ABSTRACT

Graphics provides one of the most natural means of communicating with a computer, since our highly developed 2D and 3D pattern recognition allow us to perceive and process pictorial data rapidly and efficiently. Interactive computer graphics is the most important means of producing the pictures since the invention of photography and television. It has the added advantage that with the computer we can make pictures not only of concrete real world objects but also of abstract such as survey results.

This project is a collision evasion process implemented in the form of a game. Here the user is allowed to control the movement of the plane through a collision course. The objective is to evade the obstacles and travel as far as possible. The plane's upward movement is controlled by the left mouse button. Holding and pressing the left mouse button takes the plane higher and releasing the button results in the plane descending down. As the distance covered increases the speed of the plane increases. The main obstacles are buildings and clouds. The user must make sure that no part of the plane touches either the clouds or the buildings. The plane is also equipped with a booster which the user can activate with the right mouse button. The booster makes the plane travel at a faster pace. The booster gets depleted when being used and replenishes gradually when not used. At any instance the user can pause the game by pressing "p" on the keyboard. Pressing the key again will resume the game. The library glut.h is utilized in the implementation of the game as it provides a wide range of options to draw the necessary shapes.

CONTENTS

CHAPTER	TITLE	PAGE NO
1	Introduction	1
2	Requirement Analysis	5
3	OpenGL Functions	6
4	Implementation	12
5	Results and Snapshots	34
6	Future Enhancement	37
7	Conclusion	38
	Bibliography	39

CHAPTER 1

INTRODUCTION

1.1 Introduction to Computer Graphics

Before the invention of computer graphics people used to display the information manually either by drawing or creating model which resembles real environment. For example the Greeks were able to convey their architectural ideas graphically using drawing. Today the same type of information is generated by architects, mechanical engineers and draftsman using computer based graphics system.

Our interaction with computers has become dominated by a visual paradigm that includes windows, icons, menus and a pointing device, such as a mouse. From a user's Perspective, windowing system such as the X windows system, Microsoft's Windows. Although we are familiar with the style of graphical user interface used on most workstations, advances in computer graphics have made possible other forms of interfaces.

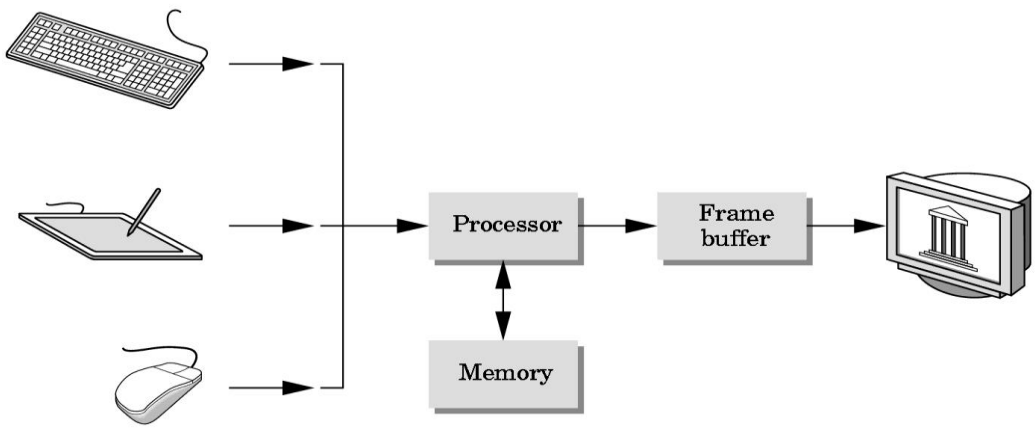

Fig 1.1 A Graphics System

1.2 Introduction to OpenGL API

OpenGL's structure is similar to that of most modern API's any effort that you put into learning OpenGL will carry over to other software systems. OpenGL is easy to learn compared with other API's, it support two and three dimensional programs that we develop. OpenGL provides the programmer with an interface to graphics hardware. It is a powerful, low-level rendering and modeling software library, available on all major platforms, with wide hardware support.

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

The main GL libraries usually have function that begins with the letters gl. The OpenGL Utility Library (GLU) uses gl functions but contains code for creating common objects and simplifying viewing. 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 glue between the window system and OpenGL that library is OpenGL Utility Toolkit (GLUT).

