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 i3 processor
Processor Speed	:	1.70 GHz
RAM	:	4 GB
Storage Space	:	40 GB
Monitor Resolution	:	1024*768 or 1336*768 or 1280*1024

3.2 SOFTWARE REQUIREMENTS

Operating System	:	Windows 8.1
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(GLdouble left, 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.

- **glLoadName(ID) :**

To replace the top of the integer-ID name stack for picking operations

- **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 integer 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 matrix 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

IMPLEMENTATION

```
#include <iostream>
#include <GL/glut.h>
#include "Source.h"
float f[9][2] = {
{300,320},{310,300},{320,290},{330,300},{335,310},{340,320},{330,340},{320,350},{31
0,340} };
float t[4][2] = { {340,320},{350,290},{345,320},{350,350} };
float m[2][2] = { {300,320} ,{310,320} };
float e = 308;
float f2[9][2] = {
{200,260},{210,240},{220,230},{230,240},{235,250},{240,260},{230,280},{220,290},{21
0,280} };
float t2[4][2] = { {240,260},{250,230},{245,260},{250,290} };
float m2[2][2] = { {200,260} ,{210,260} };
float e2 = 208;
int l = 1,o=0;
int fflag = 0;
float cc[3] = {1,1,1};
float p[5][2] = { {380,320},{380,330},{380,340},{380,350}, {380,360} };//bubble
int r = 5, r2 = 5, d = 2, c = 0;

//to reshape the window size
void reshape(int w, int h)
{
        glViewport(-100, -100, w, h);
        glMatrixMode(GL_PROJECTION);
        glLoadIdentity();
        gluOrtho2D(0, 500, 0, 500);
}
```

```
//performs the background tasks when window system events are not being received
void idle() {
        if (r < -100)
                r = 50;
        if (r2 < 2)
                r2 = 150;
        if (c > 4)
                c = 0;
        if (d > 4)
                d = 0;
        glutPostRedisplay();
}
//draws the tank
void tank()
{
        glColor3f(0, 0, 1);
        glBegin(GL_POLYGON);
        {
                glVertex2f(200, 450);
                glVertex2f(150, 400);

                glVertex2f(450, 400);
                glVertex2f(400, 450);
        }
        glEnd();

        glColor3f(0, 1, 1);
        glBegin(GL_POLYGON);
        {
                glVertex2f(200, 370);
                glVertex2f(200, 200);
                glVertex2f(400, 200);
                glVertex2f(400, 370);
        }
```

```
glEnd();

//tank
glColor3f(0, 0, 1);
glLineWidth(3);
glBegin(GL_LINE_LOOP);
{
        glVertex2f(200, 400);
        glVertex2f(200, 200);
        glVertex2f(400, 200);
        glVertex2f(400, 400);
}
glEnd();

glColor3f(0, 0, 0.8);
glBegin(GL_POLYGON);
{
        glVertex2f(350, 480);
        glVertex2f(350, 450);
        glVertex2f(400, 450);
        glVertex2f(400, 480);
}
glEnd();

//water filter
glColor3f(0.812, 0.847, 0.862);
glBegin(GL_POLYGON);
{
        glVertex2f(400, 335);
        glVertex2f(398, 335);
        glVertex2f(398, 400);
        glVertex2f(400, 400);
}
glEnd();
```

```
glColor3f(0.258, 0.258, 0.258);
glBegin(GL_POLYGON);
{
        glVertex2f(400, 335);
        glVertex2f(390, 335);
        glVertex2f(390, 320);
        glVertex2f(400, 320);
}
glEnd();

glColor3f(0.258, 0.258, 0.258);
glBegin(GL_TRIANGLES);
{
        glVertex2f(400, 320);
        glVertex2f(390, 320);
        glVertex2f(400, 312);
}
glEnd();

glColor3f(0.258, 0.258, 0.258);
glBegin(GL_POLYGON);
{
        glVertex2f(400, 315);
        glVertex2f(385, 315);
        glVertex2f(385, 310);
        glVertex2f(400, 310);
}
glEnd();

glColor3f(0.258, 0.258, 0.258);
glBegin(GL_TRIANGLES);
{
        glVertex2f(400, 314);
```

```
                glVertex2f(400, 300);
                glVertex2f(385, 300);
        }
        glEnd();

        glColor3f(0.258, 0.258, 0.258);
        glBegin(GL_POLYGON);
        {
                glVertex2f(400, 300);
                glVertex2f(385, 300);
                glVertex2f(385, 280);
                glVertex2f(400, 280);
