CHAPTER 1

INTRODUCTION

Computer graphics are pictures and films created using computers. Usually, the term refers to computer generated image data created with help from specialized graphical hardware and software. It is a vast and recent area in computer science. Some topics in computer graphics include user interface design, sprite graphics, vector graphics, 3D modeling, shaders, GPU design, implicit surface visualization with ray tracing, and computer vision, among others. The overall methodology depends heavily on the underlying sciences of geometry, optics, and physics. Computer graphics is responsible for displaying art and image data effectively and meaningfully to the user. It is also used for processing image data received from the physical world. Computer graphic development has had a significant impact on many types of media and has revolutionized animation, movies, advertising, video games, and graphic design generally.

1.1 History of Computer Graphics

Computer Graphics is the creation, manipulation, and storage of models and images of picture objects by the aid of computers. This was started with the display of data on plotters and CRT. Computer Graphics is also defined as the study of techniques to improve the communication between user and machine, thus Computer Graphics is one of the most effective medium of communication between machine and user.

William Fetter was credited with coning the term Computer Graphics in 1960, to describe his work at Boeing. One of the first displays of computer animation was future world (1976), which included an animation of a human face and hand-produced by Carmull and Fred Parkle at the University of Utah. There are several international conferences and journals where the most significant results in computer-graphics are published. Among them are the SIGGRAPH and Euro Graphics conferences and the Association for Computing Machinery (ACM) transaction on Graphics journals.

1.2 Objective

Our goal with this project is to design a traditional snake game in 2 dimensions using the features of the GLUT toolkit within the OpenGL API implemented for the windows platform. For those that are not familiar with snake, it is a video game concept where the player maneuvers a line (i.e. the snake) which grows in length, with the line and the boundaries being the obstacles they have to avoid. The objectives that are covered in Snake2D are

- To design and implement a classic snake game, restricted to a boundary within the window that randomly generates ‘fruits’ within the said area.
- To add maneuverability with the direction keys that makes the snake go up, down, left or right.
- To make the snake grow with each fruit collected, and to end the game when it hits itself or the boundaries.

- As an added effort to make it GUI friendly, to design a menu screen that gives options to start game, view high score or quit. Additional screens indicating game over and displaying options to return to the main screen or quit are also added.

1.3 Summary

The chapter discussed before is an overview about the computer graphics, its history and OpenGL interface. It even includes the OpenGL block diagram. The scope of study and objectives of the project are mentioned clearly. The organization of the report is being pictured to increase the readability. Further, coming up chapter depicts the OpenGL built-in functions used in project source code.

1.4 General Constraints

- As software is being built to run on a WINDOWS platform efficient use of the memory is very important.
- The code should be efficient and optimal with the minimal redundancies.
- Needless to say, the package should also be robust and fast.

1.5 Assumptions and Dependencies

- It is assumed that the standard output device, namely the monitor, supports colors and users' system is required to have the C/C++ compiler for the appropriate version.
- The system is also expected to have a mouse connected since most of the drawing and other graphical operations implemented assume the presence of a mouse.

CHAPTER 2

REQUIREMENTS

The requirement specifications of this project are not perfectly optimized. However, the following hardware and software specifications were done to be of our best efforts. Here are the specifications:

2.1 Hardware Requirements:

The hardware requirements given here is minimal requirements for the project to run even though the project can smoothly run on almost all i3h86 machines.

- Processor Speed - 300 MHz and above
- Ram Size - 64 Mb or above
- Storage Space - 2MB or above

2.2 Software Requirements:

- Operating System - Windows XP and later
- Compiler - CODEBLOCKS/Eclipse
- Graphics Library - glut.h
- Programming Language - C using OpenGL

CHAPTER 3

ABOUT OPENGL

The OpenGL specification describes an abstract for drawing 2D and 3D graphics. Although it is possible for the API to be implemented entirely in software, it is designed to be implemented mostly or entirely in hardware.

The API is defined as a number of functions which may be called by the client program, alongside a number of named integer constants. Although the function definition is superficially similar to those of the C programming language, they are independent. As such, OpenGL has many language bindings, some of the most noteworthy being the JavaScript binding WebGL(API based on OpenGL ES2.0);the C bindings WGL, GLX, CGL; the C binding provided by iOS and the Java and C binding provided by the Android.

In addition to being language-independent, OpenGL is also platform independent. The specification says nothing on the subject of obtaining, managing an OpenGL context, leaving this as a detail of the underlying windowing system. For the same reason, OpenGL is purely concerned with the rendering, providing no APIs related to input, audio, or windowing.

