Introduction to OpenGL
“Shading”

Shading in OpenGL

OpenGL supports two shading modes

(1) Flat Shading (default)

```
glShadeModel(GL_FLAT);
```

- For flat shading OpenGL uses the normal associated with the first vertex of each polygon
- For triangle strips OpenGL uses the normal of the 3rd vertex for the 1st polygon, 4th vertex for 2nd poly,…..

(2) Gouraud or Interpolative Shading

```
glShadeModel(GL_SMOOTH);
```

For Gouraud shading we must specify the normal for each vertex which should be computed as the average of the adjacent face normals

In OpenGL we can associate a normal with a particular vertex by:

```
glNormal3f(nx,ny,nz);
glNormal3fv(pointer_to_normal);
```

normals are defined before specifying the subsequent vertex

- normals are model variables (like colour) the specified normal is applied to all subsequent verticies until a new normal is defined

Turning the Lights On!

OpenGL support 4 light sources:

- Ambient
- Point
- Spot light
- Distant

Can have upto at least 8 sources + global ambient light in a program

- specify and enable each source

Enable lighting:

```
glEnable(GL_LIGHTING);
```

Enable individual lights:

```
glEnable(source);
```

source = GL_LIGHT0, GL_LIGHT1…

Lighting Parameters

Individual light parameters set by:

```
glLightfv(source, parameter, pointer_to_array);
glLightf(source,parameter,value);
```

Parameters (vector in homogenous coordinates):

GL_POSITION - position or direction
GL_AMBIENT - rgba ambient light
GL_DIFFUSE - rgba diffuse light
GL_SPECULAR - rgba specular light

Light position/direction determined by 4th homogenous coordinate point/vector:

GL_float light0_pos[] = {1.0,2.0,3.0,1.0}; - point
GL_float light0_dir[] = {1.0,2.0,3.0,0.0}; - direction

Lights are treated as points/vectors in OpenGL position light relative to camera using the model-view transform

Light Attenuation with Distance

Distance attenuation in OpenGL is based on the model:

$$f(d) = \frac{1}{a + bd + cd^2}$$

Parameters for light attenuation with:

GL_CONSTANT_ATTENUATION - a
GL_LINEAR_ATTENUATION -b
GL_QUADRATIC_ATTENUATION -c

Set parameters with:

```
Glfloat a=1.0;
glLightf(GL_LIGHT0, GL_CONSTANT_ATTENUATION, a);
```

Example - Lighting

Setup a point light source at position (1,5,7) with red diffuse component no ambient component and white specular component

```
glEnable(GL_LIGHTING);

glEnable(GL_LIGHT0);

Glfloat pos0[] = {1.0,5.0,7.0,1.0};
Glfloat diffuse0[] = {1.0,0.0,0.0,1.0};
Glfloat ambient0[] = {0.0,0.0,0.0,1.0};
Glfloat specular0[] = {1.0,1.0,1.0,1.0};

glLightfv(GL_LIGHT0, GL_POSITION, pos0);
glLightfv(GL_LIGHT0, GL_DIFFUSE, diffuse0);
glLightfv(GL_LIGHT0, GL_AMBIENT, ambient0);
glLightfv(GL_LIGHT0, GL_SPECULAR, specular0);
```

Spot Lights

Convert a positional light source to a directional spot light with by setting parameters:

GL_SPOT_DIRECTION - direction vector (x,y,z)
GL_SPOT_CUTOFF - angle to direction at which light stops [0,180]
GL_SPOT_EXPONENT - attenuation of spot light with direction

Setup spot light as:

```
GLfloat   spot_dir[] = {1.0,0.0,0.0}
GLfloat spot_cutoff = 45.0;
GLfloat spot_exponent = 2.0;

glLightfv(GL_LIGHT0,GL_SPOT_DIRECTION,spot_dir);
glLightf(GL_LIGHT0,GL_SPOT_CUTOFF,spot_cutoff);
glLightf(GL_LIGHT0,GL_SPOT_EXPONENT,spot_exponent);
```

Global Ambient Light

We can add a global ambient illumination independent of individual sources

```
Glfloat global_ambient[] = {0.0,0.5,0.0,1.0};
```
- green ambient light

```
glLightModelfv(GL_LIGHT_MODEL_AMBIENT,global_ambient);
```

Default Lighting Model

The default lighting model used by OpenGL assumes:

- viewer is distant from objects allowing a constant direction
- only the front faces of object are visible (ie not inside)

These assumptions allow for more efficient rendering

The default light model settings:

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

- compute direction to viewer for each vertex
- required if viewer close to scene
- default OK for many scenes

```
glLightModeli(GL_LIGHT_MODEL_TWO_SIDED,GL_TRUE);
```

- render both front and back sides of polygons
- use to view the inside of polygonal surface objects

Materials in OpenGL

Specification of material properties in OpenGL is based on the 3 different lighting components (ambient/diffuse/specular) and the Phong reflection model

Material reflection parameters are specified by:

```
glMaterialfv(face, type, pointer_to_array);
glMaterialf(face,type,value);
```

'face' parameter

```
GL_FRONT_AND_BACK   - parameters are applied for front &back
GL_FRONT            - applied to front only
GL_BACK             - applied to back only
```

'type' parameter are based on the reflection coefficients (k_a, k_d , k_s) for the Phong model:

```
GL_AMBIENT  - ambient reflection coefficient
GL_DIFFUSE
GL_SPECULAR
GL_DIFFUSE_AND_SPECULAR - equal  for diffuse and specular
```

these parameters are all specified as homogenous vectors

Example - Specification of Material Properties

Specify a material with a small ambient component, red diffuse component and white specular component

```
GLfloat ambient[] = {0.1,0.1,0.1,1.0};
GLfloat diffuse[] = {1.0,0.0,0.0,1.0};
GLfloat specular[] = {1.0,1.0,1.0,1.0};

glMaterialfv(GL_FRONT_AND_BACK, GL_AMBIENT, ambient);
glMaterialfv(GL_FRONT_AND_BACK, GL_DIFFUSE, diffuse);
glMaterialfv(GL_FRONT_AND_BACK, GL_SPECULAR, specular);
```

Material properties are modal:

- after setting the material properties they are applied to all subsequent objects specified until the next change in material properties.

Additional Material Properties

Shininess

- the shininess of a surface is defined by the exponent of the specular reflection term in the Phong model
- the shininess can be specified by

```
glMaterialf(GL_FRONT_AND_BACK, GL_SHININESS,100.0);
```

Emisive Surfaces

- self-luminous (light emitting) objects can be specified
- use for putting visible light sources into the image
- the emissive term is unaffected by other light sources & does not affect any other surfaces
- adds a fixed colour to the surface

```
GLfloat emission[]={0.0,0.0,0.5,1.0}
```
- blue light

```
glMaterialfv(GL_FRONT_AND_BACK,GL_EMISSION,emission);
```

Example - Flat Shading a Polygon Mesh

Flat Shading

- For each polygon we can compute a normal n for each face as the cross-product of the 1st 3 non-coliear verticies
- The mesh is then specified by specifying each polygon as:

```
glBegin(GL_POLYGON);
        glNormal3fv(n);
        glVertex3fv(v0);
        glVertex3fv(v1);
        glVertex3fv(v2);
glEnd();
```

Example - Gouraud Shading a Polygon Mesh

Gouraud Shading

- we require a function that computer the normal n for each vertex from the adjacent face normals:

```
compute_normal(i,n);   i is the vertex number
```

- given this function we can specify the mesh be computing a new normal for each mesh vertex

```
glShadeModel(GL_SMOOTH);
glBegin(GL_POLYGON);
        compute_normal(0,n);
         glNormal3fv(n);
         glVertex3fv(v0);
        compute_normal(1,n);
         glNormal3fv(n);
         glVertex3fv(v1);
        compute_normal(2,n);
         glNormal3fv(n);
         glVertex3fv(v2);
glEnd();
```

Summary

Shading:

`glShadeModel(m);` m = GL_FLAT or GL_SMOOTH

Lighting:

`glEnable(GL_LIGHTING);` - switch on lighting

`glEnable(s);` individual lights source = GL_LIGHT0, GL_LIGHT1…

`glLightfv(source, parameter, pointer_to_array);` - parameters

`glLightf(source,parameter,value);`

parameter = GL_POSITION/GL_AMBIENT/GL_SPECULAR/GL_DIFFUSE

`glLightModelfv(model,global_ambient);`

model = GL_LIGHT_MODEL_AMBIENT

GL_LIGHT_MODEL_LOCAL_VIEWER/GL_LIGHT_MODEL_TWO_SIDED

Material Properties:

`glMaterialfv(face, type, pointer_to_array);`

`glMaterialf(face,type,value);`

face = GL_FRONT_AND_BACK/GL_FRONT/GL_BACK

type = GL_AMBIENT/GL_DIFFUSE/GL_SPECULAR/

GL_DIFFUSE_AND_SPECULAR

GL_SHININESS/GL_EMISSION