RV COLLEGE OF ENGINEERING®
BENGALURU – 560059
(Autonomous Institution Affiliated to VTU, Belagavi)

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

“Dynamic Graph Library Using OpenGL”

COMPUTER GRAPHICS LAB (16CS73)

OPEN ENDED EXPERIMENT REPORT

VII SEMESTER

2020-2021

Submitted by

Sathvik K R	**1RV17CS141**
S Suraj	**1RV17CS130**

Under the Guidance of

Prof. Mamatha T
Department of CSE, R.V.C.E.,

Bengaluru - 56005

ABSTRACT

A 2D/3D graphics-based dynamic graphs are in demand in the industry for statistical analysis & visualization. The development of the library has large scope to learn computer graphics from scratch. The main objective of this project is to plot 5 types of graphs based on the coordinates given by the user, in real time, with interactive mouse and keyboard connectivity. OpenGL utility toolkit has been used and the object-oriented approach has been chosen for the implementation.

There is still scope left in the development of project like, increasing the number of variants of the graphs, a need to embed a button interface, increase the plotting capacity of each graph, tutorial guide, etc. In future we hope we would implement it in the source code for better experience of using this application.

Finally, we could say by developing the library, most of the primitive attributes and functions of OpenGL have been implemented which has provided a transparent understanding of the concepts of OpenGL.

RV COLLEGE OF ENGINEERING®, BENGALURU - 560059
(Autonomous Institution Affiliated to VTU, Belagavi)

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

CERTIFICATE

Certified that the **Open-Ended Experiment** titled **"Dynamic Graph Library using OpenGL"** has been carried out by **Sathvik K R (1RV17CS141) and S Suraj (1RV17CS130),** bonafide students of RV College of Engineering, Bengaluru, have submitted in partial fulfillment for the **Internal Assessment of Course: COMPUTER GRAPHICS LAB (16CS73)** during the year 2020-2021. It is certified that all corrections/suggestions indicated for the internal Assessment have been incorporated in the report.

Prof. Mamtha T	**Dr. Ramakanth P**
Faculty Incharge,	Head of Department,
Department of CSE,	Department of CSE,
R.V.C.E , Bengaluru - 59.	R.V.C.E , Bengaluru - 59.

RV COLLEGE OF ENGINEERING® , BENGALURU - 560059
(Autonomous Institution Affiliated to VTU)

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

DECLARATION

We, **Sathvik K R (1RV17CS141)** and **S Suraj (1RV17CS130),** the students of Seventh Semester B.E., Computer Science and Engineering, R.V. College of Engineering, Bengaluru hereby declare that the mini-project titled **" Dynamic Graph Library using OpenGL"** has been carried out by us and submitted in partial fulfillment for the **Internal Assessment of Course: COMPUTER GRAPHICS LAB (16CS73) - Open-Ended Experiment** during the year 2020-2021. We do declare that matter embodied in this report has not been submitted to any other university or institution for the award of any other degree or diploma.

Place: Bengaluru	**Sathvik K R**
Date:	**S Suraj**

ACKNOWLEDGEMENT

I take this opportunity to express my profound gratitude and deep regards to my guide Prof. Mamatha T for her exemplary guidance, monitoring and constant encouragement throughout the course of this project. The blessing, help and guidance given by her time to time shall carry me a long way in the journey of life on which I am about to embark.

I also take this opportunity to express a deep sense of gratitude to Dr.Ramakanth Kumar P, HoD, Department of Computer science and Engineering, R V College of Engineering ,for his valuable information and guidance, which helped me in completing this task through various stages.

I am obliged to my peers, for the valuable information provided by them in their respective fields. I am grateful for their cooperation during the period of my assignment.

Lastly, I thank almighty, my parents, brother, sisters and friends for their constant encouragement without which this assignment would not be possible.

Sathvik K R

S Suraj

Table of Contents

List of Figures

Chapter 1

Introduction

1.1 Computer Graphics

To draw a picture say, fish moving inside the water. Suddenly will get an idea to use paint, but the degree of accuracy, quality of image is not satisfied and become very sad. There is no need to worry, for every problem there will be a solution, so this problem of creating fish moving inside the water can be solved using COMPUTER GRAPHICS without any difficulties.

Computer Graphics become a powerful tool for the rapid and economical production of pictures. There is virtually no area in which Graphical displays cannot be used to some advantage so it is not surprising to find the use of CG so widespread.

Although early application in engineering & science had to rely on expensive & cumbersome equipments, advances in computer technology have made interactive computer graphics a practical tool.

Computer Graphics in a diverse area such as science, engineering, medicine, business, industry, government, art, entertainment, education and training.

Now it can be answered about computer graphics as generalized tool for drawing and creating pictures and simulates the real world situations within a small computer window.

.

1.2 OpenGL

1.2.1 OpenGL Graphics Architecture

Most of the application will be designed to access OpenGL directly through functions in three libraries. Functions in the main GL (or OpenGL in windows) library have names that begin with the letters gl and are stored in a library usually referred to as GL (or OpenGL in windows). The second is the **OpenGL Utility Library** (GLU). This library uses only GL functions but contains code for creating common objects and simplifying viewing. All functions in GLU can be created from the core GL library but application programmers prefer not to write the code repeatedly. The GLU library is available in all OpenGL implementations; functions in the GLU library begin with letters glu.

To interface with the window system and to get input from external devices into the programs, need at least one more system-specific library that provides the "glue" between the window system and OpenGL. For the X window system, this library is functionality that should be expected in any modern windowing system.

Fig 2.1 shows the organization of the libraries for an X Window System environment. For this window system, GLUT will use GLX and the X libraries. The application program, however, can use only GLUT functions and thus can be recompiled with the GLUT library for other window systems.

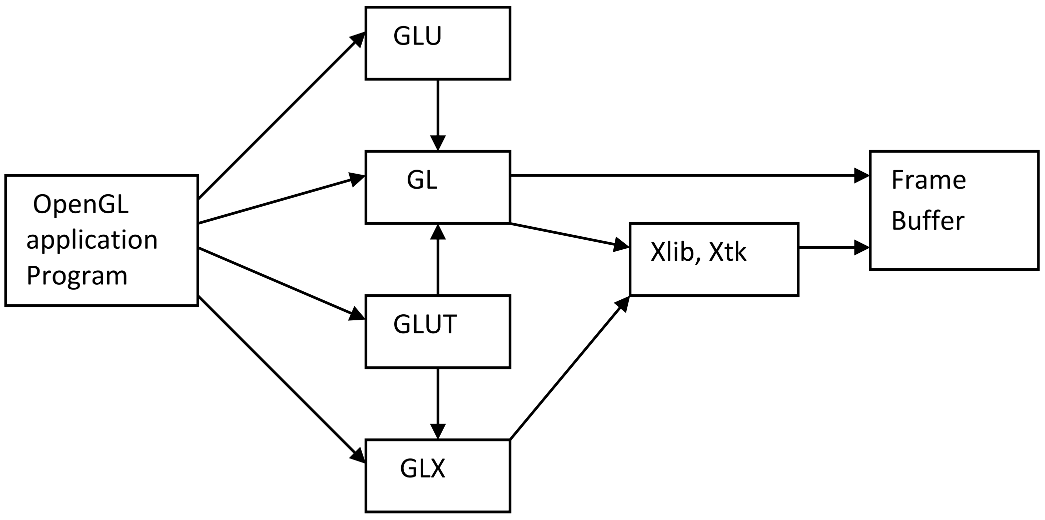

Fig 1. Library organization of OpenGL

<u>Libraries</u>

1. The **OpenGL Utility Library** (**GLU**) is a computer graphics library for OpenGL.

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. GLU is not implemented in the embedded version of the OpenGL package, OpenGL ES.

Among these features are mapping between screen- and world-coordinates, generation of texture mipmaps, drawing of quadric surfaces, NURBS, tessellation of polygonal primitives, interpretation of OpenGL error codes, an extended range of transformation routines for setting up viewing volumes and simple positioning of the camera, generally in more human-friendly terms than the routines presented by OpenGL. It also provides additional primitives for use in OpenGL applications, including spheres, cylinders and disks.

All GLU functions start with the glu prefix. An example function is gluOrtho2D which defines a two dimensional orthographic projection matrix.

2. The **OpenGL Utility Toolkit** (**GLUT**) is a library of utilities for OpenGL programs, which primarily perform system-level I/O with the host operating system. Functions performed include window definition, window control, and monitoring of keyboard and mouse input. Routines for drawing a number of geometric primitives (both in solid and wireframe mode) are also provided, including cubes, spheres and the Utah teapot. GLUT also has some limited support for creating pop-up menus.

GLUT was written by Mark J. Kilgard, author of *OpenGL Programming for the X Window System* and *The Cg Tutorial: The Definitive Guide to Programmable Real-Time Graphics*, while he was working for Silicon Graphics Inc.

The two aims of GLUT are to allow the creation of rather portable code between operating systems (GLUT is cross-platform) and to make learning OpenGL easier. Getting started with OpenGL programming while using GLUT often takes only a few lines of code and does not require knowledge of operating system–specific windowing APIs.

3. Core OpenGL (GL): consists of hundreds of functions, 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.

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.

