ABSTRACT

Computer graphics is concerned with all aspects of producing images using a computer. The graphics software system OpenGL has become a widely accepted standard for developing graphics application. This project in OpenGL is on Slytherin(Snake Game). OpenGL interface with various graphical features like orthogonal views, translation and scaling is used here.

The project is coded in C language as it is the most basic programming language and easily understandable by a programmer. The OpenGL interface is user friendly and hence enables the user to interact efficiently with the system. It also includes three OpenGL libraries that is basic GL, GLU and GLUT. The various OpenGL functions such as glTranslatef(), glFlush(), glColor3f() etc. are implemented in this project.

Our project in OpenGL is based on a snake game. A 2D graphics based snake game is great entertainment for a user who started learning computer graphics and visualization. The development of a game has a large scope to learn computer graphics from scratch. We have use openGL toolkit to implement the algorithm using C++. This project will cover the most basic programming language and can be easily understandable by the programmer. The openGL interface is user friendly and enables the user to interact efficiently with the system.

CONTENTS

FIGURE INDEX

CHAPTER 1

INTRODUCTION

Computer graphics is a branch of computer science that deals with generating images with the aid of computers. Today, computer graphics is a core technology in digital photography, film, video games, cell phone and computer displays, and many specialized applications. A great deal of specialized hardware and software has been developed, with the displays of most devices being driven by computer graphics hardware. It is a vast and recently developed area of computer science. The phrase was coined in 1960 by computer graphics researchers Verne Hudson and William Fetter of Boeing. It is often abbreviated as CG, or typically in the context of film as CGI.

In our project, we have used a graphics software system called The OpenGL. OpenGL (Open Graphics Library) is a cross language, cross platform application programming interface(API) for reducing 2D and 3D vector graphics. A API is typically used to interact with a graphics processing unit (GPU) , to achieve hardware accelerated rendering.

Snake game is a two stage game in which food pops randomly on the screen which is eaten by a snake. Task is to direct the snake using the direction keys for the movement of the snake in up,down,left,right directions respectively. The snake moves and eats the food as soon as the head of the snake touches the food. With each food the snake eats,it grows by a certain amount and when it reaches a certain length(max length ex:60) it stops growing. The game also calculates the score of the player. The snake is unable to pass through the walls as a protective wall is constructed at the boundary.

CHAPTER 2

DESCRIPTION

2.1 DESCRIPTION OF THE PROJECT

Our project makes use of 1 library – freeGLUT to achieve the desired graphics and animation. The following section gives overall idea of the objects, scenes and the method used for animating objects in the scene.

The Scenario in the project is as follows:

We are showing an animation of 2D snake which is been controlled to move and collect food popped across the screen.

The Scenes in the project are as follows:

Scene 1: Scene 1 is a welcome screen which provides options to the player of different game themes. It includes animated snakes and background music.

Scene 2: Scene 2 is a garden themed game screen where we have implemented the snake which moves and eats food that is randomly popped on the screen.With each food the snake eats it grows to a certain length and a crunch sound is heard in the background. We have also implemented objects such as trees, street lights(lamp) along with animated objects such as clouds, butterflies and ladybugs.

Scene 3: Scene 3 is a desert themed game screen where the snake moves and eats food that is randomly popped on the screen.With each food the snake eats it grows to a certain length and a crunch sound is heard in the background. We have implemented objects such as pyramids, sun, cactus plants, crows and desert turtles.

2.2 Description of the function used:

2.2.1 Interacting with Windows:

void glutInit(int *argc, char argv):

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

argc- A pointer to the unmodified argc variable from the main function.

argv- A pointer to the unmodified argv variable from the main function.

int glutCreateWindow(char *title):

Create a window on display. The string title can be used to label window. The return value provides a reference to the window that can be used when there are multiple windows. The return value is the window handler for that particular Window. title- sets the window title.

void glutInitDisplayMode(unsigned int mode):

Requests a display with the properties in mode. The value of the mode is determined byte logical OR of options including the color model (GLUT_RGB,GLUT_INDEX) and buffering (GLUT_SINGLE,GLUT_DOUBLE). mode- specifies the display mode. GLUT_SINGLE- single buffer window GLUT_DOUBLE- double buffer window, required to have smooth animation.

void glutMainLoop():

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

void glutSwapBuffers():

Swaps the front and back buffers.

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.

void glcolor3[ub][f](TYPE r, TYPE g, TYPE b):

Sets the present RGB color. Maximum and minimum value is 255.0 and 0.0 respectively.

void glBegin(glEnum mode):

Initiates a new primitive of type mode and starts the collection of vertices. Value of mode can be GL_POINTS, GL_LINES or GL_POLYGON.

void glEnd():

Terminates the list of vertices.

