Assignment No. 6

Title	Implementation of OpenGL functions
Aim/Problem Statement	a) Design and simulate any data structure like stack or queue visualization using graphics. Simulation should include all operations performed on designed data structure. Implement the same using OpenGL. **OR** b) Write C++ program to draw 3-D cube and perform following transformations on it using OpenGL i) Scaling ii) Translation iii) Rotation about an axis (X/Y/Z). **OR** c) Write OpenGL program to draw Sun Rise and Sunset.
CO Mapped	
Pre-requisite	1. Basic programming skills of C++ and OpenGL 2. 64-bit Open source Linux 3. Open Source C++ Programming tool like G++/GCC, OpenGL
Learning Objective	To implement OpenGL functions.

Theory:

OpenGL Basics:

Open Graphics Library (OpenGL) is a cross-language (language inepended), cross-platform (platform independent) API for rendering 2D and 3D Vector Graphics(use of polygons to represent image). OpenGL is a low-level, widely supported modeling and rendering software package, available across all platforms. It can be used in a range of graphics applications, such as games, CAD design, or modeling. OpenGL API is designed mostly in hardware.

- **Design**: This API is defined as a set of functions which may be called by the client program. Although functions are similar to those of C language but it is language independent.

- **Development**: It is an evolving API and Khronos Group regularly releases its new version having some extended feature compare to previous one. GPU vendors may also provide some additional functionality in the form of extension.

- **Associated Libraries:** The earliest version is released with a companion library called OpenGL utility library. But since OpenGL is quite a complex process. So in order to make it easier other library such as OpenGL Utility Toolkit is added which is later superseded by freeglut. Later included library were GLEE, GLEW and glbinding.

- **Implementation:** Mesa 3D is an open source implementation of OpenGL. It can do pure software rendering and it may also use hardware acceleration on BSD, Linux, and other platforms by taking advantage of Direct Rendering Infrastructure.

Installation of OpenGL on Ubuntu

We need the following sets of libraries in programming OpenGL:

2. **OpenGL Utility Library (GLU):** built on-top of the core OpenGL to provide important utilities and more building models (such as qradric surfaces). GLU functions start with a prefix "glu" (e.g. gluLookAt, gluPerspective).

