Computer Graphics Laboratory
Lab – 03
Applying Material Properties on Objects and Lights in a Scene
Version-1.0

Continuing from our previous laboratory, we created a 3D pyramid and place a viewer with 3D projection function. The demo program of the last lab is in **Lab 03 demo_main.cpp** file. Try to open a GLUT project and run the given file. We will see a white pyramid as we did not apply any color to the pyramid. Use key R and S to rotate the pyramid.

Today we apply material property on the pyramid rather than applying color on each vertex and also we will apply light in the scene. Before practically applying material properties, we need to learn some basics of material properties, lighting attributes etc.

Real-World and OpenGL Lighting [Ref: Redbook]
When you look at a physical surface, your eye's perception of the color depends on the distribution of photon energies that arrive and trigger your cone cells. Those photons come from a light source or combination of sources, some of which are absorbed and some of which are reflected by the surface. In addition, different surfaces may have very different properties -some are shiny and preferentially reflect light in certain directions, while others scatter incoming light equally in all directions. Most surfaces are somewhere in between.

OpenGL approximates light and lighting as if **light can be broken into red, green, and blue components**. Thus, the color of light sources is characterized by the amount of red, green, and blue light they emit, and the material of surfaces is characterized by the percentage of the incoming red, green, and blue components that is reflected in various directions.

In the OpenGL model, the light sources have an effect only when there are surfaces that absorb and reflect light. Each surface is assumed to be composed of a material with various properties. A material might emit its own light (like headlights on an automobile), it might scatter some incoming light in all directions, and it might reflect some portion of the incoming light in a preferential direction like a mirror or other shiny surface.

The OpenGL lighting model considers the lighting to be divided into four independent components: emissive, **ambient, diffuse, and specular**. All four components are computed independently and then added together.

Ambient, Diffuse, and Specular Light [Ref: Redbook]
Ambient illumination is light that's been scattered so much by the environment that its direction is impossible to determine - it seems to come from all directions. When ambient light strikes a surface, it's scattered equally in all directions.

The **diffuse** component is the light that comes from one direction, so it's brighter if it comes squarely down on a surface than if it barely glances off the surface. Once it hits a surface, however, it's scattered equally in all directions, so it appears equally bright, no matter where the eye is located. Any light coming from a particular position or direction probably has a diffuse component.

Finally, **specular** light comes from a particular direction, and it tends to bounce off the surface in a preferred direction. A well-collimated

laser beam bouncing off a high-quality mirror produces almost 100 percent specular reflection. Shiny metal or plastic has a high specular component, and chalk or carpet has almost none. You can think of specularity as shininess.

Although a light source delivers a single distribution of frequencies, the ambient, diffuse, and specular components might be different. For example, if you have a white light in a room with red walls, the scattered light tends to be red, although the light directly striking objects is white. OpenGL allows you to set the red, green, and blue values for each component of light independently.

Material Colors [Ref: Redbook]
The OpenGL lighting model makes the approximation that a material's color depends on the percentages of the incoming red, green, and blue light it reflects. For example, a perfectly red ball reflects all the incoming red light and absorbs all the green and blue light that strikes it. If you view such a ball in white light (composed of equal amounts of red, green, and blue light), all the red is reflected, and you see a red ball. If the ball is viewed in pure red light, it also appears to be red. If, however, the red ball is viewed in pure green light, it appears black (all the green is absorbed, and there's no incoming red, so no light is reflected).

Like lights, materials have different ambient, diffuse, and specular colors, which determine the ambient, diffuse, and specular reflectances of the material. A material's ambient reflectance is combined with the ambient component of each incoming light source, the diffuse reflectance with the light's diffuse component, and similarly for the specular reflectance and component. Ambient and diffuse reflectances define the color of the material and are typically similar if not identical. Specular reflectance is usually white or gray, so that specular highlights end up being the color of the light source's specular intensity. If you think of a white light shining on a shiny red plastic sphere, most of the sphere appears red, but the shiny highlight is white.

In addition to ambient, diffuse, and specular colors, materials have an emissive color, which simulates light originating from an object. In the OpenGL lighting model, the emissive color of a surface adds intensity to the object, but is unaffected by any light sources. Also, the emissive color does not introduce any additional light into the overall scene.

