CHAPTER 1

INTRODUCTION

1.1 OVERVIEW OF COMPUTER GRAPHICS

The term computer graphics has been used in a broad sense to describe almost everything on computers that is not text or sound. Typically, the term *computer graphics* refers to several different things:

- The representation and manipulation of image data by a computer
- The various technologies used to create and manipulate images
- The images so produced, and
- The sub-field of computer science which studies methods for digitally synthesizing and manipulating visual content.

Today, computers and computer-generated images touch many aspects of daily life. Computer images is found on television, in newspapers, for example in weather reports, in all kinds of medical investigation and surgical procedures. A well-constructed graph can present complex statistics in a form that is easier to understand and interpret. In the media such graphs are used to illustrate papers, reports, thesis, and other presentation material.Many powerful tools have been developed to visualize data. Computer generated imagery can be categorized into several different types: 2D, 3D, 4D, 7D, and animated graphics.

As technology has improved, 3D computer graphics have become more common. Computer graphics has emerged as a sub-field of computer science which studies methods for digitally synthesizing and manipulating visual content. Over the past decade, other specialized fields have been developed like information visualization, and scientific visualization more concerned with the visualization of three dimensional phenomena (architectural, meteorological, medical, biological, etc.), where the emphasis is on realistic renderings of volumes, surfaces, illumination sources, and so forth, perhaps with a dynamic component.

1.2 HISTORY OF COMPUTER GRAPHICS

In 1959, the TX-2 computer was developed at MIT's Lincoln Laboratory. The TX-2 integrated a number of new man-machine interfaces. A light pen could be used to draw sketches on the computer using Ivan Sutherland's revolutionary Sketchpad software.[4] Using a light pen, Sketchpad allowed one to draw simple shapes on the computer screen, save them and even recall them later. The light pen itself had a small photoelectric cell in its tip. This cell emitted an electronic pulse whenever it was placed in front of a computer screen and the screen's electron gun fired directly at it. By simply timing the electronic pulse with the current location of the electron gun, it was easy to pinpoint exactly where the pen was on the screen at any given moment. Once that was determined, the computer could then draw a cursor at that location. Also in 1961 another student at MIT, Steve Russell, created the first video game, E. E. Zajac, a scientist at Bell Telephone Laboratory (BTL), created a film called "Simulation of a two-giro gravity attitude control system" in 1963.

During 1970s, the first major advance in 3D computer graphics was created at UU by these early pioneers, the hidden-surface algorithm. In order to draw a representation of a 3D object on the screen, the computer must determine which surfaces are "behind" the object from the viewer's perspective, and thus should be "hidden" when the computer creates (or renders) the image.

In the 1980s, artists and graphic designers began to see the personal computer, particularly the Commodore Amiga and Macintosh, as a serious design tool, one that could save time and draw more accurately than other methods. In the late 1980s, SGI computers were used to create some of the first fully computer-generated short films at Pixar. The Macintosh remains a highly popular tool for computer graphics among graphic design studios and businesses. Modern computers, dating from the 1980s often use graphical user interfaces (GUI) to present data and information with symbols, icons and pictures, rather than text. Graphics are one of the five key elements of multimedia technology.

3D graphics became more popular in the 1990s in gaming, multimedia and animation. In 1996, Quake, one of the first fully 3D games, was released. In 1995, Toy Story, the first full-length computer-generated animation film, was released in cinemas worldwide. Since then, computer graphics have only become more detailed and realistic, due to more powerful graphics hardware and 3D modeling software.

1.3 APPLICATIONS OF COMPUTER GRAPHICS

The applications of computer graphics can be divided into four major areas:

- Display of information
- Design
- Simulation and animation
- User interfaces

Display of information

Computer graphics has enabled architects, researchers and designers to pictorially interpret the vast quantity of data. Cartographers have developed maps to display the celestial and geographical information. Medical imaging technologies like Computerized Tomography (CT), Magnetic Resonance Imaging (MRI), Ultrasound, Positron Emission Tomography (PET) and many others make use of computer graphics.

Design

Professions such as engineering and architecture are concerned with design. They start with a set of specification; seek cost-effective solutions that satisfy the specification. Designing is an iterative process. Designer generates a possible design, tests it and then uses the results as the basis for exploring other solutions. The use of interactive graphical tools in Computer Aided Design (CAD) pervades the fields including architecture, mechanical engineering, and the design of very-large-scale integrated (VLSI) circuits and creation of characters for animation.

