**Assignment No: 1**

**Title / Objective:** Introduction to OpenGL

**Problem Statement:** Install OpenGL on Ubuntu operating system to implement assignments. Explore the OpenGL in built functions to get familiar with structure of OpenGL function and compilation & execution of code along with error handling.

**Course Outcome:** 214457.1 Apply line& circle drawing algorithms to draw the objects.

(Applying)

**Pre-requisites:** Basic Geometry, Coordinate system, Basic engineering knowledge, C/ C++

**Study Material:** (Blogs / Videos / Courses / Web Sites / Books / e-Books)

1. **For Prerequisites** Virtual Machine:

<https://www.youtube.com/watch?v=x5MhydijWmc>

[https://docs.microsoft.com/en-us/virtualization/hyper-v-on-windows/quickstart/quick-create-virtual-machine](https://docs.microsoft.com/en-us/virtualization/hyper-v-on-windows/quick-start/quick-create-virtual-machine) dual-boot machine:

<https://vitux.com/how-to-install-ubuntu-18-04-along-with-windows-10/><https://www.youtube.com/watch?v=-iSAyiicyQY>To Install gcc : [https://linuxconfig.org/how-to-install-g-the-c-compiler-on-ubuntu-18-04-bionicbeaver-linux](https://linuxconfig.org/how-to-install-g-the-c-compiler-on-ubuntu-18-04-bionic-beaver-linux)

1. **For Assignment**
2. <https://www.opengl.org/archives/resources/code/samples/s2001/notes/opengl.pdf>
3. <https://www3.ntu.edu.sg/home/ehchua/programming/opengl/cg_introduction.html>
4. <https://csciwww.etsu.edu/barrettm/4157/OpenGL%20Functions.htm>**Requirements:** Ubuntu, OpenGL, gcc

**Theory:**

**What is Open GL?**

* **OpenGL** (Open Graphics Library) is a cross-platform, hardware-accelerated, language-independent, industrial standard API for producing 3D (including 2D) graphics. Modern computers have dedicated GPU (Graphics Processing Unit) with its own memory to speed up graphics rendering.
* OpenGL itself isn’t a programming language, or a software library. It’s the **specification** of an Application. Programming Interface (API) for computer graphics programming. In other words, OpenGL defines a set of functions for doing computer graphics.
* **Features of OpenGL**
  + It provides 3D geometric objects, such as lines, polygons, triangle meshes, spheres, cubes, quadric surfaces, NURBS curves and surfaces;
  + It provides 3D modeling transformations, and viewing functions to create views of 3D scenes using the idea of a **virtual camera**;
  + It supports high-quality rendering of scenes, including hidden-surface removal, multiple light sources, material types, transparency, textures, blending, fog;
  + It provides display lists for creating graphics caches and hierarchical models.

It also supports the interactive “picking” of objects; o It supports the manipulation of images as pixels, enabling frame-buffer effects such as antialiasing, motion blur, depth of field and soft shadows.

# How to install Steps for Installation of Opengl in Ubuntu 14.04

**sudo apt-get update sudo apt-get install freeglut3 sudo apt-get install freeglut3-dev sudo apt-get install binutils-gold sudo apt-get install g++ cmake sudo apt-get install libglew-dev sudo apt-get install g++ sudo apt-get install mesa-common-dev sudo apt-get install build-essential sudo apt-get install libglew1.5-dev libglm-dev**

After your development libraries have been installed to get information about the OpenGL and GLX implementations running on a given X display.

glxinfo | grep OpenGL

To Compile and Execute the C / C++ Program

**: ~# g++ XXX.C -lglut -lGL -lGLEW -lGLU -o XXX**

**: ~#. /XXX**

**We use 3 sets of libraries in our OpenGL programs:**

1. Core OpenGL (GL): consists of hundreds of commands, which begin with a prefix "gl" (e.g., glColor, glVertex, glTranslate, glRotate). The Core OpenGL models an object via a set of geometric primitives such as point, line and polygon.
2. OpenGL Utility Library (GLU): built on-top of the core OpenGL to provide important utilities (such as setting camera view and projection) and more building models (such as qradric surfaces and polygon tessellation). GLU commands start with a prefix "glu" (e.g., gluLookAt, gluPerspective).
3. OpenGL Utilities Toolkit (GLUT): OpenGL is designed to be independent of the windowing system or operating system. GLUT is needed to interact with the Operating System (such as creating a window, handling key and mouse inputs); it also provides more building models (such as sphere and torus). GLUT commands start with a prefix of "glut" (e.g., glutCreatewindow, glutMouseFunc). GLUT is platform independent, which is built on top of platform-specific OpenGL extension such as GLX for X Window System, WGL for Microsoft Window, and AGL, CGL or Cocoa for Mac OS.
4. Quoting from the [opengl.org](http://www.opengl.org/resources/libraries/glut/): "GLUT is designed for constructing small to medium sized OpenGL programs. While GLUT is well-suited to learning OpenGL and developing simple OpenGL applications, GLUT is not a full-featured toolkit so large applications requiring sophisticated user interfaces are better off using native window system toolkits. *GLUT is simple, easy, and small.*"
5. Alternative of GLUT includes SDL, ....
6. OpenGL Extension Wrangler Library (GLEW): "GLEW is a cross-platform opensource C/C++ extension loading library. GLEW provides efficient run-time mechanisms for determining which OpenGL extensions are supported on the target platform." Source and pre-build binary available at <http://glew.sourceforge.net/>. A standalone utility called "glewinfo.exe" (under the "bin" directory) can be used to produce the list of OpenGL functions supported by your graphics system.

