2D Obstacle Paper Plane

Computer Graphics Project

THAPAR INSTITUTE
OF ENGINEERING & TECHNOLOGY
(Deemed to be University)

Submitted by:

102097003 Simran Kumari 3CS2

102097014 Bhavya Tyagi 3CS2

102097002 Prince 3CS2

BE Third Year- COPC

Under the Mentorship of

Ms. Jasmine

TA, Ph.D. Scholar

Computer Science and Engineering Department

Thapar Institute of Engineering and Technology, Patiala

March 2022

Table of Contents

Acknowledgment

We would like to express special thanks and gratitude to our teacher Ms. Jasmine Kaur who gave us the opportunity to do this amazing project on a 2D-Obstacle Paper Plane game which helped us in researching new concepts and helped us broaden our horizons.

We are grateful for her willingness to give us valuable advice and direction whenever we approached her with a problem. We thank her from the bottom of our hearts for her exemplary guidance, monitoring, and constant encouragement throughout the course of this project without which this project would have not been possible.

We also thank the Thapar Institute of Engineering and Technology for providing us with all the educational, technological and infrastructural help required while developing this project.

Introduction

1.1 Introduction to Computer Graphics

People used to show information manually before introducing computer graphics, either by drawing or making a model that represented the real world. The Greeks, for example, were able to communicate their architectural concepts through drawing graphically. Architects, mechanical engineers, and designers now use computer-based graphics systems to generate the same type of information.

A visual paradigm that comprises windows, icons, menus, and a pointing device, such as a mouse, has dominated our computer interactions. Windowing systems such as the X windows system and Microsoft's Windows are user-friendly. Although most of us are familiar with the graphical user interface (GUI) seen on most workstations, developments in computer graphics have made other interfaces possible.

1.2 Introduction to OpenGL API

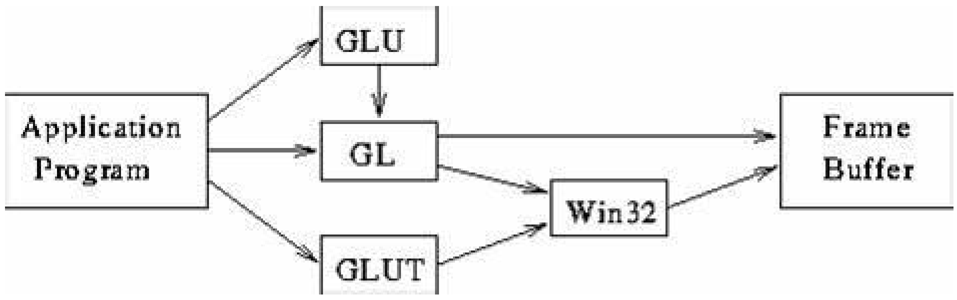
Because the structure of OpenGL is similar to that of most modern APIs, whatever time you invest in studying it will pay off in other software systems. When compared to other APIs, OpenGL is simple to understand and supports both two and three-dimensional programming.
OpenGL provides an interface to graphics hardware for programmers. It's a high-performance low-level rendering and modelling software library that runs on all major platforms and supports a wide range of hardware.

Most of our applications will be designed to access OpenGL directly through functions in three libraries. They are as follows:

1. GLU
2. GL
3. GLUT

Functions that start with GL letters are commonly found in the primary GL libraries. The OpenGL Utility Library (GLU) is a library that uses gl functions but also includes code for building everyday objects and making viewing easier. At least one more library is required to interface with the Windows system and receive input from external devices into our programs. The OpenGL Utility

Toolkit is a system-specific library that provides the glue between the window system and OpenGL for each central window system (GLUT).



1.3 OpenGL

To render a three-dimensional scene, OpenGL provides a series of commands. You submit the data in an OpenGL-compatible format, and OpenGL renders it on the screen. It is free to use and has been created by several companies.

OpenGL is a system and equipment interface. As long as an OpenGL implementation is installed, an OpenGL application will run on any platform. There are no routines to construct windows or anything like that because it is system independent, but each platform has support functions. GLUT is a handy tool.

1.4 GLUT

Mark Kilgard wrote GLUT, a comprehensive API that allows you to construct windows and manage messages. It is available for various systems, which means that GLUT-based software can be compiled on numerous platforms with little (or no) code changes.

