGL is a state machine:
- material properties stay in effect until changed

```
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);
```

26

Phong Example

α = n



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

27

Color Material Mode

- Can shortcut material properties using glColor
- Explicitly enabled and disabled if you are using lighting
  - GL_COLOR_MATERIAL is initially 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);
```

http://www.opengl.org/sdk/docs/man2/xhtml/glColorMaterial.xml

28

Outline

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

29

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 ?

30

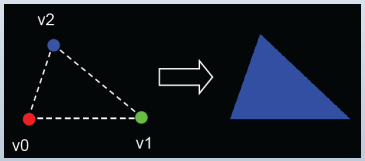
Polygonal Shading

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

31

Flat Shading

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

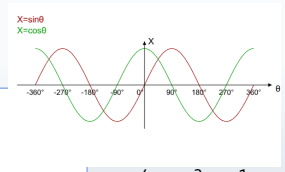


32

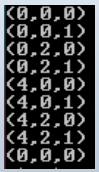
Flat Shading

- Example:

```
glClearColor(0.9f, 0.8f, 0.8f, 1.0f);
glClear(GL_COLOR_BUFFER_BIT);

glShadeModel(GL_FLAT);

glBegin(GL_TRIANGLE_FAN);
glColor3f(1.0f, 1.0f, 1.0f); // white
glVertex2f(0.0f, 0.0f);
for(i=0; i<=8; ++i)
{
   glColor3f(i&0x04, i&0x02, i&0x01);
   glVertex2f(cos(i*Pi/4), sin(i*Pi/4));
}
glEnd();
```



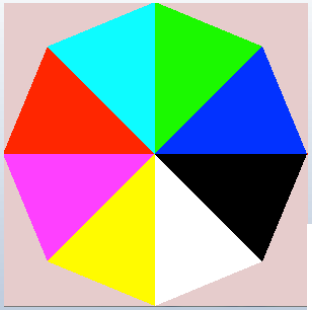
4 2 1
0100, 0010, 0001



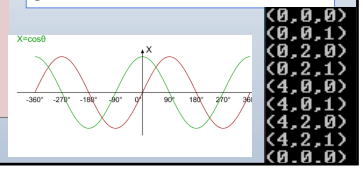
33

Flat Shading

- Result



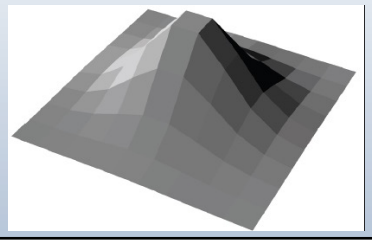
```
glShadeModel(GL_FLAT);
glBegin(GL_TRIANGLE_FAN);
glColor3f(1.0f, 1.0f, 1.0f); // white
glVertex2f(0.0f, 0.0f);
for(i=0; i<=8; ++i) {
   glColor3f(i&0x04, i&0x02, i&0x01);
   glVertex2f(cos(i*Pi/4), sin(i*Pi/4));
}
glEnd();
```



34

Flat Shading

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



35

Interpolative (smooth) Shading

- Enable with glShadeModel(GL_SMOOTH);
  - Default shade mode
- 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)

36

Interpolative Shading

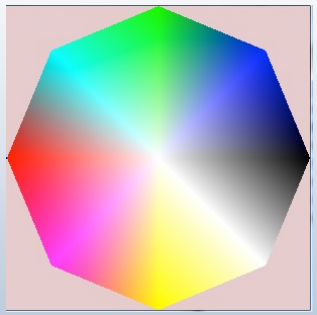
- Example:

```
glShadeModel(GL_SMOOTH);