Simulation and animation

Once the graphics system evolved to be capable of generating sophisticated images in real time, engineers and researchers began to use them as simulators. Graphical flight simulators have proved to increase the safety and to reduce the training expenses. The field of virtual reality (VR) has opened many new horizons. A human viewer can be equipped with a display headset that allow him/her to see the images with left eye and right eye which gives the effect of stereoscopic vision. This has further led to motion pictures and interactive video games.

User interfaces

Computer graphics has led to the creation of graphical user interfaces (GUI) using which even naive users are able to interact with a computer. Interaction with the computer has been dominated by a visual paradigm that includes windows, icons, menus and a pointing device such as mouse. Millions of people are internet users; they access the internet through the graphical network browsers such as Microsoft internet explorer and Mozilla Firefox.

CHAPTER 2

OpenGL

2.1 Introduction to OpenGL

OpenGL (Open Graphics Library) is a standard specification defining a cross-language, cross-platform API for writing applications that produce 2D and 3D computer graphics. The interface consists of over 250 different function calls which can be used to draw complex three-dimensional scenes from simple primitives. OpenGL was developed by Silicon Graphics Inc. (SGI) in 1992 and is widely used in CAD, virtual reality, scientific visualization, information visualization, and flight simulation.

OpenGL provides a set of commands to render a three dimensional scene. That means you provide the data in an OpenGL-useable form and OpenGL will show this data on the screen (render it). It is developed by many companies and it is free to use. You can develop OpenGL-applications without licensing. OpenGL is a hardware- and system-independent interface. An OpenGL-application will work on every platform, as long as there is an installed implementation. Because it is system independent, there are no functions to create windows etc., but there are helper functions for each platform. A very useful thing is GLUT.

2.2 OpenGL LIBRARIES

Computer Graphics are created using OpenGL, which became a widely accepted standard software system for developing graphics applications. As a software interface for graphics hardware, OpenGL's main purpose is to render two- and three-dimensional objects into a frame buffer. These objects are described as sequences of vertices (which define geometric objects) or pixels (which define images). OpenGL performs several processing steps on this data to convert it to pixels to form the final desired image in the frame buffer.

OpenGL stands for ‘open graphics library’ graphics library is a collection of API’s (Applications Programming Interface). Graphics library functions are:

1. **GL library** (OpenGL in windows) – Main functions for windows.
2. **GLU** (OpenGL utility library) - Creating and viewing objects.
3. **GLUT** (OpenGL utility toolkit)- Functions that help in creating interface of windows

OpenGL draws *primitives*—points, line segments, or polygons—subject to several selectable modes. You can control modes independently of each other; that is, setting one mode doesn't affect whether other modes are set (although many modes may interact to determine what eventually ends up in the frame buffer). Primitives are specified, modes are set, and other OpenGL operations are described by issuing commands in the form of function calls. These libraries are included in the application program using preprocessor directives

#include<GL/glut.h>

OpenGL User Interface Library (**GLUI**) is a C++ user interface library based on the OpenGL Utility Toolkit (GLUT) which provides controls such as buttons, checkboxes, radio buttons, and spinners to OpenGL applications. It is window and operating system independent, relying on GLUT to handle all system-dependent issues, such as window and mouse management.

The **OpenGL Utility Library** (**GLU**) is a computer graphics library. It consists of a number of functions that use the base OpenGL library to provide higher-level drawing routines from the more primitive routines that OpenGL provides. It is usually distributed with the base OpenGL package.

2.3 OpenGL CONTRIBUTIONS

It is very popular in the video games development industry where it competes with Direct3D (on Microsoft Windows).OpenGL is also used in CAD, virtual reality, and scientific visualization programs. OpenGL is very portable. It will run for nearly every platform in existence, and it will run well. It even runs on Windows NT 4.0 etc. The reason OpenGL runs for so many platforms is because of its Open Standard.

OpenGL has a wide range of features, both in its core and through extensions. Its extension feature allows it to stay immediately current with new hardware features, despite the mess it can cause.

2.4 LIMITATIONS

- OpenGL is case sensitive
- Line Color, Filled Faces and Fill Color not supported.

- Bump mapping is not supported.
- Shadow plane is not supported.

CHAPTER 3

ANALYSIS

3.1 HARDWARE REQUIREMENTS

The Hardware requirements are very minimal and the program can be run on most of the machines.

