

OpenGL - Lighting, Material, Shading and Texture Mapping

CS475 - Computer Graphics

Sumair Ahmed

IIT Bombay

August 28, 2009

This document briefs about the ways to deal with the prominent features existing in OpenGL. Here we discuss the Lighting, Shading, Material and Texture Mapping in programming perspective of OpenGL.

Lighting

Light makes the scene look real. Effectively, in a real world, light is present everywhere. Objects also have their own reaction in presence of a light. OpenGL supports a number of lighting effects - Directional, Spot, Ambient lights and attenuation. The Lighting and the Shading models define the light in OpenGL. But before setting these up, lighting mode has to be enabled. Lighting support needs the depth buffer to be enabled.

```
glEnable(GL_DEPTH_TEST);
glEnable(GL_LIGHTING);
```

The above function *glEnable* can enable many features of OpenGL; the feature you want to enable is provided as an input parameter. The above code enables the lighting, however we need lights as well. OpenGL has direct support for about 8 lights. To enable a light, call:

```
glEnable(GL_LIGHT0);
```

Likewise, *LIGHT1* or 2..8 shall be enabled when you want to handle multiple lights. Similar to *glEnable* function, OpenGL also has *glDisable* function that disables the features set before by *glEnable*. These functions turn a particular feature ON or OFF. OpenGL also provides a set of specific global properties to specify the visual behavior of the lighting model. This is done in two ways:

```
glLightModelf(GLenum pname, GLfloat param); // scalar params
glLightModelfv(GLenum pname, const GLfloat params); // vector params
```

For Instance, suppose you want to have a *GLOBAL* Ambient light that casts on all the rendered objects. The following code sets the *global_ambient* colour on all the objects.

```
GLfloat global_ambient[] = {0.5f, 0.5f, 0.5f, 1.0f}; // R,G,B,Alpha
glLightModelfv(GL_LIGHT_MODEL_AMBIENT, global_ambient);
```

Similarly, we have *GL_LIGHT_MODEL_TWO_SIDE* and *GL_LIGHT_MODEL_LOCAL_VIEWER* parameters. The first parameter defines if the light is to be applied on two sides of an object or one side. Second defines how specular component is calculated. The specular highlight depends on the direction from the vertex to the viewpoint and also the direction from the vertex to the light source. So, the highlight depends on the eye position. With a default infinite viewpoint, the direction from the vertex to the viewpoint remains the same for all vertices. 0 is considered as infinite viewpoint, while 1 is local view point.

```
glLightModelf(GL_LIGHT_MODEL_LOCAL_VIEWER, GL_TRUE);
```

There are two types of light properties you should consider while programming the lighting model in OpenGL. The first type describes a light source and the second type describes the light reflected by the material of an object's surface. These four independent models define the light - *GL_AMBIENT*, *GL_DIFFUSE*, *GL_SPECULAR* and *GL_EMISSIVE*

For every type of light source you need to call the *glLightfv* function with parameters. For example, To add a component of specular light to a light source, you would make the following function call:

```
GLfloat specular[] = {1.0f, 1.0f, 1.0f, 1.0f};
glLightfv(GL_LIGHT0, GL_SPECULAR, specular);
```

The first three parameters of *specular* variable are the RGB values which can range between 0.0f and 1.0f. 0 being no colour, and 1 being full colour. The final parameter, Alpha, is used to define the translucency factor of an object and is used to simulate materials made out of translucent matter such as glass. The colour format here is often referred as RGBA format.