glBegin(GL_TRIANGLE_FAN);
glColor3f(1.0f, 1.0f, 1.0f);
glVertex2f(0.0f, 0.0f);
for(i=0; i<=8; ++i) {
   glColor3f(i&0x04, i&0x02, i&0x01);
   glVertex2f(cos(i*Pi/4), sin(i*Pi/4));
}
glEnd();
```

37

Interpolative Shading

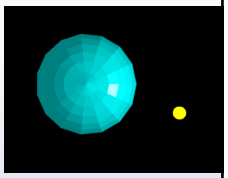
- Result:



38

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



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

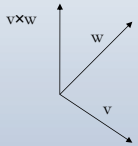
l = unit vector to light
n = surface normal

r = l reflected about n
v = vector to viewer

39

Cross Product

- The cross product of v and w: v x w
  - is a VECTOR, perpendicular to the plane defined by v and w
  - ||v x w||=||v|| ||w|| sinθ
    - θ is the angle between v and w
  - vxw=-(wxv)



40

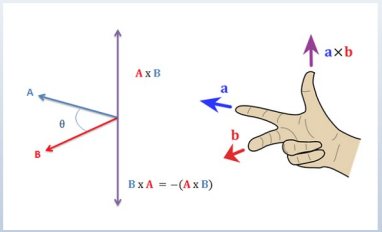
Uses of the Determinant?

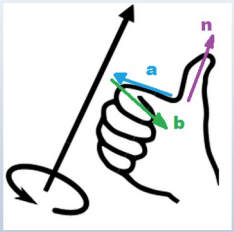
- Linear Independence of columns in a matrix
- Cross Product
  - Given 2 vectors v=[v_1 v_2 v_3], w=[w_1 w_2 w_3], the cross product is defined to be the determinant of

$$\begin{vmatrix} i & j & k \\ v_1 & v_2 & v_3 \\ w_1 & w_2 & w_3 \end{vmatrix} = \begin{pmatrix} v_2w_3 - v_3w_2 \\ v_3w_1 - v_1w_3 \\ v_1w_2 - v_2w_1 \end{pmatrix}$$

41

Right hand rule





42

Defining and Maintaining Normals

- Define unit normal before each vertex

```
glNormal3f(nx, ny, nz);
glVertex3f(x1, y1, z1);
glVertex3f(x2, y2, z2);
glVertex3f(x3, y3, z3);
```

same normal for all vertices

```
glNormal3f(nx1, ny1, nz1);
glVertex3f(x1, y1, z1);
glNormal3f(nx2, ny2, nz2);
glVertex3f(x2, y2, z2);
glNormal3f(nx3, ny3, nz3);
glVertex3f(x3, y3, z3 );
```

different normals

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

43

Normalization

- OpenGL assumes that any normals you pass to it are already of unit length.
  - Strange lighting results will get if not normalized
- Length of normals changes under some modelview transformations !
  - Enable GL_NORMALIZE
    - Alternative to manually normalizing normals
    - If the modelview matrix includes **scaling**, and are applied before lighting, the normals may not be of unit length after the operation
  - Enable GL_RESCALE_NORMALIZE (a cheaper alternative)
    - If only uniform scaling
      - Extracts the scale factor from the model view matrix, and use it to rescale the normal after the matrix is applied

44

Normalization

- Ask OpenGL to automatically re-normalize

```
glEnable(GL_NORMALIZE);
```

- Faster alternative (works only with translate, rotate and uniform scaling)

```
glEnable(GL_RESCALE_NORMAL);
```

45

Gouraud Shading

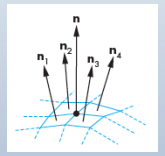
- Invented by Henri Gouraud, Univ. of Utah, 1971
- A special case of interpolative shading (smooth shading)
- How do we calculate vertex normals for a polygonal surface?

1. average all adjacent face normals
   - Requires knowledge about which faces share a vertex

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

2. use n for Phong lighting

3. interpolate vertex colors into the interior

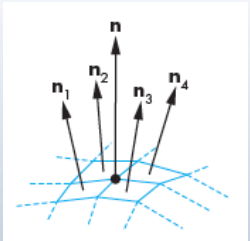


46

Gouraud Shading

- Generally, need data structure for mesh
  - Key: which polygons meet at each vertex