Processor	:	Intel Core i5 processor
Processor Speed	:	2.5 GHz
RAM	:	4 GB
Storage Space	:	400 GB
Monitor Resolution	:	1024*768 or 1336*768 or 1280*1024

3.2 SOFTWARE REQUIREMENTS

Operating System	:	Windows 7 and above
IDE	:	Microsoft Visual Studio with C++ (version 6)

OpenGL libraries, Header Files which includes GL/glut.h, Object File Libraries, glu32.lib,opengl32.lib,glut32.lib,DLLfiles,glu32.dll,glut32.dll,opengl32.dll.

CHAPTER 4

SYSTEM DESIGN

The description of all the functions used in the program is given below:

- **void glutInitDisplayMode (unsigned int mode)**

This function requests a display with the properties in mode. The value of mode is determined by the logical OR of options including the color model (GLUT_RGB, GLUT_INDEX) and buffering (GLUT_SINGLE, GLUT_DOUBLE).

- **void glutInitWindowPosition (int x, int y)**

This specifies the initial position of top-left corner of the windows in pixels.

- **void glutInitWindowSize (int width, int height)**

This function specifies the initial height and width of the window in pixels.

- **void glutCreateWindow (char *title)**

This function 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.

- **void glutDisplayFunc (void (*func) (void))**

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

- **void glClearColor(GLclampfr,GLclampf g, GLclampfb,GLclampf a)**

This sets the present RGBA clear colour used when clearing the colour buffer. Variables of type GLclampf are floating point numbers between 0.0 and 1.0.

- **void glClear(GLbitfield mask)**

It clear buffers to present values. The value of mask is determined by the bitwise OR of options GL_COLOR_BUFFER_BIT, GL_DEPTH_BUFFER_BIT

- **void glutPostRedisplay ()**

This function requests that the display callback be executed after the current callback returns.

- **void glutReshapeFunc (void *f (int width, int height)**

This function registers the reshape callback function f. The callback function returns the height and width of the new window. The reshape callback invokes a display callback.

- **void glViewport (int x, int y, GLsizei width, GLsizei height)**

This function specifies a width*height viewport in pixels whose lower left corner is at (x, y) measured from the origin of the window.

- **void glMatrixMode (GLenum mode)**

This function specifies which matrix will be affected by subsequent transformations. Mode can be GL_MODEL_VIEW, GL_PROJECTION, GL_TEXTURE.

- **void glLoadIdentity ()**

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

- **void gluOrtho2D(GLdoubleleft, GLdouble right, GLdouble bottom, GLdouble top)**

This function defines a two-dimensional viewing rectangle in the plane z=0.

- **void glutMouseFunc (void *f (int button, int state, int x, int y)**

This function registers the mouse callback function f. The callback function returns the button(GLUT_LEFT_BUTTON,GLUT_MIDDLE_BUTTON,GLUT_RIGHT_BUTTON), the state of the button after the event (GLUT_UP, GLUT_DOWN), and the position of the mouse relative to the top-left corner of the window.

- **void glVertex3f(TYPE xcoordinate, TYPE ycoordinate, TYPE zcoordinate)**
  void glVertex3fv(TYPE *coordinates)

This specifies the position of a vertex in 3 dimensions. If v is present, the argument is a pointer to an array containing the coordinates.

- **void glBegin(glEnum mode)**

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

- **void glEnd()**

This function terminates a list of vertices.

- **void glutMainLoop()**

This function causes the program to enter an event processing loop. It should be the last statement in main.

- **void glPushMatrix(void);**

Pushes all matrices in the current stack down one level. The current stack is determined by glMatrixMode(). The topmost matrix is copied, so its contents are duplicated in both the top and second-from-the-top matrix. If too many matrices are pushed, an error is generated.
Void glRotate{fd} (TYPE angle, TYPE x, TYPE y, TYPE z);
voidglTranslate{fd} (TYPE x, TYPE y, TYPE z);
Void glScale {fd} (TYPE x, TYPE y, TYPE z);

21. **Void glPopMatrix (void);**
Pops the top matrix off the stack, destroying the contents of the popped matrix. What was the second-from-the-top matrix becomes the top matrix. The current stack is determined by glMatrixMode(). If the stack contains a single matrix, calling glPopMatrix() generates an error.