        }
        glEnd();
        glShadeModel(GL_SMOOTH);
        glColor3f(1, 1, 1);
        glBegin(GL_LINES);
        glVertex2f(385, 300);
        glVertex2f(400, 300);
        glVertex2f(385, 295);
        glVertex2f(400, 295);
        glVertex2f(385, 290);
        glVertex2f(400, 290);
        glEnd();
}

float h = 40, k = 5;
//draws the pebbles in the tank
void eli()
{
        glColor3f(0.4, 0.8, .80);
        glBegin(GL_POLYGON);
        for (float i = 0; i < 180; i++)
                glVertex2f(320 + 12 * cos(i), 200 + 5 + 6 * sin(i));
        glEnd();
        glColor3f(1, 1, 1);
        glBegin(GL_POLYGON);
        for (float j = 0; j < 180; j++)
                glVertex2f(300 + 12 * cos(j), 200 + 5 + 6 * sin(j));
        glEnd();
        glColor3f(0.4, 0.8, .80);
        glBegin(GL_POLYGON);
        for (float l = 0; l < 180; l++)
                glVertex2f(280 + 12 * cos(l), 200 + 5 + 6 * sin(l));
        glEnd();
        glColor3f(1, 1, 1);
        glBegin(GL_POLYGON);
        for (float p = 0; p < 180; p++)
                glVertex2f(260 + 12 * cos(p), 200 + 5 + 6 * sin(p));
        glEnd();

        glColor3f(0.27, 0.35, 0.39);
        glBegin(GL_POLYGON);
        for (float r = 0; r < 180; r++)
                glVertex2f(385 + 12 * cos(r), 200 + 5 + 6 * sin(r));
        glEnd();
}
float i;
//draws the fish
void fish1()
{
        glColor3f(1, 0.8, 0);
        glBegin(GL_POLYGON);
        glVertex2f(f[0][0] + r, f[0][1]);
        glVertex2f(f[1][0] + r, f[1][1]);
        glVertex2f(f[2][0] + r, f[2][1]);
        glVertex2f(f[3][0] + r, f[3][1]);
        glVertex2f(f[4][0] + r, f[4][1]);
```

```
	glVertex2f(f[5][0] + r, f[5][1]);
	glVertex2f(f[6][0] + r, f[6][1]);
	glVertex2f(f[7][0] + r, f[7][1]);
	glVertex2f(f[8][0] + r, f[8][1]);
	glEnd();

	glColor3f(1, 1, 0);
	glBegin(GL_POLYGON);
	glVertex2f(t[0][0] + r, t[0][1]);
	glVertex2f(t[1][0] + r, t[1][1]);
	glVertex2f(t[2][0] + r, t[2][1]);
	glVertex2f(t[3][0] + r, t[3][1]);
	glEnd();

	glColor3f(1, 1, 1);
	glBegin(GL_LINES);
	glVertex2f(m[0][0] + r, m[0][1]);
	glVertex2f(m[1][0] + r, m[1][1]);
	glEnd();
	glColor3f(1, 1, 0);
	glBegin(GL_POLYGON);
	for (i = 0; i < 2 * 3.14; i += 0.5)
		glVertex2f(e + r + 2 * cos(i), 328 + 2 * sin(i));
	glEnd();
	Sleep(1);
	r -= 8;
}
//draws the second fish
void fish2()
{
	glColor3f(1,0.8,0);
	glBegin(GL_POLYGON);
	glVertex2f(f2[0][0] + r2, f2[0][1]);
	glVertex2f(f2[1][0] + r2, f2[1][1]);
```

```
        glVertex2f(f2[2][0] + r2, f2[2][1]);
        glVertex2f(f2[3][0] + r2, f2[3][1]);
        glVertex2f(f2[4][0] + r2, f2[4][1]);
        glVertex2f(f2[5][0] + r2, f2[5][1]);
        glVertex2f(f2[6][0] + r2, f2[6][1]);
        glVertex2f(f2[7][0] + r2, f2[7][1]);
        glVertex2f(f2[8][0] + r2, f2[8][1]);
        glEnd();

        glColor3f(1,1,0);
        glBegin(GL_POLYGON);
        glVertex2f(t2[0][0] + r2, t2[0][1]);
        glVertex2f(t2[1][0] + r2, t2[1][1]);
        glVertex2f(t2[2][0] + r2, t2[2][1]);
        glVertex2f(t2[3][0] + r2, t2[3][1]);
        glEnd();

        glColor3f(1,1,1);
        glBegin(GL_LINES);
        glVertex2f(m2[0][0] + r2, m2[0][1]);
        glVertex2f(m2[1][0] + r2, m2[1][1]);
        glEnd();
        glColor3f(1,1,0);
        glBegin(GL_POLYGON);
        for (i = 0; i < 2 * 3.14; i += 0.5)
                glVertex2f(e2 + r2 + 2 * cos(i), 268 + 2 * sin(i));
        glEnd();
        Sleep(1);
        r2 -= 8;
}
//draws the bubbles in the water
void bubble()
{
        glColor3fv(cc);
        glBegin(GL_POLYGON);
        for (i = 0; i < 2 * 3.14; i += 0.5)
                glVertex2f(p[c][0]+o + 2 * cos(i), p[c][1] + 2 * sin(i));
        glEnd();
        glBegin(GL_POLYGON);
        for (i = 0; i < 2 * 3.14; i += 0.5)
                glVertex2f(p[d][0]+o + 2 * cos(i), p[d][1] + 2 * sin(i));
        glEnd();
        Sleep(300);
        d += 1;
        c += 1;
}
//draws the plant
void plant()
{
        glColor3f(0, 0.8, 0);
        glBegin(GL_POLYGON);
