Assignment No 1

Problem Statement:
Install and explore the OpenGL

Objective:
To understand the basic concepts of OpenGl and its installation

Outcome:
To implement the basic concept of OpenGl.

CO Relevance: CO1

PO/PSOs Relevance: PO1, PO2, PO5, PO6

Theory Concepts:

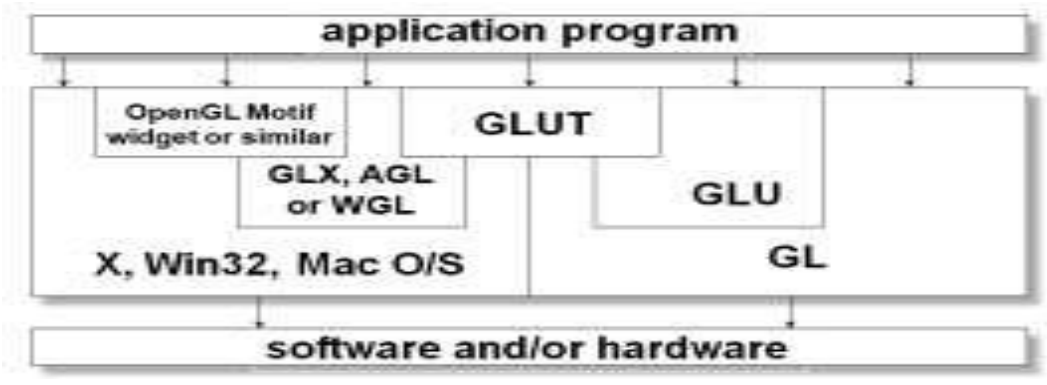
What is OpenGL?

OpenGL is a library of function calls for doing computer graphics. With it, one can create interactive applications that render high-quality color images composed of 3D geometric objects and images. The current version of OpenGL is 4.5, released on August 11, 2014, and is the eightieth revision since the original version 1.0

Software Organization in Open GL

The OpenGL API is window and operating system independent. That means that the part of the application that draws can be platform independent. However, in order for OpenGL to be able to render, it needs a window to draw into. Generally, this is controlled by the windowing system on whatever platform you are working on.

Software Organization



The OpenGL Interface consists of functions in three libraries.
i. OpenGL core library: OpenGL32 on Windows and GL on most unix/linux systems (libGL.a)
ii. OpenGL Utility Library (GLU): Provides functionality in OpenGL core but avoids having to rewrite code
iii. Links with window system: GLX for X window systems, WGL for Windows, AGL for Macintosh

4. OpenGL Utility Toolkit (GLUT)

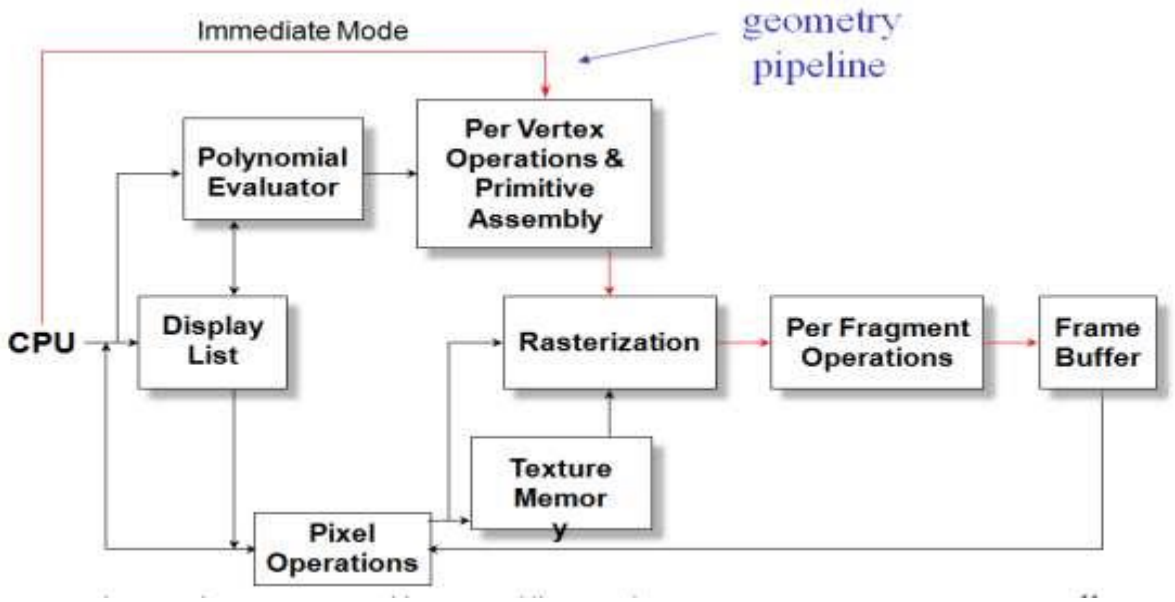
GLUT provides functionality common to all window systems such as opening a window, getting input from mouse and keyboard, menus. GLUT is event-driven and its code is portable.
GLUT is designed for constructing small to medium sized OpenGL programs. But GLUT lacks the functionality of a good toolkit for a specific platform. For example, it does not provide slide bars.