OpenGL also provides the light attenuation feature, that reduces the light intensity with the distance. Light Intensity should decrease with the distance to mimic the real world. It shall be inversely related to the distance. The Intensity attenuation formula would be :

$$\begin{aligned}\text{Attenuated Intensity} &= \text{Intensity of light Source} * \frac{1}{C+L*d+Q*d^2} \\ &= \text{Intensity of light Source} * \text{Attenuation Factor}\end{aligned}$$

where

d = distance between light source and the vertex
C = *GL_Constant_Attenuation* (default is 1)
L = *GL_Linear_Attenuation* (default is 0)
Q = *GL_Quadratic_Attenuation* (default is 0)

are the three attenuation provided by OpenGL. In order to have an attenuation with the distance, you should set to a value different to 0 the linear or quadratic. It calculates an attenuation factor (between 0 and 1) which is multiplied to the ambient, diffuse and specular colours. By default, attenuation factor is 1, it means there is no attenuation w.r.t the distance.

```
glLightf(GL_LIGHT0, GL_LINEAR_ATTENUATION, 0.2f);
```

But assigning ambient, diffuse and specular types of light to a light source is not usually enough. You also have to specify the position of the light source. You can have Directional light source instead of the below Positional light source.

```
GLfloat position1[] = {-1.5f, 1.0f, -4.0f, 1.0f}; //x,y,z,w
GLfloat position2[] = {-1.05f, 1.0f, -1.0f, 0.0f}; //x,y,z,w
glLightfv(GL_LIGHT0, GL_POSITION, position1);
glLightfv(GL_LIGHT0, GL_POSITION, position2);
```

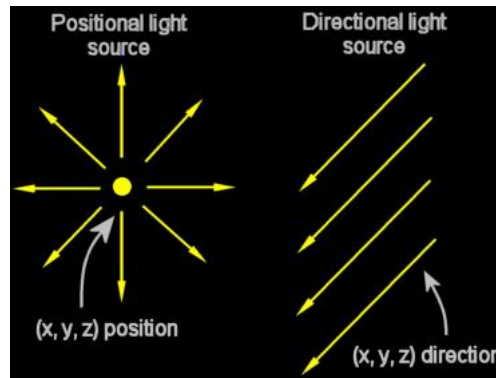


Figure 1: Positional and Directional light source

The light source can be a positional ($w > 0$) or directional ($w = 0$) light source depending on the w value. A positional light source is positioned at the location (x, y, z) as shown above. The source emits light from that particular location towards all directions. for example: lamp, bulb.

A directional one does not have any location. The source emits light from an infinite location, the rays are all parallel and have the direction (x, y, z) . A directional light is not subject to attenuation since it is at an infinite distance. for example: Sun.

Shading

The Shading model is set up with a call to *glShadeModel* and can be either set to *SMOOTH* or *FLAT* model. The *SMOOTH* shading model specifies the use of Gouraud-shaded polygons to describe light while the *FLAT* shading model specifies the use of single-colored polygons.

```
glShadeModel(GL_SMOOTH);
```

When *SMOOTH* model is selected, Lighting is evaluated at each vertex, and pixel colours are linearly interpolated across polygons. However, in *FLAT* model, Lighting is evaluated once for a polygon, and the resulting colour value is used for the complete object.

Material

This defines the reaction of an object when its surface is hit with the light. For example, some objects absorb a particular colour or reflects light. Usually when the lighting is enabled it is equally likely to assign material properties with the *glMaterialfv* command as shown in the following code sample:

```
GLfloat mcolor[] = {1.0f, 0.0f, 0.0f, 1.0f};  
glMaterialfv(GL_FRONT, GL_AMBIENT_AND_DIFFUSE, mcolor);
```

The material property of an object (defined by RGB colour format), is usually the colour reflected by that object. The first parameter of the *glMaterialfv* command indicates which face of the polygon should reflect the light specified by *mcolor*. Apparently, there are two sides to a polygon - front and back. OpenGL provides two ways to specify a polygon in 3D space in order to decide the front face. The clockwise or counterclockwise direction describes which side is the front and which is the back. OpenGL lets you specify these rules with the *glFrontFace* command. The Following code denotes that counter clockwise direction of the polygons is considered to be Front Face.

```
glFrontFace(GL_CCW);
```

glMaterial command should be called prior to defining the polygon's vertices to apply these material properties to the surface.

