Chapter 1

INTRODUCTION

1.1 Overview of Computer Graphics

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

Although early application in engineering & science had to relay on expensive & cumbersome equipments, advances in computer technology have made interactive computer graphics a practical tool. Today it found Computer Graphics in a diverse area such as science, engineering, medicine, business, industry, government, art, entertainment, education and training.

1.2 History

William fetter was credited with coining the term Computer Graphics in 1960, to describe his work at Boeng. 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.

1.3 Applications of Computer Graphics

Nowadays Computer Graphics used in almost all the areas ranges from science, engineering, medicine, business, industry, government, art, entertainment, education and training.

- **CG in the field of CAD**

Computer Aided Design methods are routinely used in the design of buildings, automobiles, aircraft, watercraft, spacecraft computers, textiles and many other applications.

- **CG in presentation Graphics**

  Another major application area presentation graphics used to produce illustrations for reports or generate slides. Presentation graphics is commonly used to summarize financial, statistical, mathematical, scientific data for research reports and other types of reports.2D and 3D bar chart to illustrate some mathematical or statistical report.

- **CG in computer Art**

  CG methods are widely used in both fine art and commercial art applications. Artists use a variety of computer methods including special purpose hardware, artist's paintbrush program (lumena), other paint packages, desktop packages, maths packages, animation packages that provide facility for designing object motion. Ex: cartoons design is an example of computer art which uses CG.

- **Entertainment**

  Computer graphics methods are now commonly used in making motion pictures, music, videos, games and sounds. Sometimes graphics objects are combined with the actors and live scenes.

- **Education and Training**

  Computer generated models of physical financial, economic system is often as education aids. For some training application special systems are designed. Ex: specialized system is simulator for practice sessions or training of ship captain, aircraft pilots and traffic control.

- **Image Processing**

  Although the methods used in CG image processing overlap, the 2 areas are concerned with fundamentally different operations.

  In CG a computer is used to create picture. Image processing on the other hand applies techniques to modify existing pictures such as photo scans, TV scans.

- **User Interface**

It is common for software packages to provide a graphical interface. A major component of a graphical interface is a window manager that allows a user to display multiple window area. Interface also displays menus, icons for fast selection and processing.

1.4 Problem Statement

In this project the aim is to design and develop a graphical game namely "STUNT PLANE GAME" where the plane has to fly by crossing the obstacles in the path of the plane, the player has to control the plane upwards and downwards using the keyboard to avoid the collisions with obstacles.

1.5 Objective of the project

To implement the concepts of computer graphics that we have learnt by, providing the graphical view of paper plane and rectangular blocks moving it in between the obstacles in order to get more score as possible.

1.6 Organization of the Report

This section deals with the Introduction and organization of the project report. Chapter 2 discusses the basic concepts OpenGL. Chapter 3 gives information about the design and implementation and the results of the project. Chapter 4 shows the result and snapshot. Chapter 5 gives the conclusion and future enhancement of the project.

Chapter 2

INTRODUCTION TO OPENGL

2.1 Introduction

OpenGL is a software interface to graphics hardware. This interface consists of about 150 distinct commands that you use to specify the objects and operations needed to produce interactive three-dimensional applications. OpenGL is designed as a streamlined, hardware-independent interface to be implemented on many different hardware platforms.

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

To interface with the window system and to get input from external devices into our programs, we need at least one more library. For each major window system there is a system-specific library that provides the “glu” between the window system and OpenGL. For the X window system, this library is called GLX, for windows, it is wgl, and for the Macintosh, it is a gl. Rather than using a different library for each system, we use a readily available library called the OpenGL Utility Toolkit (GLUT), which provides the minimum functionality that should be expected in any modern windowing system.

Fig.2.1 shows the organization of the libraries for an X Window System environment. For this window system, GLUT will use GLX and the X libraries. The application program, however, can use only GLUT functions and thus can be recompiled with the GLUT library for other window systems.Library organization of the OpenGL can be illustrated with the help of a figure which specifies various library functions supported by OpenGL

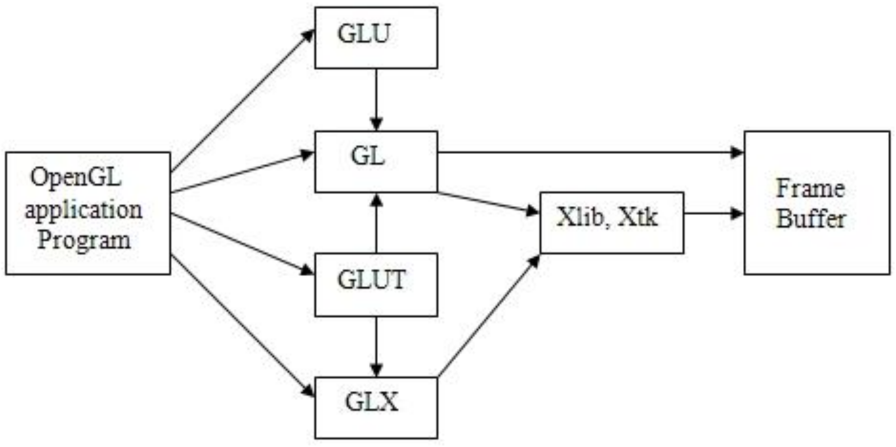


Fig 2.1Library organization of OpenGL

2.2 OpenGL Command Syntax

OpenGL commands use the prefix **gl** and initial capital letters for each word making up the command name. Similarly, OpenGL defined constants begin with GL_, use all capital letters, and use underscores to separate words (like GL_TRUE).