So, large applications requiring sophisticated user interfaces are better off using native window system toolkits. The GLUT library has C, C++ (same as C), FORTRAN, and Ada programming bindings. The GLUT source code distribution is portable to nearly all OpenGL implementations and platforms. The current version is 3.7. The toolkit supports multiple windows for OpenGL rendering, callback driven event processing, sophisticated input devices, an 'idle' routine and timers, a simple, cascading pop-up menu facility, utility routines to generate various solid and wire frame objects, support for bitmap and stroke fonts.

5. OpenGL Pipeline Architecture

Generally data flows from an application through the GPU to generate an image in the frame buffer. The application will provide vertices, which are collections of data that are composed to form geometric objects, to the OpenGL pipeline. The vertex processing stage uses a vertex shader to process each vertex, doing any computations necessary to determine where in the frame buffer each piece of geometry should go. The other shading stages like tessellation and geometry shading are also used for vertex processing.

OpenGL Pipeline



Primitives and Attributes

OpenGL Primitive	Description	Total Vertices for *n* Primitives
GL_POINTS	Render a single point per vertex (points may be larger than a single pixel)	n
GL_LINES	Connect each pair of vertices with a single line segment.	2n
GL_LINE_STRIP	Connect each successive vertex to the previous one with a line segment.	n+1
GL_LINE_LOOP	Connect all vertices in a loop of line segments.	n
GL_TRIANGLES	Render a triangle for each triple of vertices.	3n
GL_TRIANGLE_STRIP	Render a triangle from the first three vertices in the list, and then create a new triangle with the last two rendered vertices, and the new vertex.	n+2
GL_TRIANGLE_FAN	Create triangles by using the first vertex in the list, and pairs of successive vertices.	n+2

Commonly Required OpenGL Functions

void glVertex[2/3/4][sifd] (TYPE xcoordinate, TYPE ycoordinate, …)

void glVertex[2/3/4]v (TYPE *coordinates)

Specifies the position of a vertex in 2,3, or 4 dimensions. The coordinates can be specified as short s, int i, float f, or double d. If the v is present, the argument is a pointer to n array containing the coordinates. glVertex commands are used within glBegin/glEnd pairs to specify point, line, and polygon vertices. The current color, normal, texture coordinates, and fog coordinate are associated with the vertex when glVertex is called. When only x and y are specified, z defaults to 0 and w defaults to 1. When x, y, and z are specified, w defaults to 1.

void glBegin(GLenum ***mode*****)**;