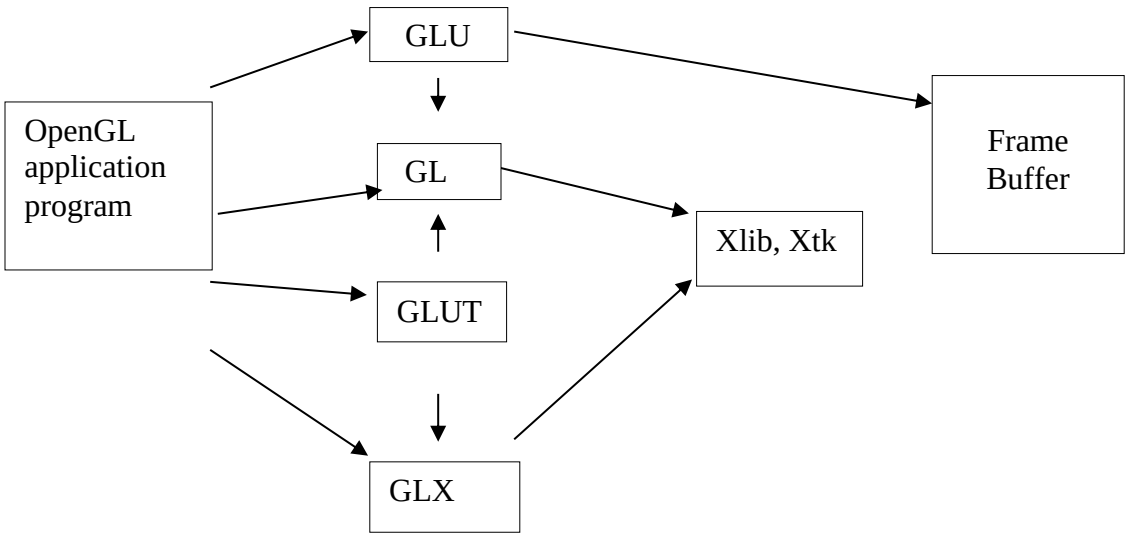

Fig 1.2 Library Organization

1.3 OpenGL supports two classes of primitives:

1 Geometric primitives

2 Image or raster primitives

Geometric primitive are specified in the problem domain and include points, line segments, polygons, curves, and surfaces. Raster primitives are those which are displayed using array of pixels, it is difficult to manipulate raster primitive objects.

OpenGL provide various viewing function that helps us to develop various views of single object, and the way in which it appears on screen. OpenGL default view is the orthographic projection. OpenGL also provide various transformation functions with the help of these functions user can render its object at the desired location on the screen. In OpenGL we obtain viewing and modeling functionality through a small set of transformation functions. We can even rotate the object along desired locations and with the desired angle on the screen.

1.4 OpenGL

OpenGL provides a set of commands to render a three dimensional scene. That means you provide the data in an OpenGL-useable form and OpenGL will show this data on the screen (render it). It is developed by many companies and it is free to use.

OpenGL is a hardware- and system-independent interface. An OpenGL-application will work on every platform, as long as there is an installed implementation. Because it is system independent, there are no functions to create windows etc., but there are helper functions for each platform. A very useful thing is GLUT.

1.5 GLUT

GLUT is a complete API written by Mark Kilgard which lets you create windows and handle the messages. It exists for several platforms, that means that a program which uses GLUT can be compiled on many platforms without (or at least with very few) changes in the code.

1.6 OpenGL architecture

OpenGL is a collection of several hundred functions providing access to all the features offered by your graphics hardware. Internally, it acts as a state machine--a collection of states that tells OpenGL what to do. Using the API, you can set various aspects of the state machine, including such things as the current color, lighting, blending, and so on. When rendering, everything drawn is affected by the current settings of the state machine. It's important to be aware of what the various states are, and the effect they have, because it's not uncommon to have unexpected results due to having one or more states set incorrectly. At the core of OpenGL is the rendering pipeline as shown in Figure.