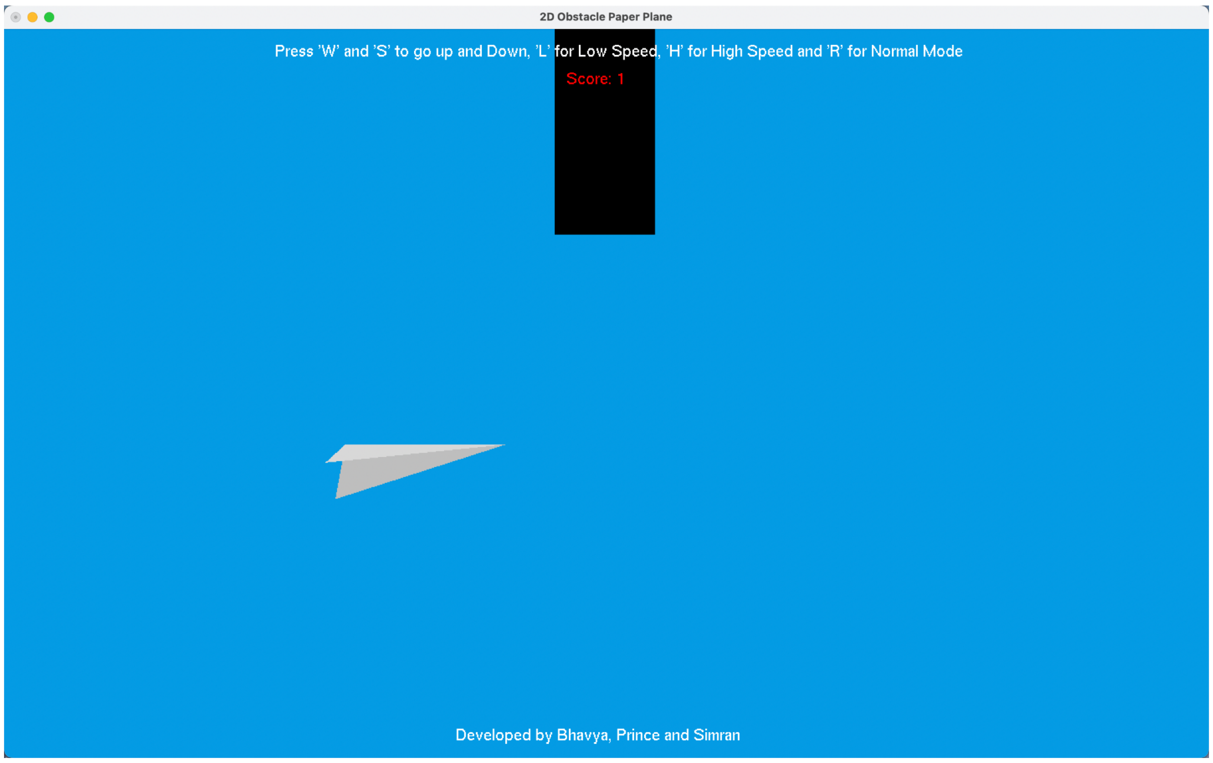
1.5 How does OpenGL work?

The state variables are the foundation of OpenGL. Many values, such as the color, remain after they have been specified. In contrast to DirectX, there are no classes. It is, however, logically organised.

OpenGL has its own set of data types. They all start with the letter "GL". For instance, GLfloat, GLint, and so on. There are numerous symbolic constants, beginning with "GL_" such as GL_POINTS and GL_POLYGON.

1.6 Project Introduction

This project implements some OpenGL functions to create an animated game.


2D Obstacle Paper Plane
Press 'W' and 'S' to go up and Down, 'L' for Low Speed, 'H' for High Speed and 'R' for Normal Mode
Score: 1
Developed by Bhavya, Prince and Simran

The project consists of a white paper plane and some obstacles that are of black colour. The paper plane can move upward, downward, and forward (not backward). We can only control upward and downward movement and the forward movement is the part of the animation i.e., the x-direction translation is simultaneously happening with the y-direction upon hitting the ‘W’ or ‘S’ key. About obstacles, they are of random sizes and are at random positions. We have no control over obstacles and they are appearing randomly which is also a part of our implementation.

Rules of the Game

2.1 Modes of the Game

The rules of the games are very simple. The user has to only save the plane from collapsing to obstacles. The game has 3 different levels.

1. **Low:-** The user has to press 'L' at the start of the game to play in this mode. The speed of the plane and obstacles in this mode is very slow as compared to other modes.
2. **Medium/Normal:-** The user has to press 'R/N' at the start of the game to play in this mode. The speed of the plane and obstacles in this mode is faster than in the lower mode but slower than in the higher mode.
3. **High:-** The user has to press 'H' at the start of the game to play in this mode. The speed of the plane and obstacles in this mode is very high as compared to other modes.

2.2 Controls of the games

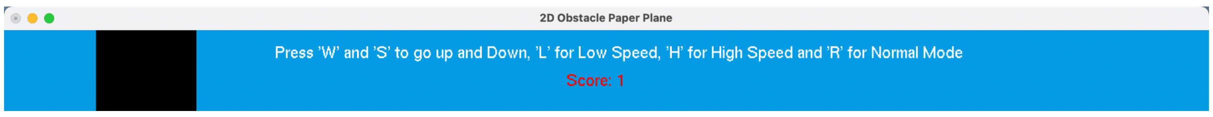
As the user can only move the plane upward and backward there are only two controls in this game.

1. Press 'W' to move the plane upward and forward.
2. Press 'S' to move the plane downward and forward.

By using these two keys the user has to save the plane from collapsing to obstacles.

2.3 Score counter

There is a score counter at the top of the screen.



Press 'W' and 'S' to go up and Down, 'L' for Low Speed, 'H' for High Speed and 'R' for Normal Mode

Score: 1

For passing every obstacle completely the score will increase by 1 point also there is no limit to the game or to the score.

The game will be over when the plane will collapse with the obstacle but you can restart the game by pressing the 'R' key and your score will start from 0 again.