Each of the software package consists of:

1. A *header* file: "gl.h" for core OpenGL, "glu.h" for GLU, and "glut.h" (or "freeglut.h") for GLUT, typically kept under "include\GL" directory.
2. A *static library*: for example, in Win32, "libopengl32.a" for core OpenGL, "libglu32.a" for GLU, "libglut32.a" (or "libfreeglut.a" or "glut32.lib") for GLUT, typically kept under "lib" directory.
3. An optional *shared library*: for example, "glut32.dll" (for "freeglut.dll") for GLUT under Win32, typically kept under "bin" or "c:\windows\system32".

It is important to locate the *directory path* and the *actual filename* of these header files and libraries in your operating platform in order to properly setup the OpenGL programming environment.

1.2.2 Primitives and Attributes

1. Line Primitives

GL_LINES: Vertices 0 and 1 are considered a line. Vertices 2 and 3 are considered a line. And so on. If the user specifies a non-even number of vertices, then the extra vertex is ignored.

GL_LINE_STRIP: The adjacent vertices are considered lines. Thus, if you pass n vertices, you will get n-1 lines. If the user only specifies 1 vertex, the drawing command is ignored.

GL_LINE_LOOP: As line strips, except that the first and last vertices are also used as a line. Thus, you get n lines for n input vertices. If the user only specifies 1 vertex, the drawing command is ignored. The line between the first and last vertices happens after all of the previous lines in the sequence.

2. Triangle Primitives

GL_TRIANGLES: Vertices 0, 1, and 2 form a triangle. Vertices 3, 4, and 5 form a triangle. And so on.

GL_TRIANGLE_STRIP: Every group of 3 adjacent vertices forms a triangle. The face direction of the strip is determined by the winding of the first triangle. Each successive triangle will have its effective face order reversed, so the system compensates for that by testing it in the opposite way. A vertex stream of *n* length will generate *n*-2 triangles.

GL_TRIANGLE_FAN: The first vertex is always held fixed. From there on, every group of 2 adjacent vertices form a triangle with the first. So with a vertex stream, you get a list of triangles like so: (0, 1, 2) (0, 2, 3), (0, 3, 4), etc. A vertex stream of *n* length will generate *n*-2 triangles.

3. Quad Primitives

GL_QUADS: Vertices 0-3 form a quad, vertices 4-7 form another, and so on. The vertex stream must be a number of vertices divisible by 4 to work.

GL_QUAD_STRIP: Similar to triangle strips, a quad strip uses adjacent edges to form the next quad. In the case of quads, the third and fourth vertices of one quad are used as the edge of the next quad. So vertices 0-3 are a quad, 2-5 are a quad, and so on. A vertex stream of *n* length will generate (*n* - 2) / 2 quads. As with triangle strips, the winding order of quads is changed for every other quad.

4. Attributes

void glColour3f(GLdouble red, GLdouble green, GLdouble blue) : Specify new red, green, and blue values for the current color.

glClearColor(): Specifies a new alpha value for the current color. Included only in the four-argument glColor4 commands.

1.2.3 Color, Viewing and Control Function

- glMatrixMode():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. Additionally, if the ARB_imaging extension is supported, GL_COLOR is also accepted.
- glTranslatef():glTranslate produces a translation by x y z . The current matrix is multiplied by this translation matrix, with the product replacing the current matrix, as if glMultMatrix were called with the following matrix for its argument:
- glColor3f():Sets the current color.
- glutInit allows application to get command line arguments and initializes system
- gluInitDisplayMode requests properties for the window (the rendering context)
  o RGB color

  - o Single buffering
  - o Properties logically ORed together
- glutWindowSize in pixels glutWindowPosition from top-left corner of display
- glutCreateWindow create window with a particular title

1.3 Proposed System

1.3.1 Objective of the project

- To implement a dynamic graphic library, consisting of line, pie, 2D bar, 3D bar and surface plot with real-time inputs.
- Graphical approach towards understanding the data to be represented.
- To make use of all essential and primitive OpenGL libraries.

1.3.2 Methodology

- The application is developed in the object oriented approach, where each graph is encapsulated into a class with appropriate attributes and functions.
- Each of the graphs are made to run on different windows simultaneously.
- A separate window is displayed to demonstrate the dynamic behavior of graph and to provide the input values, which is divided into 4 quadrants for the convenience of the user.
- Line Graph: This graph takes single array of X input and multiple arrays of Y inputs. This feature help to render multiple lines on to the graph. The scale of graph is calculated based on the maximum Y value which is given as input. The object of the line graph has different attributes which can be set according to the need.
- Pie Graph: It is a circular graph, where each input is represented as a sector. Triangle strips are used to render the sector in the graph. The number of strips, increments, etc. can be set using the object reference. It takes single array of X and Y input.
- 2D Bar Graph: Similar to pie chart, bar graph takes single array of X and Y input. For various values of X, a bar of height Y is drawn as a polygon. The width of the bars can be set as a parameter. The scale of the graph dynamically adjusts based on the input Y data.

- 3D Bar Graph: It is rendered in a three-dimensional space which takes input along x-axis, y-axis and z-axis. A mouse callback is used to move the camera to view the graph from a different perspective.
- 2D Surface Plot: For surface plot, a function pointer is passed by the user to the object. This function is used to generate a buffer of set of points. Once this buffer is generated it is passed to the rendering system for the display. There is no multiple computation of the values. The buffer is maintained at the display devices rather than moving to and from CPU to GPU.

1.3.3 Scope

The scope of this project is unbounded, finding applications in every sector where statistical analysis comes into play. It makes data interpretation easy and quick by visualizing data as multiple graphs which can be analyzed easily compared to raw figures and text. The simplicity of OpenGL makes the project very light and convenient to run even on low specification hardware. The ability of the graphs to change their form appropriately

Chapter 2

Requirement Specification

2.1 Hardware Requirements

- Processor of 2.2GHz or higher speed
- 20MB Hard Disk Space
- 1GB RAM
- Keyboard
- Mouse

2.2 Software Requirements

- Microsoft Visual Studio 9.0
- Windows-98/xp/vista/win7 Operating System
- MS-Office
- Graphics package available in Microsoft Visual Studio 9.0