- **glSelectBuffer(size,pickbuff)** : To set up a pick-buffer array
- **glRenderMode(GL_SELECT) :** To activate OpenGL picking operations
- **glInitNames() :** To activate the integer-ID name stack for the picking operations
- **glPushName(ID) :** To place an unsigned interger value on the stack
- **glGet() :** Query function to copy specified state values into an array(requires specification of data type, symbolic name of a state parameter and an array pointer)
- **gluPickMatrix(xPick,yPick,widthPick,heightPick,vpArray) :** To define a pick window within a selected viewport
- **glPopMatrix()** : To destroy the matix on top of the stack and to make the second matrix on the stack become the current matrix

- **glutMouseFunc(mousefcn)** : Specify a mouse callback function that is to be invoked when a mouse button is pressed

- **glutMotionFunc(motionfcn) :** Specify a mouse callback function that is to be invoked when the mouse cursor is moved while a button is pressed

- **glutKeyboardFunc(keyboardfcn)** : Specify a keyboard callback function that is to be invoked when a standard key is pressed

- **glutCreateMenu(menuFunc) :** To create a popup menu and specify the procedure to be invoked when a menu item is selected

- **glutAddMenuEntry(charString,menuItemNumber) :** To list the name and position for each option

- **glutAttachMenu(button)** : To specify a mouse button that is to be used to select a menu option