        glVertex2f(340 + 10, 200);
        glVertex2f(330 + 10, 220);
        glVertex2f(350 + 10, 210);
        glVertex2f(350 + 10, 250);
        glVertex2f(360 + 10, 210);
        glVertex2f(370 + 10, 220);
        glVertex2f(380, 240);
        glVertex2f(380, 200);
        glEnd();

        glColor3f(0, 0.8, 0);
        glBegin(GL_POLYGON);
        glVertex2f(340 - 130, 200);
        glVertex2f(330 - 130, 220);
        glVertex2f(350 - 130, 210);
        glVertex2f(350 - 130, 250);
        glVertex2f(360 - 130, 210);
        glVertex2f(370 - 130, 220);
        glVertex2f(380 - 130, 240);
        glVertex2f(380 - 130, 200);
        glEnd();
}
//draws the waves in the tank
void waves() {

        glPointSize(3);
        glColor3f(0, 1, 1);
        glBegin(GL_LINE_STRIP);
        for (int i = 201; i < 399; i += 1)
        {

                float x = (float)i;
                float y = 5 * sin(x*0.1);
                glVertex2f(x, y + 373);
        }
        glEnd(); glBegin(GL_LINE_STRIP);
        for (int i = 201; i < 399; i += 1)
        {

                float x = (float)i;
                float y = 5 * sin(x*0.1);
                glVertex2f(x, y + 372);
        }
        glEnd();
        glBegin(GL_LINE_STRIP);
        for (int i = 201; i < 399; i += 1)
        {

                float x = (float)i;
                float y = 5 * sin(x*0.1);
                glVertex2f(x, y + 371);
```

```
}
glEnd();
glBegin(GL_LINE_STRIP);
for (int i = 201; i < 399; i += 1)
{

        float x = (float)i;
        float y = 5 * sin(x*0.1);
        glVertex2f(x, y + 370);
}
glEnd();
glBegin(GL_LINE_STRIP);
for (int i = 201; i < 399; i += 1)
{

        float x = (float)i;
        float y = 5 * sin(x*0.1);
        glVertex2f(x, y + 369);
}
glEnd();
glBegin(GL_LINE_STRIP);
for (int i = 201; i < 399; i += 1)
{

        float x = (float)i;
        float y = 5 * sin(x*0.1);
        glVertex2f(x, y + 368);
}
glEnd();
glBegin(GL_LINE_STRIP);
for (int i = 201; i < 399; i += 1)
{

        float x = (float)i;
```

```
                float y = 5 * sin(x*0.1);
                glVertex2f(x, y + 367);
        }
        glEnd();
        glBegin(GL_LINE_STRIP);
        for (int i = 201; i < 399; i += 1)
        {

                float x = (float)i;
                float y = 5 * sin(x*0.1);
                glVertex2f(x, y + 366);
        }
        glEnd(); glBegin(GL_LINE_STRIP);
        for (int i = 201; i < 399; i += 1)
        {

                float x = (float)i;
                float y = 5 * sin(x*0.1);
                glVertex2f(x, y + 365);
        }
        glEnd();
}
//used the display the content of buffer
void display()
{
        glClearColor(1, 1, 1, 0);
        glClear(GL_COLOR_BUFFER_BIT);
        tank();
        plant();
        eli();
        fish1();
        bubble();
        if(fflag==1)
                fish2();
        waves();
        glutSwapBuffers();
        glFlush();
}
//displays the content of menu when the mouse button is pressed
void menu(int ch)
{
        if (ch == 1)
        {
                o = 0;
                d = 2; c = 0;
        }
        if (ch == 2)
                o = 30;
        if (ch == 3)
                fflag = 1;
        if (ch == 4)
                fflag = 0;
        if (ch == 5)
                exit(0);
        glutPostRedisplay();
}
int main(int argc, char *argv[])
{
        glutInit(&argc, argv);
        glutInitDisplayMode(GLUT_SINGLE | GLUT_RGB);
        glutInitWindowSize(1000, 1000);
        glutCreateWindow("Aquarium");
        glutFullScreen();
        glutDisplayFunc(display);
        glutReshapeFunc(reshape);
        glutIdleFunc(idle);
        glutCreateMenu(menu);
        glutAddMenuEntry("On Pump", 1);
```

```
	glutAddMenuEntry("Off Pump", 2);
	glutAddMenuEntry("Add 2nd Fish", 3);
	glutAddMenuEntry("Remove 2nd Fish", 4);
	glutAddMenuEntry("Close", 5);
	glutAttachMenu(GLUT_RIGHT_BUTTON);
	glutMainLoop();
	return 0;
}
```