Delimits the vertices of a primitive or a group of like primitives.
mode: Specifies the primitive or primitives that will be created from vertices presented between glBegin and the subsequent glEnd. Ten symbolic constants are accepted.
GL_POINTS, GL_LINES, GL_LINE_STRIP, GL_LINE_LOOP, GL_TRIANGLES, GL_TRIANGLE_STRIP, GL_ TRIANGLE_FAN, GL_QUADS, GL_QUAD_STRIP, andGL_POLYGON**void glEnd()** terminates a list of vertices

void glRect[sifd] (TYPE x1, TYPE y1, TYPE x2, TYPE y2)
void glRect[sifd]v(TYPE *v1, TYPE *v2)
Specify a two dimensional axis-aligned rectangle by the x, y coordinates of the diagonally opposite vertices (or pointers to the vertices) using the standard data types.

b. Specify Color
void glColor[34][b i f d ub us ui] (TYPE red,TYPE green,TYPE blue,TYPE alpha)
void glColor[34][b I f d ub us ui]v (TYPE *color)

*red, green, blue:*Specify new red, green, and blue values for the current color
Alpha: Specifies a new alpha value for the current color. Included only in the four argument glColor4 commands.
v: Specifies a pointer to an array that contains red, green, blue, and (sometimes) alpha values. The initial value for the current color is (1, 1, 1, 1). The current color can be updated at any time. In particular, glColor can be called between a call to glBegin and the corresponding call to glEnd.

void glClearColor(GLClampf r, GLClampf g, GLClampf b, GLClampf a)
specifies the red, green, blue, and alpha values used by glClear to clear the color buffers. The initial values are all 0. Values specified by glClearColor are clamped to the range [0, 1].

void glIndexi[s i f d ub] (TYPE index)
set the current color index.
index : Specifies a pointer to a one-element array that contains the new value for the current color index.

---

void glutSetColor(int cell,GLfloat red, GLfloat green, GLfloat blue);
cell: Color cell index (starting at zero).
Sets the cell color index colormap entry of the *current window*'s logical colormap for the *layer in use* with the color specified by red, green, and blue. The *layer in use* of the *current window* should be a color index window. cell should be zero or greater and less than the total number of colormap entries for the window. If the *layer in use*'s colormap was copied by reference, a glutSetColor call will force the duplication of the colormap.

void glPointSize(GLfloat size);
Size: Specifies the diameter of rasterized points. The initial value is 1.glPointSize specifies the rasterized diameter of both aliased and antialiased points. Using a point size other than 1 has different effects, depending on whether point antialiasing is enabled. To enable and disable point antialiasing, call glEnable and glDisable with argument GL_POINT_SMOOTH. Point antialiasing is initially disabled.

WORKING WITH WINDOW SYSTEM

void glFlush(*void*);

Different GL implementations buffer commands in several different locations, including network buffers and the graphics accelerator itself. glFlush empties all of these buffers, causing all issued commands to be executed as quickly as they are accepted by the actual rendering engine. Though this execution may not be completed in any particular time period, it does complete in finite time.

void glutInit(int *argcp, char **argv);
glutInit is used to initialize the GLUT library.

argcp

A pointer to the program's *unmodified* argc variable from main. Upon return, the value pointed to by argcp will be updated, because glutInit extracts any command line options intended for the GLUT library.

argv

The program's *unmodified* argv variable from main. Like argcp, the data for argv will be updated because glutInit extracts any command line options understood by the GLUT library. glutInit will initialize the GLUT library and negotiate a session with the window system. During this process, glutInit may cause the termination of the GLUT program with an error message to the user if GLUT cannot be properly initialized. Examples of this situation include the failure to connect to the window system, the lack of window system support for OpenGL, and invalid command line options. glutInit also processes command line options, but the specific options parse are window system dependent.

int glutCreateWindow(char *name);
name: ASCII character string for use as window name.glutCreateWindow creates a top-level window. The name will be provided to the window system as the window's name. The intent is that the window system will label the window with the name. Implicitly, the *current window* is set to the newly created window. Each created window has a unique associated OpenGL context. State changes to a window's associated OpenGL context can be done immediately after the window is created. The *display state* of a window is initially for the window to be shown. Until glutMainLoop is called, rendering to a created window is ineffective because the window can not yet be displayed. The value returned is a unique small integer identifier for the window. The range of allocated identifiers starts at one. This window identifier can be used when calling glutSetWindow.