CHAPTER 5

IMPLEMENTION

```
#include<iostream>
#include<fstream>
#include <GL/glut.h>
#include <GL/gl.h>

int press_x, press_y;
int x_shift,z_shift; //glutLookAT
int release_x, release_y;
float x_angle = 0.0;
float y_angle = 0.0;
float scale_size = 1;

int xform_mode = 0;

void (*varFunc)();
int menu_mode = 0;


#define XFORM_NONE    0
#define XFORM_ROTATE  1
#define XFORM_SCALE 2

int stage = 0;
int case_id = 0;
float base_translate = 0;
float low_rotate = 0;
float up_rotate = 90;
float hammer_rotate = 0;

int show_axis=-1;
int poly_fill = 0;

GLfloat objectXform[4][4] = {
            1,0,0,0,
            0,1,0,0,
            0,0,1,0,
            0,0,0,1};


void display() // THE DISPLAY FUCTION
{
   glEnable(GL_DEPTH_TEST);
```

```
glClearColor(0,0,0,1);
glClear(GL_COLOR_BUFFER_BIT|GL_DEPTH_BUFFER_BIT);
//Uses the current lighting parameters to compute the vertex color.
glEnable(GL_LIGHTING);
glEnable(GL_LIGHT0);
glLightModeli(GL_LIGHT_MODEL_TWO_SIDE, GL_TRUE);
glEnable(GL_NORMALIZE);
//If enabled, have ambient and diffuse material parameters track the current color.
glEnable(GL_COLOR_MATERIAL);

glColorMaterial(GL_FRONT_AND_BACK, GL_AMBIENT_AND_DIFFUSE);

GLfloat light_ambient[] = {.0,.0,.0,1};
GLfloat light_diffuse[] = {.8,.8,.8,1};
GLfloat light_specular[] = {1,1,1,1};
GLfloat light_pos[] = {0,0,0,1};

glMatrixMode(GL_PROJECTION);
glLoadIdentity();
gluPerspective(10, 1, .1, 150);
glMatrixMode(GL_MODELVIEW);
glLoadIdentity();

glLightfv(GL_LIGHT0,GL_AMBIENT, light_ambient);
glLightfv(GL_LIGHT0, GL_DIFFUSE, light_diffuse);
glLightfv(GL_LIGHT0, GL_SPECULAR, light_specular);
glLightfv(GL_LIGHT0, GL_POSITION, light_pos);

GLfloat mat_specular[] = {.7, .7, .7,1};
GLfloat mat_shine[] = {60};

glMaterialfv(GL_FRONT_AND_BACK, GL_SPECULAR, mat_specular);
glMaterialfv(GL_FRONT_AND_BACK, GL_SHININESS, mat_shine);

gluLookAt(30,30,30,x_shift,0,z_shift,0,1,0);

glRotatef(x_angle, 0, 1,0);
glRotatef(y_angle, 1,0,0);
glScalef(scale_size, scale_size, scale_size);

glDisable(GL_LIGHTING);
glEnable(GL_LIGHTING);

if (!poly_fill) glPolygonMode(GL_FRONT_AND_BACK, GL_LINE);
else glPolygonMode(GL_FRONT_AND_BACK, GL_FILL);
if (!poly_fill) glDisable(GL_LIGHTING);
else glEnable(GL_LIGHTING);


glPushMatrix();
glTranslatef(0,-5,0);
```

```
    glScalef(200, 1, 200);       // floor
    glutSolidCube(1);
    glPopMatrix();
    glLineWidth(1);

    std::fstream file2("plot2.txt", std::ios_base::in);

    std::fstream file("traj2.txt", std::ios_base::in);
    float x, y, theta, x_obs=5, y_obs=5, z_obs;
    while (file >> x >> y >> theta)
    {
       glColor3f(0,1,0);
       glPushMatrix();
       glTranslatef(x * 10, 0, y * 10);
       glRotatef(theta, 0, 1, 0);
       glutSolidCube(1);

       while(file2 >> x_obs>> y_obs)
       {
          if(x_obs == -1 and y_obs == -1) break;
          x_obs /= 32; //IMP
          y_obs /= 48;

          if(menu_mode == 1 || menu_mode == 2)
          {
             glPointSize(5);
             glColor3f(.5 ,1* x_obs / 25, 1 * y_obs / 25);
             glBegin(GL_POINTS);
             glVertex3f(x_obs,0,y_obs);
             glEnd();
          }
          if(menu_mode == 2)
          {
             glPointSize(5);
             glColor3f(.5,.5, .5);
             glBegin(GL_LINES);
             glVertex3f(0,0,0);
             glVertex3f(x_obs,0,y_obs);
             glEnd();
          }
       }
       glPopMatrix();
    }
    glRotatef(-90, 1, 0, 0);
    glMultMatrixf((const float*)objectXform);
    glutSwapBuffers();
}


void mymouse(int button, int state, int x, int y) // ROTATION and SCALING MAP
{
```

```
  if (state == GLUT_DOWN)
  {
    press_x = x; press_y = y;
    if (button == GLUT_LEFT_BUTTON)
       xform_mode = XFORM_ROTATE;
    else if (button == GLUT_RIGHT_BUTTON)
       xform_mode = XFORM_SCALE;
  }
  else if (state == GLUT_UP)
              xform_mode = XFORM_NONE;
}


void mymotion(int x, int y) // ROTATION and SCALING MAP