**Sample Code:**

#include <iostream> using namespace std; #include <GL/glut.h> void myInit()

{

glClearColor(1.0,1.0,1.0,0); glColor3f(0.0,0.0,0.0); glPointSize(4.0);

gluOrtho2D(0 , 640 , 0 , 480);

}

void MyDisplay()

{ glClear(GL\_COLOR\_BUFFER\_BIT | GL\_DEPTH\_BUFFER\_BIT);

glBegin(GL\_POINTS);

glVertex2d(100,100); glEnd(); glFlush(); }

int main(int argc, char \*\*argv)

{

glutInit(&argc,argv);

glutInitDisplayMode(GLUT\_SINGLE | GLUT\_RGB); glutInitWindowPosition(0,0); glutInitWindowSize(640,480); glutCreateWindow("Sample"); myInit();

glutDisplayFunc(MyDisplay);

glutMainLoop();

}

1. **#include <GL/glut.h>:** All OpenGL programs must start with this line, which accesses all the OpenGL include files: it pulls in all the function prototypes and other definitions used by OpenGL. Which is guaranteed to include "glu.h" (for GL Utility) and "gl.h" (for Core OpenGL).
2. **glClearColor(1.0,1.0,1.0,0); :** Set "clearing" or background color
3. **glColor3f(0.0,0.0,0.0); :** Color with which object to be generated Set "clearing" or background color Pixel Size
4. **gluOrtho2D(0 , 640 , 0 , 480); :** Specify Coordinate System
5. **glClear(GL\_COLOR\_BUFFER\_BIT | GL\_DEPTH\_BUFFER\_BIT); :** Clear the window
6. **glBegin(GL\_POINTS); glVertex2d(100,100);glEnd(); :** To plot single pixel
7. **glFlush();:** Force update of screen
8. **glutInit(&argc,argv);:** initializes GLUT, must be called before other GL/GLUT functions. It takes the same arguments as the main (). /**glutInit()** initializes the GLUT library, and it must be called before any other GLUT function. argcand argv should be the arguments of the application’s main() – **glutInit()** understands several commandline options,
9. **glutInitDisplayMode(GLUT\_SINGLE | GLUT\_RGB);:** creates an OpenGL window for rendering and interaction, with name displayed in its title bar.
10. **glutInitWindowPosition(0,0);:** positions the top-left corner of the initial window at

(*x*, *y*). The coordinates (*x*, *y*), in term of pixels, is measured in window coordinates,

i.e., origin (0, 0) is at the top-left corner of the screen; x-axis pointing right and y-axis pointing down. Window location where it will open

1. **glutInitWindowSize(640,480);:** specify size of output window
2. **glutCreateWindow("Sample"); :**give name to the window
3. **glutDisplayFunc(MyDisplay);:** registers the name of the callback function to be invoked when OpenGL needs to redisplay (or display for the first time) the contents of the window //Callback Functions: is a C function, written by the application programme the application never calls the callback function directly. Instead, the callback function is called **by OpenGL**, whenever OpenGL decides it needs to be called.
4. **glutMainLoop();** // this loop will carry on for as long as the program is running. Each time around the loop, GLUT checks to see if anything has changed since last time, and calls the appropriate callbackfunctions. glutDisplayFunc() registers the name of the callback function to be invoked when OpenGL needs to redisplay (or display for the first time) the contents of the window.

**Naming Convention for OpenGL Functions An OpenGL functions:**

1. Begins with lowercase gl (for core OpenGL), glu (for OpenGL Utility) or glut (for OpenGL Utility Toolkit).
2. Followed by the purpose of the function, in *camel case* (initial-capitalized), e.g., glColor to specify the drawing color, glVertex to define the position of a vertex.
3. Followed by specifications for the parameters, e.g., glColor3f takes three float parameters. glVectex2i takes two int parameters.