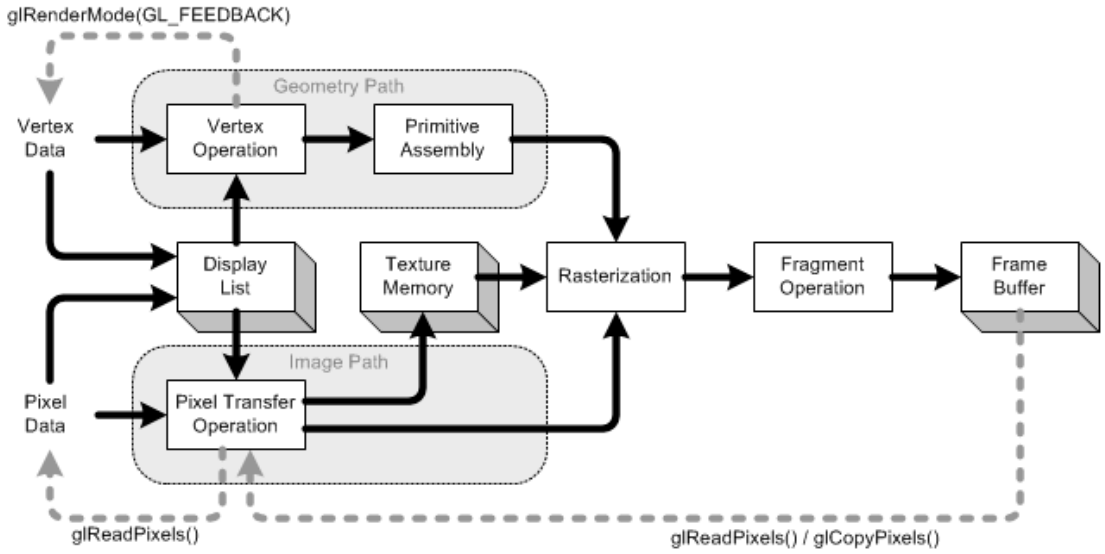

Fig 1.3 The OpenGL rendering pipeline

1.7 How does OpenGL work?

OpenGL bases on the state variables. There are many values, for example the color, that remain after being specified. There are no classes like in DirectX. However, it is logically structured. OpenGL provides its own data types. They all begin with "GL". For example- GLfloat, GLint and so on. There are also many symbolic constants; they all begin with "GL_", like GL_POINTS, GL_POLYGON.

CHAPTER 2

REQUIREMENTS ANALYSIS

2.1 RESOURCE REQUIREMENTS

The requirement analysis phase of the project can be classified into:

- Software Requirements
- Hardware Requirements

SOFTWARE REQUIREMENTS:

Input Requirement	Standard Input Device	:	Keyboard, Mouse.
Output Requirement	Standard Output Device	:	Color Monitor(60Hz)
Software Requirement	Programming Language	:	C
	Compiler Used	:	Visual C++ 2006/2008/2010/2012
	Operating System	:	Windows

HARDWARE REQUIREMENTS:

The hardware requirements are very minimal and the software can run on most of the machines.

1. Main processor : Pentium 4
2. Processor Speed : 1000 MHz or More
3. RAM Size : 64 MB DDR or More
4. Keyboard : Standard QWERTY serial or PS/2 keyboard
5. Mouse : Standard serial or PS/2 mouse
6. Compatibility : AT/T compatible
7. Cache memory : 512 KB

CHAPTER 3

OPENGL FUNCTIONS

3.1 Basic Functions

1. glPushMatrix, glPopMatrix Function

The glPushMatrix and glPopMatrix functions push and pop the current matrix stack.

SYNTAX: void glPushMatrix();

void glPopMatrix(void);

PARAMETERS: This function has no parameters.

2. glColor3f Function

Sets the current color.

SYNTAX: void glColor3f(GLfloat red, GLfloat green, GLfloat blue);

3. 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 triangle 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.

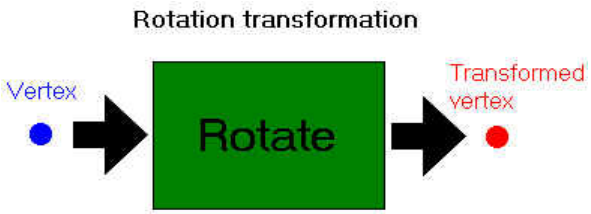

Fig 3.2 Vertex Transformation

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 color 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 that 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. glutSwapBuffers

glutSwapBuffers swaps the buffers of the current window if double buffered.

SYNTAX: void glutSwapBuffers(void);

glutSwapBuffers();

5. 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 bit masks. See values below:

  GLUT_RGB: An alias for GLUT_RGBA.

  GLUT_DOUBLE: Bit mask to select a double buffered window. This overrides GLUT_SINGLE if it is also specified.

  GLUT_DEPTH: Bit mask 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();

  glutStrokeCharacter

It renders the character with ASCII code char at the current raster position using the stoke font.

SYNTAX: Void glutStrokeCharacter(void* font, int char)

PARAMETERS:

Font: GLUT_STROKE_MONO_ROMAN & GLUT_STROKE_ROMAN.

3.6 Text Displaying Functions

1. GlRasterPos

glRasterPos specify 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) (unsigned char key, int x, int y));

CHAPTER 4

IMPLEMENTATION

4.1 Tools Used:

OpenGL is an application program interface that is used to define 2D and 3D computer graphics. This cross-platform API is generally considered to set the standard in the computer industry when it comes to this type of interaction with 2D computer graphics and has also become the usual tool for use with 3D graphics as well. Short for Open Graphics Library, OpenGL eliminated the need for programmers to rewrite the graphics section of an operating system each time a business would upgrade to a new version of the system.

The basic function of OpenGL is to issue a specific collection of executable or commands to the operating system. In doing so, the program works with the existing graphics hardware that resides on the hard drive or other specified source. Each command in the set is designed to engage a certain drawing action, or launch a specific special effect associated with the graphics.