{
   if (xform_mode==XFORM_ROTATE)
   {
      x_angle += (x - press_x);//5.0;
      if (x_angle > 180) x_angle -= 360;
      else if (x_angle <-180) x_angle += 360;
      press_x = x;

      y_angle += (y - press_y);//5.0;
      if (y_angle > 180) y_angle -= 360;
      else if (y_angle <-180) y_angle += 360;
      press_y = y;
   }
            else if (xform_mode == XFORM_SCALE)
            {
      float old_size = scale_size;
      scale_size *= (1+ (y - press_y)/60.0);
      if (scale_size <0) scale_size = old_size;
      press_y = y;
   }
            glutPostRedisplay();
}


void mykey(unsigned char key, int x, int y) // MOVING ROBOT
{
   switch(key)
   {
                        case 'q': exit(1);
                                 break;
                        case 'r':  x_shift=0, z_shift = 0;
                                 break;
                        case 'u': up_rotate +=5; show_axis=3;
                                 break;
                        case 'h': hammer_rotate +=5;show_axis=4;
                                 break;
                        case 'f': poly_fill = !poly_fill;
```

```
                        break;
                case 'n': show_axis=-1;
                        break;
                case 'w':
                        glMatrixMode(GL_MODELVIEW);
                        glLoadMatrixf((GLfloat*) objectXform);
                        glTranslatef(.1,0,0);
                        glGetFloatv( GL_MODELVIEW_MATRIX, (GLfloat *)
objectXform );
                        show_axis =1;
                        break;
                case 's':
                        glMatrixMode(GL_MODELVIEW);
                        glLoadMatrixf((GLfloat*) objectXform);
                        glTranslatef(-1,0,0);
                        glGetFloatv( GL_MODELVIEW_MATRIX, (GLfloat *)
objectXform );
                        break;
                case 'a':
                        glMatrixMode(GL_MODELVIEW);
                        glLoadMatrixf((GLfloat*) objectXform);
                        glRotatef(30, 0, 0, 1);
                        glGetFloatv( GL_MODELVIEW_MATRIX, (GLfloat *)
objectXform );
                        break;
                case 'd':
                        glMatrixMode(GL_MODELVIEW);
                        glLoadMatrixf((GLfloat*) objectXform);
                        glRotatef(-30, 0, 0, 1);
                        glGetFloatv( GL_MODELVIEW_MATRIX, (GLfloat *)
objectXform );
                        break;
            }
   glutPostRedisplay();
}


void SpecialInput(int key, int x, int y) // UPDATE gluLookAT();
{
   switch(key)
   {
      case GLUT_KEY_UP:
         z_shift -= 3;
         break;
      case GLUT_KEY_DOWN:
         z_shift += 3;
         break;
      case GLUT_KEY_LEFT:
         x_shift -= 3;
         break;
      case GLUT_KEY_RIGHT:
```

```
            x_shift += 3;
            break;
    }
    glutPostRedisplay();
}



void menuFunc(int op) // MENU FUNCTION
{
    if(op==1)
            menu_mode = 0;
    else if(op==2)
            menu_mode = 1;
    else if(op==3)
            menu_mode = 2;
    else if(op==4)  //TODO
            menu_mode = 3;
    else if(op==5)
            exit(0);
    glutPostRedisplay();
}


int main(int argc, char** argv)
{
    glutInit(&argc, argv);
    glutInitDisplayMode(GLUT_RGB|GLUT_DOUBLE|GLUT_DEPTH);
    glutInitWindowSize(1000,1000);

    glutCreateWindow("proj");
    glutPostRedisplay();
    glutCreateMenu(menuFunc);
    glutAddMenuEntry("Trajectory",1);
    glutAddMenuEntry("Interest Points",2);
    glutAddMenuEntry("Draw Interest Lines",3);
    glutAddMenuEntry("Model Objects",4);
    glutAddMenuEntry("Quit",5);
    glutAttachMenu(GLUT_MIDDLE_BUTTON);

    glutDisplayFunc(display);
    glutMouseFunc(mymouse);
    glutMotionFunc(mymotion);
    glutSpecialFunc(SpecialInput);
    glutKeyboardFunc(mykey);
    glutMainLoop();
}
```