3.1 OPENGL GRAPHICS PIPELINE ARCHITECTURE

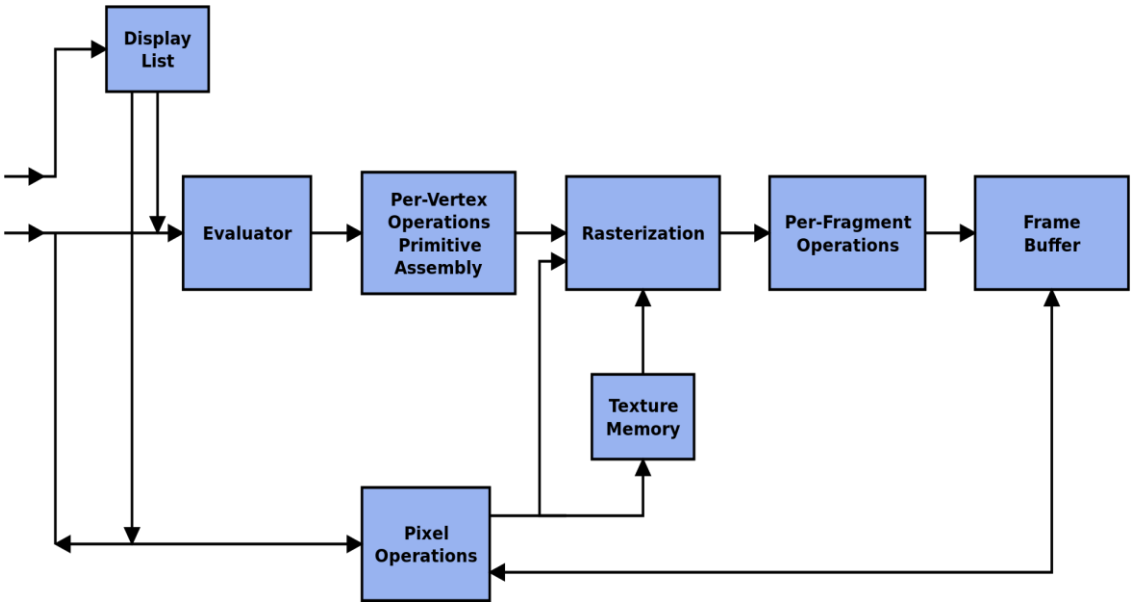


Figure 3.1 OpenGL Graphics Pipeline Architecture

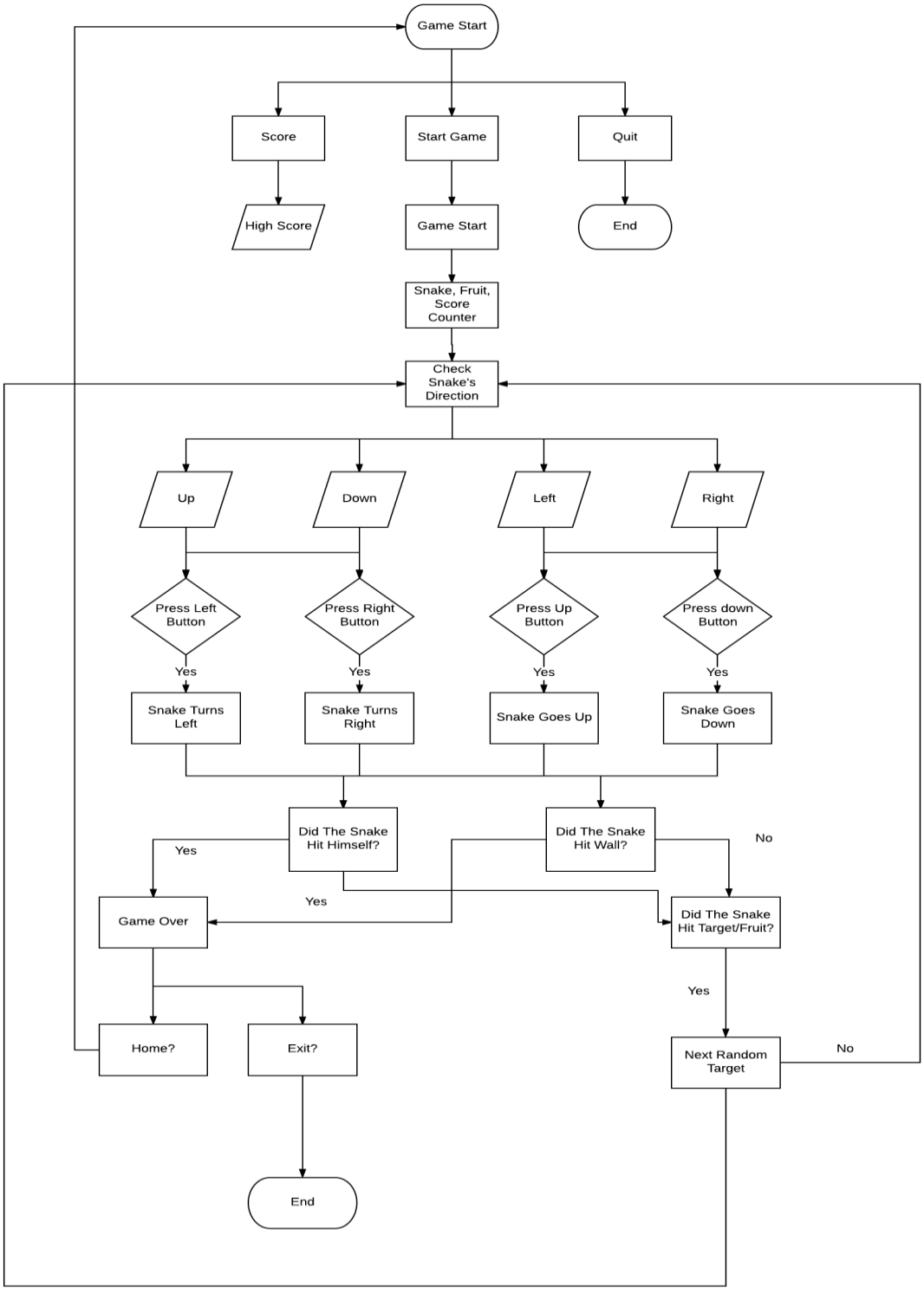
- Commands may either be accumulated in display lists, or processed immediately through the pipeline. Display lists allow for greater optimization and command reuse, but not all commands can be put in display lists.