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	**Mouse Function**	Right button pressed to continue the motion of fish and Left button pressed to stop it at that moment.	Results obtained as expected
2	**Display Function**	Computers, send window, receive window, packets, timer are displayed.	Results obtained as expected
3	**Animation Effect**	Movement of packets and timer movement.	Results obtained as expected

CHAPTER7

RESULTS AND SNAPSHOTS

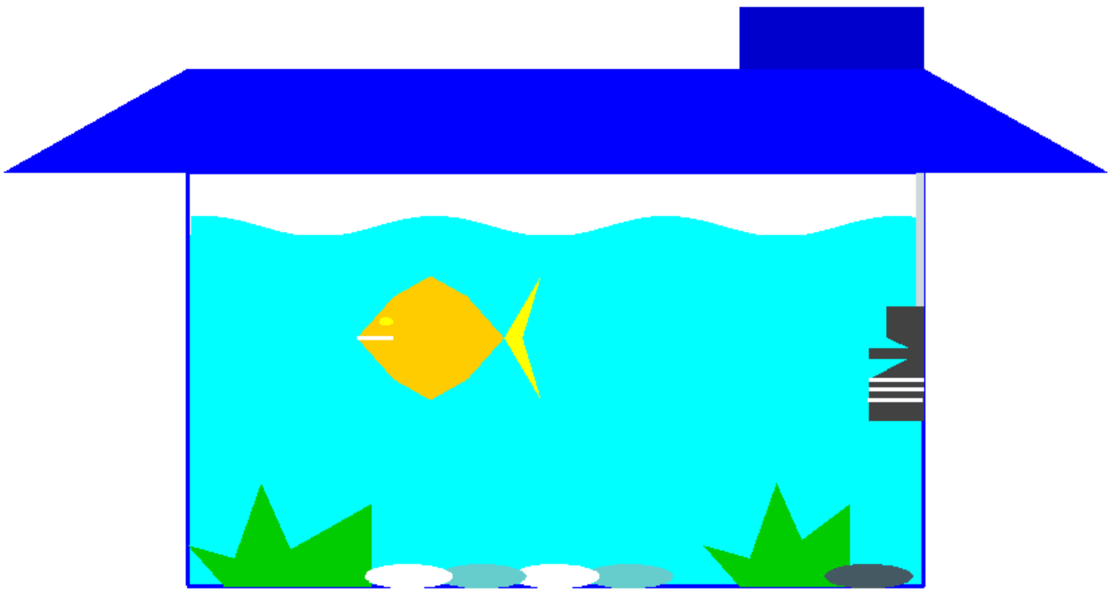

Fig 1

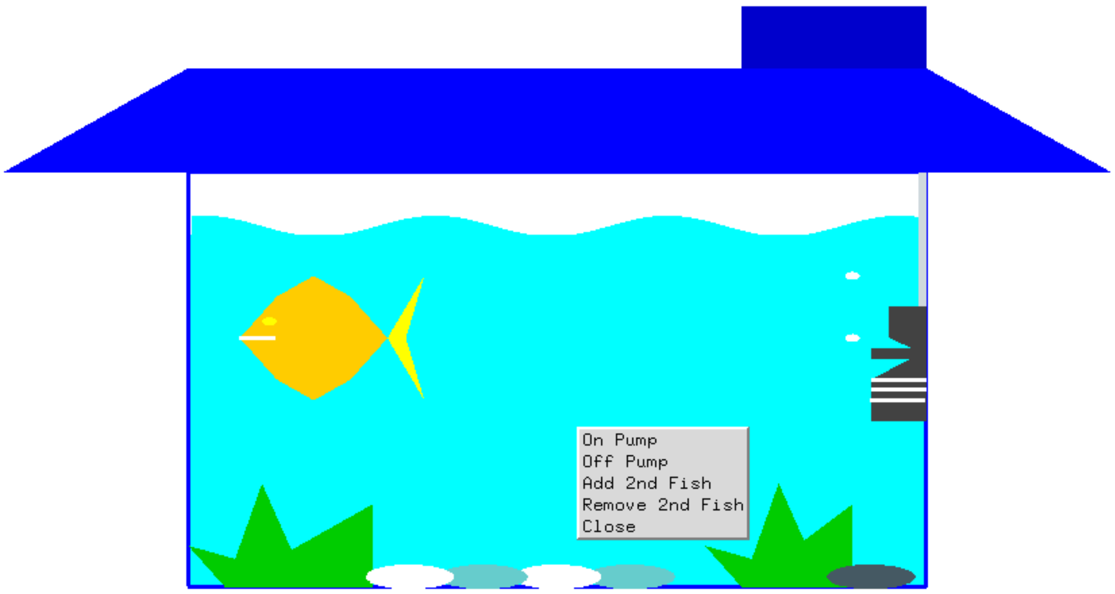

Fig 2

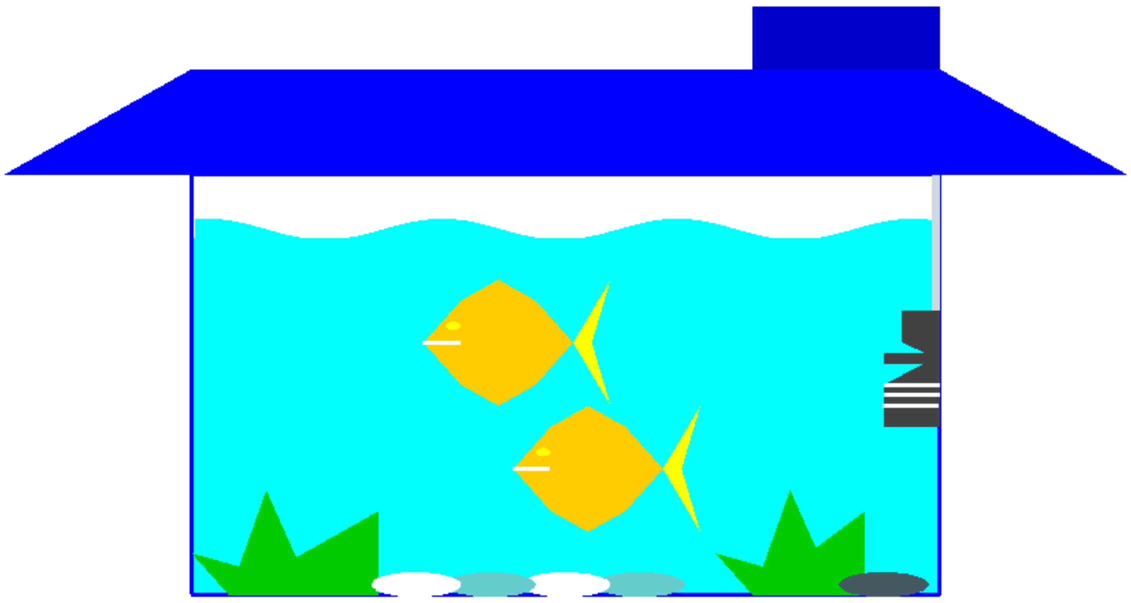

Fig 3

CHAPTER 8

CONCLUSION AND FUTURE ENHANCEMENTS

The Aquarium Simulation project gives the viewers a 2 dimensional representation of an aquarium. It provides a visual of how the fishes, plants, etc. are kept in aquariums. It also holds a water pump used in aquariums. The project used many of the OpenGL functions for its simulation.

Further in case of future enhancements, we will try to present the simulation of the model in a 3 dimensional way. We can improve the project by adding the code to provide food for the fishes. This can be done easily as the code is modularized and easy to understand. Changing the existing code or adding new modules can append improvements.

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,AndriesVanDam,StevenK.Feiner,JohnF.Hughes, ”ComputerGraphics”, Second Edition, Addison-Wesley Professional, August 14,1995

4. www.opengl.org.

5. https://www.khronos.org/opengl/

6. www.w3schools.com