CHAPTER 6

TESTING

In unit testing, the program modules that make up the system are tested individually. Unit testing focuses to locate errors in the working modules that are independent to each other. This enables to detect errors in coding and the logic within the module alone. This testing is also used to ensure the integrity of the data stored. The various routines were checked by passing the inputs and the corresponding output is tested. Test cases used in the project as follows:

Serial No	Metric	Description	Observation
1	**Left Mouse Button**	Rotate 3D-Map with Origin as pivot.	3D-Map is rotated with Origin as pivot.
2	**Right Mouse Button**	Scale 3D-Map	3D-Map increases/decreases in size as the Right Mouse Button is pressed.
3	**Mouse Function**	Has 4 options: Trajectory, Interest Points, Draw Interest Lines and Quit.	Results obtained as expected

CHAPTER7

RESULTS AND SNAPSHOTS

Reporting test execution results is very important part of testing, whenever test execution cycle is complete, tester should make a complete test results report which includes the Test Pass/Fail status of the test cycle. If manual testing is done then the test pass/fail result should be captured in an excel sheet and if automation testing is done using automation tool then the HTML or XML reports should be provided to stakeholders as test deliverable. Several errors were detected and rectified and the whole project is working as it should have to work with proper output and high efficiency.

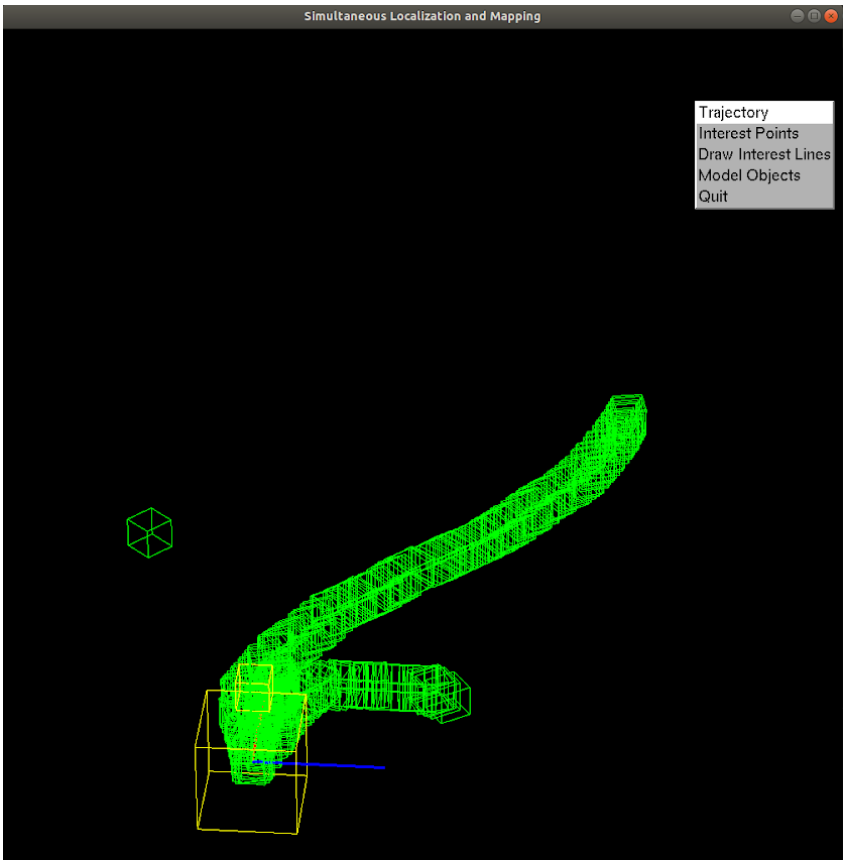

Fig 7.2.1: Display on selecting the option “Trajectory”

