Chapter 1
INTRODUCTION

1.1 Computer Graphics

Computer Graphics concerns the pictorial synthesis of real or imaginary objects from their computer-based models or in other words it may be the collection, combination and representation of imaginary objects from their computer-based models. The term computer graphics has been used in a broad sense to describe "almost everything on computers that is not text or sound". It is one of the most rapidly advancing fields of computer science and is used in almost all fields including medical and military. It is also an effective media for communication between man and the computer; the human can understand the information content of a display diagram or perspective view much faster than he can understand a table of numbers. Graphical interfaces have replaced textual interfaces as the standard means of computer interaction.

Computer Graphics has matured over the years as a discipline. Both hardware and software are available that facilitate the production of graphical images as diverse as line drawings in 2D to realistic renderings of natural objects in 3D. Nowadays Computer Graphics is also making deep inroads into the business, advertising, communication medium and entertainment industries. Today, no scientist or engineer should be without a basic understanding of underlying principles of Computer Graphics.

Computers have become a powerful tool for rapid and economical production of pictures, animation, games and simulation. Computer graphics is one of the most powerful and interesting part of a computer. There is a lot that can be done in graphics apart from drawing figures. There is virtually no area in which graphics display cannot be used to some advantage and so it is not surprising to find the use of computer graphics so widely accepted.

1.2 OpenGL

OpenGL or Open Graphics Library is a cross language, multi-platform API for rendering 2D and 3D computer graphics. The API is typically used to interact with a GPU, to achieve hardware-accelerated rendering. OpenGL was developed by Silicon Graphics Inc. from 1991 and released in January 1992. It provides the actual drawing tools through a collection of functions that are called within an application. It is easy to install and learn, and its longevity as a standard API is being nurtured and overseen by the OpenGL Architecture Review Board (ARB), an industry consortium responsible for guiding its evolution. It is widely used in CAD, virtual reality, scientific visualization, information visualization, flight simulation and video games.

The OpenGL specification describes an abstract API for drawing 2D and 3D graphics. Although it’s possible for the API to be implemented entirely in software, it’s designed to be implemented mostly or entirely in hardware. Although the function definitions are superficially similar to those of the C programming, they are language independent. In addition to being language independent, it is also platform-independent. One aspect of OpenGL that suits it so well for use in computer graphics course is its device independence or portability. A student can develop and run a program on any available computer. OpenGL offers a rich and highly usable API for 2D graphics and image manipulation, but its real power emerges with 3D graphics.

1.3 Characteristics of OpenGL

- Platform independent: can run on consumer electronics, PC’s, workstations, etc.
- Backward compatibility required in new versions.
- Supported by many hardware accelerators => fast.

Chapter 2
GRID

2.1 Problem Statement

The objective is to build a 3D graphics editor having all required operations that a normal graphics editor should offer. The coding is implemented for a single canvas graphics editor. In this editor importance is given to a simple user interface and implementation of all basic graphic routines and basic graphic editing options.

2.2 Project Goal

The aim of this project is to develop a 3-D graphics package which supports basic operations which include creating objects like cube, sphere, mesh, curves etc. and also transformation operations like translation, rotation, scale. The package must also have a user-friendly interface that may be menu-oriented, iconic or a combination of both.

“GRID” consists of user interface (GUI) which is separated into three sections:

- **Object Editor** - Individual objects are stored and edited within the object editor. Objects may be composed of primitives such as spheres, or more complex shapes made by extruding polygons and adjusting the vertexes. Materials are then applied
- **Properties Editor** – Each object created in Object Editor will have its own properties such as width & height of cube. When object is selected, it is highlighted & its properties are shown in the Properties Editor. User is allowed to modify the properties.
- **Object List** – Each object in the Object editor will be listed with their unique name. Currently selected object will be highlighted.

2.3 Features

- Support for a variety of geometric primitives, including polygon meshes, spheres, cubes, Bezier curves etc.
- Properties of objects can be adjusted interactively and results appear immediately.
- Allows rotations in three dimensions, allowing viewing of a designed object from any desired angle using camera.

Chapter 3
REQUIREMENT SPECIFICATIONS

The basic purpose of software requirement specification (SRS) is to bridge the communication between the parties involved in the development project.SRS is the medium through which the user's needs are accurately specified; indeed SRS forms the basis of software development. Another important purpose of developing an SRS is helping the users understand their own needs.