- sudo apt-get install freeglut3-dev

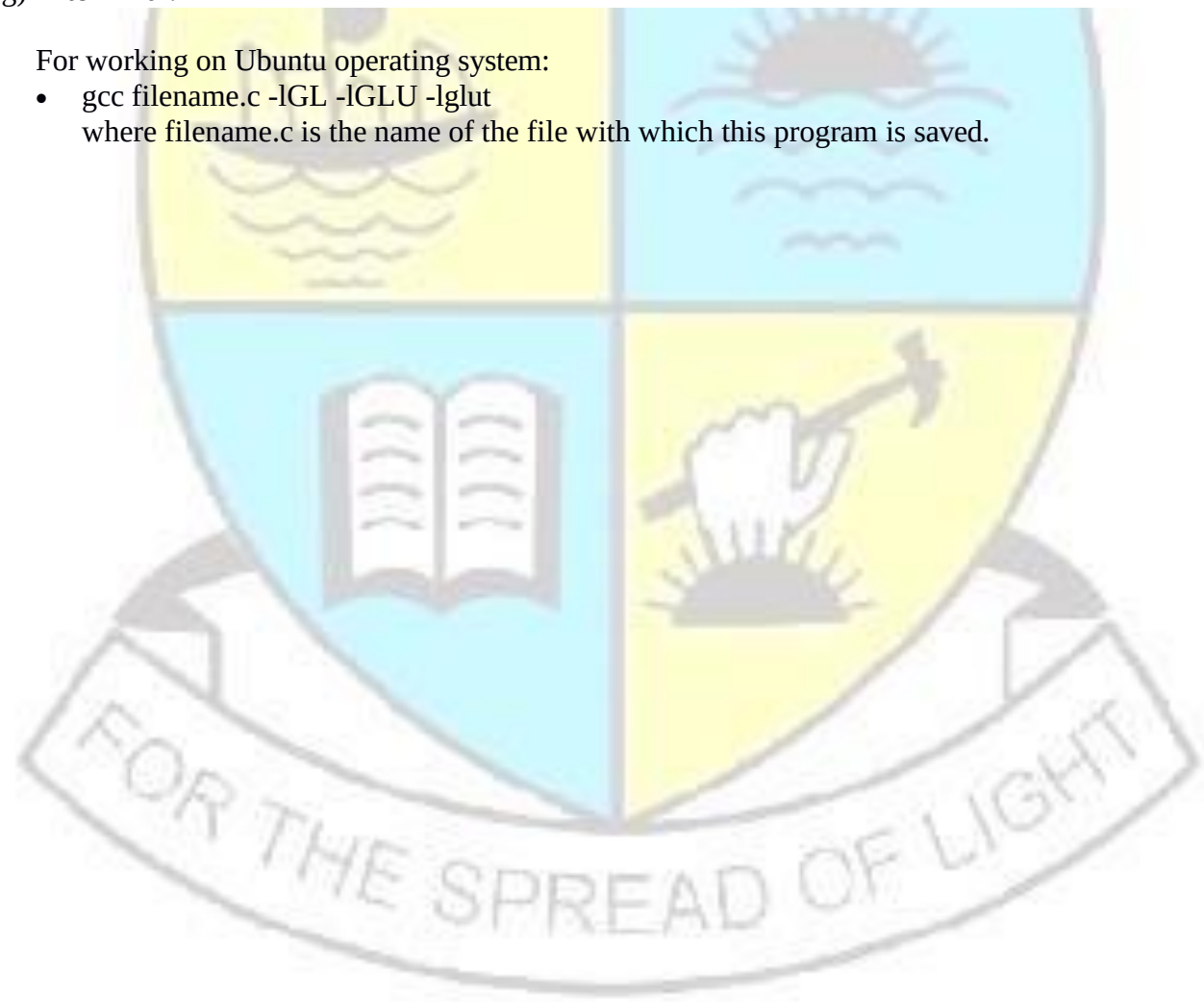
3. **OpenGL Utilities Toolkit (GLUT):** provides support to interact with the Operating System (such as creating a window, handling key and mouse inputs); and more building models (such as sphere and torus). GLUT functions start with a prefix of "glut" (e.g., glutCreatewindow, glutMouseFunc). GLUT is 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.
   Alternative of GLUT includes SDL, ...

4. **OpenGL Extension Wrangler Library (GLEW):** "GLEW is a cross-platform open-source C/C++ extension loading library. GLEW provides efficient run-time mechanisms for determining which OpenGL extensions are supported on the target platform.

For installing OpenGL on ubuntu, just execute the following command (like installing any other thing) in terminal:

For working on Ubuntu operating system:

- gcc filename.c -lGL -lGLU -lglut
  where filename.c is the name of the file with which this program is saved.



Prerequisites for OpenGL

Since OpenGL is a graphics API and not a platform of its own, it requires a language to operate in and the language of choice is C++.

Get started with OpenGL

Overview of an OpenGL program

- Main
  - Open window and configure frame buffer (using GLUT for example)
  - Initialize GL states and display (Double buffer, color mode, etc.)
- Loop
  - Check for events
    if window event (resize, unhide, maximize etc.)
    modify the viewport
    and Redraw
    else if input event (keyboard and mouse etc.)
    handle the event (such as move the camera or change the state)
    and usually draw the scene
- Redraw
  - Clear the screen (and buffers e.g., z-buffer)
  - Change states (if desired)
  - Render
  - Swap buffers (if double buffer)

OpenGL order of operations

- Construct shapes (geometric descriptions of objects – vertices, edges, polygons etc.)
- Use OpenGL to
  - Arrange shape in 3D (using transformations)
  - Select your vantage point (and perhaps lights)
  - Calculate color and texture properties of each object
  - Convert shapes into pixels on screen