OpenGL Functions

3.1 Basic Functions:-

1. **glColor3f Function:-**Sets the current colour.
   SYNTAX: void glColor3f(GLfloat red, GLfloat green, GLfloat blue);
2. **glBegin, glEnd Function:-**The glBegin, and glEnd functions delimit the vertices of a primitive or a group of like primitives.
   SYNTAX: void glBegin, glEnd (GLenum mode);
   PARAMETERS: mode
   The primitive or primitives that will be created from vertices presented between glBegin and the subsequent glEnd. The following are accepted symbolic constants and their meanings:
   GL_LINES: Treats each pair of vertices as an independent line segment. Vertices 2n - 1 and 2n define line n. N/2 lines are drawn.
   GL_LINE_STRIP: Draws a connected group of line segments from the first vertex to the last. Vertices n and n+1 define line n. N - 1 lines are drawn.
   GL_LINE_LOOP: Draws a connected group of line segments from the first vertex to the last, then back to the first. Vertices n and n + 1 define line n. The last line, however, is defined by vertices N and N lines are drawn.
   GL_TRIANGLES: Treats each triplet of vertices as an independent triangle. Vertices 3n -2, 3n - 1, and 3n define triangle n. N/3 triangles are drawn.
   GL_QUADS: Treats each group of four vertices as an independent quadrilateral. Vertices 4n - 3, 4n - 2, 4n - 1, and 4n defined quadrilateral n. N/4 quadrilaterals are drawn.

3.2 Translation Functions:-

1. **glTranslate:-**The glTranslated and glTranslatef functions multiply the current matrix by the translation matrix.
   Void glTranslate(x, y, z) ;
   Parameters
   x, y, z : The x, y, and z coordinates of a translation vector.

2. **glRotate:-**The glRotated and glRotatef functions multiply the current matrix by a rotation matrix.

Void glRotate(Glfloat angle,Glfloat x,Glfloat y,Glfloat z);
Parameters
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.

3.3 Functions Used To Display:-

1. **glClear Function:-**The glClear function clears buffers to preset values.
   SYNTAX: glClear(GLbitfield mask);
   PARAMETERS: mask
   Bitwise OR operators of masks that indicate the buffers to be cleared. The four masks are as follows:

Value	**Meaning**
GL_COLOR_BUFFER_BIT	The buffers currently enabled for colour writing.
GL_DEPTH_BUFFER_BIT	The depth buffer.

   glClear(GL_COLOR_BUFFER_BIT|GL_DEPTH_BUFFER_BIT);

2. **glMatrixMode Function:-**The glMatrixMode function specifies which matrix is the current matrix.
   SYNTAX: void glMatrixMode(GLenum mode);
   PARAMETERS: mode
   The matrix stack is the target for subsequent matrix operations. The mode parameter can assume one of three values:

Value	**Meaning**
GL_MODELVIEW	Applies subsequent matrix operations to the modelview matrix stack.

   glMatrixMode(GL_MODELVIEW);

3. **glLoadIdentity Function:-**The glLoadIdentity function replaces the current matrix with the identity matrix.
   SYNTAX: void glLoadIdentity(void);

4. **glOrtho:-**
    - This function defines orthographic viewing volume with all parameters measured from the centre of projection.
    - multiply the current matrix by a perspective matrix.

SYNTAX: void glOrtho(GLdouble left, GLdouble right, GLdouble bottom,GLdouble top, GLdouble near, GLdouble far)

3.4 Functions Used to Reshape:-

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

3.5 Main Functions:-

1. **glutInitDisplayMode Function:-**
   glutInitDisplayMode sets the initial display mode.
   SYNTAX: void glutInitDisplayMode(unsigned int mode);
   PARAMETERS:
   • mode
   Display mode, normally the bitwise OR-ing of GLUT display mode bitmasks. See values below:
   GLUT_RGB: An alias for GLUT_RGBA.
   GLUT_DOUBLE: Bitmask to select a double-buffered window. This overrides
   GLUT_SINGLE if it is also specified.
   GLUT_DEPTH: Bitmask to select a window with a depth buffer.
   • glutInitDisplayMode(GLUT_RGB|GLUT_DEPTH | GLUT_DOUBLE);

2. **glutInitWindowPosition,glutInitWindowSize Functions:-**
   glutInitWindowPosition and glutInitWindowSize set the initial window position and size
   . respectively.

SYNTAX: void glutInitWindowSize(int width, int height);
void glutInitWindowPosition(int x, int y);
PARAMETERS:
• width
Width in pixels.
• height
Height in pixels.
• x
Window X location in pixels.
• y
Window Y location in pixels.
• glutInitWindowSize(300,300);

3. **glutMainLoop Function:-**glutMainLoop enters the GLUT event processing loop.
   SYNTAX: void glutMainLoop(void);
   glutMainLoop();

<u>3.6 Text Displaying Functions:-</u>

1. **GlRasterPos:-**glRasterPos specifies the raster position for pixel operations.
   Parameters
   x,y, z, w
   Specify the x, y, z, and w object coordinates (if present) for the raster position.
2. **glutBitmapCharacter:-**
   glutBitmapCharacter renders a bitmap character using OpenGL.
   *SYNTAX: void glutBitmapCharacter(void *font, int character);*
   The available fonts are:
   GLUT_BITMAP_8_BY_13
   GLUT_BITMAP_9_BY_15
   GLUT_BITMAP_TIMES_ROMAN_10
   GLUT_BITMAP_TIMES_ROMAN_24

3.7 Interactive Functions:-

glutKeyboardfunc Function:-Registers the keyboard callback function func. The callback function returns the ASCII code of the key pressed and the position of the mouse.