Now we will be discussing the requirement analysis of the project. This report gives the description of the roles of users, the functional overviews of the project, input and output characteristics and also the hardware and software for the project.

3.1 Software Requirements

- Microsoft Visual Studio 12.0
- OpenGL Graphics software system.
- Header files used:

  glut32.dll, glut32.lib, glut.h, gl.h, glu.h.
  SDL.h

3.2 Hardware Requirements

▪ Processor	:	1.3 - GHz or higher Intel processor.
▪ Hard Disk	:	30 GB recommended.
▪ Memory	:	1 GB of RAM, 512 MB recommended
▪ Display	:	1152 x 864 or higher-resolution display with 256 colors
▪ Keyboard	:	Standard QWERTY keyboard for interface

3.3 Why C++ Language for this project?

C++ is a minimalist programming language. Among its design goal were that it could be compiled in a straight forward manner using a relatively simple compiler, provided low-level access to memory, generated only a few machine language instructions for each of its core language elements and did not require extensive run time process. As a result, C++ code is suitable for many system programming applications that had traditionally been implemented in assembly language. Despite its low level capabilities, the language was designed to encourage machine independent programming. A standard compliant and portably written C++ program can be compiled for a very wide variety of computer platform and operating systems with minimal change to its source code.

3.4 Graphics in C++

C++ itself doesn't support any graphics library. In order for C++ to be completely portable from platform to platform it has focused on providing platform independent functions, such as file access, text string manipulation, mathematical functions, etc. So in order to get graphics in C++, one needs to do one of four things:

- Write a 16 bit DOS program and use assembly language for graphics.
- Make calls to Windows API
- Make calls to the OpenGL subsystem
- Use a graphics library

3.5 OpenGL subsystems

OpenGL provides a powerful but primitive set of rendering commands, and all higher-level drawing must be done in terms of these commands. Also, OpenGL programs have to use the underlying mechanisms of the windowing system. A number of libraries exist to simplify the programming tasks, including the following:

The OpenGL Utility Library(GLU)contains several routines that use lower-level OpenGL commands to perform such tasks as setting up matrices for specific viewing orientations and projections, performing polygon tessellation, and rendering surfaces. The library is provided as part of every OpenGL implementation.

Chapter 4
SYSTEM DESIGN

The role of the user is limited in our project. User can do the following operations:

- The user has to simply click on debug option then everything is automatic…

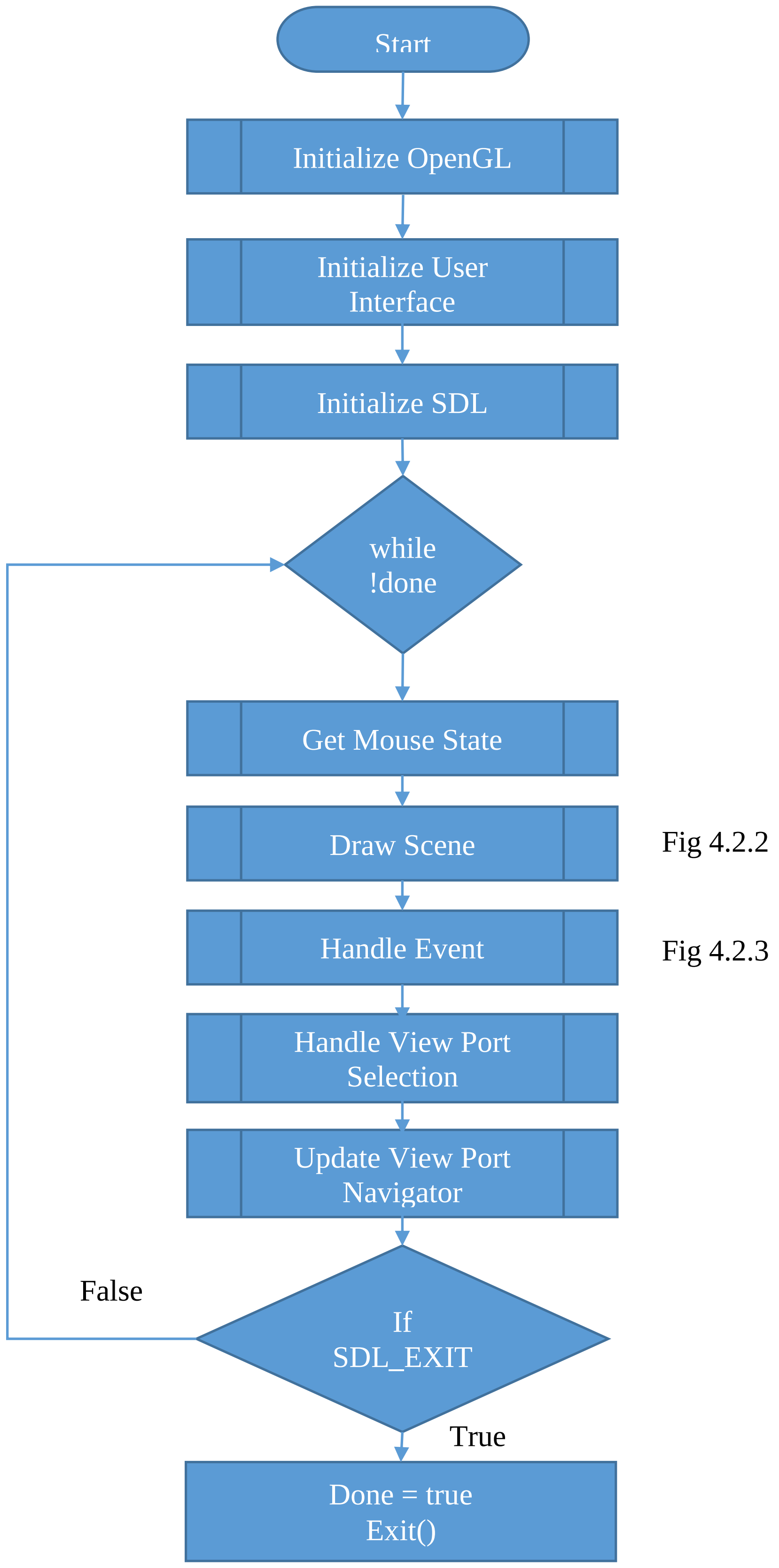
4.1 Key Features

- Provided with text to understand our concept.
- Sudden appearance of object in the screen.
- Changing of background color based on situation.

4.2 Flowchart

- A flowchart is a common type of chart that represents an algorithm or process showing the steps as boxes of various kinds and their order by connecting these with arrows. Flowcharts are used in analyzing, designing, managing a process or program in various fields.
- Flowcharts used to be a popular means for describing computer algorithms. They are still used for the purpose, modern techniques such as UML activity diagram can be considered to be extensions of the flowchart.

Fig 4.2.1 Flowchart for "GRID"



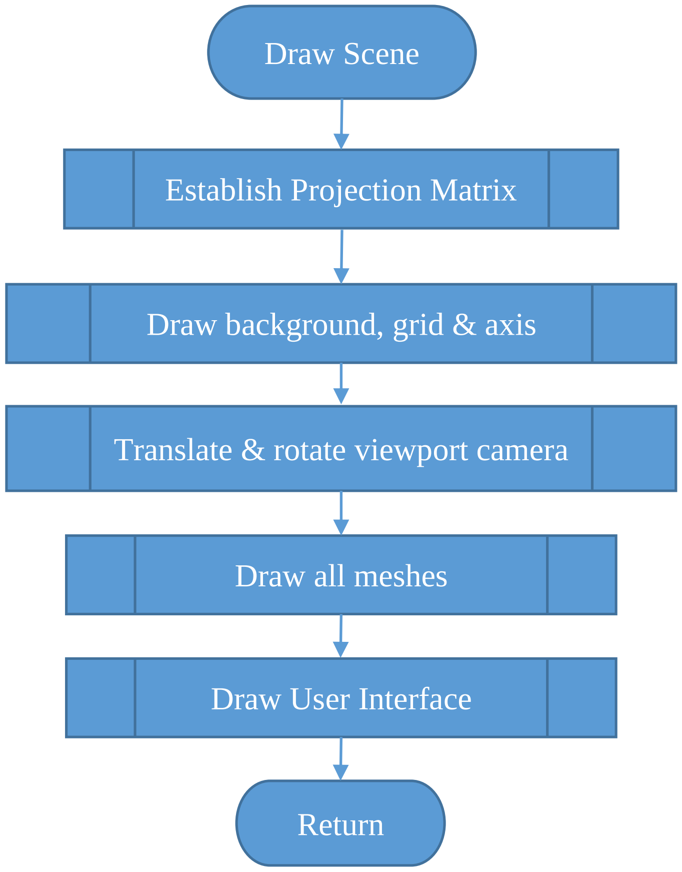

Draw Scene
Establish Projection Matrix
Draw background, grid & axis
Translate & rotate viewport camera
Draw all meshes
Draw User Interface
Return