- The first stage in the pipeline is the evaluator. This stage effectively takes any polynomial evaluator commands and evaluates them into their corresponding vertex and attribute commands.
- The second stage is the per-vertex operations, including transformations, lighting, primitive assembly, clipping, projection, and viewport mapping.
- The third stage is rasterization. This stage produces fragments, which are series of framebuffer addresses and values, from the viewport-mapped primitives as well as bitmaps and pixel rectangles. The fourth stage is the per-fragment operations. Before fragments go to the framebuffer, they may be subjected to a series of conditional tests and modifications, such as blending or z-buffering.
- Parts of the framebuffer may be fed back into the pipeline as pixel rectangles. Texture memory may be used in the rasterization process when texture mapping is enabled.

CHAPTER 4

DESIGN AND IMPLEMENTATION

A modular approach has been taken into the development of this game. The choice of programming language is C++ and header files have been used to declare and define many variables and functions that are in use in the main file. The design and workflow of the game is concisely depicted in the flowchart given below.



4.1 Basic Window

The dimensions of the window have been defined and scaled using variables present in the variables header file. A single window variable has been used to display the numerous screens present in the program. Windows are destroyed before initializing and displaying the next one.

It starts with the main screen, prompting the user to choose one of 3 options. Upon clicking the option, the respective mouse handling functions redirect the user to the next window which is saved in a function.

The main game screen includes the snake, the fruit, a bordered window and a live score counter. It can be maneuvered with the direction keys and the game gets progressively difficult with more and more fruits collected.

4.2 The Snake

The basic implementation of the snake is through an array structure that consists of the x and y position of each element or each block in the array. A rectangle function within the GLUT is utilized to graphically generate the snake with the scaling factors used to generate the play area. Any direction changes that occur during game play, result in the incrementation of either the x or y coordinate of the head element in that array structure. The trailing elements of the snake are rendered with a for loop that increments the predecessor particle in the direction of the trailing particle. All this occurs within the display function that refreshes based on a set time, that increases as the snake collects more fruits.

4.3 The Fruit

The fruit is generated randomly within the dimensions of the play area. The same rectangle function is used to generate it. The interaction between the snake and the fruit is such that, when the head of the snake i.e. the x and y co – ordinate of the snake head meets with the x and y coordinate of the fruit; the end limit of the snake is increased and the draw function of the snake and the fruit are subsequently called to increase the length of the snake and to set the position of the next fruit generated.