2.2.2 Interaction:

void glutSpecialFunc(void (*func)(int key, int x, int y));

glutSpecialFunc sets the special keyboard callback for the current window. The

special keyboard callback is triggered when keyboard function or directional keys are pressed.

2.2.3 Transformation:

void glMatrixMode(GLenum mode):

Specifies which matrix will be affected by subsequent transformations, mode can be

GL_MODELVIEW, GL_PROJECTION etc.

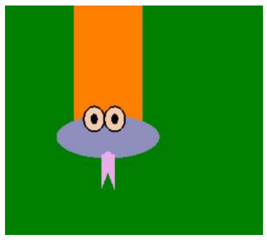
void glLoadIdentity():

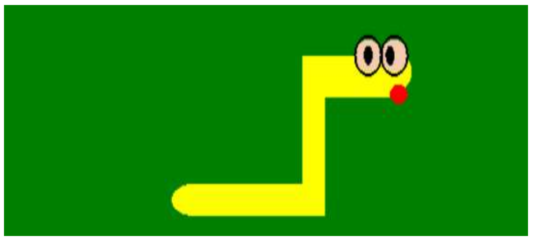
Sets the current transformation matrix to identity matrix.

2.3 Main Objects in the Project:

2.3.1 The Snakes:

The snakes in the introduction screen are drawn using rectangles, circles, and triangles. We have made use of the functions GL_QUADS for the body, GL_POLYGON for eyes, GL_LINE_LOOP for eye outline, GL_TRIANGLE_STRIP for the tongue of the snake. The circles are drawn using sin and cos functions and coordinates are converted from radians to degrees.





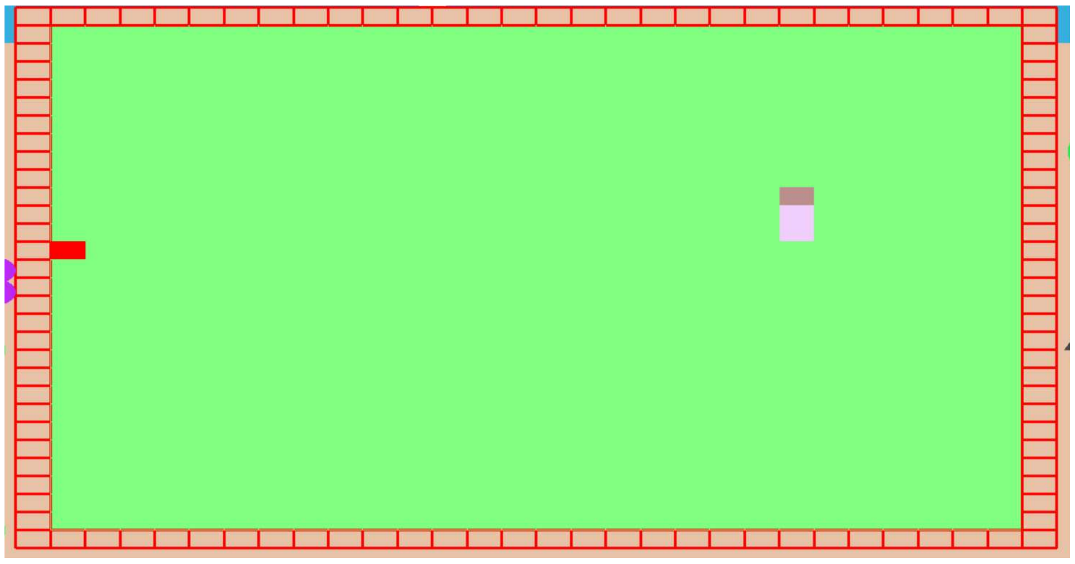
Below is the pseudo code for the snake:

```
glPushMatrix();
glTranslatef(0, sv, 0);
glColor3f(1.0 ,0.498039 ,0.0);
glBegin(GL_QUADS);
..                    //plot the vertices
glEnd();
```

```
glColor3f(0.560784 ,0.560784 , 0.737255);
glBegin(GL_POLYGON);
for (int i = 0; i < 360; i++)
{
        theta = i * 3.142 / 180;
        glVertex2f(3.0 + 1.5 * cos(theta), 20.0 + 1.2 * sin(theta));
}
glEnd();
glColor3f(0.96, 0.80, 0.69);            //snake eye code
glBegin(GL_POLYGON);
for (int i = 0; i < 380; i++)
{
        theta = i * 3.142 / 180;
        glVertex2f(2.6 + 0.3 * cos(theta), 21.0 + 0.7 * sin(theta));
}
glEnd();
..//similar code for eye number 2
glColor3f(0.0, 0.0, 0.0);       //snake eye  outline
glLineWidth(2.0);
glBegin(GL_LINE_LOOP);
for (int i = 0; i <= 100; i++) {
        glVertex2f(
                2.6 + (0.3 * cos(i * 2.0f * 3.142 / 100)),
                21.0 + (0.7 * sin(i * 2.0f * 3.142 / 100))
        );
}
..      //similar code for 2nd eye outline
glEnd();
glColor3f(0.0, 0.0, 0.0);               //eyeball 1
glBegin(GL_POLYGON);
for (int i = 0; i < 360; i++)
{
        theta = i * 3.142 / 180;
        glVertex2f(2.6 + 0.1 * cos(theta), 21.0 + 0.3 * sin(theta));
        }
        glEnd();
        ..                  //similar code for eye ball number 2 and mouth
        glColor3f(0.917647 ,0.678431 ,0.917647);    //code for tongue
        glBegin(GL_TRIANGLE_STRIP);
        ..                  //plot vertices
        glEnd();
        glPopMatrix();
```