Fig 4.2.2 Flowchart for function “Draw Scene”

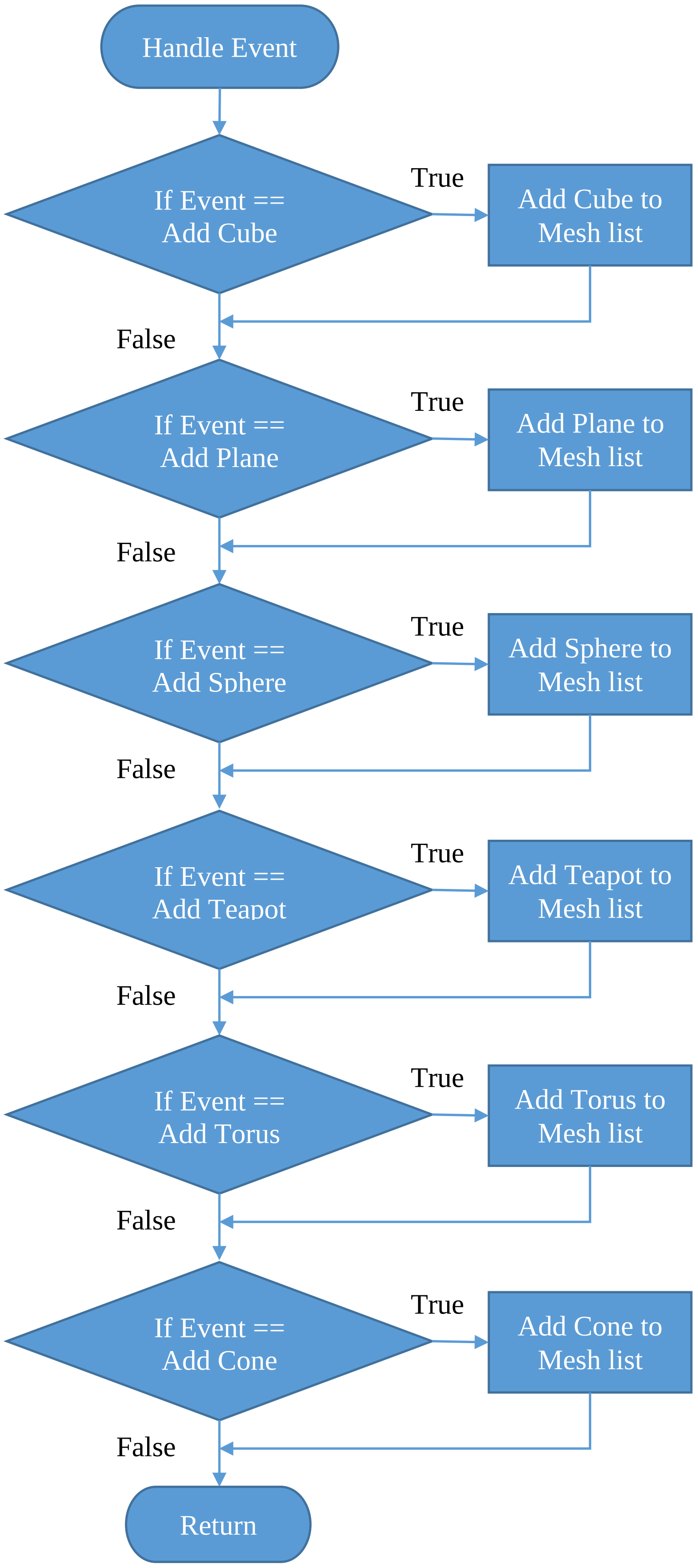


Fig 4.2.3 Flowchart for function “Handle Event”

Chapter 5

IMPLEMENTATION

The implementation stage of the model involves the following phases.

- Implementation of OpenGL built in function.
- User defined function Implementation.

5.1 Implementation of OpenGL Built In Functions

glutCreateWindow (char *name)

Creates a window on the display**.** The string name can be used to label the window. The return value provides a reference to the window that can be used when there are multiple windows.

glutDisplayFunc (void (*func)(void))

Registers the display function func that is executed when the window needs to be redrawn.

glutInitWindowSize(int width,int height)

It specifies the initial height and width of the window in pixels.

glutInitWindowPosition (int x, int y)

It specifies the initial position of the top-left corner of the window.