*SYNTAX: void glutKeyboardFunc (void (*func) (char key, int x, int y));*

CG Concepts Used

We have used various CG concepts to complete this project some of them are mentioned below:-

1. **2D transformation:**- The purpose of using computers for drawing is to provide facility to the user to view the object from different angles, enlarging or reducing the scale or shape of object called as Transformation.

   Types of transformation

   i. Translation:- It is the straight-line movement of an object from one position to another is called Translation. Here the object is positioned from one coordinate location to another.

   ii. Scaling:- It is used to alter or change the size of objects. The change is done using scaling factors.

   iii. Rotating:- It is a process of changing the angle of the object. Rotation can be clockwise or anticlockwise.

   iv. Reflecting:- It is a transformation that produces a mirror image of an object. The mirror image can be either on the x-axis or y-axis. The object is rotated by 180°.

   v. Shearing:- It is a transformation that changes the shape of an object. The sliding of layers of objects occurs. The shear can be in one direction or in two directions.

Although in our application we've majorly used translation and scaling only.

2. **2D viewing**:- The method of selecting and enlarging a portion of a drawing is called windowing. The area chosen for this display is called a window. The window is selected by world-coordinate.

   Viewport: An area on a display device to which a window is mapped [where it is to be displayed].

   Basically, the window is an area in object space. It encloses the object. After the user selects this, space is mapped on the whole area of the viewport. Almost all 2D and 3D graphics packages provide means of defining viewport size on the screen. It is possible to determine many viewports on different areas of display and view the same object in a different angle in each viewport.

3. **Animation**:- Animation refers to the movement on the screen of the display device created by displaying a sequence of still images. Animation is the technique of designing, drawing, making layouts, and preparation of photographic series which are integrated into multimedia and gaming products. Animation connects the exploitation and management of still images to generate the illusion of movement. A person who creates animations is called an animator. He/she uses various computer technologies to capture the pictures and then to animate these in the desired sequence.

   Animation includes all the visual changes on the screen of display devices. These are

   i. Change of shape

   ii. Change in size

   iii. Change in color

   iv. Change in structure

   v. Change in angle

User-Defined Functions

Starting from the header file “glut.h” our program consists of various functions.

4.1 User Defined functions:-

There are the following user-defined functions:

- **void drawScore(char* text)**:- This function is used for showing the score of the player which is at the top of the screen.
- **void drawInstructions(const char* text):-** This function is used to show the instructions for the game at the top of the display.
- **void drawTitleText(const char* text):-** This function is printing **"Developed by Bhavya, Prince and Simran"** on the bottom of the screen.
- **void drawGameOverText()**:- This function is used to show the “**Game over**” string when the plane will collapse with any obstacle.
- **void keyboard(char key, int x, int y) :-** This function is used to accept the keys required to move the plane, change the mode of the game and restart the game.
- **void plane():-** This function is used to create the plane. It is defining the shape, size, and color of the plane.
- **void obstacle():-** This function is used to create the blocks or obstacles. It is defining the shape, scale, and color of the obstacles.
- **void myDisplay(void) :-** This function is used to show objects which, is paper plane and obstacles, string which is “Press L for Lower mode, H for Higher mode and R for Normal Mode” and score on the player's display.
- **void myInit(void) :-** This function is setting the colour of the display.

4.2 Inbuilt functions:-

There are some inbuilt functions used in the program. These inbuilt functions are part of the "glut.h" header file.

- **glutInitDisplayMode():-** The main function of this function is to **specify the type of display mode when creating a window**. The function prototype is: **void glutInitDisplayMode(unsigned int mode);**

- **glutInitWindowSize():-** This function Requests future windows to open at a given width/height. The function prototype is **void glutInitWindowSize(int width, int height);**

- **glutInitWindowPosition():-** This function is used to set the initial window position and size respectively. The function prototype is **void glutInitWindowPosition(int x, int y);**

- **glutCreateWindow():-** This function creates a top-level window. The name will be provided to the window system as the window's name. The intent is that the window system will label the window with the name. Implicitly, the *current window* is set to the newly-created window.