4.1 Implementation of User Defined Functions

User Defined Functions

The project contains several main user defined functions performing the essential tasks for implementing the game. They include:

- boundHit( ) : Checks if the plane exceeds the boundary.

```
bool boundHit()
{
    if(plane_mvmt+50>=100||plane_mvmt+50 <=18)
            return true;
    else
     return false;
}
```

- BuildingBlock( ) : Constructing the building.

```
void buildingBlock()
{
    b.block_x=50.0;
    srand(time(0));
    b.no_floors = rand()%3+4;
    buildColor = rand()%3;

    b.block_y=b.no_floors*10 +15;
    b.state=true;
    s.state=false;
 }
```

- buildingHit( ) : Checking if the airplane has crashed into the building.

```
bool buildingHit()
{
    if (((int)b.block_x<=8 &&(int)b.block_x>=-7 && ((int)plane_mvmt+50)-b.block_y<=3))
            return true;
    else if (((int)b.block_x<=10 &&(int)b.block_x>=-5 && ((int)plane_mvmt+50)-
b.block_y<=0))
            return true;
    else if(((int)b.block_x<=6 &&(int)b.block_x>=-3 && ((int)plane_mvmt+47)-
b.block_y<=0))          return true;
    else if(((int)b.block_x<=4 &&(int)b.block_x>=-4 && ((int)plane_mvmt+47)-
b.block_y<=3))
            return true;
    else
            return false;
}
```

- Circle( ) : Draw circle.

```
void Circle(float x1,float y1,float radius)
{
    float x2,y2;
```

```
    float angle;
    glBegin(GL_POINTS);

    for (angle=1.0f;angle<360.0f;angle++)
    {
        x2 = x1+sin(angle)*radius;
        y2 = y1+cos(angle)*radius;
        glVertex2f(x2,y2);
    }
    glEnd();
  }
```

- CloudBlock( ) : Constructing the cloud.

```
void CloudBlock()
{
    s.block_x=50.0;
    srand(time(0));
    s.block_y=(rand()%30)+50;
    s.state=true;
    b.state=false;
}
```

- CloudHit( ) : Checking if the airplane has hit the cloud.

```
bool cloudHit()
{
    if(s.block_x < 13 && s.block_x> -5)
        if(plane_mvmt-3+50 >s.block_y-3 && plane_mvmt-3+50 < s.block_y+3)
            return true;

    if(s.block_x < 12 && s.block_x> -4)
        if(plane_mvmt-3+50 >s.block_y-5 && plane_mvmt-3+50 < s.block_y+5
            return true;

    if(s.block_x < 10 && s.block_x> -1)
```

```
            if(plane_mvmt-3+50 >s.block_y-6 && plane_mvmt-3+50 < s.block_y-2)
                return true;

        //for top wing and bottom wing
        if(s.block_x < 5 && s.block_x> -3)
            if(plane_mvmt-3+50 >s.block_y-11 && plane_mvmt-3+50 < s.block_y+13)
                return true;

        return false;
    }
```