Some extraneous letters are appended to some command names (for example, the 3f in glColor3f() and glVertex3f()). It's true that the Color part of the command name glColor3f() is enough to define the command as one that sets the current color. However, more than one such command has been defined so as to use different types of arguments. In particular, the 3 part of the suffix indicates that three arguments are given; another version of the Color command takes four arguments. The part of the suffix indicates that the arguments are floating-point numbers.

2.3 OpenGL as a State Machine

OpenGL is a state machine. It is put into various states (or modes) that then remain in effect until it is changed. The current color is a state variable. The current colour can be set to white, red, or any other color, and thereafter every object is drawn with that color until the current color is set to something else.

Many state variables refer to modes that are enabled or disabled with the command glEnable() or glDisable(). Each state variable or mode has a default value, and at any point the system b can be queried for each variable's current value.

2.4 Pixel Operations

Pixels from an array in system memory are first unpacked from one of a variety of formats into the proper number of components. Next the data is scaled, biased, and processed by a pixel map. The results are clamped and then either written into texture memory or sent to the Rasterization step. If pixel data is read from the frame buffer, pixel-transfer operations (scale, bias, mapping, and clamping) are performed. Then these results are packed into an appropriate format and returned to an array in system memory.

There are special pixel copy operations to copy data in the frame buffer to other parts of the frame buffer or to the texture memory. A single pass is made through the pixel transfer operations before the data is written to the texture memory or back to the frame buffer.

2.5 Texture Assembly

An OpenGL application may wish to apply texture images onto geometric objects to make them look more realistic. Some OpenGL implementations may have special resources to accelerate texture performance. There may be specialized, high-performance texture memory.

2.6 Rasterization

Rasterization is the conversion of both geometric and pixel data into fragments. Each fragment square corresponds to a pixel in the frame buffer. Line and polygon stipples, line width, point size, shading model, and coverage calculations to support antialiasing are taken into consideration as vertices are connected into lines or the interior pixels are calculated for a filled polygon. Color and depth values are assigned for each fragment square.

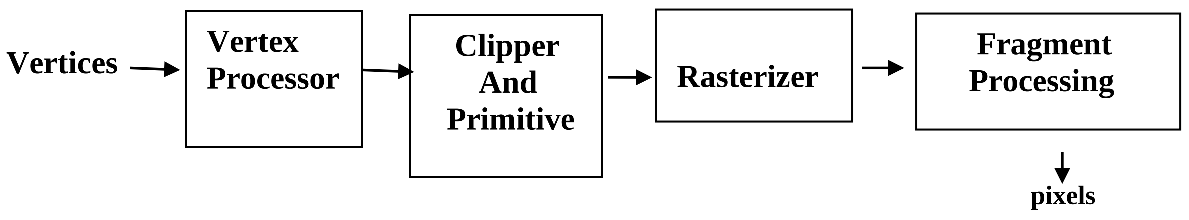


Figure 2.2 Block diagram showing Rasterisation

2.7 Immediate Mode and Display Lists

All data, whether it describes geometry or pixels, can be saved in a display listfor current or later use. When a display list is executed, the retained data is sent from the display list just as if it were sent by the application in immediate mode.

- **Transforming to Window Coordinates**

Before clip coordinates can be converted to windowcoordinates, they are normalized by dividing by the value of `w` to yield normalized device coordinates. After that, the viewport transformation applied to these normalized coordinates produces window coordinates. You control the viewport, which determines the area of the on-screen window that displays an image, with `glDepthRange()` and `glViewport()`.

- **Matrix Transformations**

Vertices and normals are transformed by the modelview and projection matrices before they're used to produce an image in the frame buffer. You can use commands such as `glMatrixMode ()`, `glMultMatrix ()`, `glRotate ()`, `glTranslate ()`, and `glScale()` to compose the desired transformations, or you can directly specify matrices with `glLoadMatrix()` and `glLoadIdentity()`. Use `glPushMatrix ()` and `glPopMatrix ()` to save and restore modelview and projection matrices on their respective stacks.

The basic model for OpenGL command interpretation is immediate mode, in which a command is executed as soon as the server receives it; vertex processing, for example may begin even before specification of the primitive of which it is a part has been completed. Immediate mode execution is well-suited to interactive applications in which primitives and modes are constantly altered. In OpenGL, the fine-grained control provided by immediate mode is taken as far as possible: even individual lighting parameters (the diffuse reflectance color of a material, for instance) and texture images are set with individual commands that have immediate effect. While immediate mode provides flexibility, its use can be inefficient if unchanging parameters or objects must be re-specified. To accommodate such situations, OpenGL provides display lists. A display list encapsulates a sequence of OpenGL commands and is stored on the server. The display list is given a numeric name by the application when it is specified; the application need only name the display list to cause the server to effectively execute all the commands contained within the list. This mechanism provides a straightforward, effective means for an application to transmit a group of commands to the server just once even when those same commands must be executed many times.

2.7.1 Display List Optimization

Accumulating commands into a group for repeated execution presents possibilities for optimization. Consider, for example, specifying a texture image. Texture images are often large, requiring a large, and therefore possibly slow, data transfer from client to server (or from the server to its graphics subsystem) whenever the image is re-specified. For this reason, some graphics subsystems are equipped with sufficient storage to hold several texture images simultaneously. If the texture image definition is placed in a display list, then the server may be able to load that image just once when it is specified. When the display list is invoked (or re-invoked), the server simply indicates to the graphics subsystem that it should use the texture image already present in its memory, thus avoiding the overhead of re-specifying the entire image. Examples like this one indicate that display list optimization is required to achieve the best performance. In the case of texture image loading, the server is expected to recognize that a display list contains texture image information and to use that information appropriately. It also places a burden on the application writer to know to use display lists in cases where doing so could improve performance.