2.3.2 The Game Grid , Snake and food:

The area the snake is moving in i.e, the grid is made up of lines. We have used the GL_LINE_LOOP function for this grid. The snake is drawn using a rectangle and we have used glRectd for this purpose. We have used glRectf function to draw food. The food is placed at a random position using the rand function.



Below is the pseudo code for grid:

```
void drawGrid()
{
        for (int x = 5; x < gridX-5 ; x++)
        {
                for (int y = 5; y < gridY-5 ; y++)
                {
                        unit(x, y);
                }
        }
}

void unit(int x, int y)
{
        if (x == 5 || y == 5 || x == gridX - 6 || y == gridY - 6)
        {
                glLineWidth(3.0);
                glColor3f(ra, rb, rc);          //red
        }
        else {
                glLineWidth(1.0);
                glColor3f(rr, gg, bb);          //green
        }
        glBegin(GL_LINE_LOOP);
        glVertex2f(x, y);
        glVertex2f(x + 1, y);
        glVertex2f(x + 1, y + 1);
        glVertex2f(x, y + 1);
        glEnd();
}
```

Below is the pseudo code for snake:

```
void drawSnake()
{
        for (int i = snake_length - 1; i > 0; i--)
        {
                posX[i] = posX[i - 1];
                posY[i] = posY[i - 1];
        }
        if (sDirection == UP)
                posY[0]++;
        else if (sDirection == DOWN)
                posY[0]--;
        else if (sDirection == RIGHT)
                posX[0]++;
        else if (sDirection == LEFT)
                posX[0]--;
        for (int i = 0; i < snake_length; i++)
        {
                if (i == 0)
                        glColor3f(0.737255,  0.560784 , 0.560784);  //head
                else
                        glColor3f(o1,o2,o3);
                glRectd(posX[i], posY[i], posX[i] + 1, posY[i] + 1);

        }
}
```

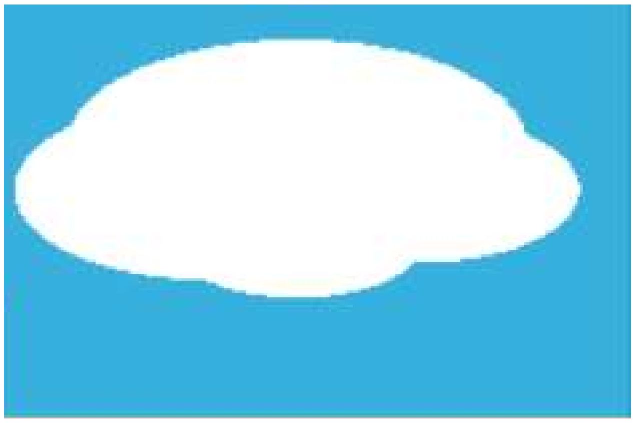
Below is the pseudo code for food:

```
void drawFood()
{
        if (food)
                random(foodX, foodY);
        food = false;
        glColor3f(1.0, 0.0, 0.0);
        glRectf(foodX, foodY, foodX + 1, foodY + 1);

}
void random(int& x, int& y)
{
        int _maxX = gridX - 7;
        int _maxY = gridY - 7;
        int _min = 6;
        srand(time(NULL));
        x = _min + rand() % (_maxX - _min);
        y = _min + rand() % (_maxY - _min);
}
```

2.3.3 The Cloud:

The Cloud is drawn completely using circles. Four overlapping circles are used to draw the cloud. Here, the circles are drawn using sin and cos functions and coordinates are converted from radians to degrees. In order to draw it, we used GL_Polygon function here.