void glutInitDisplayMode(unsigned int mode);
glutInitDisplayMode sets the *initial display mode*.
mode: Display mode, normally the bitwise *OR*-ing of GLUT display mode bit masks below:
GLUT_RGBA: Bit mask to select an RGBA mode window. This is the default if either
GLUT_RGBA nor GLUT_INDEX are specified.
GLUT_RGB: An alias for GLUT_RGBA.
GLUT_INDEX: Bit mask to select a color index mode window. This overrides GLUT_RGBA if
it is also specified.
GLUT_SINGLE: Bit mask to select a single buffered window. This is the default if
neither GLUT_DOUBLE or GLUT_SINGLE are specified.
GLUT_DOUBLE: Bit mask to select a double buffered window. This overrides
GLUT_SINGLE if it is also specified.
GLUT_ACCUM: Bit mask to select a window with an accumulation buffer.
GLUT_ALPHA: Bit mask to select a window with an alpha component to the color buffer(s).

GLUT_DEPTH: Bit mask to select a window with a depth buffer.
GLUT_STENCIL: Bit mask to select a window with a stencil buffer.
Also used are GLUT_MULTISAMPLE, GLUT_STEREO, GLUT_LUMINANCE

void glutInitWindowSize(int width, int height);
void glutInitWindowPosition(int x, int y);
glutInitWindowPosition and glutInitWindowSize set the *initial position* and *size* respectively.
Width: Width in pixels.Height: Height in pixels.
X: Window X location in pixels.
Y: Window Y location in pixels.
Windows created by glutCreateWindow will be requested to be created with the current *initial window position* and *size*. The initial value of the *initial window position* GLUT state is -1 and -1. If either the X or Y component to the *initial window position* is negative, the actual window position is left to the window system to determine. The initial value of the *initial window size* GLUT state is 300 by 300. The *initial window size* components must be greater than zero.

void **glViewport**(Glint *x*, GLint *y*, GLsizei *width*, GLsizei *height*);
Sets the viewport. *x, y:*Specify the lower left corner of the viewport rectangle, in pixels. The initial value is (0,0). *width, height:* Specify the width and height of the viewport. When a GL context is first attached to a window, *width* and *height* are set to the dimensions of that window.

void glutMainLoop(void);
glutMainLoop enters the GLUT event processing loop. This routine should be called at most once in a GLUT program. Once called, this routine will never return. It will call as necessary any callbacks that have been registered.

void glutDisplayFunc(void (*func)(void));
func: The new display callback function. glutDisplayFunc sets the display callback for the *current window*. GLUT determines when the display callback should be triggered based on the window's redisplay state. The redisplay state for a window can be either set explicitly by calling glutPostRedisplay or implicitly as the result of window damage reported by the window system. Multiple posted redisplays for a window are coalesced by GLUT to minimize the number of display callbacks called.

void glutPostRedisplay(void);
Mark the normal plane of *current window* as needing to be redisplayed. The next iteration through glutMainLoop, the window's display callback will be called to redisplay the window's normal plane. Multiple calls to glutPostRedisplay before the next display callback opportunity generates only a single redisplay callback.

void glutSwapBuffers(void);
glutSwapBuffers swaps the buffers of the *current window* if double buffered.
Specifically, glutSwapBuffers promotes the contents of the back buffer of the *layer in use* of the *current window* to become the contents of the front buffer. The contents of the back buffer then become undefined.

void glutSetWindow(int win);
int glutGetWindow(void);
win: Identifier of GLUT window to make the *current window*.
glutSetWindow sets the *current window*; glutGetWindow returns the identifier of the *current window*. If no windows exist or the previously *current window* was destroyed, glutGetWindow returns zero.

INTERACTION

void glutMouseFunc(void (*func)(int button, int state, int x, int y));

func:The new mouse callback function. glutMouseFunc sets the mouse callback for the *current window*. When a user presses and releases mouse buttons in the window, each press and each release generates a mouse callback. The button parameter is one of GLUT_LEFT_BUTTON, GLUT_MIDDLE_BUTTON, or GLUT_RIGHT_BUTTON.