(This is needed as C Language does not support function overloading. Different versions of the function need to be written for different parameter lists.) Convention:

*returnType* gl*Function*[234][sifd] (*type value*, ...); // 2, 3 or 4 parameters *returnType* gl*Function*[234][sifd]v (*type* \**value*); // an array parameter

**OpenGL defines its own *data types*:**

1. typedef unsigned int GLenum;
2. typedef unsigned char GLboolean;
3. typedef unsigned int GLbitfield;
4. typedef void GLvoid;
5. typedef signed char GLbyte; /\* 1-byte signed \*/
6. typedef short GLshort; /\* 2-byte signed \*/
7. typedef int GLint; /\* 4-byte signed \*/
8. typedef unsigned char GLubyte; /\* 1-byte unsigned \*/
9. typedef unsigned short GLushort; /\* 2-byte unsigned \*/
10. typedef unsigned int GLuint; /\* 4-byte unsigned \*/
11. typedef int GLsizei; /\* 4-byte signed \*/
12. typedef float GLfloat; /\* single precision float \*/
13. typedef float GLclampf; /\* single precision float in [0,1] \*/
14. typedef double GLdouble; /\* double precision float \*/
15. typedef double GLclampd; /\* double precision float in [0,1] \*/

# Color

We use glColor function to set the *foreground color*, and glClearColor function to set the *background* (or *clearing*) color.

|  |  |
| --- | --- |
| 1. | void glColor3f(GLfloat *red*, GLfloat *green*, GLfloat *blue*) |
| 2. | void glColor3fv(GLfloat \**colorRGB*) |
| 3. | void glColor4f(GLfloat *red*, GLfloat *green*, GLfloat *blue*, GLfloat *alpha*) |
| 4. | void glColor4fv(GLfloat *\*colorRGBA*) |

5. void glClearColor(GLclampf *red*, GLclampf *green*, GLclampf *blue*, GLclampf *alpha*) // GLclampf in the range of 0.0f to 1.0f Notes:

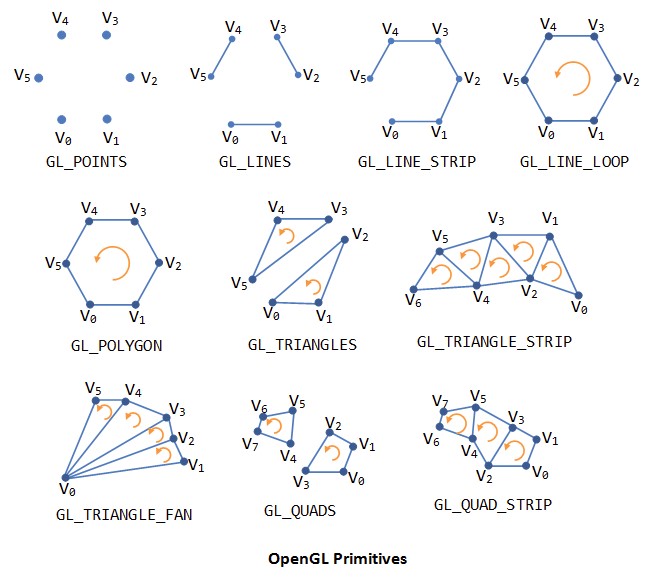
* Color is typically specified in float in the range 0.0f and 1.0f.
* Color can be specified using RGB (Red-Green-Blue) or RGBA (Red-Green-BlueAlpha) components. The 'A' (or alpha) specifies the transparency (or opacity) index, with value of 1 denotes opaque (non-transparent and cannot see-thru) and value of 0 denotes total transparent. We shall discuss alpha later.

In the above example, we set the background color via glClearColor in initGL(), with R=0, G=0, B=0 (black) and A=1 (opaque and cannot see through).

**Example:**

glClearColor(0.0f, 0.0f, 0.0f, 1.0f); // Black and opague glColor3f(1.0f, 0.0f, 0.0f); // Red

# Geometric Primitives



A geometric primitive is defined by specifying its vertices via glVertex function, enclosed within a pair glBegin and glEnd.

void **glBegin**(GLenum *shape*)

void **glVertex[234][sifd]** (*type* *x*, *type* *y*, *type* *z*, ...) void **glVertex[234][sifd]v** (*type* \**coords*) void **glEnd**()

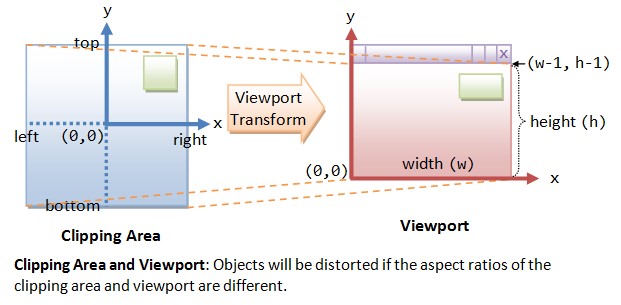
glBegin specifies the type of geometric object, such as GL\_POINTS, GL\_LINES, GL\_QUADS, GL\_TRIANGLES, and GL\_POLYGON. For types that end with 'S', you can define multiple objects of the same type in each glBegin/glEnd pair. For example, for GL\_TRIANGLES, each set of three glVertex's defines a triangle.