- display( ) : Used to manifest all the objects onto the screen.

```
    void display()
    {
        glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
        //GameOver Checking
        if(gameEndStatus == true)
        {
            gameEnd();
        }
        else if(wflag==true)//Welcome Screen
        {
            welcome();
        }
        else if (instflag == true)
        {
            glColor3f(0.3,0.56,0.84);
            glBegin(GL_POLYGON);
            glVertex3f(0.0,0.0,0.0);
            glColor3f(0.137,0.137,0.556);
            glVertex3f(100.0,0.0,0.0);
            glColor3f(0.196,0.196,0.8);
            glVertex3f(100.0,100.0,0.0);
```

```
glVertex3f(0.0,100.0,0.0);
glEnd();
glPushMatrix();
glScalef(0.8,0.8,0);
drawJet();
glPopMatrix();
glColor3f(0.137,0.137,0.556);
glRectf(20.0,20.0,80.0,80.0);
glColor3f(0.8,0.8,0.8);
glRectf(21.0,21.0,79.0,79.0);


glColor3f(0.196,0.196,0.8);
glRectf(40,5,60,10);
glColor3f(0.8,0.8,0.8);
glRectf(40.5,5.5,59.5,9.5);

glColor3f(0.137,0.137,0.556);
drawString(46,6,0,GLUT_BITMAP_TIMES_ROMAN_24,"BACK");

glColor3f(0.137,0.137,0.556);
drawString(37,75,0,GLUT_BITMAP_TIMES_ROMAN_24,"HOW TO PLAY");
drawString(23,69,0,GLUT_BITMAP_HELVETICA_18,"- Click and hold mouse left key to gain altitude of ");
drawString(23,65,0,GLUT_BITMAP_HELVETICA_18,"    the plane.");
drawString(23,61,0,GLUT_BITMAP_HELVETICA_18,"- Release the mouse left key to reduce the altitude.");
drawString(23,57,0,GLUT_BITMAP_HELVETICA_18,"- Use the Right mouse key to speed up the plane(NOS)");
drawString(23,53,0,GLUT_BITMAP_HELVETICA_18,"- Main aim of the game is to avoid the obstacles ");
```

```
            drawString(23,49,0,GLUT_BITMAP_HELVETICA_18,"    such as buildings and
clouds.");
            drawString(23,45,0,GLUT_BITMAP_HELVETICA_18,"- Also the meter at the
bottom shows the distance ");
            drawString(23,41,0,GLUT_BITMAP_HELVETICA_18,"    travelled,NITROS
left,Atitude and the LEVEL.");
            drawString(23,37,0,GLUT_BITMAP_HELVETICA_18,"- As you reach distance
multples of 50 tour level ");
            drawString(23,33,0,GLUT_BITMAP_HELVETICA_18,"    increases as well as
the speed of the plane.");
            drawString(33,27,0,GLUT_BITMAP_HELVETICA_18," ENJOY PLAYING
THE GAME");

            glutPostRedisplay();

    }
    else if (abtflag == true)
    {
            glColor3f(0.3,0.56,0.84);
            glBegin(GL_POLYGON);
            glVertex3f(0.0,0.0,0.0);
            glColor3f(0.137,0.137,0.556);
            glVertex3f(100.0,0.0,0.0);
            glColor3f(0.196,0.196,0.8);
            glVertex3f(100.0,100.0,0.0);
            glVertex3f(0.0,100.0,0.0);
            glEnd();
            glPushMatrix();
            glScalef(0.8,0.8,0);
            drawJet();
            glPopMatrix();
            glColor3f(0.137,0.137,0.556);
```

```
            glRectf(20.0,20.0,80.0,80.0);
            glColor3f(0.8,0.8,0.8);
            glRectf(21.0,21.0,79.0,79.0);
            glColor3f(0.196,0.196,0.8);
            glRectf(40,5,60,10);
            glColor3f(0.8,0.8,0.8);
            glRectf(40.5,5.5,59.5,9.5);
            glColor3f(0.137,0.137,0.556);
            drawString(46,6,0,GLUT_BITMAP_TIMES_ROMAN_24,"BACK");
            glColor3f(0.137,0.137,0.556);
            drawString(44,75,0,GLUT_BITMAP_TIMES_ROMAN_24,"ABOUT");
            drawString(21,61,0,GLUT_BITMAP_HELVETICA_18," COMPUTER
GRAPHICS PROJECT BY ");
            drawString(23,53,0,GLUT_BITMAP_HELVETICA_18,"
ARPITHA.N.KASHYAP :1BG12CS022");
            drawString(33,40,0,GLUT_BITMAP_HELVETICA_18," ENJOY PLAYING
THE GAME");
            glutPostRedisplay();
}
    else if( pause == true)
    {
            drawBg();
            glPushMatrix();
            glScalef(0.8,0.8,0);
            drawJet();
            glPopMatrix();
            glPushMatrix();
            glColor3f(0.196,0.196,0.8);
            glRectf(35.0,40.0,65.0,60.0);
            glColor3f(0.8,0.8,0.8);
            glRectf(36.0,41.0,64.0,59.0);
            glPopMatrix();
```

```
            glColor3f(0.137,0.137,0.556);
        drawString(40,55,0,GLUT_BITMAP_HELVETICA_18," GAME PAUSED");
            drawString(37,45,0,GLUT_BITMAP_HELVETICA_18," PRESS 'P' to
continue");
            glutPostRedisplay();

    }
    else if((b.state == true && buildingHit() == true)|| boundHit()== true)
    {
            gameEndStatus = true;
            gameEnd();
    }
    else if(s.state == true && cloudHit() == true)
    {

            gameEndStatus = true;
            gameEnd();
    }
    else
    {
             if((int)score%50==0 && lflag==true)// l-level
            {
                    lflag=false;
                    level++;
                    bspd+=0.01;
            }
            else if((int)score%50!=0 && lflag==false)
            {
                    lflag=true;
            }

            glPushMatrix();
```

```
drawBg();
glPushMatrix();
glTranslatef(0.0,plane_mvmt,0.0);
drawJet();                          //code for jet
glPopMatrix();
if( booster <= BOOSTER_MAX && !boost)
        booster+=0.005;
if( (b.state == true && b.block_x < -10) || (s.state == true && s.block_x < -10))
{
        srand(time(NULL));
        int random = rand()%2;//for random building or cloud
        if( random == 0)
        {
                buildingBlock();
        }
        else
        {
                CloudBlock();
        }
}

else if(b.state == true)
{
        if(booster >0 && boost)
        {
                b.block_x-=bspd+boost;
                booster = booster-0.02;
        }
        else
                b.block_x-=bspd;
}
else if( s.state == true)
```

```
            {
                if(booster >0 && boost)
                {
                    s.block_x-=bspd+boost;
                    booster = booster-0.02;

                }
                else
                {
                    s.block_x-=bspd;
                }
            }
            if(b.state == true)
            {
                glTranslatef(b.block_x,0.0,0.0);
                drawBuilding();
            }
            else if( s.state == true)
            {
                glTranslatef(s.block_x,0.0,0.0);
                drawCloud();
            }
            glPopMatrix();

            printScore();
        }
        glFlush();
        glutSwapBuffers();
    }
```