CHAPTER 5

SNAPSHOTS

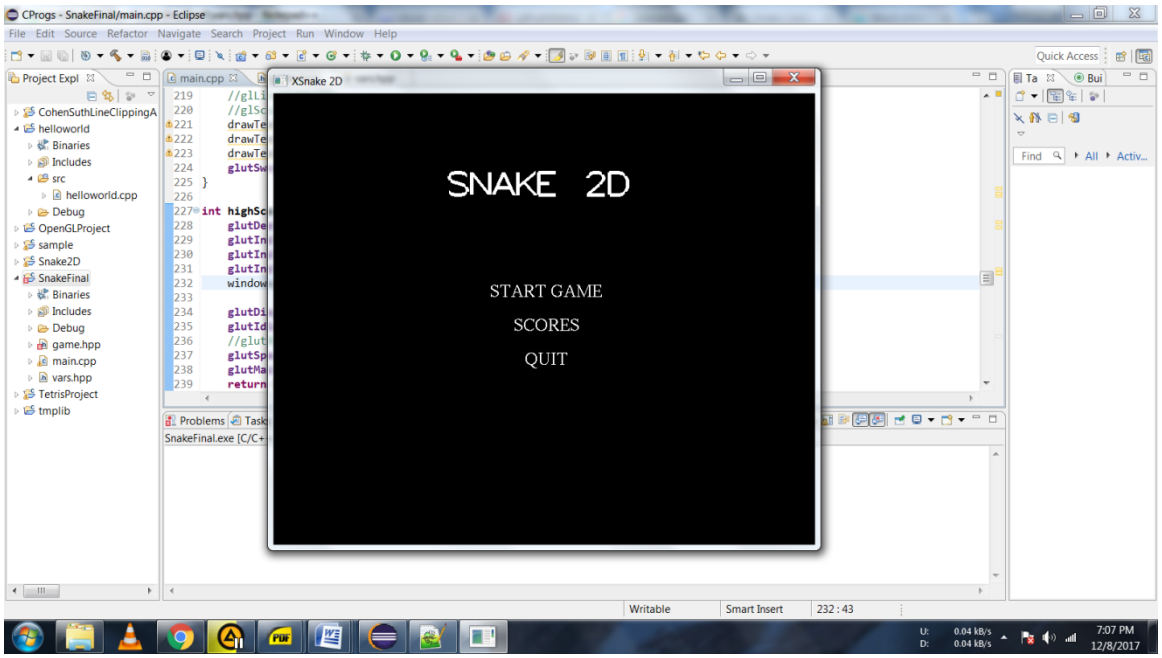


Figure 9.1 Main Screen Display

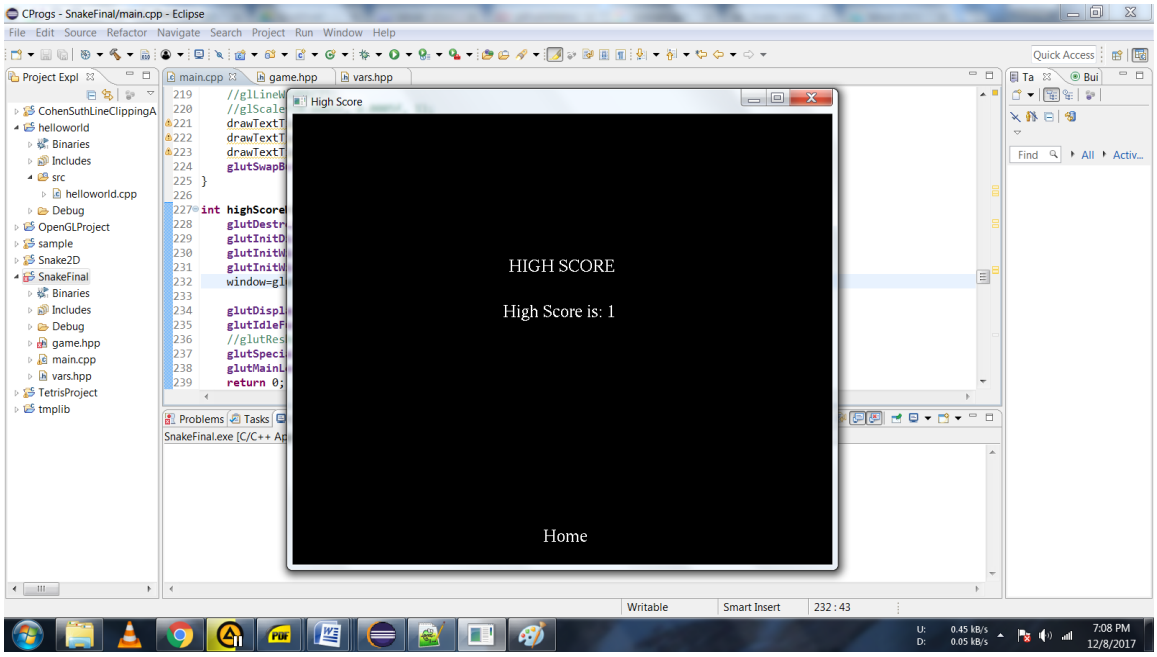


Figure 9.2 High Score Screen

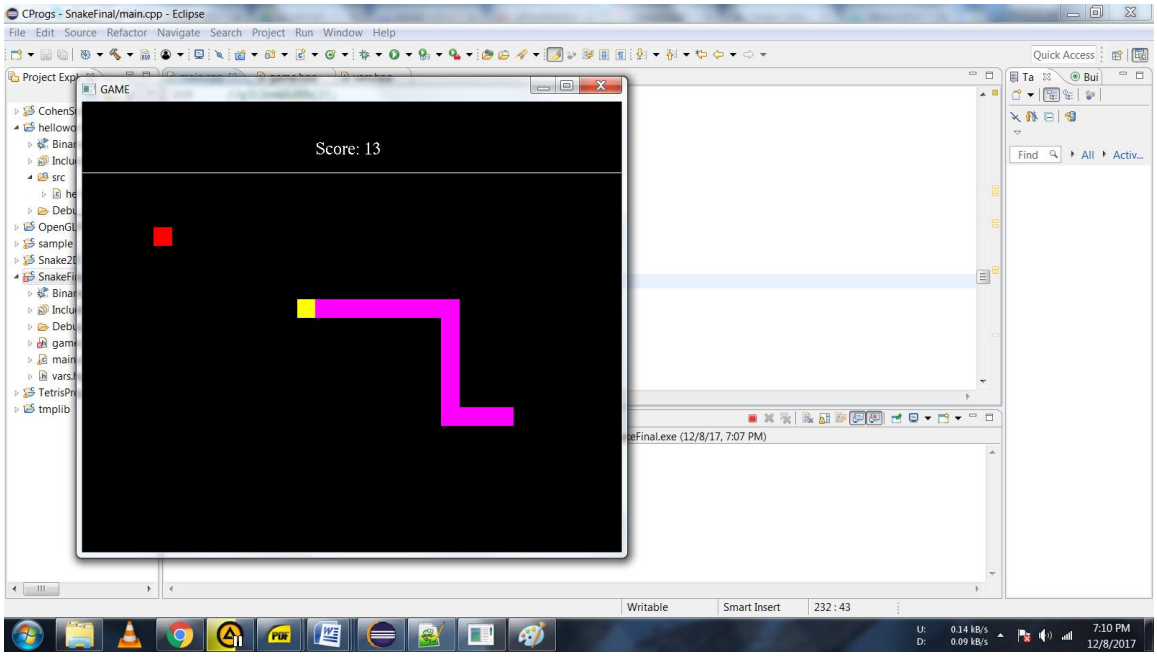


Fig 9.3 Gameplay screen

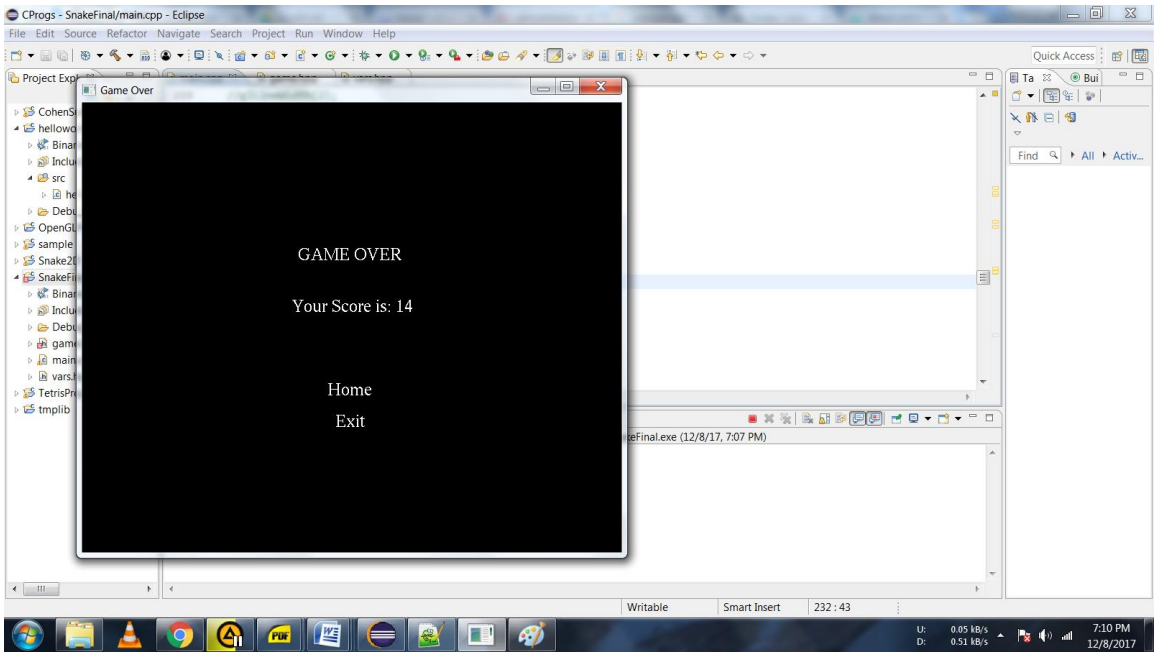


Figure 9.4 Game Over Screen

CHAPTER 6

CONCLUSION

The project entitled "SNAKE 2D GAME" is developed with the best of our effort. The project was developed using a modular approach.

The graphics creation and manipulation algorithm in the package have been implemented and tested to ensure the efficiency of operation and they were found to be quite satisfactory. The graphics editor has a good and user-friendly interface which enable the user to get better and easier interaction with the software.

After the completion of the development and study of the project we came to conclusion that the computer graphics in C using OpenGL can be used to develop much better and complex software that include 2D,3D image processing and animation.

The program is also mainly concerned with animation and more of animation implementations. This type of implementation is nowadays used in game development.

REFERENCE

- www3.ntu.sed.org
- opengl.org
- stackoverflow.com
- Graphics Under C by Yashwanth Kanetkar
- Cplusplus.com/reference
- www.khronos.org/registry/OpengGL-Refpages

APPENDIX

a) GLUT Functions: -

1. glutDisplayFunc: - glutDisplayFunc sets the display callback for the current window.
Syntax: - **void glutDisplayFunc(void (*func)(void));**
func: - The new display callback function.

2. glutReshapeFunc: - glutReshapeFunc sets the reshape callback for the current window.