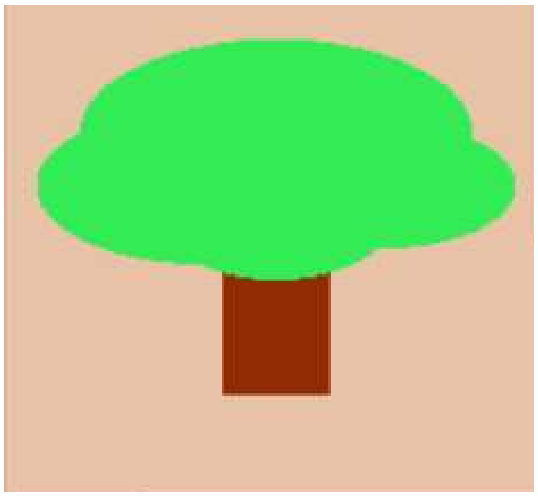


Below is the pseudo code for the cloud:

```
void clouds()
{
        float theta;
        glTranslatef(ch, 0, 0);
        glColor3f(1.0, 1.0, 1.0);                                   //top
        glBegin(GL_POLYGON);
        for (int i = 0; i < 360; i++)
        {
                theta = i * 3.142 / 180;
                glVertex2f(37.5 + 1.8 * cos(theta), 38 + 1.8 * sin(theta));
        }
        glEnd();
}
```

2.3.4 The Tree:

The tree is drawn using a rectangle and 4 overlapping circles. The circles are drawn using sin and cos functions and coordinates are converted from radians to degrees. In order to draw it, we used GL_Polygon function here.

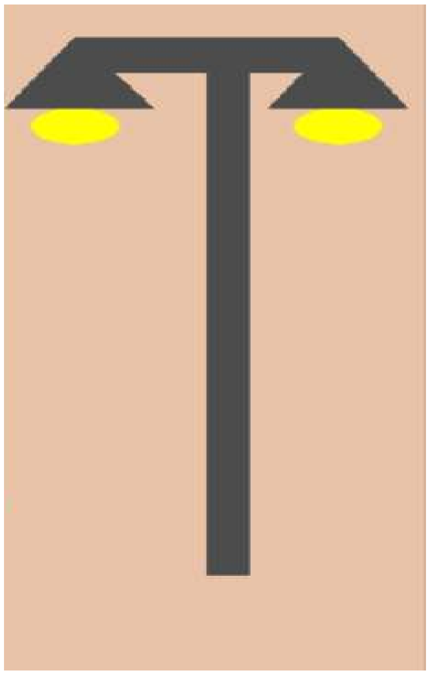


Below is the pseudo code for the tree:

```
void tree()
{
        //tree 1
        glColor3f(0.57, .17, 0.017);
        glBegin(GL_POLYGON);    //tree bark
        ..                  //plot the vertices
        glEnd();
        float theta;
        glColor3f(0.2, 0.92, 0.33);                             //top
        glBegin(GL_POLYGON);
        for (int i = 0; i < 360; i++)
        {
                theta = i * 3.142 / 180;
                glVertex2f(2.5+ 1.8 * cos(theta), 7 + 1.8* sin(theta));
        }
        glEnd();
        ..                          //similar code for 3 more circles
        //tree 2
        glColor3f(0.57, .17, 0.017);
        glBegin(GL_POLYGON);                              //tree bark
        ..              //plot vertices
        glEnd();
```

2.3.5 The Lamp:

The lamp is drawn using triangles, rectangles and circles. We make use of GL_POLYGON and the circles are drawn using sin and cos functions and coordinates are converted from radians to degrees



Below is the pseudo code for lamp:

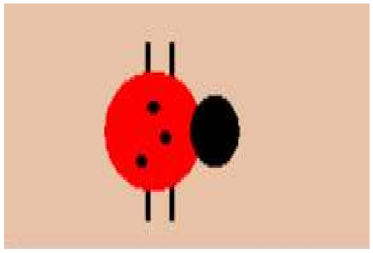
```
{
        //lamp
        glColor3f(0.3,0.3,0.3);                   //lamp pole
        glBegin(GL_POLYGON);
        …                              // plot the vertices
        glEnd();
        glColor3f(0.3, 0.3, 0.3);            //horizontal pole part
        glBegin(GL_POLYGON);
        …               //plot vertices
        glEnd();
        //left bulb
```

```
glColor3f(0.3, 0.3, 0.3);
glBegin(GL_POLYGON);
…                  //plot vertices
glEnd();
float theta;
glColor3f(1.0, 1.0, 0.0);
glBegin(GL_POLYGON);
for (int i = 0; i < 360; i++)
{
        theta = i * 3.142 / 180;
        glVertex2f(2.0 + 0.5 * cos(theta), 36.5+ 0.5 * sin(theta));
}
glEnd();
//right bulb
glColor3f(0.3, 0.3, 0.3);
glBegin(GL_POLYGON);
..                          //plot vertices
glEnd();


glColor3f(1.0, 1.0, 0.0);
glBegin(GL_POLYGON);
for (int i = 0; i < 360; i++)
{
        theta = i * 3.142 / 180;
        glVertex2f(5.5 + 0.5 * cos(theta), 36.5 + 0.5 * sin(theta));
}
glEnd();
```

2.3.6 The Bug:

The bug body is drawn using circles and its feet are drawn using lines. We make use of GL_LINES and GL_POLYGON functions for the same. The circle is drawn using sin and cos functions and coordinates are converted from radians to degrees.