Chapter 3

System Design and Implementation

3.1 Data Flow Diagram

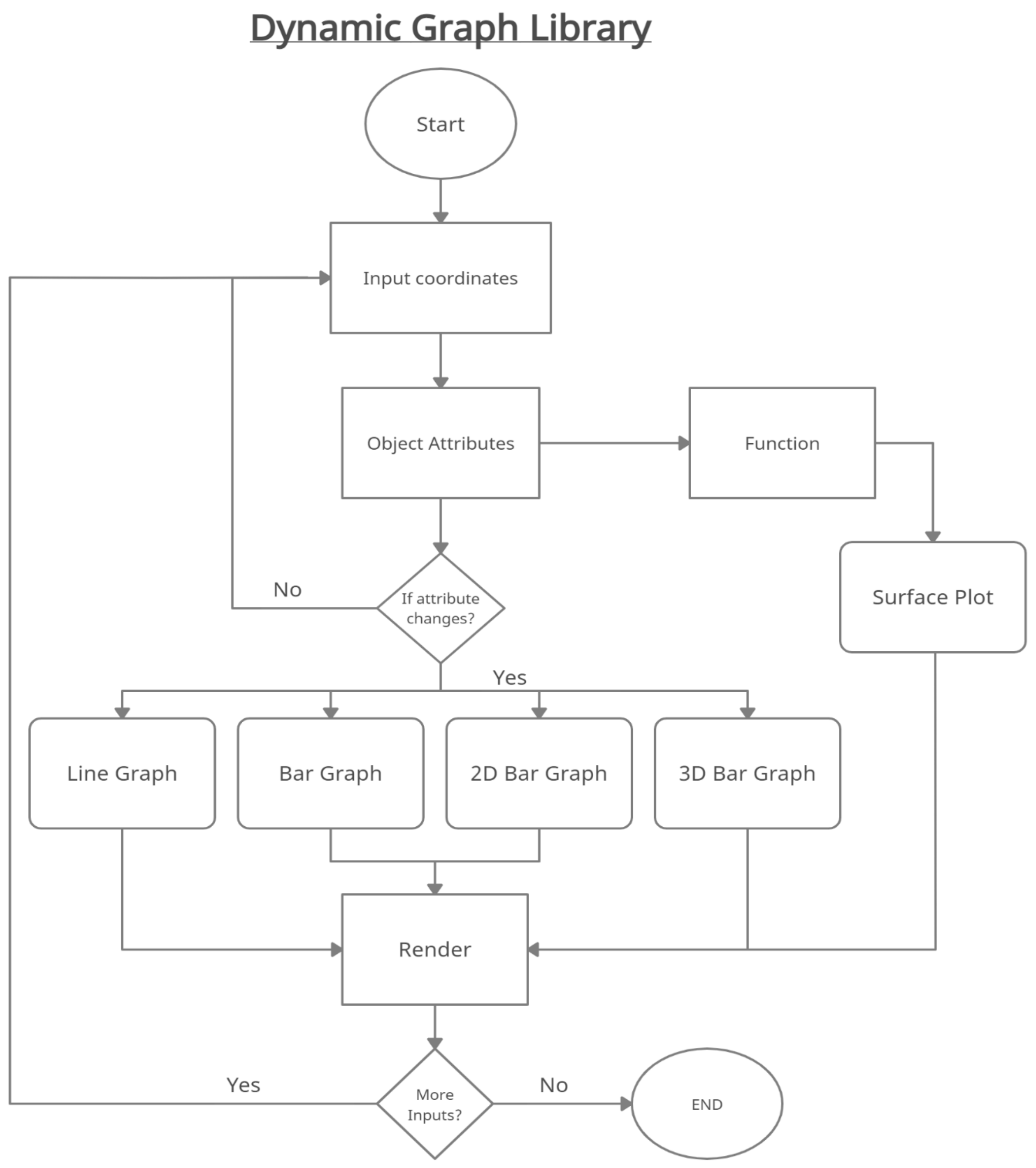

Fig 2. Flow Chart of Dynamic Graph Library

3.2 Structure Chart

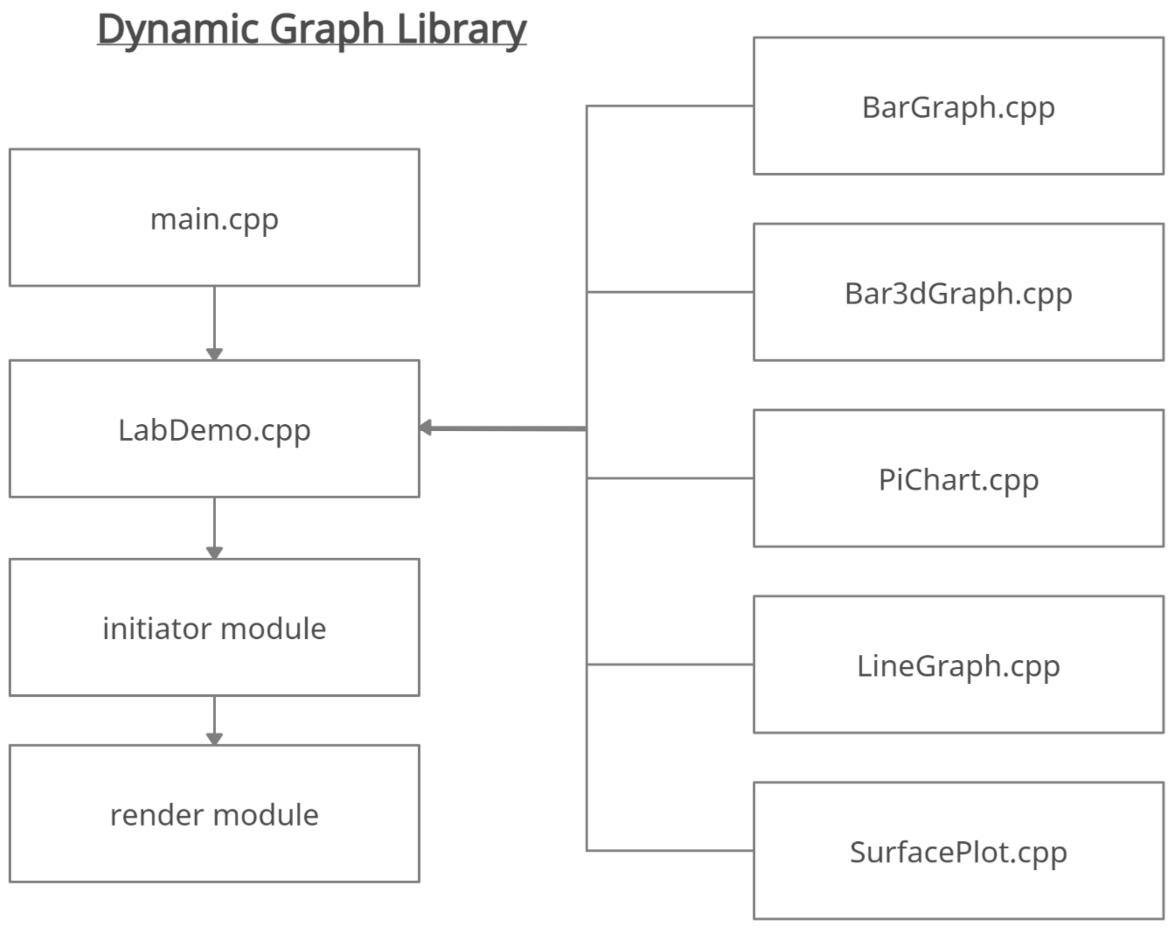

Fig 3. Structure Graph of Dynamic Graph Library

3.3 Modular Description

- **main.cpp:** This is the module which kickstarts the application. The main() function calls and initializes all the graph modules and ensures that this function which is used to display/draw the graphs, loops indefinitely.
- **LabDemo.cpp:** This is the module which implements the majority of the functionalities of the application. It binds together all other modules belonging to different graphs and prepares the final output for rendering. It instantiates the objects of various graph classes and defines the complete implementation of each of the member functions of each graph time. In addition to this, it also defines the keyboard and mouse functionalities and also the function used for fitting the surface plot.
- **utility.cpp:** This module is used for drawing 2 dimensional and 3 dimensional lines when required.
- **Graph.cpp:** Backbone module which defines the basic and generic attributes shared by each graph.
- **LineGraph.cpp:** Enables:
    - Plotting multiple lines.
    - Coloring of these lines in order to differentiate.
    - Automatic scaling.
    - Setting Ydivisions.
- **SurfacePlot.cpp:** Enables:
    - Plotting curve with either line/points.
    - Keyboard bindings for the above function.
    - Function pointer.
    - Buffer utilization with glGenBuffer(), glBindBuffer().
    - Aggregating pointers and vertices.
- **PiChart.cpp:** Enables:
    - Changing color, radius and center of pi circle.
    - Displaying the percentage coverage of each element.
    - Updating and appending new inputs.
- **BarGraph.cpp:** Enables:
    - Setting of maximum and division size of the scale.
    - Changing bar width, color, etc.
    - Dynamic scaling and plotting.

- **Bar3dGraph.cpp:** Enables:
    - Zooming in and out of the plotting area.
    - Rotation and sideward movement of the viewer.
    - Perspective viewing.

OpenGL FUNCTIONS:

- **void glBegin(glEnum mode);**

  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( );**

  Terminates a list of vertices.

- **void glColor3f[i f d] (TYPE r, TYPE g, TYPE b);**

  Sets the present RGB colors. Valid types are int (i), float (f) and double (d). The maximum and minimum values of the floating-point types are 1.0 and 0.0, respectively.

- **void glClearColor(GLclampf r,GLclampf g,GLclampf b,GLclampf a);**

  Sets the present RGBA clear color used when clearing the color buffer. Variables of GLclampf are floating-point numbers between 0.0 and 1.0.

- **int glutCreateWindow(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 where there are multiple windows.

  void glutInitWindowSize(int width, int height);

  Specifies the initial height and width of the window in pixels.

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

  Specifies the initial position of the top-left corner of the window in pixels.

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

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

- **void glFlush( );**

  Forces any buffered any OpenGL commands to execute.

- **void glutInit (int argc, char **argv);**

  Initializes GLUT. The arguments from main are passed in and can be used by the application.

- **void glutMainLoop( );**

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

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

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

- **gluOrtho2D(GLdouble left, GLdouble right, GLdouble bottom, GLdouble top);**

  Defines a two-dimensional viewing rectangle in the plane Z=0;

- **void glutBitmapCharacter(void *font, int char);**

  Renders the character with ASCII code char at the current raster position using the raster font given by font. Fonts include GLUT_BITMAP_TIMES_ROMAN_10 and GLUT_BITMAP_TIMES_ROMAN_8_Y_13. The raster position is incremented by the width of the character.

- **void glClear(GL_COLOR_BUFFER_BIT);**

  To make the screen solid and white.

- **void MouseFunc(myMouse);**

  It is used for the implementation of mouse interface.
  Passing the control to

void myMouse(int button,int state,int x,int y);

- **void KeyboardFunc(key);**

  It is used for the implementation of keyboard interface.
  Passing control to
  void key(unsigned char key,int x,int y);

- **void translate[fd](TYPE x,TYPE y,TYPE z);**

  Alters the current matrix by displacement of (x,y,z).Type is either GLfloat or GLdouble.

- **void glPushMatrix(); void glPopMatrix();**

  Pushes to and pops from the matrix stack corresponding to current matrix mode.

- **void glLoadMatrix[fd](TYPE *m);**

  Loads the 16 element array of TYPE GLfloat or GLdouble as a current matrix.

- **void glutPostRedisplay():**
  Re-render the scene again as and when the input is added dynamically.