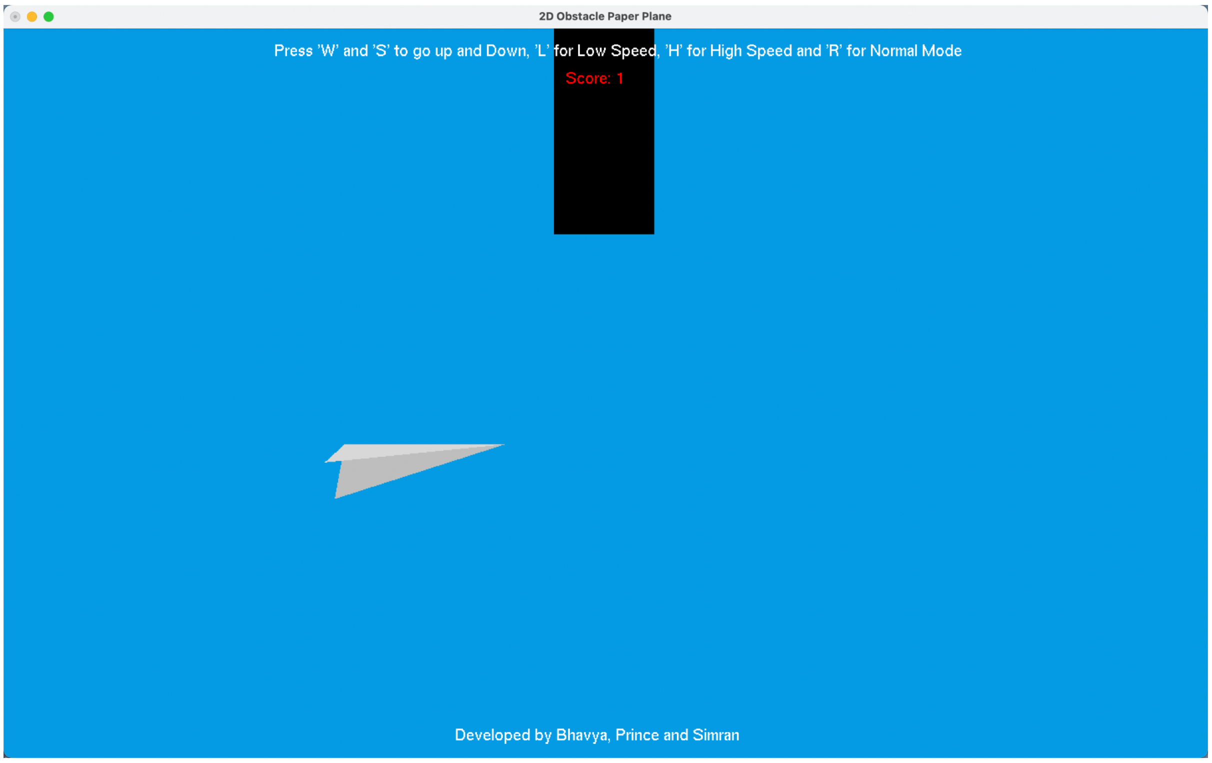
- **glutDisplayFunc():-** This function sets the display callback for the *current window*. When GLUT determines that the normal plane for the window needs to be redisplayed, the display callback for the window is called. Before the callback, the *current window* is set to the window needing to be redisplayed and (if no overlay display callback is registered) the *layer in use* is set to the normal plane. The display callback is called with no parameters. The entire normal plane region should be redisplayed in response to the callback (this includes ancillary buffers if your program depends on their state).

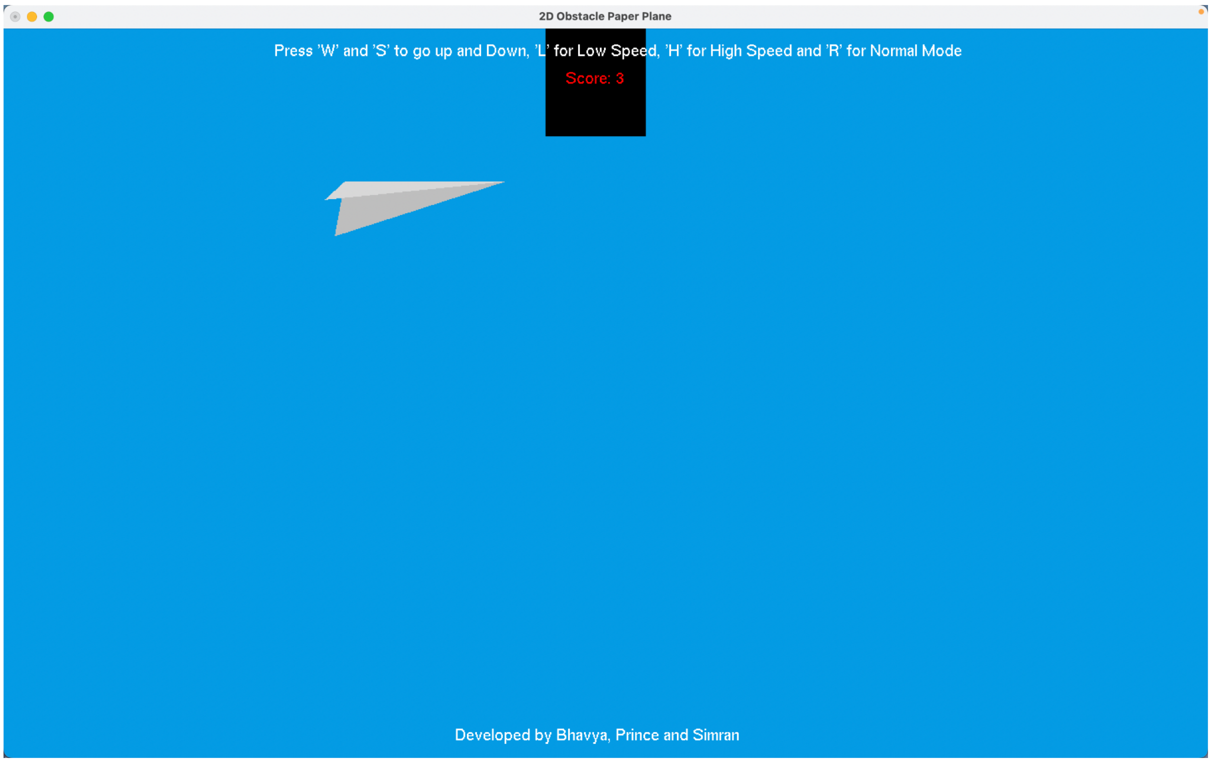
- **glutKeyboardFunc() :-** This function sets the keyboard callback for the *current window*. When a user types into the window, each key press generating an ASCII character will generate a keyboard callback. The key callback parameter is the generated ASCII character. The state of modifier keys such as Shift cannot be determined directly; their only effect will be on the returned ASCII data. The x and y callback parameters indicate the mouse location in window relative coordinates when the key was pressed. When a new window is created, no keyboard callback is initially registered, and ASCII keystrokes in the window are ignored. Passing NULL to glutKeyboardFunc disables the generation of keyboard callbacks.