The vertices are usually specified in float precision. It is because integer is not suitable for trigonometric operations (needed to carry out transformations such as rotation). Precision of float is sufficient for carrying out intermediate operations, and render the objects finally into pixels on screen (with resolution of says 800x600, integral precision). double precision is often not necessary.

# Clipping-Area & Viewport

Try dragging the corner of the window to make it bigger or smaller. Observe that all the shapes are distorted.

We can handle the re-sizing of window via a callback handler reshape(), which can be programmed to adjust the OpenGL clipping-area according to the window's aspect ratio.



Clipping Area: *Clipping area* refers to the area that can be seen (i.e., captured by the camera), measured in OpenGL coordinates.

The function gluOrtho2D can be used to set the clipping area of 2D orthographic view.

Objects outside the clipping area will be *clipped* away and cannot be seen.

void **gluOrtho2D**(GLdouble *left*, GLdouble *right*, GLdouble *bottom*, GLdouble *top*)

// The default clipping area is (-1.0, 1.0, -1.0, 1.0) in OpenGL coordinates, // i.e., 2x2 square centered at the origin.

To set the clipping area, we need to issue a series of commands as follows: we first select the so-called *projection matrix* for operation, and reset the projection matrix to identity. We then choose the 2D orthographic view with the desired clipping area, via gluOrtho2D().

// Set to 2D orthographic projection with the specified clipping area

glMatrixMode(GL\_PROJECTION); // Select the Projection matrix for operation glLoadIdentity(); // Reset Projection matrix

gluOrtho2D(-1.0, 1.0, -1.0, 1.0); // Set clipping area's left, right, bottom, top

Viewport: *Viewport* refers to the display area on the window (screen), which is measured in pixels in screen coordinates (excluding the title bar).

The clipping area is mapped to the viewport. We can use glViewport function to configure the viewport.

void **glViewport**(GLint *xTopLeft*, GLint *yTopLeft*, GLsizei *width*, GLsizei *height*)

Suppose the the clipping area's (left, right, bottom, top) is (-1.0, 1.0, -1.0, 1.0) (in OpenGL coordinates) and the viewport's (xTopLeft, xTopRight, width, height) is (0, 0, 640, 480) (in screen coordinates in pixels), then the bottom-left corner (-1.0, -1.0) maps to (0, 0) in the viewport, the top-right corner (1.0, 1.0) maps to (639, 479). It is obvious that if the *aspect ratios* for the clipping area and the viewport are not the same, the shapes will be distorted. **Output:** OpenGL installed on system. Students are able to execute sample code to display single pixel on screen. Students can design various graphical shapes using in build functions. **Inference:** Students will be able to create OpenGL environment to implement computer graphics assignments. Student will be familiar with the OpenGL program structure. They are able to make use of OpenGL in built function to design various graphics shapes on screen.

**Oral questions:**

1. What is OpenGL (or Open Graphics Library)? Give Brief introduction about it.
2. Name major competitors of OpenGL. Also give main advantages & disadvantages OpenGL have over other graphics libraries in the market.
3. Give the main advantages that OpenGL have over Microsoft’s proprietary Direct3D.
4. OpenGL is written in which language? Is it possible to implement (or use) same library in programming languages other than that?
5. Is OpenGL API platform independent? Is it possible to port the library to embedded systems such as mobile phones etc.?
6. Name few OpenGL related libraries that simplifies the programming task by providing a layer of abstraction over OpenGL.
7. How OpenGl can be considered as a state machine?
8. What is term named Rasterization means? How is it different from vector graphics?
9. What is difference between glColor3f() & glClearColor() ?
10. Under which circumstances glFlush () method is used? How it is different from glFinish ()?
11. What kind of restrictions OpenGL imposes on primitive polygons? Why?
12. Using glBegin() & glEnd() how do we create primitive geometric drawings such as quadrilaterals, polygons etc.?
13. What are the different types of modes
14. What are changes needed to execute graphics program.
15. How to initialize graphics editor.
16. What is coordinate system? What is exact resolution of the computer screen or the screen orientation