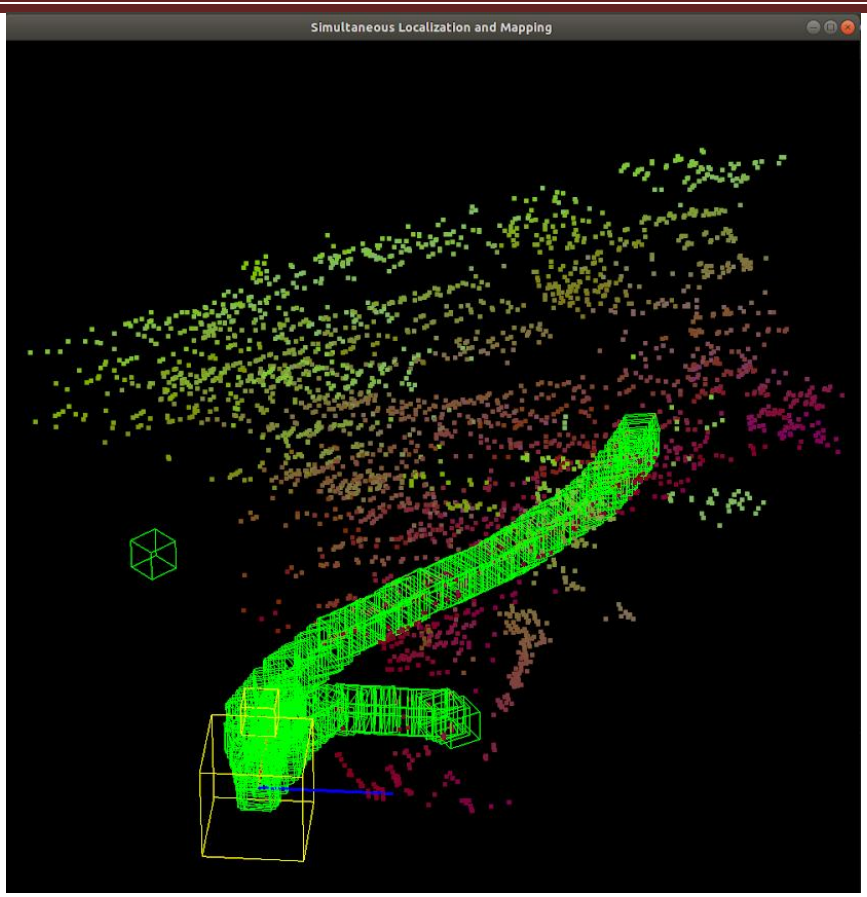

Fig 7.2.2: Interest Points

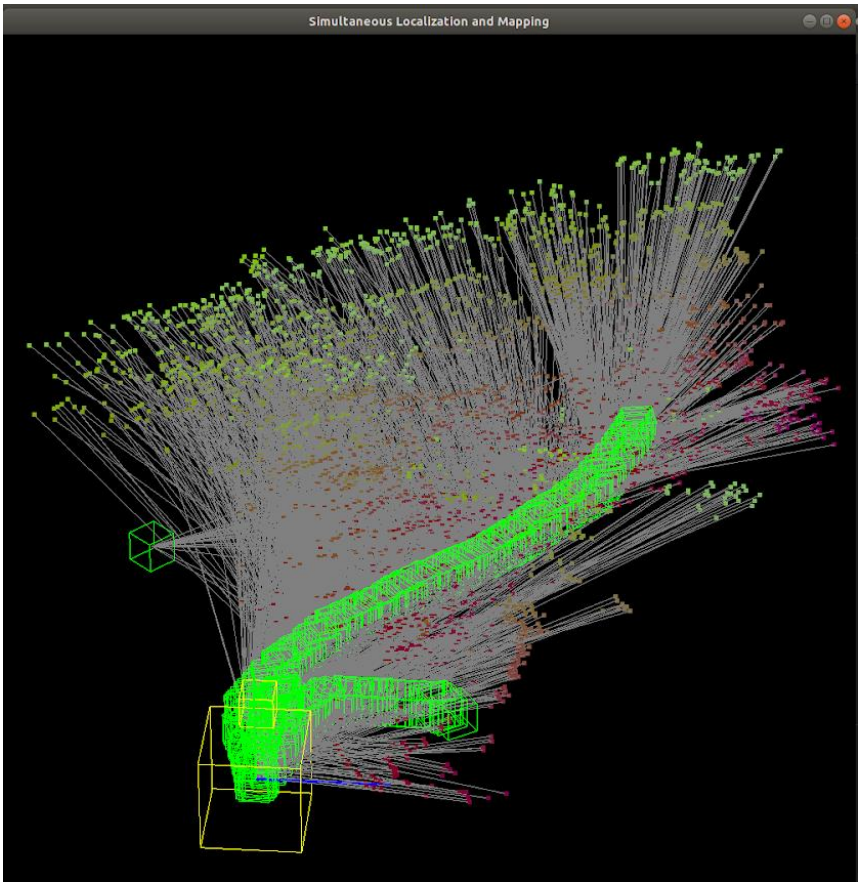

Fig 7.2.3: Draw Interest Lines

CHAPTER 8

CONCLUSION AND FUTURE ENHANCEMENTS

This project is an effort in the development of a Graphical Software package which is the building block of graphical application. During the development of this package, effort leads to understanding the display of geometric primitives like rectangles, lines, line loops, polygon, modes of display, and features like translation were also designed. Various functions and operations of the graphical library provide the learning platform to get the maximum performance of the OpenGL functions.

The project "SIMULTANEOUS LOCALIZATION AND MAPPING" has been successfully implemented using OpenGL. The illustrations of graphical principles and OpenGL features are included and application program is efficiently developed.

The project enlightens the basic idea of scaling, rotation and translation of the objects. Since it uses interaction with both keyboard and mouse it is sufficiently easy for any kind of end user to run it. However user requirement changes and provision for changes in design has been allowed. If no major requirements are demanded from the user, the current design holds. Thus this project meets the basic requirements successfully and is flexible in all aspects. On conclusion, this mini project is implemented with standard OpenGL functions.

An attempt has been made to develop a Graphical Software package, which meets the necessary requirement of the user successfully. The objects in the project can move with respect to x and z axis, it can be enhanced to 3D movements more precisely. We can also change the color and provide more effects

Implementation of the other graphical applications will definitely enhance the representation and view of the project. The mini project can further be enhanced to demonstrate the use of other OpenGL functions of higher degree.

Since it is user friendly it enables the user to interact efficiently and easily. Thus the package designed demonstrates the Basic Structures Orientation of some objects to the user.

REFERENCES

1. Edward Angel, “Interactive Computer Graphics A Top-Down Approach with OpenGL” 5th Edition, Addison-Wesley, 2008.

2. F.S. Hill, Jr., “Computer Graphics Using OpenGL”, 2nd Edition, Pearson Education, 2001.

3. JamesD.Foley, AndriesVan Dam, StevenK.Feiner, JohnF.Hughes, ”Computer Graphics”, Second Edition, Addison-Wesley Proffesional, August 14,1995

4. www.opengl.org.
5. https://www.khronos.org/opengl/
6. http://www.opengl-tutorial.org/
7. https://learnopengl.com/
8. https://stackoverflow.com/