Syntax: - **void glutReshapeFunc(void (*func)(int width, int height));**
func: - The new reshape callback function.

3. glutMouseFunc: - glutMouseFunc sets the mouse callback for the current window.
Syntax: - void glutMouseFunc(void (*func)(int button, int state, int x, int y));
func: - The new mouse callback function.

4. glutKeyboardFunc: - glutKeyboardFunc sets the keyboard callback for the current window.
Syntax: - **void glutKeyboardFunc(void (*func)(unsigned char key, int x, int y));**
func: - The new keyboard callback function.

5. glutCreateWindow: - glutCreateWindow creates a top-level window.
Syntax: - **intglutCreateWindow(char *name);**
name: - ASCII character string for use as window name.

6. glutInit: - glutInit is used to initialize the GLUT library.
Syntax: - void**glutInit (int *argcp, char **argv)**
Argcp:- A pointer to the program's unmodified argc variable from main. Upon return, the value pointed to by argcp will be updated, because glutInit extracts any command line options intended for the GLUT library.
Argv:- The program's unmodified argv variable from main. Like argcp, the data for argv will be updated because glutInit extracts any command line options understood by the GLUT library.

7. **glutInitWindowPosition, glutInitWindowSize** :- glutInitWindowPosition and glutInitWindowSize set the initial window position and size respectively.
Syntax: - **void glutInitWindowSize (int width, int height);**
voidglutInitWindowPosition (int x, int y);
width: Width in pixels.
Height: - Height in pixels.
X: - Window X location in pixels.
Y: - Window Y location in pixels.