Chapter 4

<u>Results and Snapshots</u>

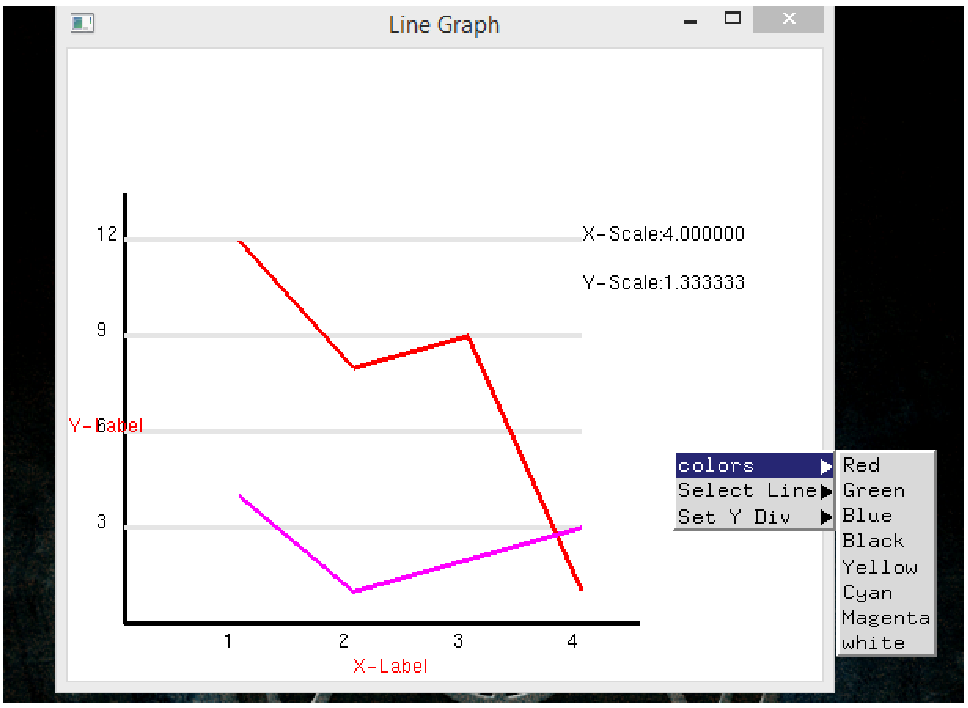

Fig 4. Line graph

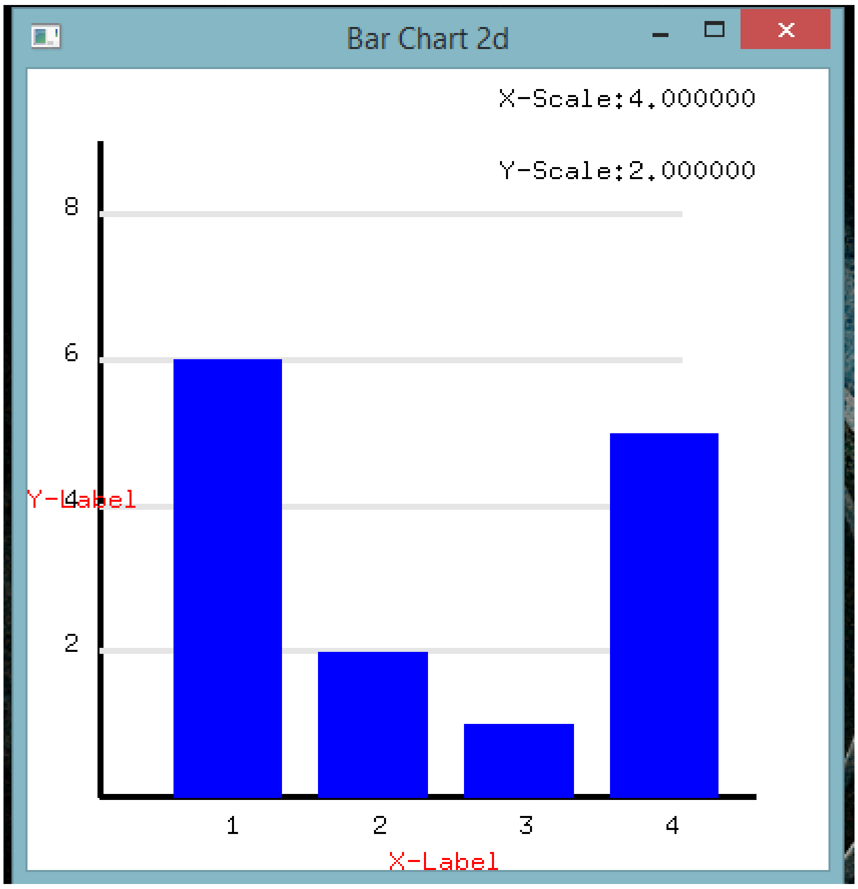

Fig 5. Bar Graph

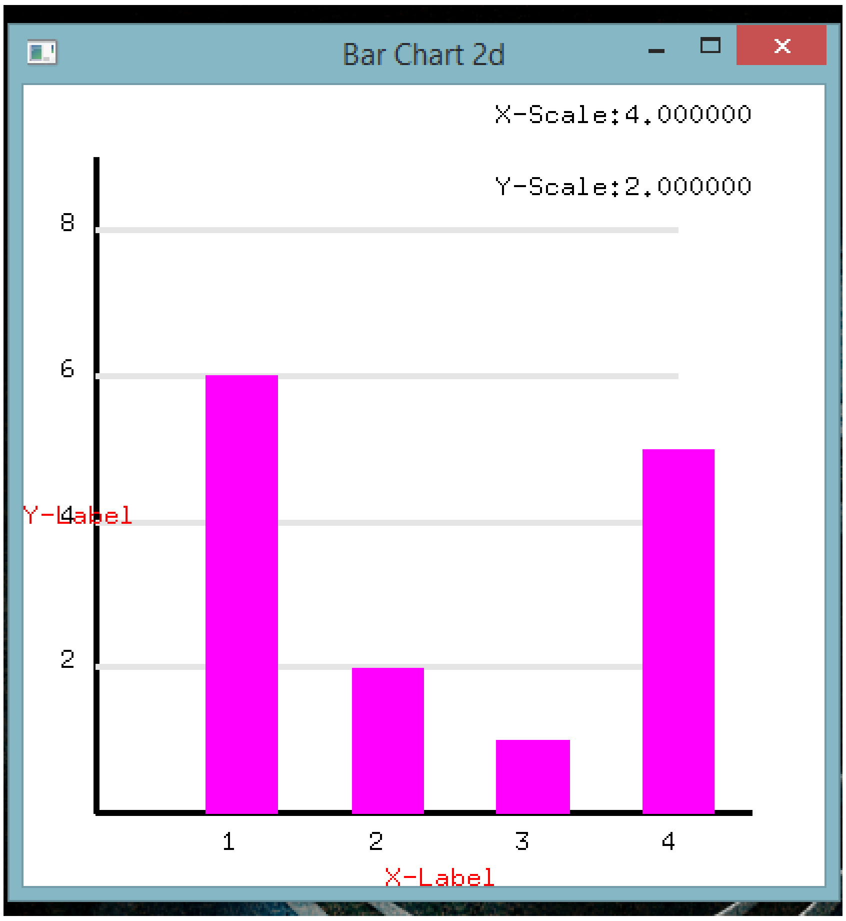

Fig 6. Bar graph (with bar width and color changed)