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

This function sets the keyboard callback for the current window.

glutMainLoop()

This function enters the glut event processing loop. The routine should be called at most once in a glut program.It should be the last statement in main.

glutIdleFunc(void (*f)(void))

This function registers the display callback function f that is executed when there are no other events to be handled.

glTranslate[i d] (TYPE x, TYPE y, TPYE z)

This function is used to move objects from one position to another. It alters the current matrix by a displacement of (x, y, z).Type is either GLfloat or GLdouble.

glPushMatrix ( )

It pushes to the matrix stack corresponding to the current matrix mode.

glPopMatrix ( )

Pops from the matrix stack corresponding to the current matrix mode.

glColor3[i f] (TYPE r, TYPE g, TYPE b)

It sets the present RGB colors. Valid types are byte (b), int (i), float (f), double (d) and many more. The maximum and minimum values of the floating point types are 1.0 and 0.0.

glBegin (glEnum mode)

It initiates a new primitive of type mode and starts the collection of vertices. Values of mode include GL_POINTS, GL_LINES, and GL_POLYGON.

glVertex[2 3][s i d f](TYPE xcoordinate, TYPE ycoordinate,…)

It specifies the position of a vertex in 2 or 3 dimensions. The coordinates can be specified as shorts s, int i, floats f or doubles d.

glEnd ( )

It terminates a list of vertices.

glMatrixMode(GLenum mode)

Specifies which matrix will be affected by subsequent transformations. Mode can be GL_MODELVIEW,GL_PROJECTION.

void glLoadIdentity()

This function sets the current transformation matrix to an identity matrix.

void glOrtho(GLdouble left, GLdouble right, GLdouble bottom, GLdouble top, GLdouble near,GLdouble far)

glOrtho defines an 3 dimensional orthographic viewing volume with parameters left, right, bottom, top, near, far measured from the center of the projection plane.

glFlush()

It flushes the color buffer and depth buffer.

void glutPostRedisplay ()

Request that the display call back be executed after the current call back returns.

glRasterPos2f()

This function specifies raster postion. Parameters are the same for glVertex().

5.2 Working with the window system

void GLflush()

Forces any buffered OpenGL command to execute.

void glutInit(int argc,char **argv)

glutInit() should be called before any other glut routine, because it initializes the GLUT library. glutInit() will also process command line options, but he specific options are window system dependent.

intglutCreateWindow(char *title)

Creates a window on the display. The string title can be used to label the window. The return value provides a reference to the window that can be used when there are multiple windows

5.3 User Defined Functions

Class: GRID_ENGINE

Member functions:

GRID_Engine::initializeGLWindow(char _name[], GLfloat _w, GLfloat _h)
Initializes OpenGL Window

GRID_Engine::establishProjectionMatrix(GLsizei width, GLsizei height)
Establish Projection Matrix

GLvoid GRID_Engine::initializeSDL(char _name[], GLsizei windowWidth, GLsizei windowHeight, Color c, ViewType v)
Intialize the SDL window

Main.cpp

void updateViewPortNavigator()
Used for viewport navigation

void DrawHeirarchyList()
Draws all the objects names in the left side frame

void HandleEvent(Inspector * currentInspector)
Used for creating objects based on input

void HandleViewPortSelection()
Used for picking objects in the viewport

void DrawInspectorList()
Draws the user interface flies such as buttons, sliders etc.

GLvoid drawScene(bool display)
Draws all the objects in the mesh list

GLvoid displayFPS(GLvoid)
Calculates the FPS(Frames Per Second) of the window

GRID

Class: CUBE

Member functions:
void Cube::create(char* _name)
Creates Cube with dimension of 1 unit

Class: PLANE

Member functions:
void Plane::setSize(float w, float h)
Set the width & height of the plane

void Plane::create(char* _name)
Creates the Plane

Class: SPHERE

Member functions:
void Sphere::Init(float _radius, float _stacks, float _slices)
Set the radius, Stacks & Slices of the Sphere

void Sphere::create(char* _name)
Create the Sphere (uses glutSolidSphere() function)

Class: TRANSFORM

Member functions:

Transform::Transform()
Constructor – Sets the scale to 1,1,1 & Position to 0,0,0

void Transform::Translate(vec3 vec)
Used to translate the given object to x,y,z position

void Transform::Rotation(vec3 vec)
Rotate the given object

void Transform::Scale(vec3 vec)
Scale the given object

Class: MESH

MESH is the parent class for all basic primitives
Member functions:

GRID

bool Mesh::drawMesh(bool drawChildren, Mesh* currentMesh, bool drawAxes,bool mode)

Draws the mesh

Class: UTILITIES

Member functions:

void Utilities::glEnable2D(void)

Enables 2D i.e. Disables DEPTH_TEST & GL_LIGHTING

void Utilities::glDisable2D(void)

Disables 2D i.e. Enables DEPTH_TEST & GL_LIGHTING

void Utilities::glPrintf(int x, int y, void* font, const char* string, ...)

Used for printing text

Inspector* Utilities:: addInspector(Inspector *_inspector)

Used to set the position for Top frame buttons

Inspector* Utilities::addInspectorRightFrame(Inspector *_inspector)

Used to set the position for Right frame buttons

void Utilities::draw_axes(void)

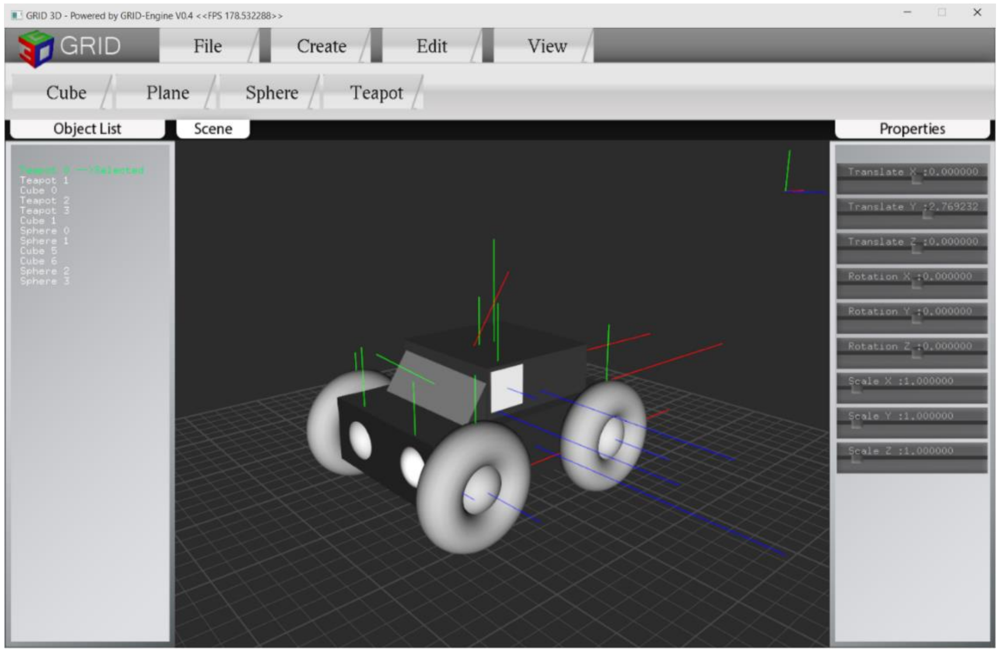
Draws the 3 lines representing x, y & z axis in red, green & blue colors respectively

void Utilities::draw_grid()