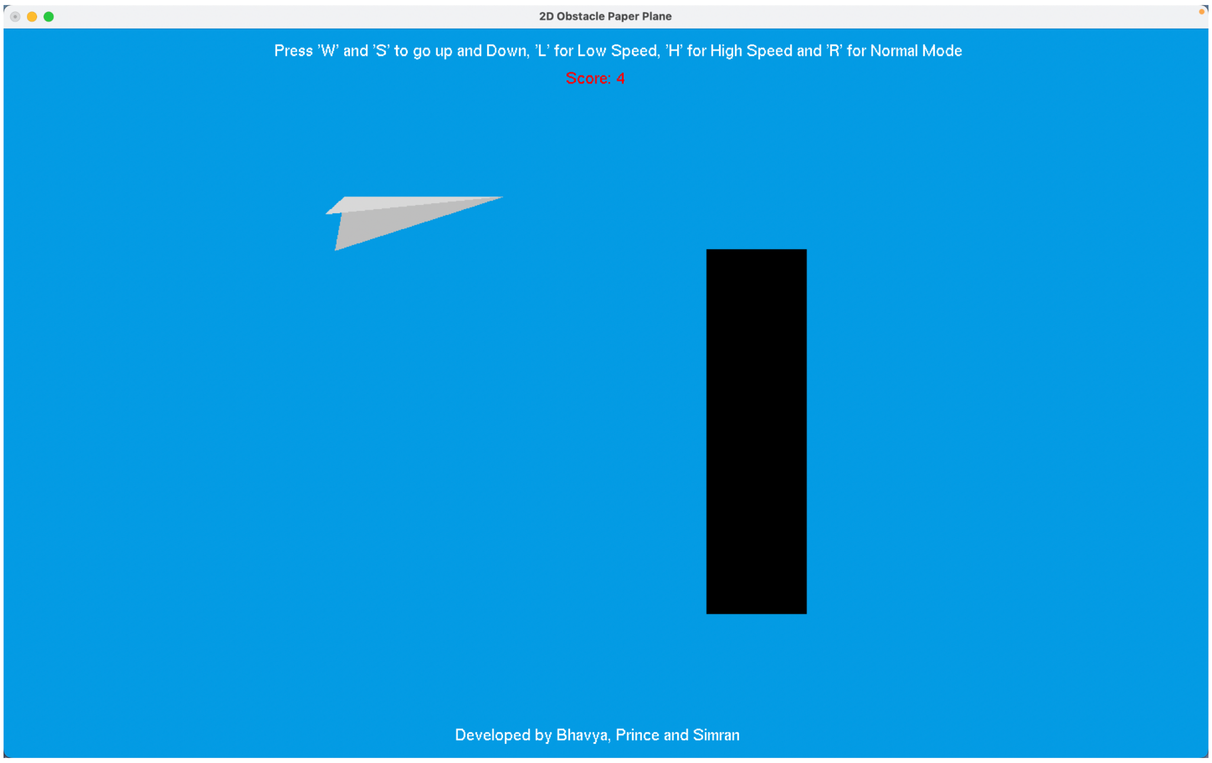
- **glutMainLoop() :-** This function enters the GLUT event processing loop. This routine should be called at most once in a GLUT program. Once called, this routine will never return. It will call as necessary any callbacks that have been registered.

Proof of Concept

You can find the source code at this link https://github.com/bhavyatyagi/2D-Obstacle-Paper-Plane.git. Hereby we are attaching a few screenshots as proof of concept. You can clone and run the code using the repository link to set it up locally.


2D Obstacle Paper Plane
Press 'W' and 'S' to go up and Down, 'L' for Low Speed, 'H' for High Speed and 'R' for Normal Mode
Score: 1
Developed by Bhavya, Prince and Simran


2D Obstacle Paper Plane
Press 'W' and 'S' to go up and Down, 'L' for Low Speed, 'H' for High Speed and 'R' for Normal Mode
Score: 3
Developed by Bhavya, Prince and Simran


2D Obstacle Paper Plane

Press ’W’ and ’S’ to go up and Down, ’L’ for Low Speed, ’H’ for High Speed and ’R’ for Normal Mode