Now let's try to apply material properties and light on the scene. First we will apply material properties to our created pyramid. Lets see the following code segment:

```
GLfloat no_mat[] = { 0.0, 0.0, 0.0, 1.0 };
GLfloat mat_ambient[] = { 0.5, 0.0, 0.0, 1.0 };
GLfloat mat_diffuse[] = { 1.0, 0.0, 0.0, 1.0 };
GLfloat mat_specular[] = { 1.0, 1.0, 1.0, 1.0 };
GLfloat mat_shininess[] = {10};

glMaterialfv( GL_FRONT, GL_AMBIENT, mat_ambient);
glMaterialfv( GL_FRONT, GL_DIFFUSE, mat_diffuse);
glMaterialfv( GL_FRONT, GL_SPECULAR, mat_specular);
glMaterialfv( GL_FRONT, GL_SHININESS, mat_shininess);
```

Here we first define the reflectance properties of four color component for the material. To apply these reflectance properties as material properties we use function glMaterialfv) which has three parameters.

1. Specifies whether the front, back, or both material properties of the polygons are being set by this function. May be either GL_FRONT, GL_BACK, or GL_FRONT_AND_BACK.
2. Specifies the single-valued material parameter being set for the first two variations. Currently, the only single-valued material parameter is GL_SHININESS. The second two variations, which take arrays for their parameters, can set the following material properties: GL_AMBIENT, GL_DIFFUSE, GL_SPECULAR, GL_EMISSION, GL_SHININESS etc.
3. Specifies the value to which the parameter specified by parameter 2.

Before using these code segment, we need to enable lighting features by using **glEnable(GL_LIGHTING)** function in the main function. Now use the above code segment (drawpyramid()) where you created the pyramid as this properties only applicable to pyramid. **Now run the program. You will see dim red pyramid as we still do not add any light in the scene**. This is because of ambient property.

At this stage we need to add one or more light source in the scene. We can add at most 8 lights simultaneously in our scene. The code segment below adds one light in the scene.

```
GLfloat no_light[] = { 0.0, 0.0, 0.0, 1.0 };
GLfloat light_ambient[]  = {1.0, 1.0, 1.0, 1.0};
GLfloat light_diffuse[]  = { 1.0, 1.0, 1.0, 1 };
GLfloat light_specular[] = { 1, 1, 1, 1 };
GLfloat light_position[] = { 2.0, 25.0, 3.0, 1.0 };

glEnable( GL_LIGHT0);
                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);
```

Here, first we define the four color component for light No. 0 (zero). Then define the light position using x,y,z values and the fourth parameter of light position, **if 1.0, specifies that the light is at this position. Otherwise, the light source is directional and all rays are parallel**. Then enable light 0 by glEnable(GL_LIGHT0) function. Finally apply these color component on light 0 using glLightfv() function. It also has three parameters like glMaterialfv() function, but the first parameter is the light number to which we apply the color component. Now use this code segment globally (in main function) in your program. **Now run the program. You will see a bright red pyramid**.

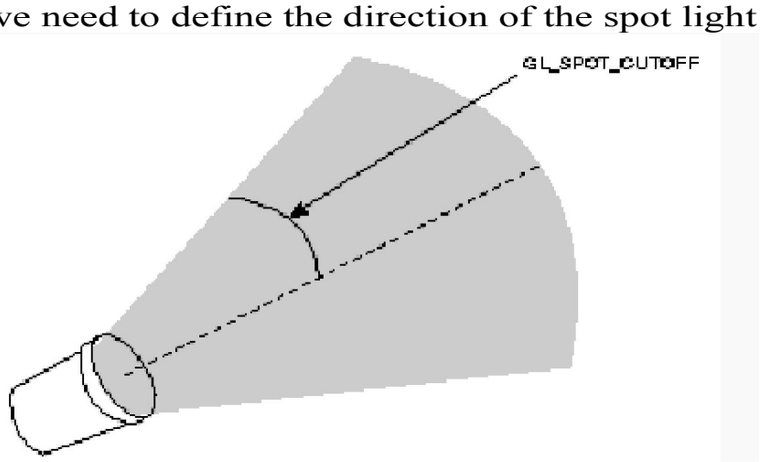
Now try to analyze above material properties and lighting properties. Omit any one or more color component properties and try to understand the effects. Also change the color component values for both material and light and see the effects and justify it with your basic knowledge about lighting.

Remember, the specular effects depends upon the light position and the viewer's position.

We can also add spot lights in our scene. This is very simple. We just need to add some more properties in the light to make it a spot light. Append the following code segment along with your light property code segment.

```
GLfloat spot_direction[] = { 0.0, -1.0, 0.0 };
glLightfv(GL_LIGHT0, GL_SPOT_DIRECTION, spot_direction);
glLightf( GL_LIGHT0, GL_SPOT_CUTOFF, 25.0);
```

Here, first we need to define the direction of the spot light like below.



Then set the GL_SPOT_CUTOFF to specify the angle between the axis of the cone and a ray along the edge of the cone. [Ref: Redbook]

You must have to set the direction of the spot light properly to focus your desired object.

Solve class work 01and 02 at this stage.