Unit-10

Introduction to OpenGL :

OpenGL is a software interface that allows a programmer to communicate with graphics hardware. OpenGL is the short form for "Open Graphics Library". It is an application programming interface (API) designed for rendering 2D and 3D graphics. It provides a common set of commands that can be used to manage graphics in different applications and on multiple platforms. Examples of OpenGL commands include drawing polygons, assigning colors to shapes, zooming in and out, rotating objects etc.

OpenGL Utility Toolkit (GLUT) has been created to aid in the development of more complicated 3D objects such as a sphere, a torus and even a teapot. Like all software libraries, OpenGL consists of a series of function calls that can be invoked from our own programs.

⊛. Why OpenGL ?

→ The first vendor-independent API for development of graphics applications.
→ There is no need to license it to use it.
→ It was designed to use the graphics card where possible to improve performance.
→ Originally based on a state machine, procedural model; thus it can be used with a wide variety of programming languages.

⊛. Callback Functions :-

A callback function is a function which the library (GLUT) calls when it needs to know how to process something. Eg. When GLUT gets a key down event it uses the glutKeyboardFunc callback routine to find out what to do with a key press. The following are some valid callback functions :-

Callback	Description
GLU_TESS_BEGIN	The GLU_TESS_BEGIN callback is invoked like glBegin to indicate the start of a (triangle) primitive. The function takes a single argument of type GLenum.
GLU_TESS_BEGIN_DATA	GLU_TESS_BEGIN_DATA is same as GLU_TESS_BEGIN callback except that it takes an additional pointer argument.
GLU_TESS_EDGE_FLAG	The GLU_TESS_EDGE_FLAG callback is similar to glEdgeFlag. This function takes a single boolean flag that indicates which edges lay on the polygon boundry
GLU_TESS_EDGE_FLAG_DATA	The GLU_TESS_EDGE_FLAG_DATA callback is the same as GLU_TESS_EDGE_FLAG callback except it takes an additional pointer argument.
GLU_TESS_END	The GLU_TESS_END callback indicates the end of a primitive, and it takes no arguments.

⊛. Color Commands:-

OpenGL has two color models: the RGBA mode and color index mode. In RGBA mode, a color is specified by three intensities (for the Red, Green and Blue components of the color) and optionally a fourth value, Alpha, which controls transparency. The function, glColor4f (red, green, blue, alpha)

maps available red, green, and blue intensities onto (0.0, 1.0) where 0.0 means that the color component is absent (0.0, 0.0, 0.0) which indicates black and 1.0 is a saturated color (1.0, 1.0, 1.0) which indicates white.

The number of bits used to represent each color depends upon the graphics card. Current graphics cards have several Mbytes of memory and use 24 or 32 bits for color. The term bit plane refers to an image of single-bit values. Thus a system of 24 bits of color has 24 bit planes.

⊛. Drawing pixels, lines, polygons using OpenGL:-

Creating Lines in OpenGL → In OpenGL, the term line refers to a line segment. There are easy ways to specify a connected series of line segments, or even a closed, connected series of segments. In all cases the lines constituting the connected series are specified in terms of the vertices at their endpoints.