A convenient alternative to *glmaterial* is color tracking. Material properties are specified by merely calling the *glColor* command prior to each object or polygon. Also it has to be enabled.

```
glEnable(GL_COLOR_MATERIAL);  
glColorMaterial(GL_FRONT, GL_AMBIENT_AND_DIFFUSE);  
glColor3f(0.0f, 0.0f, 1.0f); // blue reflective properties
```

Example: This code would typically be placed with the OpenGL initialization code

```
//set the global lighting / shading  
glShadeModel(GL_SMOOTH); // or GL_FLAT  
glEnable(GL_NORMALIZE);  
glEnable(GL_LIGHTING);  
  
//set the global ambient light  
GLfloat ambient = {.2, .2, .2, 1};  
glLightModelfv(GL_LIGHT_MODEL_AMBIENT, globalAmb);  
  
//set up a light and enable it  
GLfloat diffuse[] = {1, 0, 0, 1};  
GLfloat ambient[] = {.5, 0, 0, 1};  
GLfloat specular[] = {1, 1, 1, 1};
```

```

glLightfv(GL_LIGHT0, GL_DIFFUSE, diffuse);
glLightfv(GL_LIGHT0, GL_AMBIENT, ambient);
glLightfv(GL_LIGHT0, GL_SPECULAR, specular);
glEnable(GL_LIGHT0); //enable the light

//set light position
// set last term to 0 for a spotlight
GLfloat lightpos[] = {1, 1, 1, 1};
glLightfv(GL_LIGHT0, GL_POSITION, lightpos);

//This code sets a simple material property
GLfloat ambient[] = {.5, 0, 0, 1};
GLfloat specular[] = {1, 1, 1, 1};

//set params for front and back separately (GL_BACK, GL_FRONT_AND_BACK)
glMaterialfv(GL_FRONT, GL_AMBIENT_AND_DIFFUSE, ambient);
glMaterialfv(GL_FRONT, GL_SPECULAR, specular);

```

Example2: A Cube in an environment with two diffuse lights and an ambient light. Two Diffuse lights are of different colours (blue & green) whereas the ambient light is red colour.

```

GLfloatDiffuseLight1[] = {0, 0, 1};
GLfloatDiffuseLight2[] = {0, 1, 0};
GLfloatAmbientLight[] = {1, 0, 0};
glLightfv(GL_LIGHT0, GL_DIFFUSE, DiffuseLight1);
glLightfv(GL_LIGHT1, GL_AMBIENT, AmbientLight);
glLightfv(GL_LIGHT2, GL_DIFFUSE, DiffuseLight2);
GLfloatLightPosition1[] = {0, 0, 3, 0};
GLfloatLightPosition2[] = {3, 0, 0, 0};
glLightfv(GL_LIGHT0, GL_POSITION, LightPosition1);
glLightfv(GL_LIGHT2, GL_POSITION, LightPosition2);
gluLookAt(3.0, -2.0, 4.0, 0.0, 0.0, 0.0, 0.0, 1.0, 0.0);

```

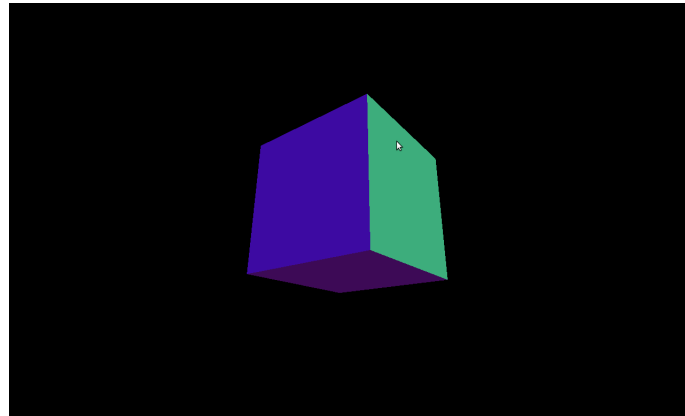


Figure 2: Output of Example2

You can see three colours on three sides of a cube. There is one diffuse light at position (3,0,0) and the other at (0,0,3). There is also an Ambient light of red colour. The eye position is at (3,-2,4) and is looking at (0,0,0) i.e., the center of cube.