2.8 Advantages of Using OpenGL

- **Industry standard:** An independent consortium, the OpenGL Architecture Review Board, guides the OpenGL specification. With broad industry support, OpenGL is the only truly open, vendor-neutral, multiplatform graphics standard.
- **Stable:** OpenGL implementations have been available for more than seven years on a wide variety of platforms. Additions to the specification are well controlled, and proposed updates are announced in time for developers to adopt changes. Backward compatibility requirements ensure that existing applications do not become obsolete.
- **Reliable and portable:** All OpenGL applications produce consistent visual display results on any OpenGL API-compliant hardware, regardless of operating system or windowing system.
- **Evolving**: Because of its thorough and forward-looking design, OpenGL allows new hardware innovations to be accessible through the API via the OpenGL extension mechanism. In this way, innovations appear in the API in a timely fashion, letting application developers and hardware vendors incorporate new features into their normal product release cycles.

CHAPTER 3

STUNT PLANE GAME

3.1 History

Games are an integral part of all societies. Like work and relationships, they are an expression of some basic part of the human nature. Games are formalized expressions of play which allow people to go beyond immediate imagination. Games capture the ideas and behaviors of people at one period of time and carry that through time to their descendants.

Games like ludo, pagade, chennamane, etc illustrate the thinking of the people centuries ago. When archaeologists excavate an ancient society they find artifacts related to living, working, family and social activities, games often become an archival record of how individuals and groups played in earlier times. With the time the games also evolved to meet the latest technologies, as such we can see many electronic games nowadays like play stations, online flash games, etc.

3.2 Concept of the game

1. Game is very simple, it consists of plane and obstacles which are rectangular blocks.
2. The player will move the plane using keys and have to avoid collision with obstacles.
3. For crossing an obstacle the player will get a point.
4. If plane gets collide with any of the obstacle, game will be over.
5. Player will get retry option to play the game once again.

3.3 How to play

1. Press ‘a’ in order to start game.
2. Move plane with help of ‘w’ and ‘s’ keys.
3. Plane should not touch the blocks.
4. If plane got crash with the block then game over and player have chance to retry.

CHAPTER 4

DESIGN AND IMPLEMENTATION

The design and implementation part is consider with coding of the algorithms and analyzed portion of project.

The Game is presented by using various standard OpenGL and user defined functions. The following list gives an overview of the functions used and an implementation section defines how actually the project is implemented.

4.1 Graphic Functions and Requirements

4.1.1 Header Files

The Header files used in OpenGL are

•**GL**, for which the commands begin with GL;

•**GLUT**, the GL Utility Toolkit, opens windows, develops menus, and manages events.

4.1.2 Functions

void glBegin (glEnum mode)	Initiates a new primitive of type mode and starts the collection of vertices. Values of mode include GL_POINTS, GL_LINES and GL_POLYGON.
void glEnd ( )	Terminates a list of vertices.
void glColor3f (TYPE r, TYPE g,TYPE b)	Sets the present RGB colors. Valid types are int (i), float (f) and double (d). The maximum and minimum values of the floating-point types are 1.0 and 0.0, respectively.

void glClearColor (GLclampf r, GLclampf g, GLclampf b, Glclampf a)	Sets the present RGBA clear color used when clearing the color buffer. Variables of GLclampf are floating-point numbers between 0.0 and 1.0.
int glutCreateWindow (char *title)	Creates a window on the display. The string title can be used to label the window. The return value provides a reference to the window that can be used where there are multiple windows.
void glVertex2i(GLint x, GLint y)	This function commands are used within glBegin/glEnd pairs to specify point, line, and polygon vertices
void glutInitWindowSize (int width, int height)	Specifies the initial height and width of the window in pixels.
void glutInitWindowPosition (int x, inty)	Specifies the initial position of the top-left corner of the window in pixels.
void glutInitDisplayMode (unsigned int mode)	Request a display with the properties in mode. The value of mode is determined by the logical OR of operation including the color model (GLUT_RGB, GLUT_INDEX) and buffering (GLUT_SINGLE,GLUT_DOUBLE).
void glFlush ( )	Forces and buffers any OpenGL commands to execute.
void glutSwapBuffers(void)	Swaps the buffers of the current window if double buffered
void glutInit (int argc, char **argv)	Initializes GLUT. The arguments from main are passed in and can be used by the application.

void glutMainLoop ( )	Cause the program to enter an event processing loop. It appears at the end of main.
void glutDisplayFunc(void(*func) (void))	Registers the display function func that is executed when the window needs to be redrawn.
void glRasterPos2f(Glfloat x,Glfloat y)	Specifies raster position for pixel operations.
void gluOrtho2D (GLdouble left, GLdouble right, GLdouble bottom, GLdouble top)	Defines a two-dimensional viewing rectangle in the plane Z=0.
void glutBitmapCharacter(void *font, int char)	Renders the character with ASCII code char at the current raster position using the raster font given by font. Fonts include GLUT_BITMAP_TIMES_ROMAN_10 and GLUT_BITMAP_TIMES_ROMAN_8_Y_13. The raster position is incremented by the width of the character.
glClear(GL_COLOR_BUFFER_BIT)	To clear entire screen to background color.
void glutFullScreen(void)	To requests that the current window be made full screen.
void KeyboardFunc(KeyBoard)	It is used for the implementation of keyboard interface. Passing control to void keyboard(unsigned char key, int x, int y)