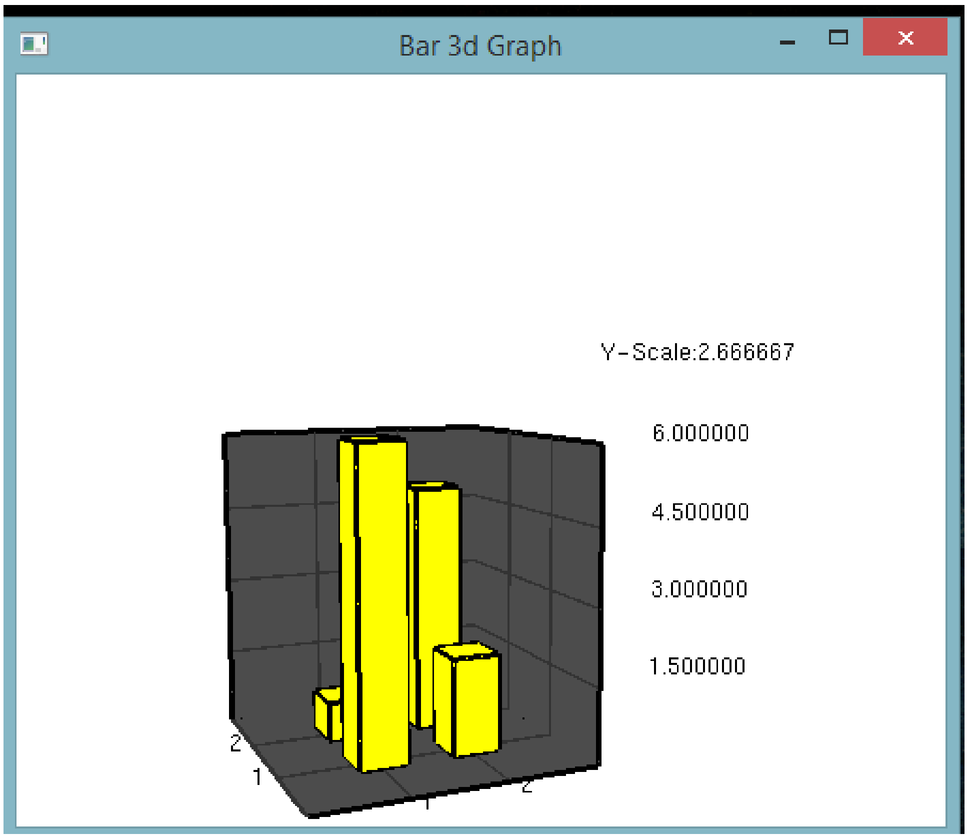

Fig 7. Bar Chart 3d

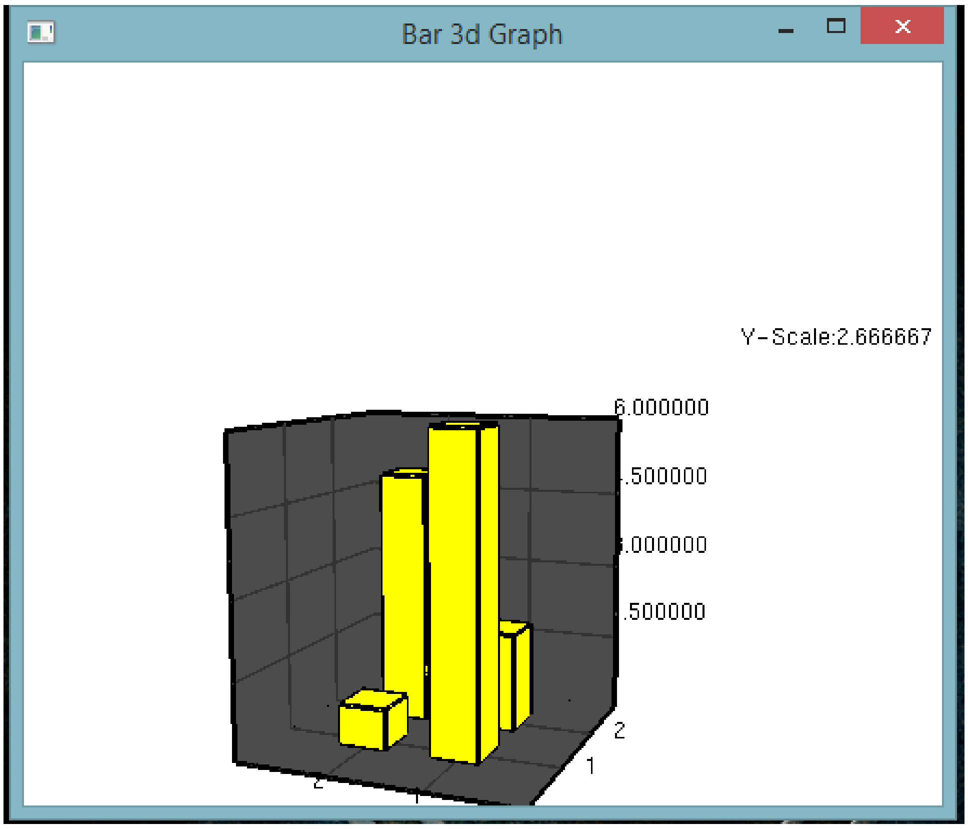

Fig 8. Bar Chart 3d from different perspective

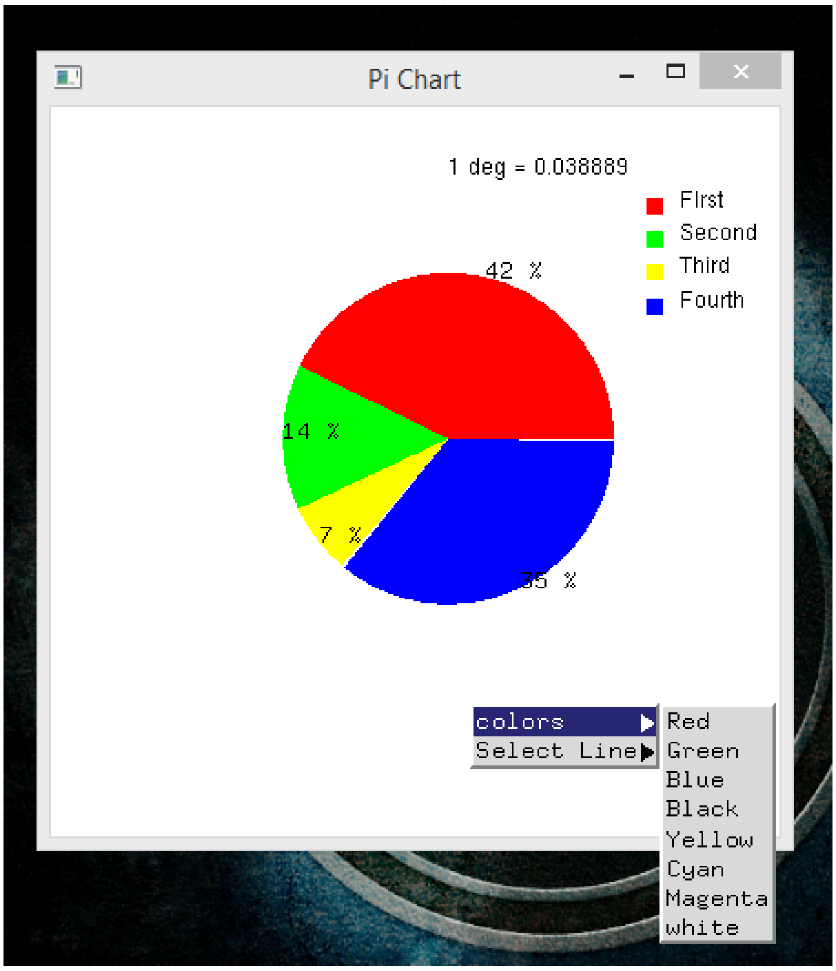

Fig 9. Pi Chart

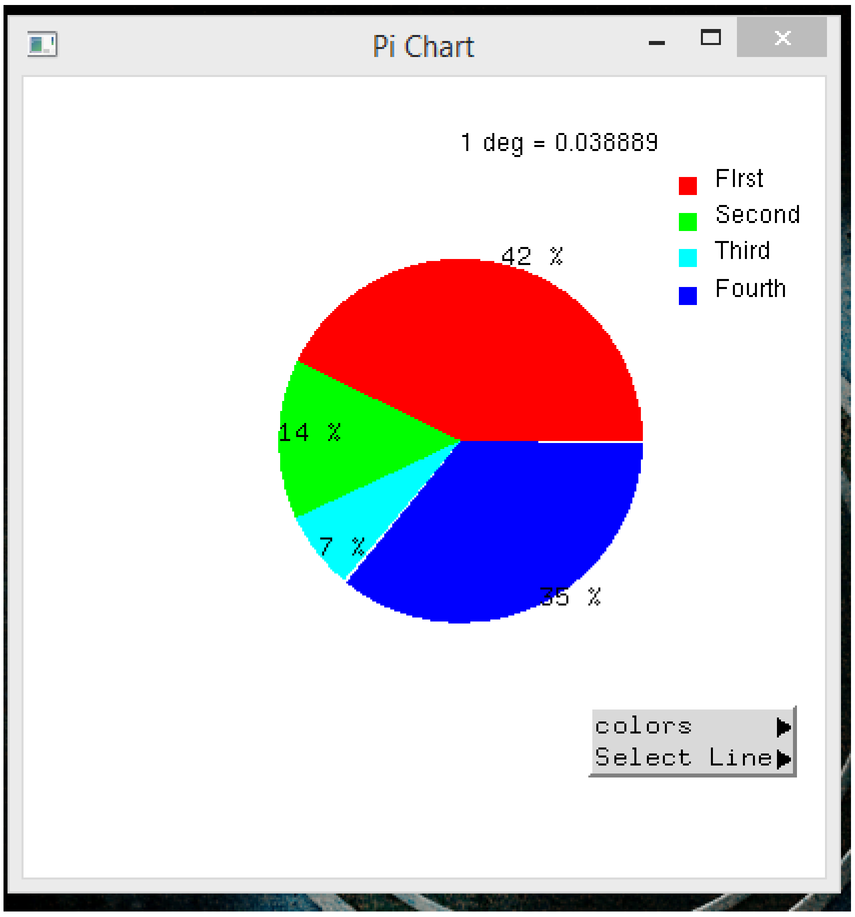

Fig 10. Pi Chart with color changed

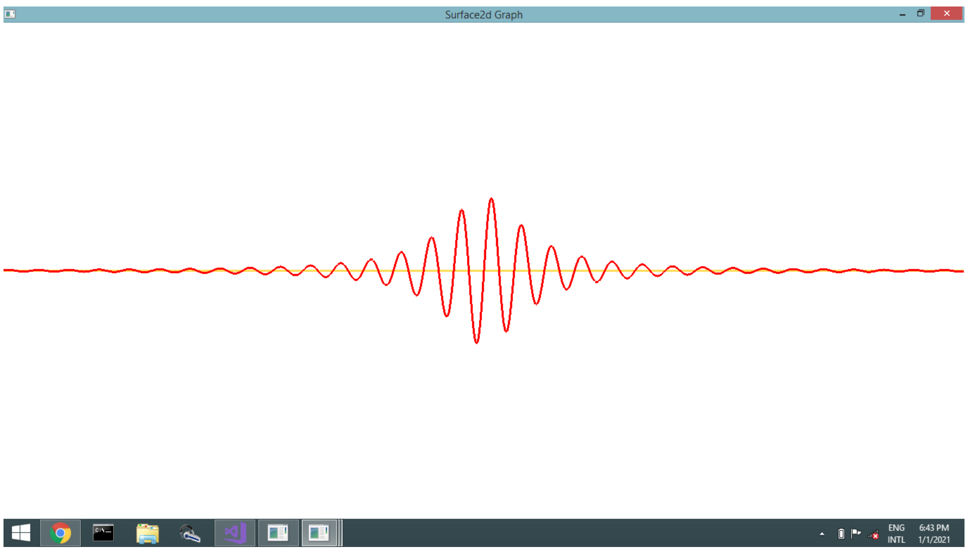

Fig 11. Surface plot for sin (x) / (1+x)^3

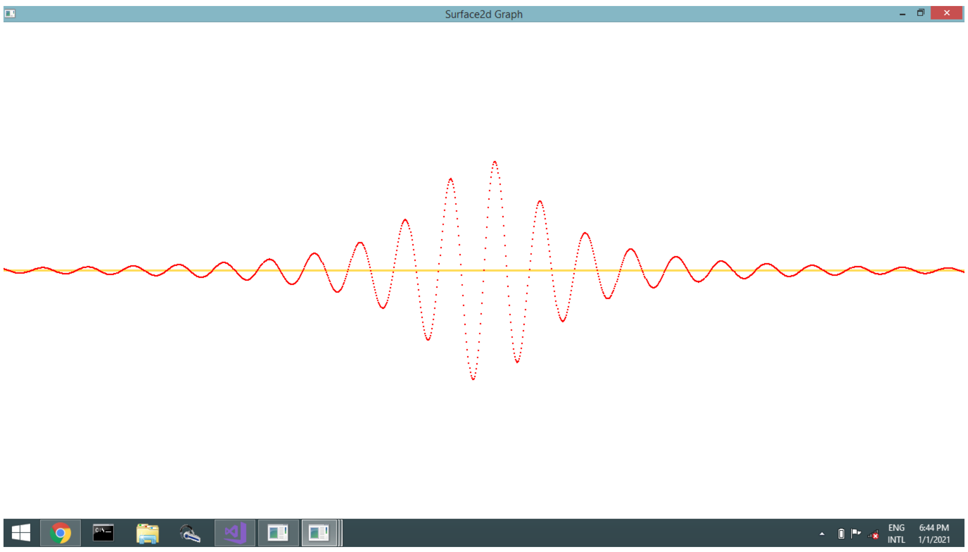

Fig 12. Surface plot zoomed and plotted using points

Chapter 5

CONCLUSION

This project was successfully done and has helped understand the core features and libraries of OpenGL. Construction and plotting of all the graphs, movement of the lines, bars and curves with real-time data, choice of color for the graphs, keyboard keys to control the projective view of the graph in the window, function estimation of the surface plot was efficiently implemented and demonstrated. This project was an interactive computer graphics assignment to demonstrate the knowledge acquired through the course.

4.1 Future scope

1. Introduce further new variants of graphs.

2. Introduce more 3D aspects to 2D graphs.

3. Make the UI more appealing and responsive.

4. Extend the application to statistical inferences, interpolations and predications.

References

1. https://www.khronos.org/registry/OpenGL/index_gl.php
2. https://www.researchgate.net/publication/255623788_Advanced_Computer_Graphics_using_OpenGL
3. Edward Angel-Interactive Computer Graphics:A Top-Down Approach using OpenGLFifth Edition, Published by Pearson Education, 2009
4. The OpenGL Programming Guide, 5thEdition. The Official guide to learning OpenGL Version 2.1 by OpenGL Architecture Review Board
5. Bruneton, E.& Neyret, F. Real-time Realistic Rendering and Lighting of Forests. Computer Graphics Forum 31.2 (2012), available at http://hal.inria.fr/hal-00650120/en.
6. available at http://www.sharecg.com/v/36271/gallery/5/3D-Model/Alder-Tree-OBJ?interstitial_displayed=Yes.
7. Paul Martz Generating Random Fractal Terrain available at http://gameprogrammer.com/fractal. html.
8. Dave Rusin Topics on Sphere Distributions available at http://www.math.niu.edu/~rusin/ known-math/95/sphere.faq. [5] WildfireGames
9. OpenGL capabilities report: GTX 470 available at http://feedback.wildfiregames. com/report/opengl/device/GeForce%20GTX%20470

BIBLIOGRAPHY

1.Computer Graphics – Principles And Practice (Foley, Van Dam, Fenier and Hughes) helped me to understand graphics generation algorithms, user interface and dialogue design