```
glNormal3f(nx, ny, nz);
glVertex3f(x1, y1, z1);
glVertex3f(x2, y2, z2);
glVertex3f(x3, y3, z3);
glVertex3f(x4, y4, z4);
```



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

47

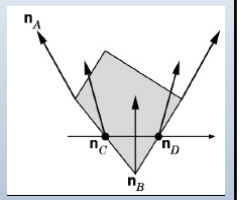
Phong Shading ("per-pixel lighting")

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

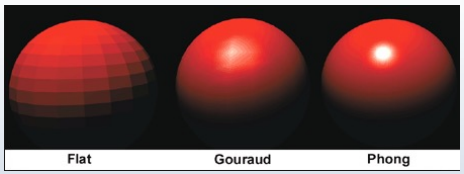
$$I = \frac{1}{a+bq+cq^2}(k_d L_d (l \cdot n) + k_s L_s (r \cdot v)^{\alpha}) + k_a L_a$$

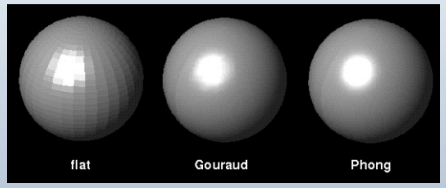


Invented by Bui Tuong Phong, Univ. of Utah, 1973

48

Comparison





49

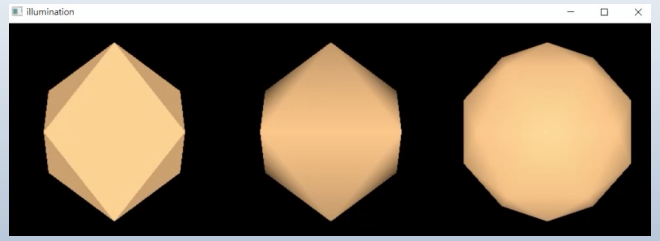
Polygonal Shading Summary

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

50

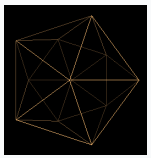
Subdivision

- Keyboard
  - Rotate along the center: (left-right & up-down)
  - Polygon mode: (Line, fill)
  - Subdivide depth (+ , -)



51

Example



- Approximating a Sphere: using 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}
    };
```

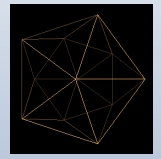
a polyhedron with 20 faces

52

Defining the Faces

- Index into vertex data array

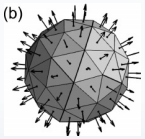
```
static GLuint tindices[20][3] = {
        {1,4,0}, {4,9,0}, {4,9,5}, {8,5,4}, {1,8,4},
        {1,10,8}, {10,3,8}, {8,3,5}, {3,2,5}, {3,7,2},
        {3,10,7}, {10,6,7}, {6,11,7}, {6,0,11}, {6,1,0},
        {10,1,6}, {11,0,9}, {2,11,9}, {5,2,9}, {11,2,7}
};
```



53

Drawing the Icosahedron

(b)



- Normal vector calculation next

```
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

54

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);
}
```

55

The Normalized Cross Product

```
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);
}
```

Cross Product

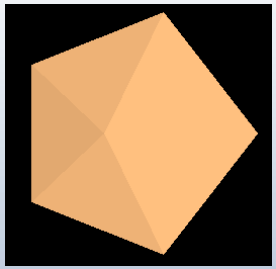
- Given 2 vectors $v=[v_1, v_2, v_3]$, $w=[w_1, w_2, w_3]$, the cross product is defined to be the determinant of

$$\begin{vmatrix} i & j & k \\ v_1 & v_2 & v_3 \\ w_1 & w_2 & w_3 \end{vmatrix} = \begin{pmatrix} v_2w_3 - v_3w_2 \\ v_3w_1 - v_1w_3 \\ v_1w_2 - v_2w_1 \end{pmatrix}$$