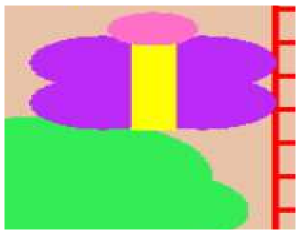
Below is the pseudo code for bug:

```
void bug()
{
        float theta;
        glPushMatrix();
        glTranslatef(lx, 0, 0);
        glBegin(GL_LINES);                      //legs
        glLineWidth(2.0);
        ..                      //plot vertices
        glEnd();
        glColor3f(1.0, 0.0, 0.0);                               //body
        glBegin(GL_POLYGON);
        for (int i = 0; i < 360; i++)
        {
                theta = i * 3.142 / 180;
                glVertex2f(5.0 + 0.8 * cos(theta), 2.0 + 1.0 * sin(theta));
        }
        glEnd();
        glColor3f(0.0, 0.0, 0.0);                               //head
        glBegin(GL_POLYGON);
        for (int i = 0; i < 360; i++)
        {
                theta = i * 3.142 / 180;
                glVertex2f(6.0 + 0.4 * cos(theta), 2.0 + 0.6 * sin(theta));
        }
        glEnd();
        glColor3f(0.0, 0.0, 0.0);
```

```
glBegin(GL_POLYGON);                                          //bug spots
for (int i = 0; i < 360; i++)
{
        theta = i * 3.142 / 180;
        glVertex2f(4.8 + 0.1 * cos(theta), 1.5 + 0.1 * sin(theta));
}
glEnd();
glBegin(GL_POLYGON);
for (int i = 0; i < 360; i++)
{
        theta = i * 3.142 / 180;
        glVertex2f(5.2 + 0.1 * cos(theta), 1.9 + 0.1 * sin(theta));
}
glEnd();
glBegin(GL_POLYGON);
for (int i = 0; i < 360; i++)
{
        theta = i * 3.142 / 180;
        glVertex2f(5.0 + 0.1 * cos(theta), 2.4 + 0.1 * sin(theta));
}
glEnd();
glPopMatrix();
```

2.3.7 The Butterfly:

The butterfly is drawn using circles as wings and the body is drawn using a rectangle. We make use of the GL_POLYGON function for the same. The circle is drawn using sin and cos functions and coordinates are converted from radians to degrees.



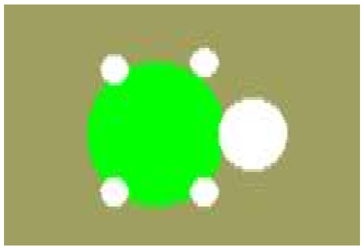
Below is the pseudo code for butterfly:

```
void butterfly()
{
	float theta;	//wings
	//butterfly1
	glPushMatrix();
	glTranslatef(b1x, b1y, 0);
	glColor3f(0.73, 0.16, 0.96);
	glBegin(GL_POLYGON);
	for (int i = 0; i < 360; i++)
	{
		theta = i * 3.142 / 180;
		glVertex2f(1.7 + 0.8 * cos(theta), 19.0 + 0.8 * sin(theta));
	}
	glEnd();
	glColor3f(0.73, 0.16, 0.96);
	glBegin(GL_POLYGON);
	for (int i = 0; i < 360; i++)
	{
		theta = i * 3.142 / 180;
		glVertex2f(1.7 + 0.8 * cos(theta), 17.8 + 0.8 * sin(theta));
	}
	glEnd();
	glColor3f(0.73, 0.16, 0.96);
	glBegin(GL_POLYGON);
	for (int i = 0; i < 360; i++)
	{
		theta = i * 3.142 / 180;
		glVertex2f(2.8 + 0.8 * cos(theta), 19.0 + 0.8 * sin(theta));
	}
	glEnd();
	glColor3f(0.73, 0.16, 0.96);
	glBegin(GL_POLYGON);
```

```
        for (int i = 0; i < 360; i++)
        {
                theta = i * 3.142 / 180;
                glVertex2f(2.8 + 0.8 * cos(theta), 17.8 + 0.8 * sin(theta));
        }
        glEnd();
        //body
        glColor3f(1.0, 1.0, 0.0);
        glBegin(GL_POLYGON);
        …                //plot vertices
        glEnd();
        //head
        glColor3f(1.00, 0.43, 0.78);
        glBegin(GL_POLYGON);
        for (int i = 0; i < 360; i++)
        {
                theta = i * 3.142 / 180;
                glVertex2f(2.25 + 0.5 * cos(theta), 20.0 + 0.5 * sin(theta));
        }
        glEnd();
        glPopMatrix();
}
```

2.3.8 The Turtle:

The turtle is drawn using circles. The circle is drawn using sin and cos functions and coordinates are converted from radians to degrees. We have made use of GL_POLYGON.