4.1.3 User-Defined Functions

void draw(float x, float y, const char* text)	This function is used to display text on screen
void fpage(void)	This function is used to display introductory page.
void menu()	This displays menu on the screen.
void inst()	Display the instructions of the game.
void drawGameOverText()	Display gameover text when plane touches obstacles.
void keyboard(unsigned char key, int x, int y)	Defined to handle keyboard inputs.
void plane ()	This function is defined to draw the plane on the screen.
void box()	This function is defined to draw the obstacles.
void myDisplay(void)	This function is defined to handle the overall display of game.
void display(void)	This function is used to display front page and game interface.
void myInit(void)	This function is defined to initialize various OpenGL settings.
void main()	Execution begins from here.

4.2 Flowchart

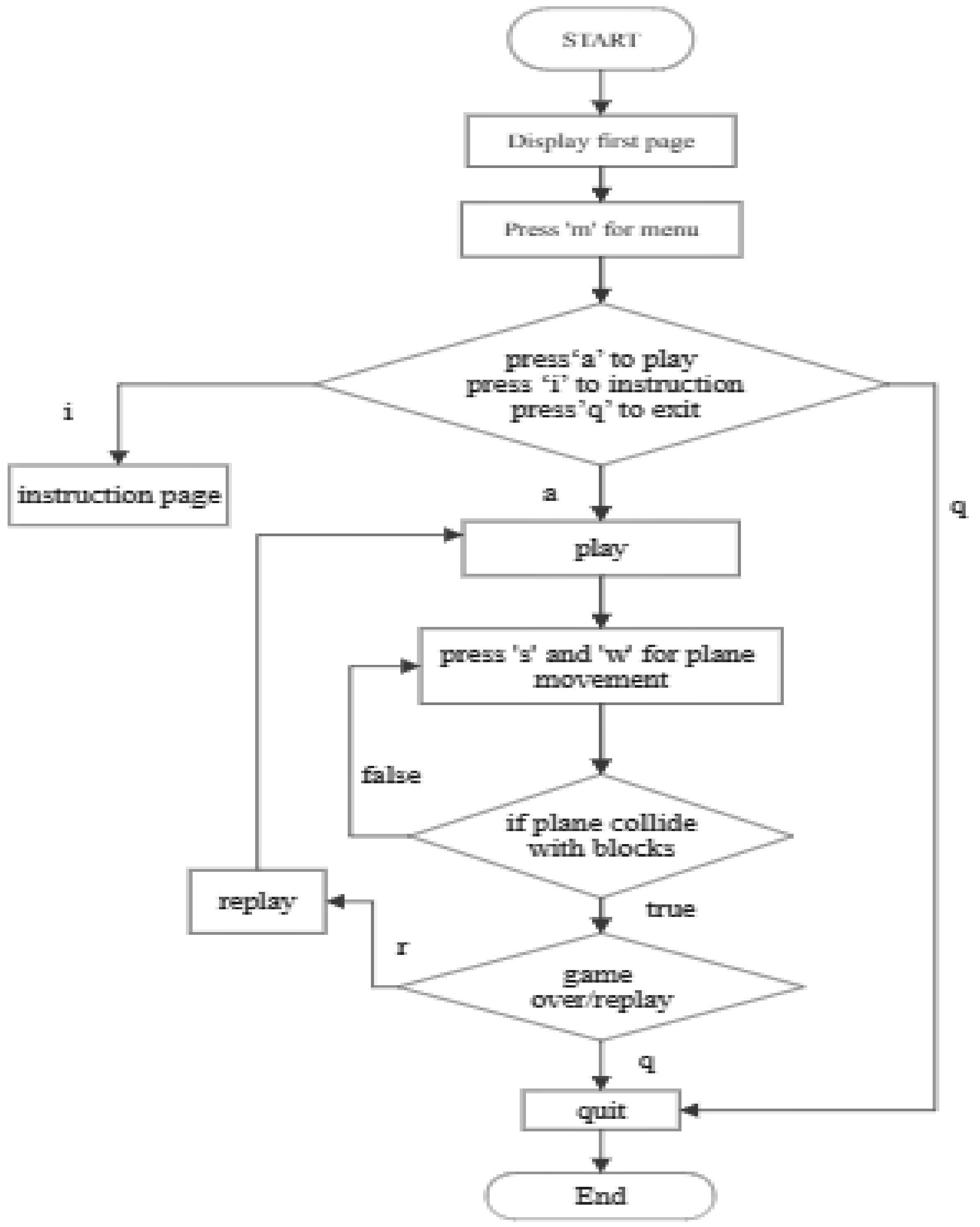


Fig 4.1 Flowchart

Shows how the flow of control will go in project.