OpenGL Syntax

- All functions have the form: gl*
  - *glVertex3f()* – 3 means that this function take three arguments, and f means that the type of those arguments is float.
  - *glVertex2i()* – 2 means that this function take two arguments, and i means that the type of those arguments is integer
- All variable types have the form: GL*
  - In OpenGL program it is better to use OpenGL variable types (portability)
    - *Glfloat* instead of *float*
    - *Glint* instead of *int*

OpenGL primitives

Drawing two lines

```
glBegin(GL_LINES);
glVertex3f(_,_,_); // start point of line 1
glVertex3f(_,_,_); // end point of line 1
glVertex3f(_,_,_); // start point of line 2
glVertex3f(_,_,_); // end point of line 2
glEnd();
```

We can replace GL_*LINES* with *GL_POINTS, GL_LINELOOP, GL_POLYGON* etc.

OpenGL states

- On/off (e.g., depth buffer test)
  - *glEnable(GLenum)*
  - *glDisable(GLenum)*
  - Examples:
    - *glEnable(GL_DEPTH_TEST);*
    - *glDisable(GL_LIGHTING);*
- Mode States
  - Once the mode is set the effect stays until reset
  - Examples:
    - *glShadeModel(GL_FLAT)* or *glShadeModel(GL_SMOOTH)*
    - *glLightModel(…)* etc.

Drawing in 3D

- Depth buffer (or z-buffer) allows scene to remove hidden surfaces. Use *glEnable(GL_DEPTH_TEST)* to enable it.
  - glPolygonMode(Face, Mode)
- Face: GL_FRONT, GL_BACK, GL_FRONT_AND_BACKMode: GL_LINE, GL_POINT, GL_FILL
  - glCullFace(Mode)
- Mode: GL_FRONT, GL_BACK, GL_FRONT_AND_BACK
  - glFrontFace(Vertex_Ordering)
  - Vertex Ordering: GL_CW or GL_CCW

Viewing transformation

- glMatrixMode (Mode)
  - Mode: GL_MODELVIEW, GL_PROJECTION, or GL_TEXTURE
- glLoadIdentity()
- glTranslate3f(x,y,z)
- glRotate3f(angle,x,y,z)
- glScale3f(x,y,z)

OpenGL provides a consistent interface to the underlying graphics hardware. This abstraction allows a single program to run a different graphics hardware easily. A program written with OpenGL can even be run in software (slowly) on machines with no graphics acceleration. OpenGL function names always begin with *gl*, such as *glClear()*, and they may end with characters that indicate the types of the parameters, for example *glColor3f(GLfloat red, GLfloat green, GLfloat blue)* takes three floating-point color parameters and *glColor4dv(const GLdouble *v)* takes a pointer to an array that contains 4 double-precision floating-point values. OpenGL constants begin with *GL*, such as *GL DEPTH*. OpenGL also uses special names for types that are passed to its functions, such as *GLfloat* or *GLint*, the corresponding C types are compatible, that is *float* and *int* respectively.

GLU is the OpenGL utility library. It contains useful functions at a higher level than those provided by OpenGL, for example, to draw complex shapes or set up cameras. All GLU functions are written on top of OpenGL. Like OpenGL, GLU function names begin with glu, and constants begin with GLU.

GLUT, the OpenGL Utility Toolkit, provides a system for setting up callbacks for interacting with the user and functions for dealing with the windowing system. This abstraction allows a

program to run on different operating systems with only a recompile. Glut follows the convention of prepending function names with glut and constants with GLUT.

Writing an OpenGL Program with GLUT

An OpenGL program using the three libraries listed above must include the appropriate headers. This requires the following three lines:

```
#include <GL/gl.h>
#include <GL/glu.h>
#include <GL/glut.h>
```

Before OpenGL rendering calls can be made, some initialization has to be done. With GLUT, this consists of initializing the GLUT library, initializing the display mode, creating the window, and setting up callback functions. The following lines initialize a full color, double buffered display:

```
glutInit(&argc, argv);
glutInitDisplayMode(GLUT_DOUBLE | GLUT_RGB);
```

Double buffering means that there are two buffers, a front buffer and a back buffer. The front buffer is displayed to the user, while the back buffer is used for rendering operations. This prevents flickering that would occur if we rendered directly to the front buffer.