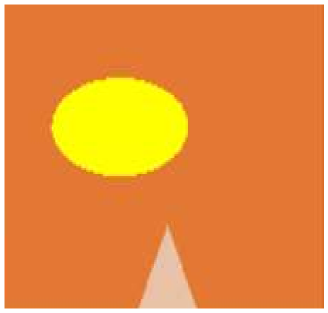
Below is the pseudo code for turtle:

```
void turtle()
```

```
{
        glPushMatrix();
        glTranslatef(0, v, 0);
        glColor3f(0.5, 1.0, 0.0);
        glBegin(GL_POLYGON);
        for (int i = 0; i < 360; i++)
        {
                theta = i * 3.142 / 180;
                glVertex2f(2.0 + 1.0 * cos(theta), 25.0 + 1.0 * sin(theta));
        }
        glEnd();
        ..      // similar code for creating other parts of the turtle with different vertices
        glPopMatrix();
}
```

2.3.9 The Sun:

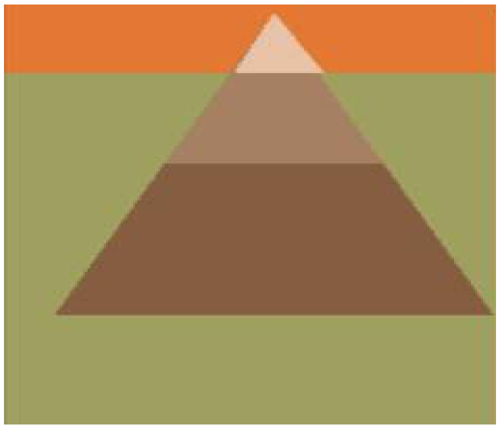
The sun is drawn using a circle . The circle is drawn using sin and cos functions and coordinates are converted from radians to degrees. We have made use of GL_POLYGON.



```
        glColor3f(1.0 ,1.0 , 0.0);
        glBegin(GL_POLYGON);
        for (int i = 0; i < 360; i++)
        {
                theta = i * 3.142 / 180;
                glVertex2f(37 + 1.0 * cos(theta), 37 + 1.0 * sin(theta));
        }
        glEnd();
```

2.3.10 The Pyramid:

The pyramid is drawn using triangles. We have used 3 triangles that overlap on each other. GL_POLYGON function is used for this.

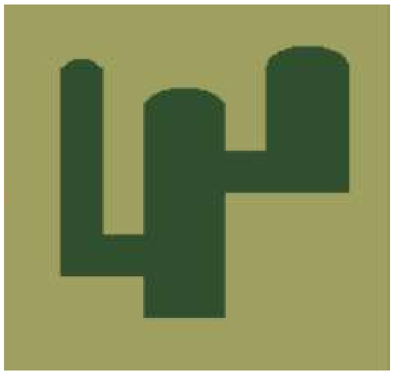


Below is the pseudo code for pyramid:

```
glColor3f(0.52, 0.37, 0.26);
glBegin(GL_POLYGON);
..                    //plot vertices
glEnd();
..          // similar code for the 2 more triangles
```

2.3.11 The Cactus:

The cactus is made using rectangles and semi circles. We use the GL_POLYGON function for this. The semi circle is drawn using sin and cos functions and coordinates are converted from radians to degrees.