The state parameter is either GLUT_UP or GLUT_DOWN indicating whether the callback was due to a release or press respectively. The x and y callback parameters indicate the window relative coordinates when the mouse button state changed. Passing NULL to glutMouseFunc disables the generation of mouse callbacks.

void glutReshapeFunc(void (*func)(int width, int height));

glutReshapeFunc sets the reshape callback for the *current window*. The reshape callback is triggered when a window is reshaped, immediately before a window's first display callback after a window is created. The width and height parameters of the callback specify the new window size in pixels. Before the callback, the *current window*is set to the window that has been reshaped.If a reshape callback is not registered for a window or NULL is passed to glutReshapeFunc (to deregister a previously registered callback), the default reshape callback is used. This default callback will simply call glViewport(0,0,width,height) on the normal plane (and on the overlay if one exists).

void glutKeyboardFunc(void (*func)(unsigned char key, int x, int y));

glutKeyboardFunc sets the keyboard callback for the *current window*. When a user types into the window, each key press generating an ASCII character will generate a keyboard callback.

The x and y callback parameters indicate the mouse location in window relative coordinates when the key was pressed. When a new window is created, no keyboard callback is initially registered, and ASCII key strokes in the window are ignored. During a keyboard callback, glutGetModifiers may be called to determine the state of modifier keys when the keystroke generating the callback occurred.

void glutIdleFunc(void (*func)(void));

glutIdleFunc sets the global idle callback to be func so a GLUT program can perform background processing tasks or continuous animation when window system events are not being received.

void glutTimerFunc(unsigned int msecs, void (*func)(int value), value);

glutTimerFunc registers the timer callback func to be triggered in at least msecs milliseconds.

The value parameter to the timer callback will be the value of the value parameter to glutTimerFunc. GLUT attempts to deliver the timer callback as soon as possible after the expiration of the callback's time interval.

TRANSFORMATIONS

void glMatrixMode(GLenum *mode*);

mode: Specifies which matrix stack is the target for subsequent matrix operations. Three values are accepted: GL_MODELVIEW, GL_PROJECTION, and GL_TEXTURE. The initial value is GL_MODELVIEW.

glMatrixMode sets the current matrix mode.

mode can assume one of four values:

GL_MODELVIEW, GL_PROJECTION, GL_TEXTURE, GL_COLOR.

To find out which matrix stack is currently the target of all matrix operations, call glGet with argument GL_MATRIX_MODE. The initial value is GL_MODELVIEW.

void glLoadIdentity(void):glLoadIdentity replaces the current matrix with the identity matrix. It is semantically equivalent to calling glLoadMatrix with the identity matrix but in some cases it is more efficient.

void glPushMatrix (void);
void glPopmatrix(void);
Push and pop the current matrix. There is a stack of matrices for each of the matrix modes. In GL_MODELVIEW mode, the stack depth is at least 32. In the other modes, GL_COLOR, GL_PROJECTION, and GL_TEXTURE, the depth is at least 2. The current matrix in any mode is the matrix on the top of the stack for that mode. Initially, each of the stacks contains one matrix, an identity matrix.

void glRotate[df] (GLDouble angle, GLDouble x, GLDouble y, GLDouble z);
multiply the current matrix by a rotation matrix
angle - Specifies the angle of rotation, in degrees.
x, *y*, *z*- Specify the *x*, *y*, and *z* coordinates of a vector, respectively.
glRotate produces a rotation of *angle* degrees around the vector x y z .

void glScale[df] (GLDouble x, GLDouble y, GLDouble z);
glScale multiply the current matrix by a general scaling matrix
x, *y*, *z* Specify scale factors along the *x*, *y*, and *z* axes, respectively. glScale produces a nonuniform scaling along the *x*, *y*, and *z* axes. The three parameters indicate the desired scale factor along each of the three axes.

void glTranslate[df] (GLDouble x, GLDouble y, GLDouble z);
glTranslate multiply the current matrix by a translation matrix. *x*, *y*, *z:* Specify the *x*, *y*, and *z* coordinates of a translation vector.