2.OpenGL Programming Guide (Addison-Wesley Publishing Company) helped me to get through all OpenGL functions and Commands and understandings of all aspects of them.
3.www.cplusplus.com: - provided references regarding all C++ functions and their uses.
4.www.stackoverflow.com: - help to get rid of all types of error occurred regarding uses of OpenGL functions.
5.www.lighthouse3d.com: - OpenGL tutorial for implementing the OpenGL functions in Source code.

APPENDIX A - SOURCE CODE

main.cpp:

```
#include <gl/glew.h>
#include<gl/glut.h>
#include <iostream>
#include <vector>
#include "LabDemo.h"
//#include "graph.h"
//#include "PiChart.h"
//#include "SurfacePlot.h"
//#include "Surface3dPlot.h"

/*
PiChart p1;
SurfacePlot s1;
Surface3dPlot s2;
void display()
{
	glClear(GL_COLOR_BUFFER_BIT);
	gluLookAt(0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0);
	glPushMatrix();
		s2.draw();
	glPopMatrix();
	glFlush();
}

float calc(float x)
{
	return  sin(x * 10.0) / (1.0 + x * x);
}

void minit()
{
	std::vector<std::string> x = { "CRIC", "FOOT", "TT" , "BADMI" };
	std::vector<float> y = { 3.0, 1.0, 1.0, 1.0 };
	p1.appendXData(x);
	p1.appendyData(y);
	p1.setColor({ {1.0f,1.0f,0.0f}, {0.0f,1.0f,1.0f}, {1.0f,0.0f,0.0f},
{1.0f,0.0f,1.0f} });
	s1.SetFunction(calc);
	s1.fillBuffer();
	s2.fillBuffer();
	/*
	b1.appendXData(x);
	std::vector<int> y1 = { 3,1,6,9 };
	b1.appendYData(y,1);
	b1.setLineColor({ 0.0f, 1.0f, 1.0f },1);
	b1.appendYData(y1, 2);
	b1.setLineColor({ 1.0f, 0.0f, 1.0f }, 2);
	glClearColor(1, 1, 1, 1);
	glColor4f(1.0, 0.0, 0.0, 1.0f);
	glLineWidth(2.5);

	glMatrixMode(GL_PROJECTION);
	glLoadIdentity();
	gluPerspective(45.0, 1.0f, 0.1f, 10.0f);
	//gluOrtho2D(-2, 20, -2, 20);
	glMatrixMode(GL_MODELVIEW);
}
void mouseAction(int button, int state, int x, int y)
{
	if (button == GLUT_LEFT_BUTTON && state == GLUT_DOWN)
	{
		std::vector<float> y = { 12.0, 8.0, 9.0, 1.0 };
		p1.updateYData(y);
	}
	if (button == GLUT_RIGHT_BUTTON && state == GLUT_DOWN)
	{

	}
}
void special(int key, int x, int y) {
	s1.SpecialFunc(key, x, y);
	glutPostRedisplay();
}
*/
int main(int argc, char *argv[])
{
	glutInit(&argc, argv);
	glutInitDisplayMode(GLUT_RGBA);
	glutInitWindowSize(600, 600);
	glutInitWindowPosition(5, 5);

	//glutCreateWindow("CSE");
	DemoFinalInit();

	//setLineGraph(2);
	//glutMouseFunc(lineMouseAction);
	//glutMouseFunc(mouseButton);
	//glutMotionFunc(mouseMove);
	//glutDisplayFunc(barGraph3dDraw);
	//glutSpecialFunc(special);
	//barGraph3dInit();
	surfacePlotFinalInit();

	Bar3dGraphInit();
	LineGraphInit();
	piChartFinalInit();
	bar2dFinalInit();

	//surfacePlotFinalInit();
	glutMainLoop();
	return 1;
}
```