4.3 Pseudo-code

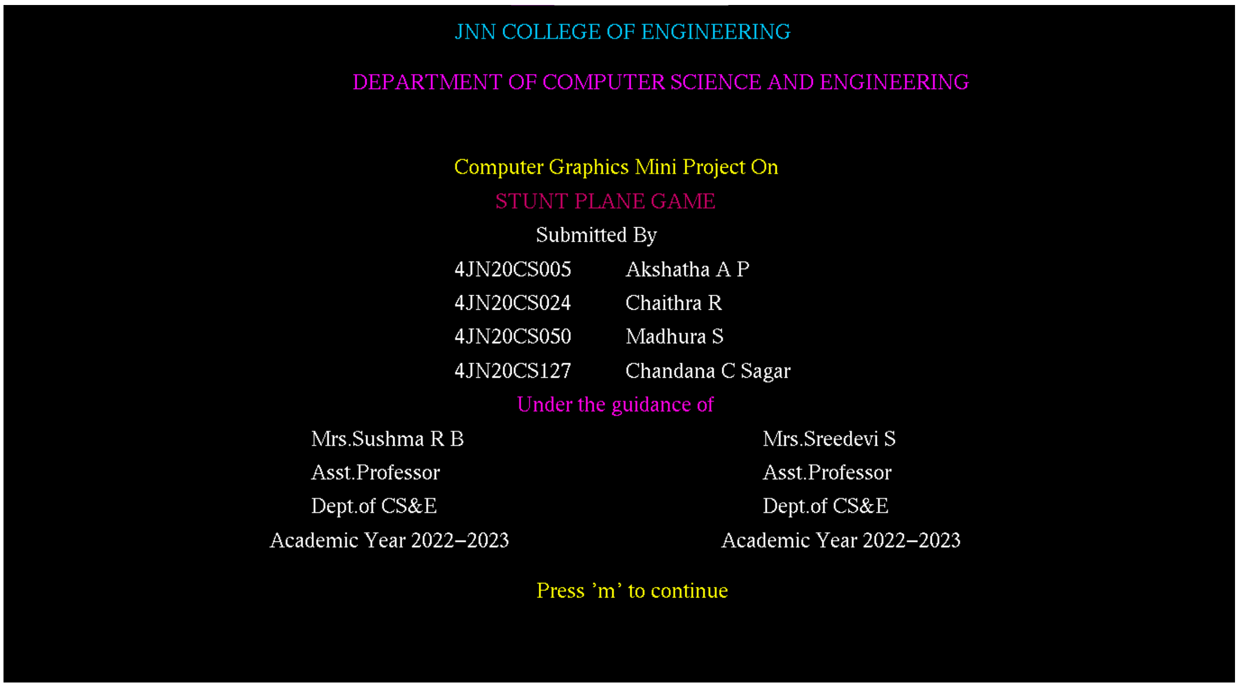
```
int main(int argc, char** argv)
{
        glutInit(&argc, argv);
        glutInitDisplayMode(GLUT_SINGLE | GLUT_RGB);
        glutInitWindowSize(700, 500);
        glutInitWindowPosition(10, 10);
        glutCreateWindow("Stunt plane");
        glutDisplayFunc(display);
        glutKeyboardFunc(keyboard);
        myInit();
        glutMainLoop();
}
void inst()
{       glClearColor(0.0, 0.0, 0.0, 0.0);
        glClear(GL_COLOR_BUFFER_BIT);
        glColor3f(1.0, 1.0, 0.0);
        draw(250, 300, "INSTRUCTIONS");
        glColor3f(1.0, 1.0, 1.2);
        draw(210, 240, "Press 'w' for the upward movement of plane");
        draw(210, 210, "Press 's' for the downward movement of plane");
        draw(210, 180, "Press 'r' to Retry the game");
        glColor3f(0.0, 0.8, 1.0);
        draw(230, 90, "Press  'a' to start the game");
        glFlush();
}
void keyboard(unsigned char key, int x, int y)
{
        if (key == 'a')
        {if (flag == 0)
                myDisplay();
                else {   flag = 0;
                      myDisplay();
        }}
        if (key == 'm')
                menu();
        if (key == 'i')
                inst();
        if (key == 'q')
                exit(0);
        if (key == 'w')
        {if (py1 > bymin && py1<bymax && px1>bxmin && px1 < bxmax)
                {       bx = bx;
                        by = by;
                        count_r++;
                }
                else if (py2 > bymin && py2<bymax && px2>bxmin && px2 < bxmax)
                {       bx = bx;
                        by = by;
                        count_r++;
                }
                else if (py3 > bymin && py3<bymax && px3>bxmin && px3 < bxmax)
                {       bx = bx;
                        by = by;
                        count_r++;
                }
                else if (pyvmax < 365)
                {
                        py = py + 2;
                        bx = bx - 8;
                        if (bx < -600)
                        {
                                bx = 0;
                                score++;
                                by = rand() % 365;
                        }
                        x++;}
                else if (bx < -600)
                {
                        bx = 0;
                        score++;
                        by = rand() % 365;
                        x++;
                }
                else {
                        bx = bx - 8;
                }
                glutPostRedisplay();
        }
        else if (key == 's')
        {
                if (py1 > bymin && py1<bymax && px1>bxmin && px1 < bxmax)
                {
                        bx = bx;
                        by = by;
                        count_r++;
                }
                else if (py2 > bymin && py2<bymax && px2>bxmin && px2 < bxmax)
                {
                        bx = bx;
                        by = by;
                        count_r++;
                }
                else if (py3 > bymin && py3<bymax && px3>bxmin && px3 < bxmax)
                {
                        bx = bx;
                        by = by;
                        count_r++;
                }
                else if (pyvmin > 0)
                {
```

```
                        py = py - 2;
                        bx = bx - 8;
                        if (bx < -600)
                        {           bx = 0;
                                    score++;
                                    by = rand() % 365;
                }           }
                else if (bx < -600)
                {           bx = 0;
                            score++;
                            by = rand() % 365;
                }
                else
                            bx = bx - 8;
                glutPostRedisplay();
        }
        else if (key == 'r')
        {           py = 0;
                    bx = 0;
                    by = 0;
                    score = 0;
                    glutPostRedisplay();
        }
}
void plane()
{
        px1 = 250 + px;
        py1 = 340 + py;
        pyvmax = py1;
        pyvmin = 310 + py;
        px2 = 165 + px;
        py2 = pyvmin;
        px3 = 170 + px;
        py3 = py1;
        //down triangle of plane
        glColor3f(0.747, 0.747, 0.747);
        glBegin(GL_POLYGON);
        glVertex2i(px2, pyvmin);
        glVertex2i(px1, py1);
        glVertex2i(170 + px, 340 + py);
        glEnd();
        //upper triangle of plane
        glColor3f(0.847, 0.847, 0.847);
        glBegin(GL_POLYGON);
        glVertex2i(160 + px, 330 + py);
        glVertex2i(250 + px, 340 + py);
        glVertex2i(px3, 340 + py);
        glEnd();
}
void box()
{       bxmax = 600 + bx;
        bxmin = 550 + bx;
        bymax = 200 + by;
        bymin = 0 + by;

        glColor3f(0.6123, 1.0, 0.6863);
        glBegin(GL_POLYGON);
        glVertex2i(bxmin, bymin);
        glVertex2i(bxmax, bymin);
        glVertex2i(600 + bx, bymax);
        glVertex2i(550 + bx, bymax);
        glEnd();
}
```