8. glutInitDisplayMode: - glutInitDisplayMode sets the initial display mode.
Syntax: - **void glutInitDisplayMode (unsigned int mode);**
mode: - Display mode, normally the bitwise OR-ing of GLUT display mode bit masks.

9. glutMainLoop: - glutMainLoop enters the GLUT event processing loop.
Syntax: - **void glutMainLoop(void);**

10. glutPostRedisplay: -glutPostRedisplay marks the current window as needing to be redisplayed.
Syntax: -**void glutPostRedisplay(void);**

11. glutBitmapCharacter: -glutbitmapCharacter renders a bitmap character using OpenGL.
Syntax: -**void glutBitmapCharacter(void *font, int character);**

12. glEnable: -
Syntax: -**void glEnable(GLenum cap);**
Cap: - specifies a symbolic constant indicating a GL capability.

13. glBegin&glEnd: - The **glBegin** and **glEnd** functions delimit the vertices of a primitive or a group of like primitives.
Syntax**: - void glBegin(GLenum mode);**
mode: - The primitive or primitives that will be created from vertices presented between **glBegin**and the subsequent **glEnd**

14. glClear: - The **glClear** function clears buffers to preset values.
Syntax: - **void glClear(GLbitfield mask);**
mask: - Bitwise OR operators of masks that indicate the buffers to be cleared

15. glClearColor: - The **glClearColor** function specifies clear values for the color buffers.
Syntax**: -void glClearColor(red, green, blue, alpha);**
red: - The red value that **glClear** uses to clear the color buffers. The default value is zero.
green: -The green value that **glClear** uses to clear the color buffers. The default value is zero.
blue: - The blue value that **glClear** uses to clear the color buffers. The default value is zero.
alpha: -The alpha value that **glClear** uses to clear the color buffers. The default value is zero.

16. glColor3i: - Sets the current color.
Syntax: - **void glColor3i(GLint red, GLint green, GLint blue);**
red: - The new red value for the current color.
green: -The new green value for the current color.
blue: - The new blue value for the current color.

17. glColor3fv: - Sets the current color from an already existing array of color values.
Syntax: - **void glColor3fv(constGLfloat *v);**
V: - A pointer to an array that contains red, green, and blue values

18. glFlush: - The **glFlush** function forces execution of OpenGL functions in finite time.
Syntax: - **void glFlush(void);**
This function has no parameters.

19. glLoadIdentity: - The **glLoadIdentity** function replaces the current matrix with the identity matrix.

20. glOrtho: - The **glOrtho** function multiplies the current matrix by an orthographic matrix.
Syntax: - **void glOrtho(GLdouble left, GLdouble right, GLdouble bottom, GLdouble top, GLdoublezNear, GLdoublezFar);**
left: -The coordinates for the left vertical clipping plane.
right: - The coordinates for the right vertical clipping plane.
Bottom: - The coordinates for the bottom horizontal clipping plane.
top: - The coordinates for the top horizontal clipping plans.
zNear: - The distances to the nearer depth clipping plane. This distance is negative if the plane is to be behind the viewer.
zFar: - The distances to the farther depth clipping plane. This distance is negative if the plane is to be behind the viewer.

21. glPointSize: - The **glPointSize** function specifies the diameter of rasterized points.
Syntax: - **void glPointSize(GLfloat size);**
Size: -The diameter of rasterized points. The default is 1.0.

22.glPushMatrix & glPopMatrix: - The **glPushMatrix** and **glPopMatrix** functions push and pop the current matrix stack.
Syntax: **- void WINAPI glPopMatrix(void);**

23. glRotatef: - The **glRotatef** function multiplies the current matrix by a rotation matrix.
Syntax: - **void glRotatef(GLfloat angle, GLfloat x, GLfloat y, GLfloat z);**
angle: - The angle of rotation, in degrees.
X: - The x coordinate of a vector.
y: - The y coordinate of a vector.
z: - The z coordinate of a vector.

24. glScalef: - The **glScaled** and **glScalef** functions multiply the current matrix by a general scaling matrix.
Syntax: - **void glScalef(GLfloat x, GLfloat y, GLfloat z);**
x: - Scale factors along the x axis.
y: - Scale factors along the y axis.
Z: - Scale factors along the z axis.

25. glTranslatef: - The **glTranslatef** function multiplies the current matrix by a translation matrix.
Syntax: - **void glTranslatef(GLfloat x, GLfloat y, GLfloat z);**
x: - The x coordinate of a translation vector.