LabDemo.cpp:

```
#include <gl/glew.h>

#include<gl/glut.h>

#include <iostream>

#include <vector>

#include <string>

#include "graph.h"

#include "PiChart.h"

#include "SurfacePlot.h"

#include "LabDemo.h"

#include "utility.h"


int window1;

int window2;

int window3;

int window4;


void reShape(int width, int height)

{

	float ar;

	if (height == 0)

		height = width;

	ar = (float)width / height;


	glMatrixMode(GL_PROJECTION);

	glLoadIdentity();
```

```
      if (width <= height)

      {

            gluOrtho2D(-2, 20, -2 * ((float)height / (float)width), 20 *
((float)height / (float)width));

      }

      else

      {

            gluOrtho2D(-2, 20, -2 * ((float)width / (float)height), 20 *
((float)width / (float)height));

      }

      glMatrixMode(GL_MODELVIEW);

}


void reShape2(int width, int height)

{

      //if (width != height)

      //{

            //glViewport(0, 0, width, width);

      //}

      glViewport(0, 0, width, height);


}


LineGraph *l1;

std::vector<int> ypoints = {1, 1, 1, 4};

int lineNumber = 1;

void setLineGraph(int _nol)

{

      l1 = new LineGraph(_nol);

}

void setLineNumberMenu(int id)
```

```
{
      lineNumber = id;
}



void colorMenu(int id) {
      switch (id) {
      case 0:
            break;
      case 1:
            l1->setLineColor({ 0.0,0.0, 1.0 }, lineNumber);
            break;
      case 2:
            l1->setLineColor({ 0.0,1.0, 0.0 }, lineNumber);
            break;
      case 4:
            l1->setLineColor({ 1.0,0.0, 0.0 }, lineNumber);
            break;
      case 3:
            l1->setLineColor({ 0.0,1.0, 1.0 }, lineNumber);
            break;
      case 5:
            l1->setLineColor({ 1.0,0.0, 1.0 }, lineNumber);
            break;
      case 6:
            l1->setLineColor({ 1.0,1.0, 0.0 }, lineNumber);
            break;
      case 7:
            l1->setLineColor({ 1.0,1.0, 1.0 }, lineNumber);
            break;
```

```
	default:
		break;
	}
	lineGraphDraw();
}
void main_menu(int id) {



}
void setYdivision(int id)
{
	l1->setYDivision(id);
}
void lineGraphInit()
{
	std::vector<int> x = { 1 , 2, 3, 4 };
	std::vector<int> y = ypoints;
	std::vector<int> y1 = { 4, 1, 2, 3 };
	int setLineId = 0;
	int setYd = 0;
	if (l1)
	{
		l1->appendXData(x);
		l1->appendYData(y, 1);
		l1->appendYData(y1, 2);

		l1->setLineColor({ 1.0f, 0.0f, 0.0f }, 1);
		l1->setLineColor({ 1.0f, 0.0f, 1.0f }, 2);
		setLineId = glutCreateMenu(setLineNumberMenu);
		for (int i = 1; i <= l1->getLineNumber(); i++)
		{
			std::string str = "Line " + std::to_string(i);

			glutAddMenuEntry(str.c_str() , i);
		}
		glutAttachMenu(GLUT_LEFT_BUTTON);

		setYd = glutCreateMenu(setYdivision);
		for (int i = 3; i <= 6; i++)
		{
			std::string str = std::to_string(i);

			glutAddMenuEntry(str.c_str(), i);
		}
		glutAttachMenu(GLUT_LEFT_BUTTON);
	}

	int colorId = glutCreateMenu(colorMenu);
	glutAddMenuEntry("Red", 4);
	glutAddMenuEntry("Green", 2);
	glutAddMenuEntry("Blue", 1);
	glutAddMenuEntry("Black", 0);
	glutAddMenuEntry("Yellow", 6);
	glutAddMenuEntry("Cyan", 3);
	glutAddMenuEntry("Magenta", 5);
	glutAddMenuEntry("white", 7);
	glutAttachMenu(GLUT_LEFT_BUTTON);
	glutCreateMenu(main_menu);
```

```
      glutAddSubMenu("colors", colorId);
      glutAddSubMenu("Select Line", setLineId);
      glutAddSubMenu("Set Y Div", setYd);
      glutAttachMenu(GLUT_LEFT_BUTTON);

      glClearColor(1, 1, 1, 1);
      glColor4f(1.0, 0.0, 0.0, 1.0f);
      glLineWidth(2.5);

      glMatrixMode(GL_PROJECTION);
      glLoadIdentity();
      gluOrtho2D(-2, 20, -2, 20);
      glMatrixMode(GL_MODELVIEW);
}

void lineGraphDraw()
{
            glClear(GL_COLOR_BUFFER_BIT);
            if(l1)
                  l1->draw();
            glFlush();
}

void lineMouseAction(int button, int state, int x, int y)
{
      if (button == GLUT_LEFT_BUTTON && state == GLUT_DOWN)
      {

      }
      if (button == GLUT_RIGHT_BUTTON && state == GLUT_DOWN)
```

```
	{
		std::vector<int> y = { 12, 8, 9, 1 };
		l1->updateData(y, 1);
	}
}

void LineGraphInit()
{
	glutInitWindowPosition(400, 5);
	glutInitWindowSize(400, 400);
	window1 = glutCreateWindow("Line Graph");
	setLineGraph(2);
	lineGraphInit();
	glutMouseFunc(lineMouseAction);
	glutReshapeFunc(reShape);
	glutDisplayFunc(lineGraphDraw);
}



//PiChart
PiChart p1;
int piNumber = 0;
void setPiNumberMenu(int id)
{
	piNumber = id;
}
void picolorMenu(int id) {
	switch (id) {
	case 0:
```

```
            break;
      case 1:
            p1.updateColor({ 0.0,0.0, 1.0 }, piNumber);
            break;
      case 2:
            p1.updateColor({ 0.0,1.0, 0.0 }, piNumber);


            break;
      case 4:
            p1.updateColor({ 1.0,0.0, 0.0 }, piNumber);


            break;
      case 3:
            p1.updateColor({ 0.0,1.0, 1.0 }, piNumber);


            break;
      case 5:
            p1.updateColor({ 1.0,0.0, 1.0 }, piNumber);


            break;
      case 6:
            p1.updateColor({ 1.0,1.0, 0.0 }, piNumber);


            break;
      case 7:
            p1.updateColor({ 1.0,1.0, 1.0 }, piNumber);
            break;
      default:
            break;
      }
      piChartDraw();

}

void piChartInit()

{


      glClearColor(1, 1, 1, 1);

      glColor4f(1.0, 0.0, 0.0, 1.0f);

      glLineWidth(2.5);




      glMatrixMode(GL_PROJECTION);

      glLoadIdentity();

      gluOrtho2D(-2, 20, -2, 20);

      glMatrixMode(GL_MODELVIEW);


      p1.appendXData({ "FIrst", "Second", "Third" , "Fourth" });

      std::vector<float> yvalues = { (float)ypoints[0],
(float)ypoints[1],(float)ypoints[2],(float)ypoints[3] };

      p1.appendyData(yvalues);

      p1.setColor({ {1.0f, 0.0f, 0.0f}, {0.0f, 1.0f, 0.0f}, {1.0f, 1.0f,
0.0f}, {0.0f, 0.0f, 1.0f} });


      int piId = glutCreateMenu(setPiNumberMenu);

      for (int i = 1; i <= p1.xData.size(); i++)

      {

            std::string str = p1.xData[i - 1];

            glutAddMenuEntry(str.c_str(), i-1);

      }

      glutAttachMenu(GLUT_LEFT_BUTTON);


      int colorId = glutCreateMenu(picolorMenu);
```

```
	glutAddMenuEntry("Red", 4);
	glutAddMenuEntry("Green", 2);
	glutAddMenuEntry("Blue", 1);
	glutAddMenuEntry("Black", 0);
	glutAddMenuEntry("Yellow", 6);
	glutAddMenuEntry("Cyan", 3);
	glutAddMenuEntry("Magenta", 5);
	glutAddMenuEntry("white", 7);
	glutAttachMenu(GLUT_LEFT_BUTTON);

	glutCreateMenu(main_menu);
	glutAddSubMenu("colors", colorId);

	glutAddSubMenu("Select Line", piId);
	glutAttachMenu(GLUT_LEFT_BUTTON);

}
void piChartDraw()
{
	glClear(GL_COLOR_BUFFER_BIT);
	p1.draw();
	glFlush();
}
void piChartFinalInit()
{
	glutInitWindowPosition(800, 5);

	glutInitWindowSize(400, 400);
	window3 = glutCreateWindow("Pi Chart");
	piChartInit();
	//glutMouseFunc(lineMouseAction);
	glutReshapeFunc(reShape);
	glutDisplayFunc(piChartDraw);
}



//Bar2gGraph
Bar2DGraph b1;
void bar2dcolormenu(int id) {
	switch (id) {
	case 0:
		break;
	case 1:
		b1.setBarColor({ 0.0,0.0, 1.0 });
		break;
	case 2:
		b1.setBarColor({ 0.0,1.0, 0.0 });

		break;
	case 4:
		b1.setBarColor({ 1.0,0.0, 0.0 });

		break;
	case 3:
		b1.setBarColor({ 0.0,1.0, 1.0 });

		break;
	case 5:
		b1.setBarColor({ 1.0,0.0, 1.0 });
```

```
            break;
      case 6:
            b1.setBarColor({ 1.0,1.0, 0.0 });


            break;
      case 7:
            b1.setBarColor({ 1.0,1.0, 1.0 });
            break;
      default:
            break;
      }
      bar2dChartDraw();
}
void setWidthDiv(int id)
{
      b1.setBarWidth(id);
      bar2dChartDraw();
}
void bar2dMenus()
{
      int colorId = glutCreateMenu(bar2dcolormenu);