56

The icosahedron

- using simple lighting setup

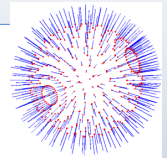


57

Sphere Normals

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

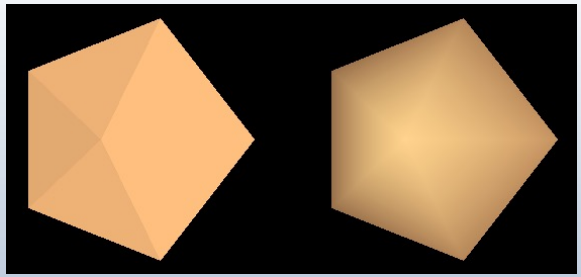
```
glBegin(GL_TRIANGLES);
for (i = 0; i < 20; i++)
{
      glNormal3fv(&vdata[tindices[i][0]][0]);
      glVertex3fv(&vdata[tindices[i][0]][0]);
      glNormal3fv(&vdata[tindices[i][1]][0]);
      glVertex3fv(&vdata[tindices[i][1]][0]);
      glNormal3fv(&vdata[tindices[i][2]][0]);
      glVertex3fv(&vdata[tindices[i][2]][0]);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



58

icosahedron with Sphere Normals

- flat shading effect v.s. Interpolation



flat shading Interpolation

59

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

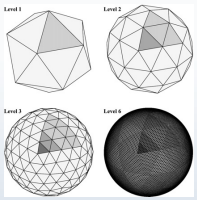
Here: simplest case

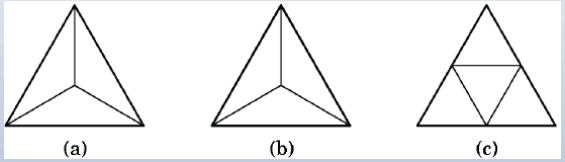
- Approximate sphere by subdividing

60

Methods of Subdivision

- Bisecting angles
- Computing center
- Bisecting sides

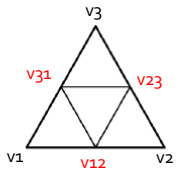




61

Bisection of Sides

```
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); //Draw if no further subdivision requested
        return;
    }
    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;
    }
    ...
}
```



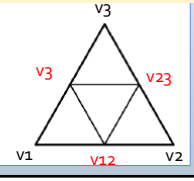
62

Extrusion of Midpoints

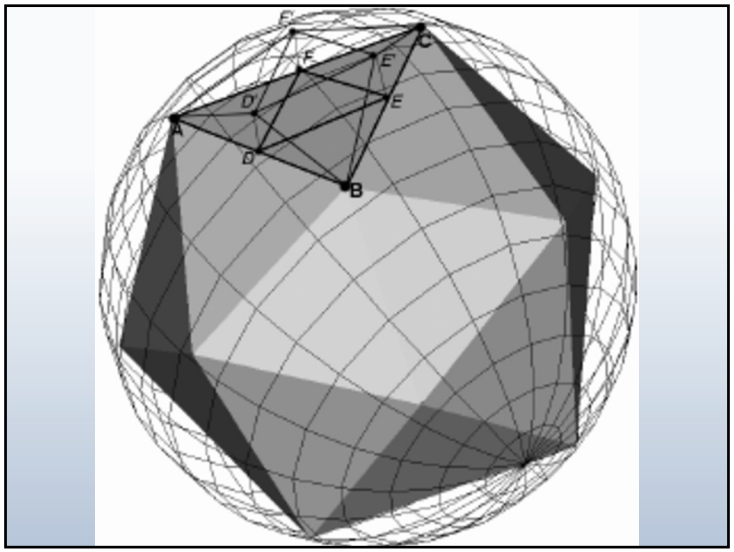
- Re-normalize midpoints to lie on unit sphere

```
void subdivide(GLfloat v1[3], GLfloat v2[3], GLfloat v3[3], int depth)
{
    //...
    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;
    }
    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);
}
```

normalize會將這些新的頂點調整至unit sphere之surface上



63



64

display

- 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();
}
```

65

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);
```

66

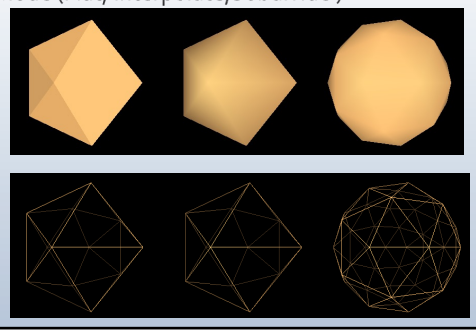
Example Material Properties

```
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 */
```

67

Lab

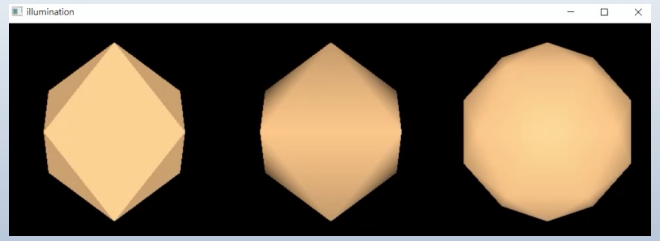
- 3 Mode (Flat, Interpolate,Subdivide)



68

Subdivison

- Keyboard
  - Rotate along the center: (left-right & up-down)
  - Polygon mode: (Line, fill)
  - Subdivide depth (+ , -)



69