b) Complete Snake 2D Source Code

main.cpp

```
#include <GL/glut.h>
#include <stdlib.h>
#include <fstream>
#include "vars.hpp"
#include "game.hpp"

fruct fru;
game snake;
static int window;
static intgameOver=0;

void gameOverMouseFunc(int, int, int, int);
void backSpaceFunc(int, int, int);
void mouseFunc(int, int, int, int);
void homeMouseFunc(int, int, int, int);
void displayStartWindow();

void display () {
        glClear(GL_COLOR_BUFFER_BIT);
        snake.drawField();
        snake.drawSnake();
        snake.drawCounter();
        fru.draw();
        glutSwapBuffers();
}

void firstRecordSetup() {
        std::ifstream RI("save.snk");
```

```
RI >> MX;
        myMX = MX;
}

void firstGameSetup() {
        for(inti = 0; i<sh; i++) {
                s[i].x = VM_N / 2;
                s[i].y = (VM_M + sh) / 2 - i;
        }
srand(time(0));
        fru.gen();
}

void firstSetup() {
        glClearColor(0, 0, 0, 0);
        glColor3f(1, 1, 1);
}

void drawTextTimes(char* string,float x, float y)
{
        glColor3f(1,1,1);
        glPushMatrix();
        intlen, i;
        glRasterPos2f(x,y);
        len=(int) strlen(string);
        for(i = 0; i<len; i++)
        {
                glutBitmapCharacter(GLUT_BITMAP_TIMES_ROMAN_24,string[i]);
        }
        glPopMatrix();
```

```
}

void strokeTextTimes(char* string,float x, float y)
{
	glColor3f(1,1,1);
	glPushMatrix();
	intlen, i;
	glTranslatef(x,y,0);
	glScalef(0.0010f, 0.0010f, 1);
	len=(int) strlen(string);
	for(i = 0; i<len; i++)
	{
		glutStrokeCharacter(GLUT_STROKE_ROMAN, string[i]);
	}
	glPopMatrix();
}

void gameOverScreen()
{
	glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
	glMatrixMode(GL_MODELVIEW);
	glLoadIdentity ();
	glColor3f(1,1,1);
	drawTextTimes("GAME OVER",-0.2,0.3);
	char score[10];
	sprintf(score, "%d", sh-5);
	char str[30] = "Your Score is: ";
	strcat(str, score);
	drawTextTimes(str,-0.22,0.07);
	drawTextTimes("Home", -0.09, -0.3);
```

```
        drawTextTimes("Exit", -0.06, -0.44);
        glutSwapBuffers();
}

intgameOverWindow()
{
std::ofstream RO("save.snk");
   RO <<myMX;
RO.close();

gameOver=1;
        glutDestroyWindow(window);
        glutInitDisplayMode( GLUT_DOUBLE | GLUT_RGB);
        glutInitWindowSize (VD_W, VD_H+100);
        glutInitWindowPosition (100, 100);
        window=glutCreateWindow("Game Over");
        glutDisplayFunc(gameOverScreen);
        glutIdleFunc(gameOverScreen);
        glutMouseFunc(gameOverMouseFunc);
        glutSpecialFunc(backSpaceFunc);
        glutMainLoop();
        return 0;
}

void highScoreScreen() {
        glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
        glMatrixMode(GL_MODELVIEW);
        glLoadIdentity ();
        glColor3f(1,1,1);
        drawTextTimes("HIGH SCORE ",-0.2,0.3);
```

```
        firstRecordSetup();
        char hscore[10];
        itoa(MX, hscore, 10);
        char str[30] = "High Score is: ";
        strcat(str, hscore);
        drawTextTimes(str,-0.22,0.1);
        drawTextTimes("Home", -0.07, -0.9);
        glutMouseFunc(homeMouseFunc);
        glutSwapBuffers();
}


void stroke() {
        for(inti = sh; i> 0; i--) {
                s[i].x = s[i-1].x;
                s[i].y = s[i-1].y;
        }

        if(dir == 1)
                s[0].y += 1;
        if(dir == 2)
                s[0].x += 1;
        if(dir == 3)
                s[0].y -= 1;
        if(dir == 4)
                s[0].x -= 1;

        if(fru.x == s[0].x &&fru.y == s[0].y) {
                sh++;
                fru.gen();
```

```
                score++;
                if(sh> MX)
                        myMX = score;

                if(step > 1)
                        step -= step / 20;
        }

        if(s[0].x < 0 || s[0].x >= VM_N || s[0].y < 0 || s[0].y >= VM_M)
                gameOverWindow();

        for(inti = 1; i<sh; i++)
                if(s[0].x == s[i].x && s[0].y == s[i].y)
                        gameOverWindow();
}

void timer(int = 0) {
        display();
        stroke();
        glutTimerFunc(step, timer, 0);
}

void displayStartScreen()
{
glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
glLineWidth(5);
strokeTextTimes("SNAKE 2D",-0.36,0.55);
drawTextTimes("START GAME", -0.2,0.1);
drawTextTimes("SCORES", -0.11,-0.05);
drawTextTimes("QUIT", -0.07,-0.2);
```

```
glutSwapBuffers();
}

inthighScoreWindow() {
        glutDestroyWindow(window);
        glutInitDisplayMode( GLUT_DOUBLE | GLUT_RGB);
        glutInitWindowSize (VD_W, VD_H+100);
        glutInitWindowPosition (100, 100);
        window=glutCreateWindow("High Score");
        glutDisplayFunc(highScoreScreen);
        glutIdleFunc(highScoreScreen);
        glutSpecialFunc(backSpaceFunc);
        glutMainLoop();
        return 0;
}

intgameWindow() {
glutDestroyWindow(window);
        glutInitDisplayMode( GLUT_DOUBLE | GLUT_RGB);
        glutInitWindowSize (VD_W, VD_H+100);
        glutInitWindowPosition (100, 100);
        window=glutCreateWindow("GAME");
        glMatrixMode(GL_PROJECTION);
        glLoadIdentity();
        gluOrtho2D(0, VD_W, 0, VD_H+100);
        firstSetup();
        firstGameSetup();
        firstRecordSetup();

        glutSpecialFunc(game::keyboard);
```

```
        glutTimerFunc(step, timer, 0);
        glutDisplayFunc(display);
        glutIdleFunc(display);
        glutMainLoop();
   return 0;
}

void gameOverMouseFunc(intbtn, int state, int x, int y){
if(btn == GLUT_LEFT_BUTTON && state == GLUT_DOWN){
if( gameOver==1 && x >= 341 && x<= 404 &&  y >= 385 && y <= 410){
glutDestroyWindow(window);
displayStartWindow();
      }
      if (gameOver==1 && x >= 350 && x <= 395 && y >= 429 && y <= 453) {
exit(0);
         }
   }
}
void homeMouseFunc(intbtn, int state, int x, int y) {
        if(btn == GLUT_LEFT_BUTTON && state == GLUT_DOWN){
if( gameOver==0 && x >= 348 && x <= 410 && y >= 575 && y <=595) {
glutDestroyWindow(window);
displayStartWindow();
      }
   }
}
void mouseFunc(intbtn, int state, int x, int y){
if(btn == GLUT_LEFT_BUTTON && state == GLUT_DOWN){
                if(x >= 300 && x <= 455 && y >= 260 && y<= 285){
                        gameWindow();
```

```
                }
        if (x >= 333 && x <= 425 && y >= 308 && y <= 330) {
                highScoreWindow();
        }
        if (gameOver==0 && x >= 350 && x <=415 && y >=354 && y <=379) {
                exit(0);
        }
   }
}

void displayStartWindow() {
gameOver = 0;
        glutInitDisplayMode(GLUT_DOUBLE | GLUT_RGBA);
        glutInitWindowSize(VD_W, VD_H+100);
        glutInitWindowPosition(100, 100);
        window = glutCreateWindow("XSnake 2D");
        glMatrixMode(GL_PROJECTION);
        glLoadIdentity();
        glutMouseFunc(mouseFunc);
        glutDisplayFunc(displayStartScreen);
        glutIdleFunc(displayStartScreen);
        glutSpecialFunc(backSpaceFunc);
        glutMainLoop();
}

void backSpaceFunc(int key, int x, int y) {
        if (key == GLUT_KEY_HOME) {
                glutDestroyWindow(window);
                displayStartWindow();
        }
```

```
	if (key == GLUT_KEY_END) {
exit(0);
	}
}

int main(intargc, char **argv) {
	glutInit(&argc, argv);
	displayStartWindow();
	return 0;
}
```