Chapter 5

RESULTS AND SNAPSHOTS



JNN COLLEGE OF ENGINEERING

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

Computer Graphics Mini Project On

STUNT PLANE GAME

Submitted By

4JN20CS005	Akshatha A P
4JN20CS024	Chaithra R
4JN20CS050	Madhura S
4JN20CS127	Chandana C Sagar

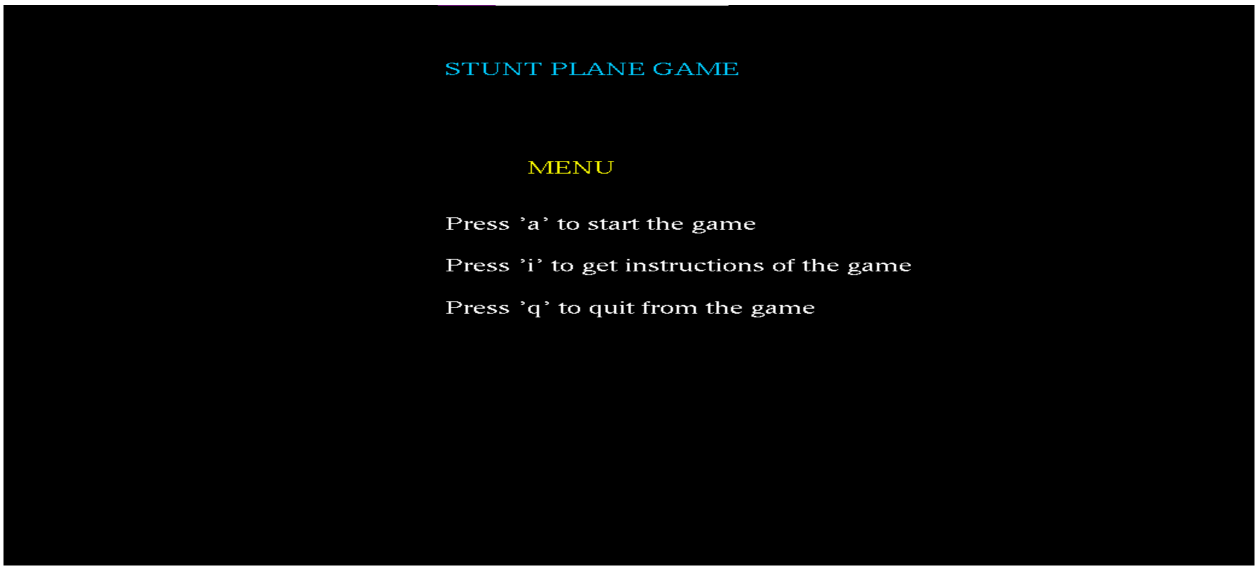
Under the guidance of

Mrs.Sushma R B	Mrs.Sreedevi S
Asst.Professor	Asst.Professor
Dept.of CS&E	Dept.of CS&E
Academic Year 2022–2023	Academic Year 2022–2023

Press 'm' to continue

Fig 5.1 Front screen of the game

This screen shot shows the front screen of the project.