Score: 4

Developed by Bhavya, Prince and Simran

Source Code

```
// ----------------------------- HEADER FILES STARTS --------------------------------
#include<iostream>
#include<stdio.h>
#include<string>
#include<math.h>
#include <GLUT/glut.h>
using namespace std;
// ----------------------------- HEADER FILES ENDS -----------------------------------

// ----------------------------- CONSTANTS STARTS ------------------------------------
// VARIABLE INITIALIZATIONS AND DECLRATIONS
int xSpeed = 12, ySpeed = 6; // Variables to adjust speed/translation coord in different
modes
std::string scoring; // Score String Global
int score = 0; // Score
int gameEndFlag = 0; // flag variable to check wheather the game is ended or still
playing
int x = 0; // Score x point
int y = 0; // Score y point
int px = 0; // Plane x coord0
int px1 = 0; // Plane x coord1
int px2 = 0; // Plane x coord2
int px3 = 0; // Plane x coord3
int py = 0; // Plane y coord0
int pyvmax = 0; // Plane y coord max
int pyvmin = 0; // Plane y coord min
int py1 = 0; // Plane y coord1
int py2 = 0; // Plane y coord2
int py3 = 0; // Plane y coord3
int bx = 0; // Obstacle x coord
int by = 0; // Obstacle y coord
int bxmax = 0; // Obstacle x coord max
int bxmin = 0; // Ostacle x coord min
int bymax = 0; // Ostacle y coord max
int bymin = 0; // Ostacle y coord min
// ----------------------------- CONSTANTS ENDS --------------------------------------

// ----------------------------- HELPER FUNCTIONS STARTS -----------------------------

void drawScore(const char* text) {
   int x = 280, y = 370; // defining score coord
```

```
    glColor3f(0.95, 0, 0); // Score String color to shade of red
    glRasterPos2f(x, y); // defining raster position for pixel operations
    while (*text) { // for each text character create a glut Bitmap Character to show on
screen
        glutBitmapCharacter(GLUT_BITMAP_HELVETICA_18, *text);
        text++;
    }
}
void drawInstructions(const char* text) {
    int x = 135, y = 385; // instruction string coord
    glColor3f(1, 1, 1); // set color to white
    glRasterPos2f(x, y); // defining raster position for pixel operations [raster positions
are maintain with subpixel accuracy hence required for us when objects overlaps]
    while (*text) { // for each text character create a glut Bitmap Character to show on
screen
        glutBitmapCharacter(GLUT_BITMAP_HELVETICA_18, *text);
        text++;
    }
}
void drawTitleText(const char* text) {
    int x = 225, y = 10; // title text coord
    glColor3f(1, 1, 1); // set color to white
    glRasterPos2f(x, y); // defining raster position for pixel operations
    while (*text) { // for each text character create a glut Bitmap Character to show on
screen
        glutBitmapCharacter(GLUT_BITMAP_HELVETICA_18, *text);
        text++;
    }
}
void drawGameOverText() {
    glColor3f(0.95, 0, 0); // set color to shade of red
    std::string str;
    str = "Game Over !";
    const char* text = str.data(); // retriving string data from string and typecasting to
char*
    int x = 280, y = 230; // coord
    glRasterPos2f(x, y);
    while (*text) {
        glutBitmapCharacter(GLUT_BITMAP_HELVETICA_18, *text);
        text++;
    }

}
void keyboard(unsigned char key, int x, int y)
```

```
{
  //    cout<<(key);
  if (key == 'w') // move up key for plane, move left for obstacle
  {
    // base cases -> If fail then endTheGame
    if (py1 > bymin && py1<bymax && px1>bxmin && px1 < bxmax)
    {
      bx = bx;
      by = by;
      gameEndFlag++;
    }
    else if (py2 > bymin && py2<bymax && px2>bxmin && px2 < bxmax)
    {
      bx = bx;
      by = by;
      gameEndFlag++;
    }
    else if (py3 > bymin && py3<bymax && px3>bxmin && px3 < bxmax)
    {
      bx = bx;
      by = by;
      gameEndFlag++;
    }
    else if (pyvmax < 365)
    { // if game is still playing then adjust according to curSpeeds
      py = py + ySpeed;
      bx = bx - xSpeed;
      if (bx < -600)
      {
        bx = 0;
        score++;
        by = rand() % 365; // random obstacle height
      }
    }
    else if (bx < -600)
    {
      bx = 0;
      score++;
      by = rand() % 365; // random obstacle height
    }
    else {
      bx = bx - xSpeed;
    }
    glutPostRedisplay(); // marks the current window as needing to be redisplayed.
}
else if (key == 's')
{
    // base cases -> If fail then endTheGame
    if (py1 > bymin && py1<bymax && px1>bxmin && px1 < bxmax)
    {
        bx = bx;
        by = by;
        gameEndFlag++;
    }
    else if (py2 > bymin && py2<bymax && px2>bxmin && px2 < bxmax)
    {
        bx = bx;
        by = by;
        gameEndFlag++;
    }
    else if (py3 > bymin && py3<bymax && px3>bxmin && px3 < bxmax)
    {
        bx = bx;
        by = by;
        gameEndFlag++;
    }
    else if (pyvmin > 0)
    {
        py = py - ySpeed;
        bx = bx - xSpeed;
        if (bx < -600)
        {
            bx = 0;
            score++;
            by = rand() % 365;
        }
    }
    else if (bx < -600)
    {
        bx = 0;
        score++;
        by = rand() % 365;
    }
    else {
        bx = bx - xSpeed;
    }
    glutPostRedisplay(); // marks the current window as needing to be redisplayed.
}
```

```
    else if (key == 'r'||key=='n')
    {
        // if restart or normal is hit then reinitiailize everything to 0 and set speeds
        py = 0;
        bx = 0;
        by = 0;
        score = 0;
        xSpeed = 12;
        ySpeed = 6;
        glutPostRedisplay(); // marks the current window as needing to be redisplayed.
    }
    else if (key == 'l')
    {
        // if low mode is hit then reinitiailize everything to 0 and set speeds to low
        py = 0;
        bx = 0;
        by = 0;
        score = 0;
        xSpeed = 8;
        ySpeed = 2;
        glutPostRedisplay(); // marks the current window as needing to be redisplayed.
    }
    else if (key == 'h')
    {
        // if high mode is hit then reinitiailize everything to 0 and set speeds to high
        py = 0;
        bx = 0;
        by = 0;
        score = 0; xSpeed = 20;
        ySpeed = 8;
        glutPostRedisplay(); // marks the current window as needing to be redisplayed.
    }
}
void plane()
{
    // re-initialising positions of plane according to key hit
    px1 = 250 + px;
    py1 = 340 + py;
    pyvmax = py1;
    pyvmin = 310 + py;
    px2 = 165 + px;
    py2 = pyvmin;
    px3 = 170 + px;
    py3 = py1;
    glColor3f(0.747, 0.747, 0.747); // first triangle color
    glBegin(GL_POLYGON); // plotting triangle first
    glVertex2i(px2, pyvmin);
    glVertex2i(px1, py1);
    glVertex2i(170 + px, 340 + py);
    glEnd();
    glColor3f(0.847, 0.847, 0.847); // second triangle color
    glBegin(GL_POLYGON); // plotting triangle second
    glVertex2i(160 + px, 330 + py);
    glVertex2i(250 + px, 340 + py);
    glVertex2i(px3, 340 + py);
    glEnd();
}
void obstacle()
{
    // re-initialising positions of obstacle according to key hit
    bxmax = 600 + bx;
    bxmin = 550 + bx;
    bymax = 200 + by;
    bymin = 0 + by;
    glColor3f(0.0, 0.0, 0.0); // black color obstacle
    glBegin(GL_POLYGON); // plot 4 points of a rectangle
    glVertex2i(bxmin, bymin);
    glVertex2i(bxmax, bymin);
    glVertex2i(600 + bx, bymax);
    glVertex2i(550 + bx, bymax);
    glEnd();
}
void myDisplay(void)
{
    glClear(GL_COLOR_BUFFER_BIT);

    plane(); // calling to create plane
    obstacle(); // calling to create obstacle
    string title="Developed by Bhavya, Prince and Simran";
    string instructions="Press 'W' and 'S' to go up and Down, 'L' for Low Speed, 'H' for
High Speed and 'R' for Normal Mode";
    drawTitleText(title.data());
    drawInstructions(instructions.data());
    scoring = "Score: " + std::to_string(score);
    drawScore(scoring.data());
    glEnd();
    if (gameEndFlag > 0) { // checking if positive then end game
        drawGameOverText();
        gameEndFlag = 0;
    }
    glFlush();
}
void myInit(void)
{
    glClearColor(0.27f,0.6f,0.87f, 1.0f);
    glColor3f(0.0f, 0.0f, 1.0f);
    glPointSize(4.0);
    glMatrixMode(GL_PROJECTION); // set matrix mode to GL_PROJECTION
    glLoadIdentity(); // set proj matrix to I
    gluOrtho2D(0.0, 600.0, 0.0, 400.0);
}
// ---------------------------- HELPER FUNCTIONS EDNS --------------------------------

// ---------------------------- MAIN PROGRAM STARTS ----------------------------------
int main(int argc, char** argv)
{
    glutInit(&argc, argv);
    glutInitDisplayMode(GLUT_SINGLE | GLUT_RGB);
    glutInitWindowSize(1980, 1080); // set intiial window size
    glutInitWindowPosition(10, 10); // set window position
    glutCreateWindow("2D Obstacle Paper Plane"); // Window Title Text
    glutDisplayFunc(myDisplay); // main display function
    glutKeyboardFunc(keyboard); // keyboard function to take inputs
    myInit(); // rest initializations
    glutMainLoop(); // for further executing methods
    return 0;
}
// ---------------------------- MAIN PROGRAM ENDS -----------------------------------
```