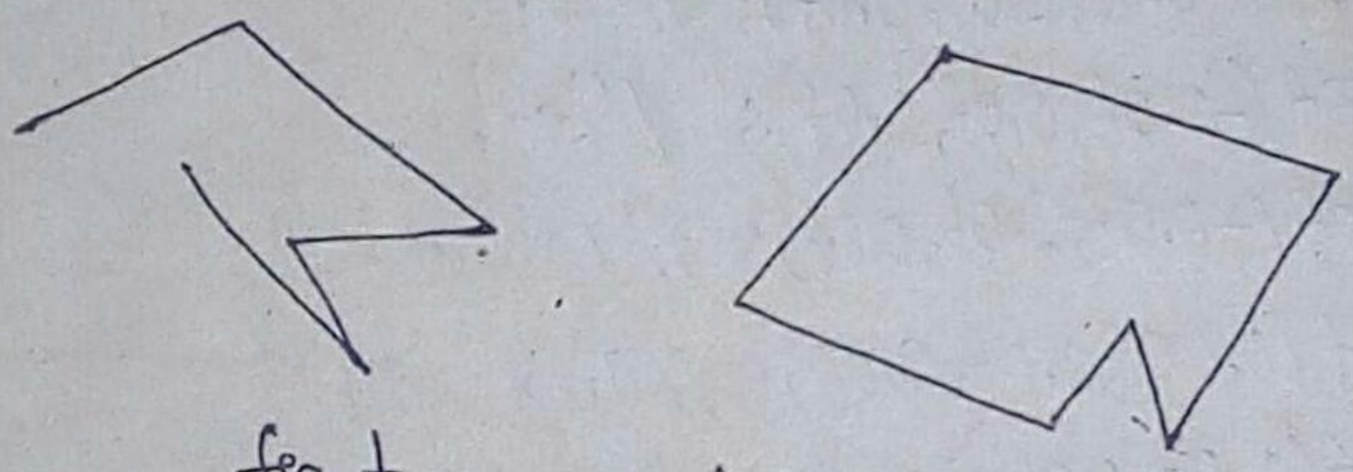

fig. two connected series of line segments.

With OpenGL, we can specify lines with different widths and lines that are stippled in various ways - dotted, dashed and so on. Drawing a line with OpenGL is apparently quite simple using the following code.

```
glBegin(GL_LINES);
    glVertex2i(x0, y0);
    glVertex2i(x1, y1);
glEnd();
```

Creating Polygons in OpenGL → Polygons are typically drawn by filling in all the pixels enclosed within the boundry, but we can also draw them as outlined polygons or simply points at the vertices. A filled polygon might be solidly filled or stippled with a certain pattern.

A polygon has two sides - front and back and might be rendered differently depending on which side is facing the viewer. This allows us to have cutaway views of solid objects in which there is an obvious distinction between the parts that are inside and those that are outside. By default, both front and back faces are drawn in the same way. To change this, or to draw only outlines or vertices, use

glPolygonMode().

We can draw polygon by using following code segment;

```
glBegin(GL_POLYGON); //Draw a Quad
glVertex3f(-0.5f, 1.0f, 0.0f); //Top.
glVertex3f(-1.0f, 0.0f, 0.0f); //Left.
glVertex3f(-0.5f, 1.0f, 0.0f); // Bottom Left.
glVertex3f(0.05f, 1.0f, 0.0f); // Top right
glVertex3f(1.0f, 0.0f, 0.0f); // Right
glVertex3f(0.5f, 1.0f, 0.0f); // Bottom Right
glEnd();
```

⊛. Basic Lighting:- Lighting in the real world is extremely complicated and depends on way to many factors. Lighting in OpenGL is based on approximations of reality using simplified models that are much easier to process and looks relatively similar. These lighting models are based on the physics of light as we understand it. One of those models is called the Phong lighting model. The major building blocks of the Phong model consists of following three components;

i) Ambient lighting → Even when it is dark there is usually still some light somewhere in the world (the moon, a distant light) so objects are ~~never~~ almost never completely dark. To simulate this we use an ambient lighting constant that always gives the object some color. Ambient illumination is light that is been scattered so much by the environment that its direction is imposible to determine. It seems to come from all directions.

ii) Diffuse Lighting → This is the most visually significant component of the lighting model. The more a part of an object faces the light source, the brighter it becomes. The diffuse component is the light that comes from one direction, so its brighter if it comes squarely down on a surface than if it barely glances off the surface. Once it hits a surface it is scattered equally in all directions, so it appears equally bright.

iii) Specular lighting → It simulates the bright spot of a light that appears on shiny objects. Specular highlights are often more inclined to the color of the light than the color of the object. Specular light comes from a particular direction, and it tends to bounce off the surface in a preferred direction. Shiny metal or plastic has a high specular component.