Draws the base grid

Chapter 6

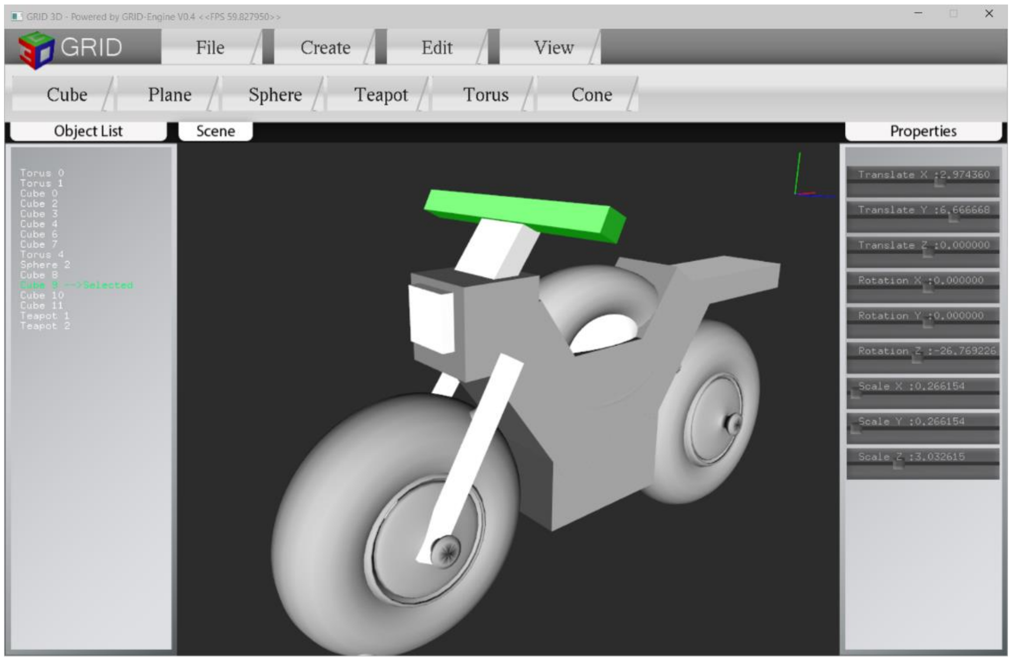
RESULTS



```
Teapot 0 -->Selected
Teapot 1
Cube 0
Teapot 2
Teapot 3
Cube 1
Sphere 0
Sphere 1
Cube 5
Cube 6
Sphere 2
Sphere 3
```

```
Translate X :0.000000
Translate Y :2.769232
Translate Z :0.000000
Rotation X :0.000000
Rotation Y :0.000000
Rotation Z :0.000000
Scale X :1.000000
Scale Y :1.000000
Scale Z :1.000000
```

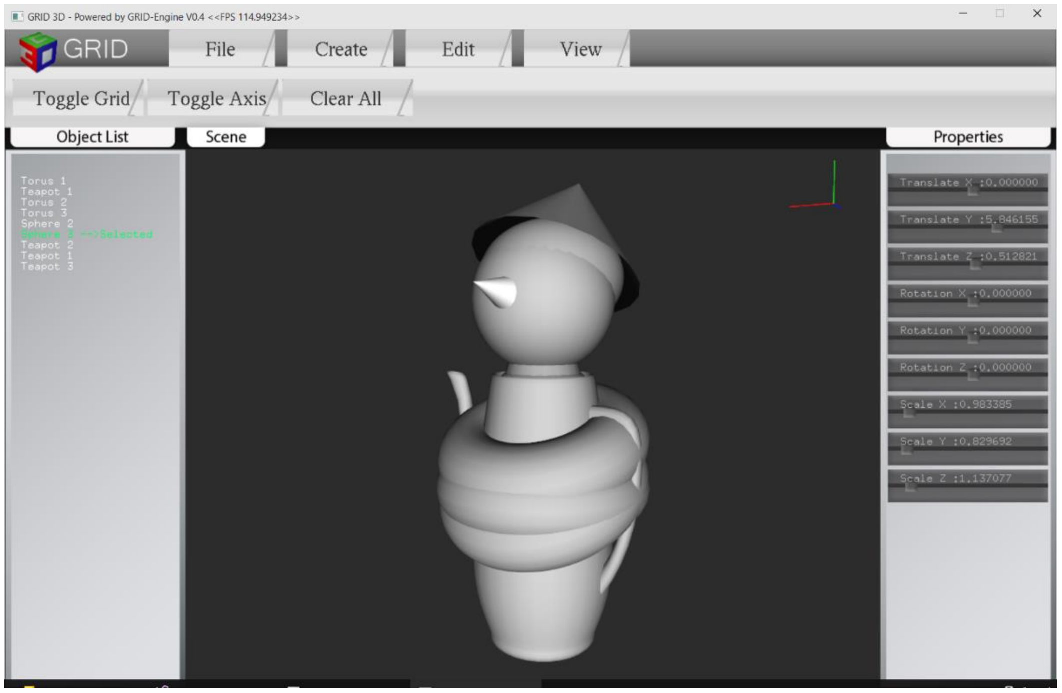
Fig.6.1 3d model of a car designed using basic solid objects like Cube, Sphere, and Torus etc.



```
Torus 0
Torus 1
Cube 0
Cube 2
Cube 3
Cube 4
Cube 6
Cube 7
Torus 4
Sphere 2
Cube 8
Cube 9 -->Selected
Cube 10
Cube 11
Teapot 1
Teapot 2
```

```
Translate X :2.974360
Translate Y :6.666668
Translate Z :0.000000
Rotation X :0.000000
Rotation Y :0.000000
Rotation Z :-26.769226
Scale X :0.266154
Scale Y :0.266154
Scale Z :3.032615
```