Conclusions

The **"2D Obstacle Paper Plane"** project was successfully finished on the Windows platform using C++. It uses various translating, scaling, and rotating techniques to create a simple application game. The utilities and functions provided by the OpenGL have been used in an optimal way to achieve a complete working project.

During the project's completion, the work we did has helped us better understand the many functions available more practically. We have realized that the OpenGL platform is a great game development launchpad. As a result, it is helpful in the development of numerous games. In and of itself, OpenGL is useful for low-cost and easy game creation. It's an excellent place to start if you want to branch out into other areas of computer graphics design and applications.

Working on this project has helped us learn the fundamentals of 2D animation, transformation, modeling, and gaming, the following significant things. This game is quite simple and joyful to play. This project was developed as a Lab Component part of UCS505 Computer Graphics for the year 2022, semester 6th, Computer Science and Engineering.

References

[1]: https://docs.microsoft.com/en-us/previous-versions/dd318362(v=vs.85)

[2]: https://kwpowers.medium.com/loading-scenes-in-unity-def40d45276a

[3]: https://docs.microsoft.com/en-us/windows/win32/opengl/glmatrixmode

[4]: https://vtunotesinfo.blogspot.com/2014/02/v-behaviorurldefaultvmlo.html

[5]: http://www.cs.uccs.edu/~ssemwal/man.html

[6]:
https://www.mcqhubb.com/cad-cam-and-automation/allows-the-user-to-view-the-object-from-different-angles-526me/

[7]: https://maxwell.ict.griffith.edu.au/1304ENG/Lecture/Appendix_G.pdf

[8]: https://sarata.com/manpages/glutKeyboardUpFunc.3.html

[9]: http://www.globalschoolnet.org/gsncf/narrative_view.cfm?narrid=1357