STUNT PLANE GAME

MENU

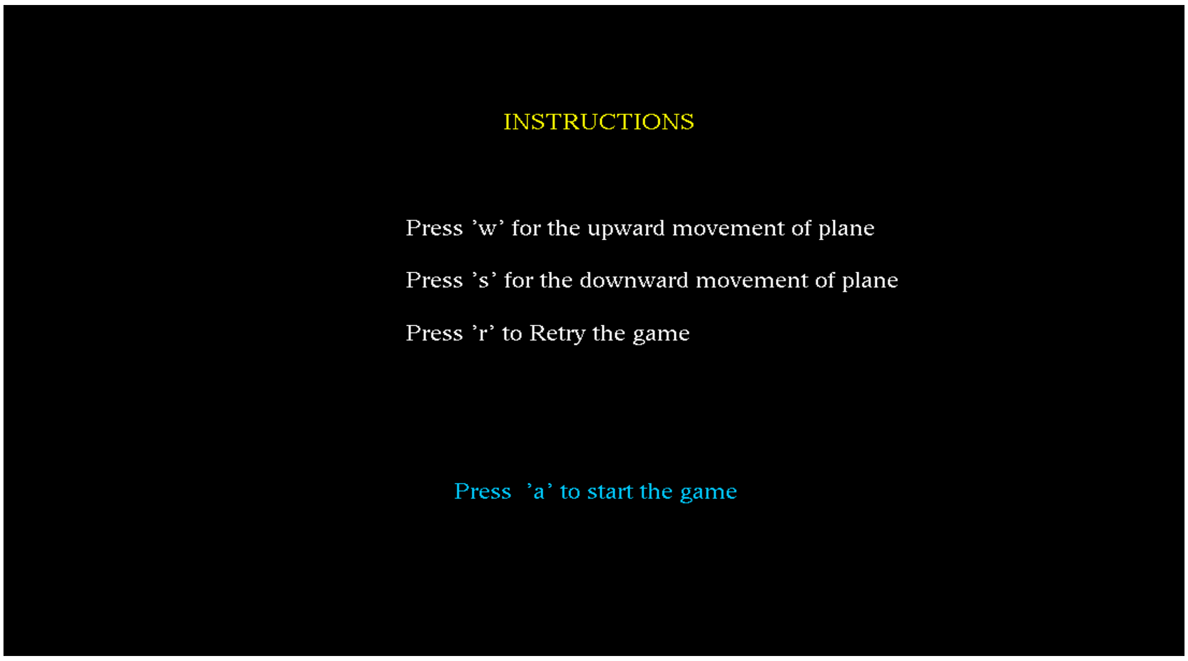
Press 'a' to start the game

Press 'i' to get instructions of the game

Press 'q' to quit from the game

Fig 5.2 Option screen of the game

This screen gives the options for user to start the game , to read the instruction how to play the game or to exit from the game.


INSTRUCTIONS

Press ’w’ for the upward movement of plane

Press ’s’ for the downward movement of plane

Press ’r’ to Retry the game

Press ’a’ to start the game

Fig. 5.3 Instructions page

This screen shows the instructions of the game.

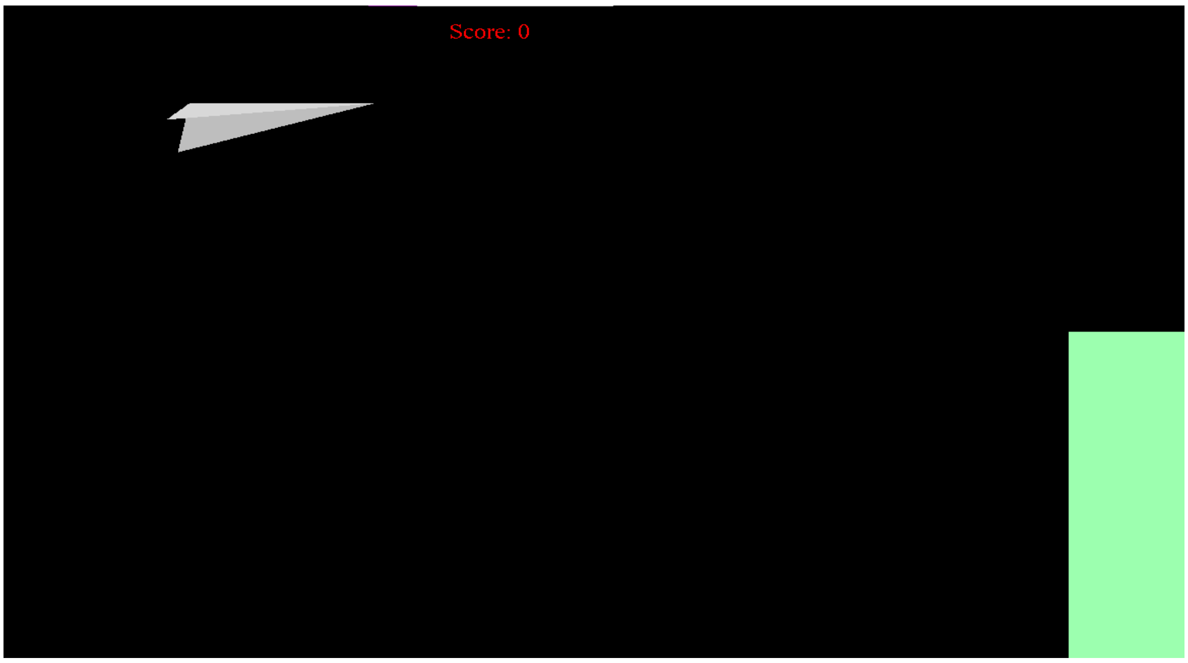


Fig 5.4 Start of game

This screen shows the initial state of game.