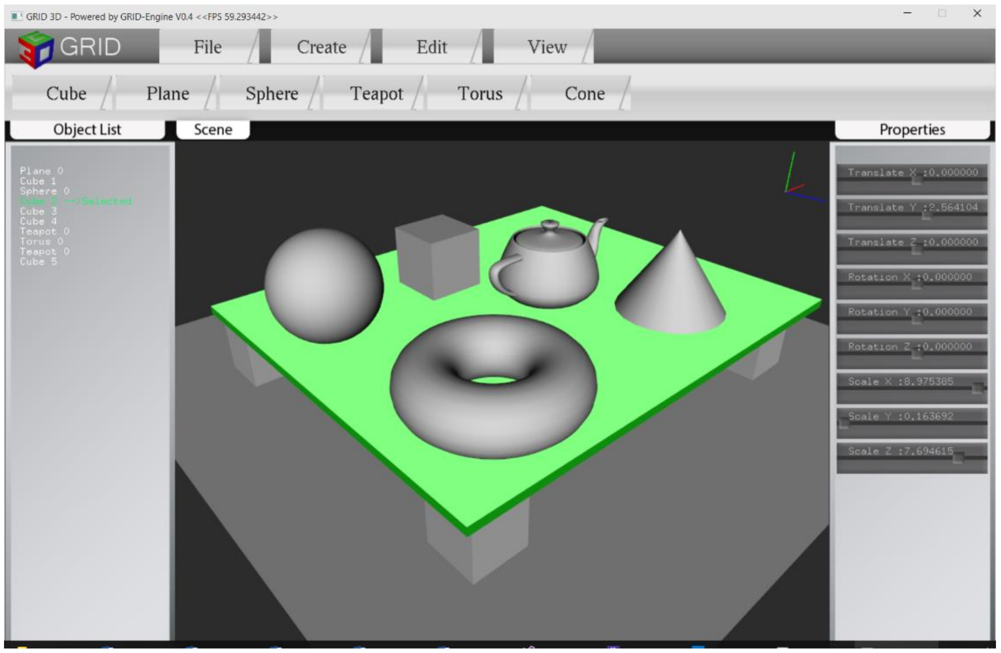
Fig.6.2 Motor Bike



```
Torus 1
Teapot 1
Torus 2
Torus 3
Sphere 2
Sphere 3 -->Selected
Teapot 2
Teapot 1
Teapot 3
```

```
Translate X :0.000000
Translate Y :5.846155
Translate Z :0.512821
Rotation X :0.000000
Rotation Y :0.000000
Rotation Z :0.000000
Scale X :0.983385
Scale Y :0.829692
Scale Z :1.137077
```

Fig.6.3 3d Statue



```
Plane 0
Cube 1
Sphere 0
Cube 2 -->Selected
Cube 3
Cube 4
Teapot 0
Torus 0
Teapot 0
Cube 5
```

```
Translate X :0.000000
Translate Y :2.564104
Translate Z :0.000000
Rotation X :0.000000
Rotation Y :0.000000
Rotation Z :0.000000
Scale X :8.975385
Scale Y :0.163692
Scale Z :7.694615
```

Fig.6.3 3d Solid objects

Chapter 7
CONCLUSION

Computer Graphics plays a major role in today’s world where visualization takes the upper hand as compared to text based interactions. This is largely true as we can see user interfaces more attractive. All thanks to the major leaps in the field of Computer Graphics.

Working with computer graphics and exploring its new features is really interesting. In conclusion, it can be said that OOP's concept along with the use of graphics has a very deep impact in programming. The usage of input device along with these makes the work easier though the usage of mouse has been restricted to some parts in the software. But, in spite of this limitation, it is seen that a user can handle the software without much difficulty.

This is being my first project using an Open GL implementation. It was an effort to demonstrate the basic Open GL functions which will help the user to gain some knowledge about Computer Graphics. I hope this will be a stepping stone to my next such projects.