- DrawBg( ) : Draws the background.
- drawBuilding( ) : Draws the constructed building.

```
void drawBuilding()
```

```
{
    glPushMatrix();
    if(buildColor==0)
            glColor3f(0.1,0.0,0.0);
    else if (buildColor ==1)
            glColor3f(0.1,0.1,0.0);
    else
            glColor3f(0.0,0.1,0.1);
    glTranslatef(b.block_x,b.no_floors*10.0+10,0.0);
    glBegin(GL_POLYGON);
    glVertex3f(0.0,0.0,0.0);
    glVertex3f(5.0,3.0,0.0);
    glVertex3f(20.0,3.0,0.0);
    glVertex3f(20.0,-b.no_floors*10.0,0.0);
    glVertex3f(0.0,-b.no_floors*10.0,0.0);
    glEnd();
    glPopMatrix();

    for(int i=1;i<=b.no_floors;i++)
    {
            glPushMatrix();

            if(buildColor==0)
                    glColor3f(0.8,0.0,0.0);
```

```
else if (buildColor ==1)
        glColor3f(0.8,0.8,0.0);
else
        glColor3f(0.0,0.8,0.8);
glTranslatef(b.block_x,10.0*i,0.0);
glBegin(GL_POLYGON);
glVertex3f(0.0,0.0,0.0);
glVertex3f(15.0,0.0,0.0);
glVertex3f(15.0,10.0,0.0);
glVertex3f(0.0,10.0,0.0);
glEnd();
glColor3f(1.0,1.0,1.0);
glBegin(GL_POLYGON);
glVertex3f(2.5,5.0,0.0);
glVertex3f(5.5,5.0,0.0);
glVertex3f(5.5,8.0,0.0);
glVertex3f(2.5,8.0,0.0);
glEnd();
glColor3f(1.0,1.0,1.0);
glBegin(GL_POLYGON);
glVertex3f(2.5,0.0,0.0);
glVertex3f(5.5,0.0,0.0);
glVertex3f(5.5,3.0,0.0);
glVertex3f(2.5,3.0,0.0);
```

```
        glEnd();
        glColor3f(1.0,1.0,1.0);
        glBegin(GL_POLYGON);
        glVertex3f(12.5,5.0,0.0);
        glVertex3f(9.5,5.0,0.0);
        glVertex3f(9.5,8.0,0.0);
        glVertex3f(12.5,8.0,0.0);
        glEnd();
        glColor3f(1.0,1.0,1.0);
        glBegin(GL_POLYGON);
        glVertex3f(12.5,.0,0.0);
        glVertex3f(9.5,0.0,0.0);
        glVertex3f(9.5,3.0,0.0);
        glVertex3f(12.5,3.0,0.0);
        glEnd();
        glPopMatrix();
}
glPushMatrix();
if(buildColor==0)
        glColor3f(0.8,0.0,0.0);
else if (buildColor ==1)
        glColor3f(0.8,0.8,0.0);
else
        glColor3f(0.0,0.8,0.8);
```

```
        glTranslatef(b.block_x,10.0,0.0);
        glBegin(GL_POLYGON);
        glVertex3f(0.0,0.0,0.0);
        glVertex3f(15.0,0.0,0.0);
        glVertex3f(15.0,10.0,0.0);
        glVertex3f(0.0,10.0,0.0);
        glEnd();
        glColor3f(1.0,1.0,1.0);
        glBegin(GL_POLYGON);
        glVertex3f(5.5,0.0,0.0);
        glVertex3f(9.5,0.0,0.0);
        glVertex3f(9.5,6.0,0.0);
        glVertex3f(5.5,6.0,0.0);
        glEnd();
        glPopMatrix();
    }
```

- drawCloud( ) : Draws the constructed cloud.
- drawJet( ) : Draws the jet airplane.
- init( ) : Initializes the glOrtho and randomizes between the cloud or building for the first time.
- keyPressed( ) : Used to interact with the keyboard interface.

```
    void keyPressed(unsigned char key,int x, int y)
    {
```

```
    if(key == 27)
    {
        exit(0);
    }
    else if(key == 'p' || key == 'P')
    {   if(pause == true)
            pause = false;
        else
            pause = true;
     }
    glutPostRedisplay();
}
```