Texture Mapping

Another feature in OpenGL is Texture Mapping feature where you can apply the textures to your geometry. Once a texture is uploaded to the video memory it can be used throughout the program. There are certain steps to be followed before a texture is readily available to the program. We first need a texture name. This is essentially a number that OpenGL uses to index all the different textures.

```
GLuint texture; // allocate a texture name  
glGenTextures(1,&texture); //get a free texture id
```

Now that we have our texture name, it has to be bound before doing anything to it. Note that there are two forms of textures in OpenGL, 1D and 2D. You can load different textures, however only one is selected at a time.

```
// select our current texture  
glBindTexture(GL_TEXTURE_2D,texture);
```

Now we need to set some texture parameters and load the texture data on the current texture. OpenGL has four texture parameters to setup. Here, it defines several effects like bilinear, trilinear texture filtering, and mipmapping. We also can define whether the texture wraps over at the edges or is clamped at the ends.

```
// the texture wraps over at the edges (repeat)  
glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_S, GL_REPEAT);  
glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_T, GL_REPEAT);  
  
// when texture area is large, bilinear filter the original  
glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER, GL_LINEAR);  
glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER, GL_LINEAR);
```

The default state of *MIN_FILTER* is *GL_LINEAR_MIPMAP_NEAREST*, if it is not defined. In such a case, the texture is considered incomplete and it renders a white texture on the object.

Also we need to set environment variables for the current texture. This tells the OpenGL how the texture should act when it is rendered into a scene.

```
glTexEnvf(GL_TEXTURE_ENV, GL_TEXTURE_ENV_MODE, GL_MODULATE);
```

Here it sets the active texture to *GL_MODULATE*. This attribute allows to apply effects such as lighting and colouring to your texture. If you would like to display the texture unchanged then replace it with *GL_DECAL*.

After all these parameters are set, OpenGL calls *glTexImage2D* that will upload the texture to the video memory and will be ready for us to use in our programs.

```
glTexImage2D(GL_TEXTURE_2D, level, internalFormat, width, height,  
border, format, type, pixels);
```

- *internalFormat* - This tells OpenGL how many colour components are needed to represent internally from the texture that is uploaded. ex: *GL_RGB*
- *format* - Format of the pixel data that will be uploaded. ex: *GL_RGB*
- *type* - Type of data that will be uploaded. ex: *GL_UNSIGNED_BYTE*
- *pixels* - Pointer to the image data.

Note that after your call to *glTexImage2D* you can free this memory with *free* function since the texture is already uploaded into video memory. A good alternative to *glTexImage2D* is to build your texture mipmaps. This can be done by:

```
gluBuild2DMipmaps(GL_TEXTURE_2D, 3, width, height, GL_RGB,  
GL_UNSIGNED_BYTE, data);
```

Now the texture is ready to be applied to your geometry, with all the above parameters set. Remember Texturing has to be enabled.

Example: Texture Quad

```
// enable texturing  
glEnable(GL_TEXTURE_2D);  
glBegin(GL_QUADS);  
glTexCoord2d(0.0, 0.0); glVertex2d(0.0, 0.0);  
glTexCoord2d(1.0, 0.0); glVertex2d(1.0, 0.0);  
glTexCoord2d(1.0, 1.0); glVertex2d(1.0, 1.0);  
glTexCoord2d(0.0, 1.0); glVertex2d(0.0, 1.0);  
glEnd();
```

References

- OpenGL Tutorials - [http : //www.swiftless.com/tutorials/opengl/opengl_tuts.html](http://www.swiftless.com/tutorials/opengl/opengl_tuts.html)
- Texture Mapping - [http : //www.opengl.org/wiki/Texture_Mapping](http://www.opengl.org/wiki/Texture_Mapping)
- OpenGL Lighting - [http : //www.falloutssoftware.com/tutorials/gl/gl8.htm](http://www.falloutssoftware.com/tutorials/gl/gl8.htm)
- OpenGL Texture Mapping - [http : //www.gamedev.net/reference/articles/article947.asp](http://www.gamedev.net/reference/articles/article947.asp)
- OpenGL Texture Tutorial - [http : //www.nullterminator.net/gltexture.html](http://www.nullterminator.net/gltexture.html)