game.hpp

```
#include<time.h>
#include<stdlib.h>
#include<GL/glut.h>
#include<iostream>

using namespace std;

class fruct{
	public:
		int x, y;
		void gen() {
			srand(time(NULL));
			x = rand() % VM_N;
			y = rand() % VM_M;
			for(inti = 0; i<sh; i++)
				if(s[i].x == x && s[i].y == y)
					gen();
		}
```

```
		void draw() {
			glColor3f(1, 0, 0);
			glRecti(x * VM_Scale - 1, y * VM_Scale, (x + 1) * VM_Scale, (y + 1) *
VM_Scale + 1);
		}
};

class game {
	public:
		void over() {
			std::ofstream RO("save.snk");
			RO <<myMX;
			RO.close();
			exit(0);
		}
		void drawField() {

			glColor3f(1,1,1);
			glBegin(GL_LINES);
				glVertex2i(0, VD_H+2);
				glVertex2i(VD_W,VD_H+2);
			glEnd();
		}

		void drawSnake() {
			glColor3f(1, 1, 0);
			glRecti(s[0].x * VM_Scale - 1, s[0].y * VM_Scale, (s[0].x + 1) * VM_Scale,
(s[0].y + 1) * VM_Scale + 1);

			glColor3f(1, 0, 1);
			for(inti = 1; i<sh; i++)
```

```
                    glRecti(s[i].x * VM_Scale - 1, s[i].y * VM_Scale, (s[i].x + 1) *
VM_Scale, (s[i].y + 1) * VM_Scale + 1);
            }

            void drawCounter() {
                char str[10] = "Score: ";
                char number[10];
                sprintf(number, "%d", score);
strcat(str, number );
        glColor3f(1,1,1);
glPushMatrix();
intlen, i;
glRasterPos2f(13*VM_Scale,VD_H+28);
len=(int) strlen(str);
for(i = 0; i<len; i++)
        {
glutBitmapCharacter(GLUT_BITMAP_TIMES_ROMAN_24,str[i]);
        }
glPopMatrix();
            }

            static void keyboard(int key, int a, int b) {
                switch(key) {
                    case 101: if(dir != 3)
                                dir = 1;
                                break;
                    case 102:
                                if(dir != 4)
                                dir = 2;
                                break;
```

```
                    case 103:
                        if(dir != 1)
                        dir = 3;
                        break;
                    case 100:
                        if(dir != 2)
                        dir = 4;
                        break;
                }
            }
};
```

vars.hpp

```
#define VM_N 30
#define VM_M 21
#define VM_Scale 25
#define DB 100
#define VD_W VM_N * VM_Scale
#define VD_H VM_M * VM_Scale

struct snk {
    int x;
    int y;
};
int dir = 1, sh = 5;
int score = 0;
snks[100];
int step = 300;
int MX, myMX;
```