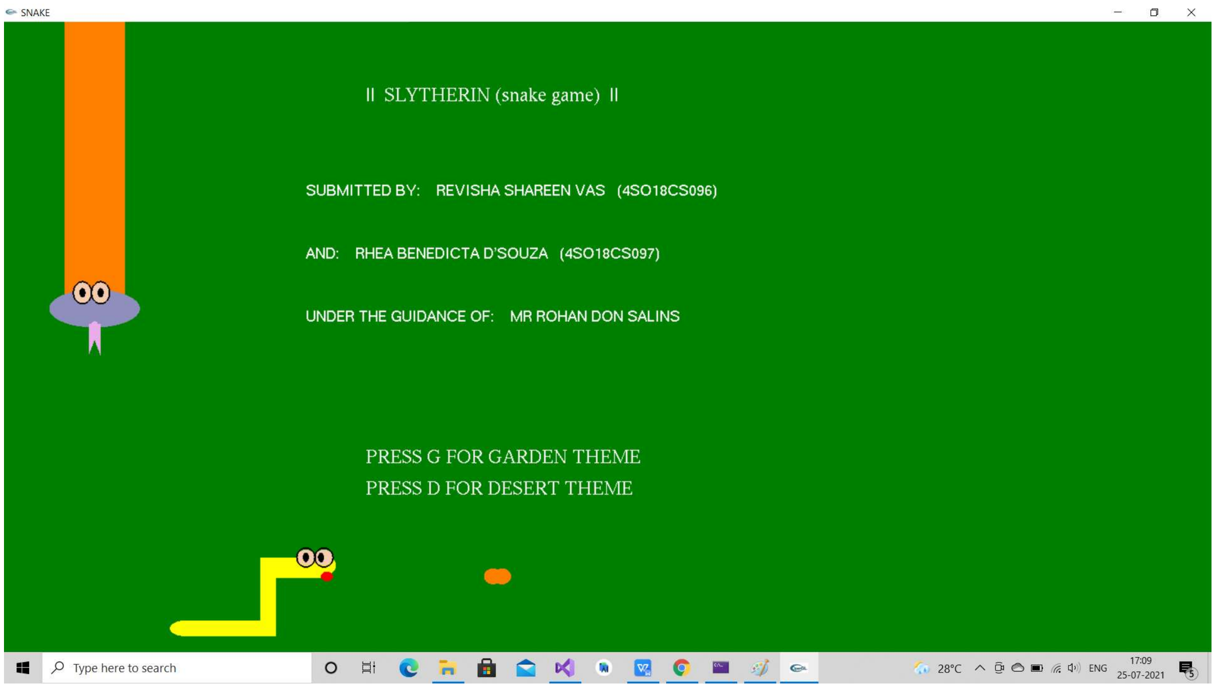
Below is the pseudo code for cactus:

```
glColor3f(0.184314, 0.309804, 0.184314);
glBegin(GL_POLYGON);
glVertex2d(2, 0);
glVertex2d(3, 0);
```

```
glVertex2d(3, 5);
glVertex2d(2, 5);
glEnd();
glColor3f(0.184314 , 0.309804,  0.184314);
glBegin(GL_POLYGON);
for (int i = 0; i < 180; i++)
{
        theta = i * 3.142 / 180;
        glVertex2f(2.5 + 0.5 * cos(theta), 5.0 + 0.5 * sin(theta));
}
glEnd();
…       // similar code for remaining part of the cactus
```

CHAPTER 3

SCREENSHOTS



SNAKE

|| SLYTHERIN (snake game) ||

SUBMITTED BY: REVISHA SHAREEN VAS (4SO18CS096)

AND: RHEA BENEDICTA D'SOUZA (4SO18CS097)