      glutAddMenuEntry("Red", 4);
      glutAddMenuEntry("Green", 2);
      glutAddMenuEntry("Blue", 1);
      glutAddMenuEntry("Black", 0);
      glutAddMenuEntry("Yellow", 6);
      glutAddMenuEntry("Cyan", 3);
      glutAddMenuEntry("Magenta", 5);
      glutAddMenuEntry("white", 7);
```

```
	glutAttachMenu(GLUT_LEFT_BUTTON);

	int setWidth = glutCreateMenu(setWidthDiv);
	for (int i = 1; i <= 3; i++)
	{
		std::string str = std::to_string(i);
		glutAddMenuEntry(str.c_str(), i);
	}
	glutAttachMenu(GLUT_LEFT_BUTTON);

	glutCreateMenu(main_menu);
	glutAddSubMenu("colors", colorId);
	glutAddSubMenu("Width", setWidth);
	glutAttachMenu(GLUT_LEFT_BUTTON);
}
void bar2dChartInit()
{
	glClearColor(1, 1, 1, 1);
	glColor4f(1.0, 0.0, 0.0, 1.0f);
	glLineWidth(2.5);

	glMatrixMode(GL_PROJECTION);
	glLoadIdentity();
	gluOrtho2D(-2, 20, -2, 20);
	glMatrixMode(GL_MODELVIEW);

	b1.appendXData({ 1,2,3,4 });
	b1.appendYData(ypoints);
	b1.setBarColor({ 1.0f, 0.0f, 1.0f });
	bar2dMenus();
```

```
}
void bar2dChartDraw()
{
	glClear(GL_COLOR_BUFFER_BIT);
	b1.draw();
	glFlush();
}
void bar2dFinalInit()
{
	glutInitWindowPosition(400, 405);

	glutInitWindowSize(400, 400);
	window4 = glutCreateWindow("Bar Chart 2d");
	bar2dChartInit();
	//glutMouseFunc(lineMouseAction);
	glutReshapeFunc(reShape2);
	glutDisplayFunc(bar2dChartDraw);
}

//BarGraph
Bar3dGraph b3;
void barGraph3dInit()
{
	b3.appendXData({ 1,2 });
	b3.appendZData({ 1,2 });
	b3.appendYData(ypoints);
	glClearColor(1, 1, 1, 1);
	glColor4f(1.0, 0.0, 0.0, 1.0f);
	glLineWidth(2.5);
	glEnable(GL_DEPTH_TEST);
```

```
      glMatrixMode(GL_PROJECTION);
      glLoadIdentity();
      gluPerspective(45.0, 1.0f, 0.1f, 100.0f);
      //gluPerspective(45.0, 1.0f, 0.1f, 10.0f);
      //glOrtho(-2.0, 20.0, -2.0, 20.0, -2.0, 20.0);
      glMatrixMode(GL_MODELVIEW);
}
// the key states. These variables will be zero
//when no key is being presses
float deltaAngle = 0.0f;
float deltaMove = 0;
int xOrigin = -1;
// angle of rotation for the camera direction
float angle = 0.0f;

// actual vector representing the camera's direction
float lx = 0.0f, lz = -1.0f;

// XZ position of the camera
float x = 0.0f, z = 40.0f;

void computePos(float deltaMove) {

      x += deltaMove * lx * 0.1f;
      z += deltaMove * lz * 0.1f;
}

void barGraph3dDraw()
{
      if (deltaMove)
```

```
            computePos(deltaMove);
      glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
      glPushMatrix();
      // Set the camera
      gluLookAt(x, 20.0f, z,
                  x + lx, 19.9f, z + lz,
            0.0f, 1.0f, 0.0f);
      b3.draw();
      glPopMatrix();
      glFlush();
}

void barMouseMove(int x, int y) {
      // this will only be true when the left button is down
      if (xOrigin >= 0) {

            // update deltaAngle
            deltaAngle = (x - xOrigin) * 0.001f;

            // update camera's direction
            lx = sin(angle + deltaAngle);
            lz = -cos(angle + deltaAngle);
      }
}

void barMouseButton(int button, int state, int x, int y) {

      // only start motion if the left button is pressed
      if (button == GLUT_LEFT_BUTTON) {
            // when the button is released
            if (state == GLUT_UP) {
                  angle += deltaAngle;
                  xOrigin = -1;
            }
            else {// state = GLUT_DOWN
                  xOrigin = x;
            }
      }
}

void pressKey(int key, int xx, int yy) {
      switch (key) {
      case GLUT_KEY_LEFT:
            x = x + lz * 0.1f;
            z = z - lx * 0.1f;
            break;
      case GLUT_KEY_RIGHT:
            x = x - lz * 0.1f;
            z = z + lx * 0.1f;
            break;
      case GLUT_KEY_UP: deltaMove = 0.5f; break;
      case GLUT_KEY_DOWN: deltaMove = -0.5f; break;
      }
      barGraph3dDraw();
}
void releaseKey(int key, int x, int y) {
      switch (key) {
      case GLUT_KEY_UP:
      case GLUT_KEY_DOWN: deltaMove = 0;break;
      }
      barGraph3dDraw();
}
void Bar3dGraphInit()
{
      glutInitWindowPosition(900, 405);

      glutInitWindowSize(400, 400);
      window2 = glutCreateWindow("Bar 3d Graph");

      glutSpecialFunc(pressKey);
      glutSpecialUpFunc(releaseKey);
      glutMouseFunc(barMouseButton);
      glutMotionFunc(barMouseMove);
      glutDisplayFunc(barGraph3dDraw);
      barGraph3dInit();
}
//
SurfacePlot s1;
float calc(float x)
{
      return  3*sin(x * 10.0) / (1.0 + x * x);
}

void surfacePlotInit()
{
      glClearColor(1, 1, 1, 1);
      glColor4f(1.0, 0.0, 0.0, 1.0f);
      glLineWidth(2.5);
```

```
	glMatrixMode(GL_PROJECTION);

	glLoadIdentity();

	gluOrtho2D(0, 20, 0, 20);

	glMatrixMode(GL_MODELVIEW);


	s1.SetFunction(calc);

	s1.fillBuffer();

}

void surfacePlotDraw()

{

	glClear(GL_COLOR_BUFFER_BIT);

	s1.draw();

	glFlush();

}

void special(int key, int x, int y) {

	s1.SpecialFunc(key, x, y);

	glutPostRedisplay();

}

void surfacePlotFinalInit()

{

	glutInitWindowPosition(5, 405);

	glutInitWindowSize(400, 400);

	window2 = glutCreateWindow("Surface2d Graph");

	GLenum glew_status = glewInit();


	if (GLEW_OK != glew_status) {

		fprintf(stderr, "Error: %s\n", 
glewGetErrorString(glew_status));

		return;

	}
	surfacePlotInit();

	glutSpecialFunc(special);

	glutDisplayFunc(surfacePlotDraw);


}





//Add Draw Functions here

void UpdateDraw()

{

	glutSetWindow(window1);

	glutPostRedisplay();


	glutSetWindow(window2);

	glutPostRedisplay();


	glutSetWindow(window3);

	std::vector<float> yvalues = { (float)ypoints[0],
(float)ypoints[1],(float)ypoints[2],(float)ypoints[3] };

	p1.updateYData(yvalues);

	glutPostRedisplay();


	glutSetWindow(window4);

	glutPostRedisplay();

}


void DemoMouseAction(int button, int state, int x, int y)

{

	x = x - 200;

	y = 200 - y;
```

```
std::cout << x << " " << y << "\n";

if (button == GLUT_LEFT_BUTTON && state == GLUT_DOWN)
{
      if (x > 0)
      {
            if (y > 0)
            {
                  ypoints[0]++;
                  l1->updateData(ypoints, 1);
                  b3.updateYData(ypoints);
                  b1.updateData(ypoints);

            }
            else
            {
                  ypoints[3]++;
                  l1->updateData(ypoints, 1);
                  b3.updateYData(ypoints);
                  b1.updateData(ypoints);

            }
      }
      else
      {
            if (y > 0)
            {
                  ypoints[1]++;
                  l1->updateData(ypoints, 1);
                  b3.updateYData(ypoints);
```

```
                        b1.updateData(ypoints);

                  }
                  else
                  {
                        ypoints[2]++;
                        l1->updateData(ypoints, 1);
                        b3.updateYData(ypoints);
                        b1.updateData(ypoints);

                  }
            }
      }
      if (button == GLUT_RIGHT_BUTTON && state == GLUT_DOWN)
      {
      }
      UpdateDraw();
}

void DemoInit()
{
      glClearColor(1, 1, 1, 1);
      glColor4f(1.0, 0.0, 0.0, 1.0f);
      glLineWidth(2.5);

      glMatrixMode(GL_PROJECTION);
      glLoadIdentity();
      gluOrtho2D(-5, 5, -5, 5);
      glMatrixMode(GL_MODELVIEW);
}
```

```
void DemoDraw()
{
      glClear(GL_COLOR_BUFFER_BIT);
      glColor3f(0.0f, 0.0f, 0.0f);
      drawLine(0, -5, 0, 5);
      drawLine(-5, 0, 5, 0);
      glFlush();
}
void DemoFinalInit()
{
      glutInitWindowSize(500, 500);
      int windowId = glutCreateWindow("Main Screen");
      glutMouseFunc(DemoMouseAction);
      glutDisplayFunc(DemoDraw);
      DemoInit();
}
```