void glMultMatrixd(const GLdouble m);
void glMultMatrixf(const GLfloat m);
glMultMatrix multiplies the current matrix with the one specified using *m*, and replaces the current matrix with the product.

void glLoadMatrixd(const GLdouble m);
void glLoadMatrixf(const GLfloat m);
glLoadMatrix replace the current matrix with the specified matrix. *m s*pecifies a pointer to 16 consecutive values, which are used as the elements of a 4 x 4 column-major matrix. The current matrix is the projection matrix, modelview matrix, or texture matrix, depending on the current matrix mode.

VIEWING

void glOrtho(GLdouble left, GLdouble right, GLdouble bottom, GLdouble top, GLdouble nearVal, GLdouble farVal);
glOrtho describes a transformation that produces a parallel projection. It multiplies the current matrix with an orthographic matrix. *left*, *right s*pecify the coordinates for the left and right vertical clipping planes. *bottom*, *top s*pecify the coordinates for the bottom and top horizontal clipping planes. *nearVal*, *farVal s*pecify the distances to the nearer and farther depth clipping planes. These values are negative if the plane is to be behind the viewer.

void gluOrtho2D (GLdouble left, GLdouble right, GLdouble bottom, GLdouble top);
define a 2D orthographic projection matrix. *left*, *right* - Specify the coordinates for the left and right vertical clipping planes. *bottom*, *top* - Specify the coordinates for the bottom and top horizontal clipping planes.

gluOrtho2D sets up a two-dimensional orthographic viewing region. This is equivalent to calling glOrtho with near = -1 and far = 1.

void gluLookAt(GLdouble eyeX, eyeY,eyeZ, centerX, centerY, centerZ, upX, upY, upZ);
eyeX, *eyeY*, *eyeZ* - Specifies the position of the eye point. *centerX*, *centerY*, *centerZ* - Specifies the position of the reference point. *upX*, *upY*, *upZ* - Specifies the direction of the *up* vector. gluLookAt creates a viewing matrix derived from an eye point, a reference point indicating the center of the scene, and an *UP* vector.

void gluPerspective(GLDouble fovy, GLDouble aspect, GLDouble zNear, GLDouble zFar);
gluPerspective specifies a viewing frustum into the world coordinate system and sets up a perspective projection matrix. *Fovy* specifies the field of view angle, in degrees, in the *y* direction. *Aspect* specifies the aspect ratio that determines the field of view in the *x* direction. The aspect ratio is the ratio of *x* (width) to *y* (height). *zNear* specifies the distance from the viewer to the near clipping plane (always positive). *zFar* specifies the distance from the viewer to the far clipping plane (always positive).

- **OpenGl Installation:**

- **Open Terminal by using following command**
  Ctrl+Alt+T

- **Go to root directory by using following command**
  su –

- **Then press enter**
  Enter password

- **Run the following commands to install OpenGL.**

```
$ sudo apt-get update
$ sudo apt-get install libglu1-mesa-dev freeglut3-dev mesa-common-dev
```

- **Now to test if OpenGl libraries are working fine on our Linux, we will create a C++ program and test it. For creating program type $ gedit on Terminal Then save the program in root directory with .cpp extension**

- **Now give the command below to compile your code**

```
$ g++ programname.cpp -o batchfilename -lglut -lGLU –lGL
```

- **Now run your OpenGl program with following command**

```
$ ./batchfilename
```

Output:
(Execute the program and attach the printout here)

Conclusion :

In This way we have studied that how to Install and explore the OpenGL.

Viva Questions:

1.What is openGL?

2.What libraries should we include to run the program of OpenGL?

3.Explain GLClear, GLClearColor,gluOrtho2D (with arguments).

4.Explain glutInit, glutInitDisplayMode(GLUT_SINGLE | GLUT_RGB).

5.Explain glutInitWindowPosition(), glutInitWindowSize(), glutCreateWindow().

Date:	
Marks obtained:	
Sign of course coordinator:	
Name of course Coordinator :	