UNDER THE GUIDANCE OF: MR ROHAN DON SALINS

PRESS G FOR GARDEN THEME
PRESS D FOR DESERT THEME

Fig 3.1 Introduction Scene

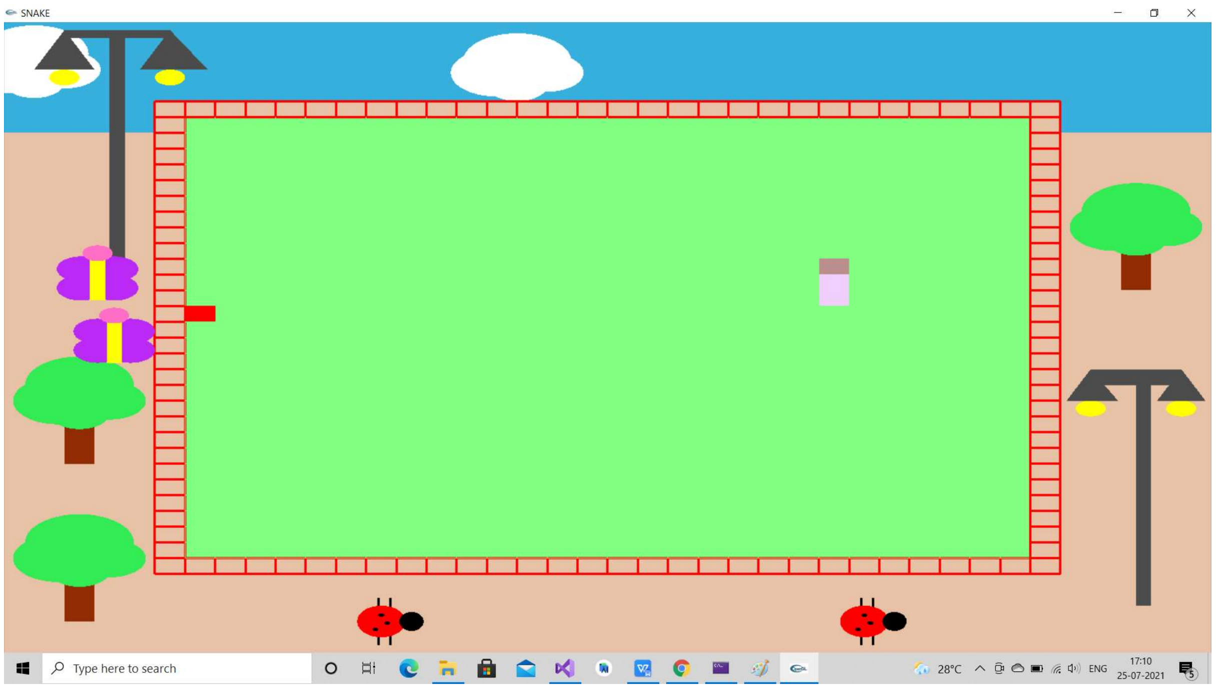


Fig 3.2 Garden Scene

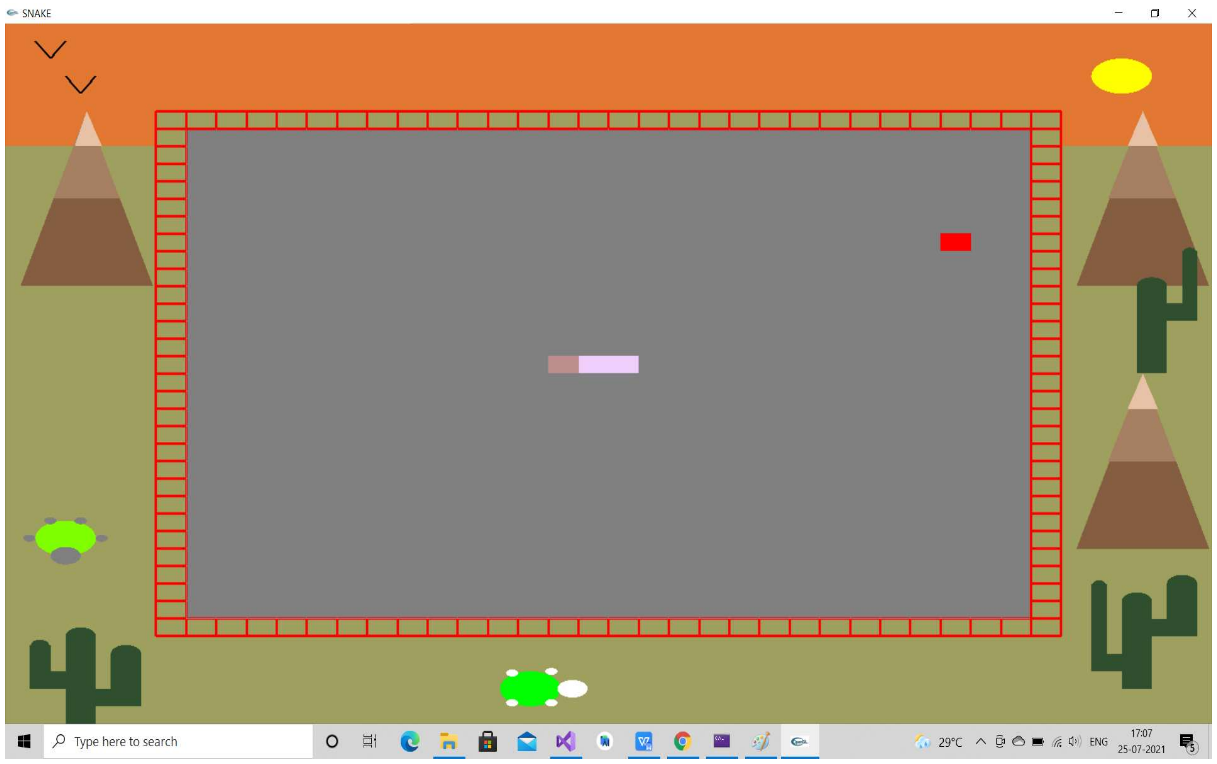


Fig 3.3 Desert Scene

CHAPTER 4

CONCLUSION & FUTURE WORK

5.1 Conclusion

The primary goal of this project is to showcase the concepts that we have learnt in theory and how these concepts can be used to visually demonstrate algorithms in computer science. By visually representing transformations in an animation form, the viewer gets a clear idea of it. The keyboard driven interfaces are used to show the various scenes in the project. This reduces the complexity and improves the case with which any kind of user can run it. It also provides a base for developing advanced projects on realistic view in 2D that has a core demand in the graphics and Animation Industry. The project has also helped in understanding the working of computer graphics using OpenGl and the various concepts, functions and methodologies required for the development of this project.

5.2 Future work

In the near future, we may add additional obstacles like rocks, boulders and so on. The game could provide a multiplayer option too. In addition to this we may also add collectables like coins and other boosts, where, on obtaining them the players points shoot up to an additional five. We may also add extra lives to the snake, where each player can have a maximum of 3 lives before the game ends.

CHAPTER 5

REFERENCES

1) Edward Angel: Interactive Computer Graphics, 5th Edition, Pearson Education, 2008

2) https://www.javatpoint.com/computer-graphics-tutorial

3) https://www.ambiera.com/irrklang/