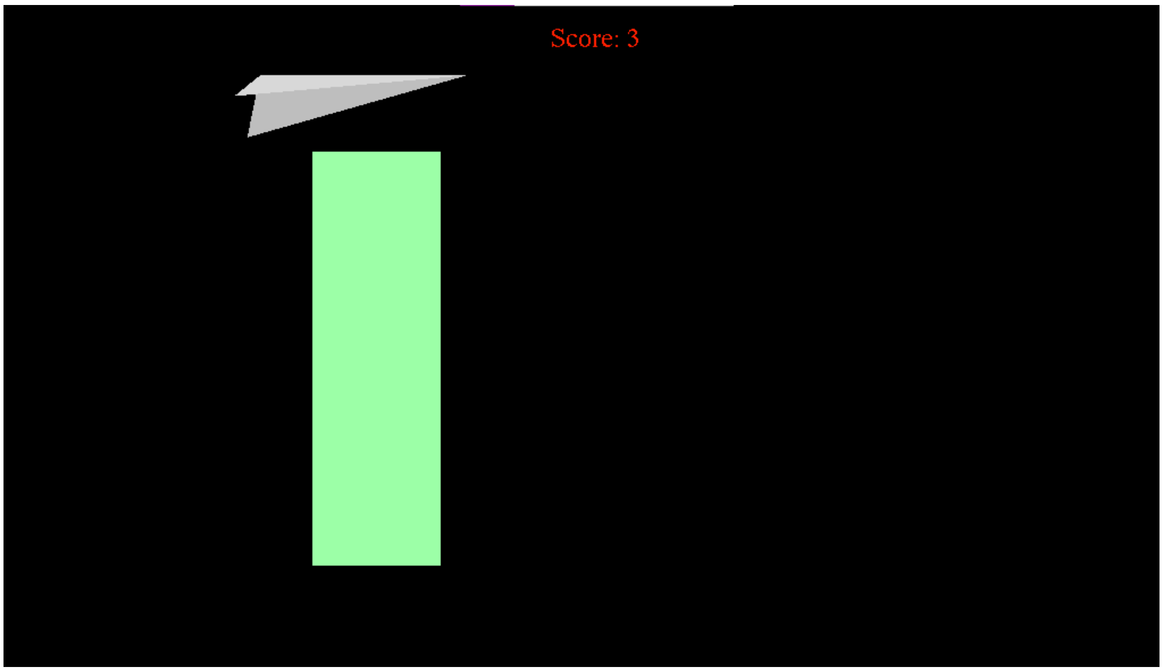


Fig 5.5 Plane movement and score display

This screen shows plane movement and score increment.

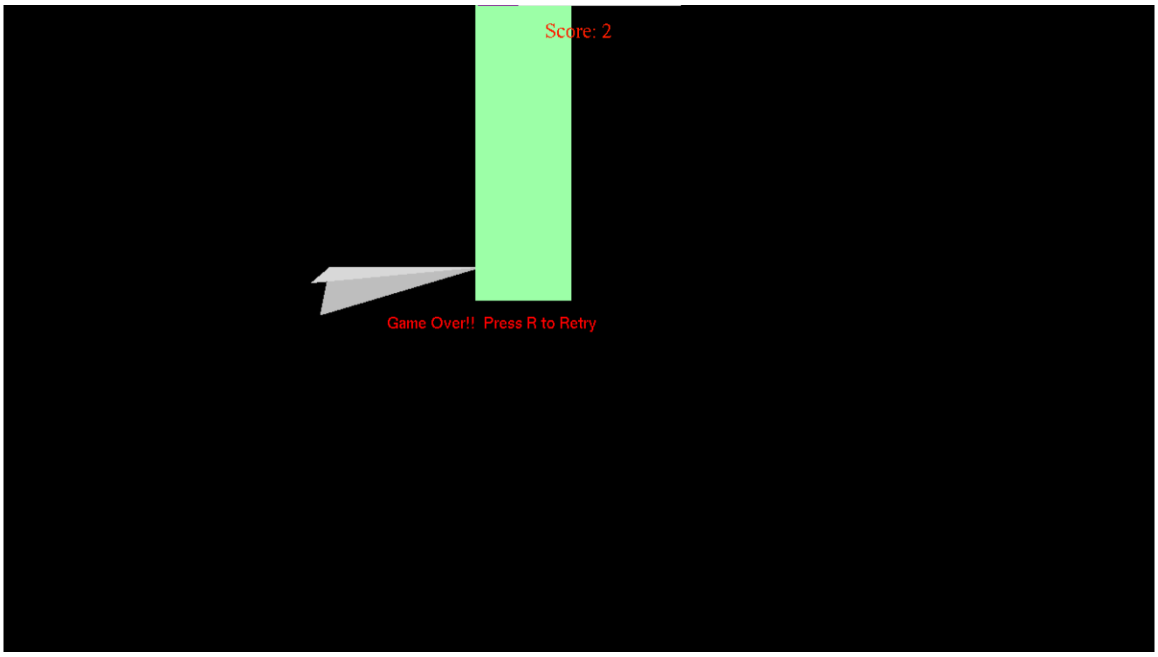


Fig 5.6 Game over

This screen shot shows the plane has been collided with block and gameover condition and retry option.

Chapter 6

CONCLUSION AND FUTURE SCOPE

6.1 Conclusion

The above developed game is a simpler version with easily understandable code. It can be advanced more with lot more levels.

The project helped know about the proper utilization of various graphics library functions.It provides user friendly interfaces like keyboard interactions.The game is simple and interesting.

6.2 Future Scope

Further development in the project can be done by illustrating graphically some other conditions other than what we have illustrated. It can be made still more attractive using 3D implementation. The project is widely open to anyone who wishes to improve it further on graphical way.

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] Donald Hearn and Pauline Baker: Computer Graphics Using Opengl, Pearson Education ,2004

[4] James D Foley, Andries Van Dam, Steven K Feiner, John F Hughes, ComputrGraphics, Pearson Education 1997

[5] www.opengl.org/