Next, a window is created with GLUT that will contain the viewport which displays the OpenGL front buffer with the following three lines:

```
glutInitWindowPosition(px, py);
glutInitWindowSize(sx, sy);
glutCreateWindow(name);
```

To register callback functions, we simply pass the name of the function that handles the event to the appropriate GLUT function.

```
glutReshapeFunc(reshape);
glutDisplayFunc(display);
```

Here, the functions should have the following prototypes:

```
void reshape(int width, int height);
void display();
```

In this example, when the user resizes the window, reshape is called by GLUT, and when the display needs to be refreshed, the display function is called. For animation, an idle event handler that takes no arguments can be created to call the display function to constantly redraw the scene with *glutIdleFunc*. Once all the callbacks have been set up, a call to *glutMainLoop* allows the program to run.

In the display function, typically the image buffer is cleared, primitives are rendered to it, and the results are presented to the user. The following line clears the image buffer, setting each pixel color to the clear color, which can be configured to be any color:

```
glClear(GL_COLOR_BUFFER_BIT);
```

The next line sets the current rendering color to blue. OpenGL behaves like a state machine, so certain state such as the rendering color is saved by OpenGL and used automatically later as it is needed.

glColor3f(0.0f, 0.0f, 1.0f);

To render a primitive, such as a point, line, or polygon, OpenGL requires that a call to *glBegin* is made to specify the type of primitive being rendered.

glBegin(GL_LINES);

Only a subset of OpenGL commands is available after a call to *glBegin*. The main command that is used is *glVertex*, which specifies a vertex position. In GL LINES mode, each pair of vertices define endpoints of a line segment. In this case, a line would be drawn from the point at (x0, y0) to (x1, y1).

glVertex2f(x0, y0); glVertex2f(x1, y1);

A call to glEnd completes rendering of the current primitive. *glEnd()*; Finally, the back buffer needs to be swapped to the front buffer that the user will see, which GLUT can handle for us:

glutSwapBuffers();

Developer-Driven Advantages

- **Industry standard**
  An independent consortium, the OpenGL Architecture Review Board, guides the OpenGL specification. With broad industry support, OpenGL is the only truly open, vendor-neutral, multiplatform graphics standard.

- **Stable**
  OpenGL implementations have been available for more than seven years on a wide variety of platforms. Additions to the specification are well controlled, and proposed updates are announced in time for developers to adopt changes. Backward compatibility requirements ensure that existing applications do not become obsolete.

- **Reliable and portable**
  All OpenGL applications produce consistent visual display results on any OpenGL API-compliant hardware, regardless of operating system or windowing system.

- **Evolving**
  Because of its thorough and forward-looking design, OpenGL allows new hardware innovations to be accessible through the API via the OpenGL extension mechanism. In this way, innovations appear in the API in a timely fashion, letting application developers and hardware vendors incorporate new features into their normal product release cycles.

- **Scalable**
  OpenGL API-based applications can run on systems ranging from consumer electronics to PCs, workstations, and supercomputers. As a result, applications can scale to any class of machine that the developer chooses to target.

- **Easy to use**

  OpenGL is well structured with an intuitive design and logical commands. Efficient OpenGL routines typically result in applications with fewer lines of code than those that make up programs generated using other graphics libraries or packages. In addition, OpenGL drivers encapsulate information about the underlying hardware, freeing the application developer from having to design for specific hardware features.

- **Well-documented:**

  Numerous books have been published about OpenGL, and a great deal of sample code is readily available, making information about OpenGL inexpensive and easy to obtain.

Conclusion:

Questions:

1. What are the advantages of Open GL over other API's?
2. Explain rendering pipeline with reference to OpenGL.
3. Write OpenGl Advantages