- mouse( ) : Used to interact with the mouse.

```
void mouse(int button, int state, int x, int y)
{
    int mx=x*100/SCREENW,my=(SCREENH-y)*100/SCREENH;
    if(instflag || abtflag || gameEndStatus)
    {
        if(mx>40 && mx<60)
        {
            if(my>5 && my<10)
            {
                wflag = true;
                if(instflag)
```

```
                    instflag = false;
                else if (abtflag)
                    abtflag = false;
                if(gameEndStatus)
                {
                    wflag = true;
                    gameEndStatus = false;
                    plane_mvmt = 0;
                    start = false;
                    init();
                    bspd = BLOCKSPEED;//restarting the game
                    booster=BOOSTER_MAX;
                    score=1;
                    level=1;
                    glutPostRedisplay();
                }
            }
        }
    }
    if(wflag == true)
    {
        if(mx>40 && mx<60)
        {
            if(my>40 && my<45)
```

```
                {
                        start = true;
                        wflag=false;
                }
                else if(my>30 && my<35)
                {
                        instflag = true;
                        wflag = false;
                }
                else if(my>20 && my<25)
                {
                        abtflag = true;
                        wflag = false;
                }
                else if(my>10 && my<15)
                {
                        exit(0);
                }
        }
}
 else
{
         if(button == GLUT_LEFT_BUTTON)
        {
```

```
                if (state == GLUT_DOWN )
                        glutIdleFunc(moveJetU);

                else if (state == GLUT_UP )
                        glutIdleFunc(moveJetD);
        }
        if(button == GLUT_RIGHT_BUTTON)
        {
                if(state == GLUT_DOWN)
                {
                        if(booster>0)
                        {
                                boost = 0.05;
                        }
                }
                if(state == GLUT_UP)
                {
                        boost = 0;
                }
        }
    }
}
```

- moveJetD( ) : Used to move the jet airplane down.

```
void moveJetD()
```

```
{
    if(start == false)
            glutPostRedisplay();
    else if(pause == false )
    {
            //alti_ang+=0.15;
            plane_mvmt-=0.05;
            glutPostRedisplay();
    }
}
```

- moveJetU( ) : Used to move the jet airplane up.

```
void moveJetU()
{
    if(start == false)
            glutPostRedisplay();
    else if(pause == false)
    {
            plane_mvmt+=0.05;
            glutPostRedisplay();
    }
}
```

- printScore( ) : Used to print the final score.

```
void printScore()
{
```

```
glColor3f(1.0,1.0,0.0);//score
sprintf_s(slevel, "%d", (int)level );
drawString(58,1.8,0,GLUT_BITMAP_TIMES_ROMAN_10,"Level");
drawString(58,3.5,0,GLUT_BITMAP_TIMES_ROMAN_24,slevel);
if(booster >0 && boost)
        score+=0.03;//SCORE with booster
else
        score+=0.005;//SCORE without booster

drawString(38,1.5,0,GLUT_BITMAP_TIMES_ROMAN_10,"Distance");
sprintf_s(score_Str, "%d m",(int)score );
drawString(38,3,0,GLUT_BITMAP_TIMES_ROMAN_24, score_Str);
}
```

- welcome( ) : Displays the welcome screen.

Built in 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[i f d] (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 glutCeateWindow(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 glutInitWindowSize(int width, int height):

Specifies the initial height and width of the window in pixels.

void glutInitWindowPosition(int x, int y) :

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 any buffered any OpenGL commands to execute.

void glutInit (intargc, 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 should be the last statement in main.

void glutDisplayFunc(void (*func) (void)):

Registers the display function func that is executed when the window needs to be redrawn.

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_24 etc. The raster position is incremented by the width of the character.

void glClear(GL_COLOR_BUFFER_BIT):

To make the screen solid and white.

void glutpostRedisplay():

Registers the display function that is executed when the window needs to be redrawn.

void LoadIdentity():

Sets the current transformation matrix to an identity matrix.

void glEnable(glenum features):

Enables the opengl features.

void glMatrixMode(glenum mode):

Specifies which matrix will be affected by subsequent transformations.

void glViewport(intx,inty,GLsizeiwidth,GLsizei height):

Specifies a width*height viewport in pixel whose lower left corner is at (x,y) measured from the origin of the window.

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_24 etc. The raster position is incremented by the width of the character.

CHAPTER 5

RESULTS AND SNAP SHOTS

Fig 5.1 Screen with Menu

The start screen has four options:

1. Play – starts the game.

2. Instructions – Displays the instructions to play the game.

3. About – Details of the student.

4. Exit – Exits from the screen.

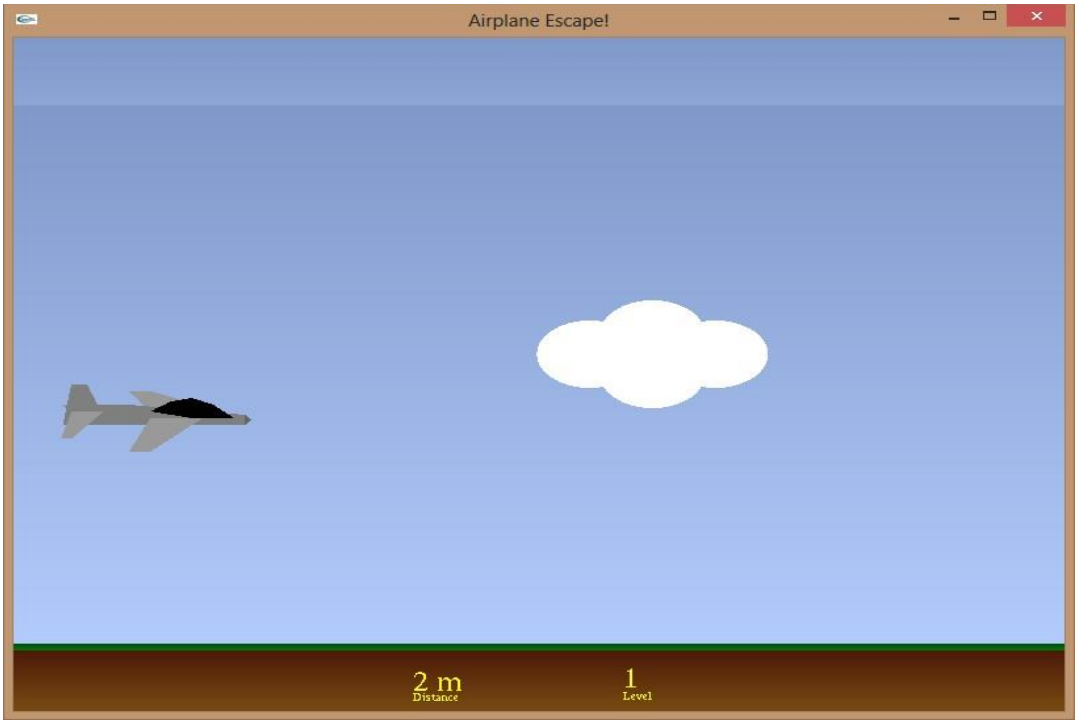

Fig 5.2 Cloud as an obstacle

The plane has to escape the cloud/building using mouse interaction. By pressing the left mouse button, the plane moves upwards and escapes from the cloud/building. If the plane collides with the cloud/building, the game ends.

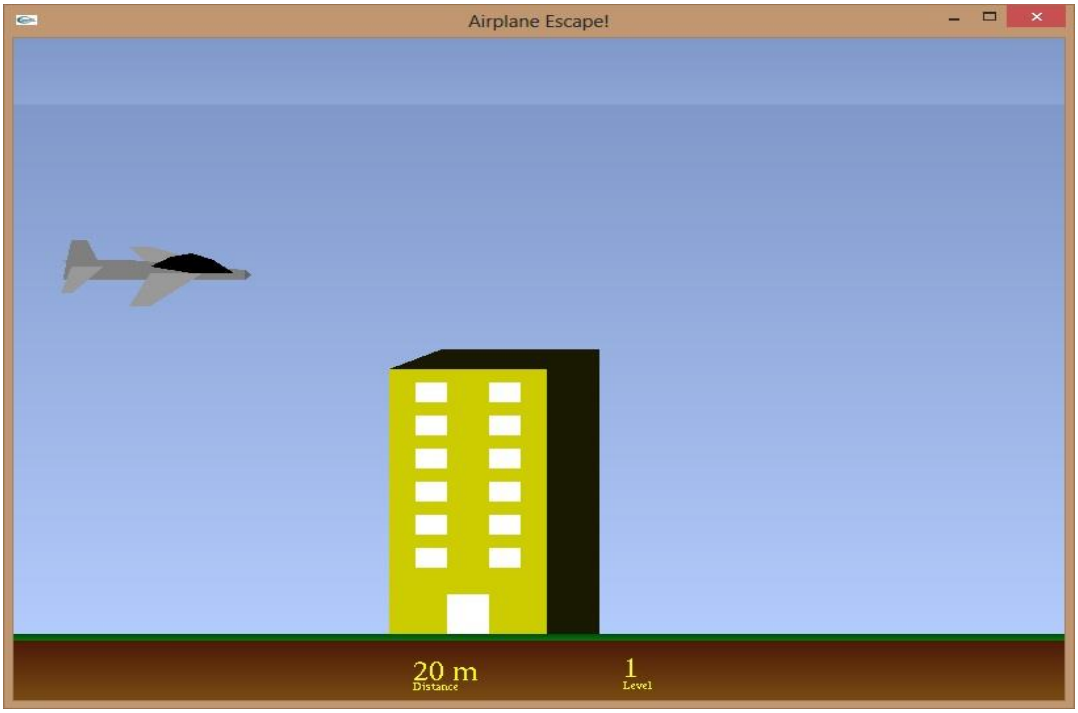

Fig 5.3 Building as an obstacle

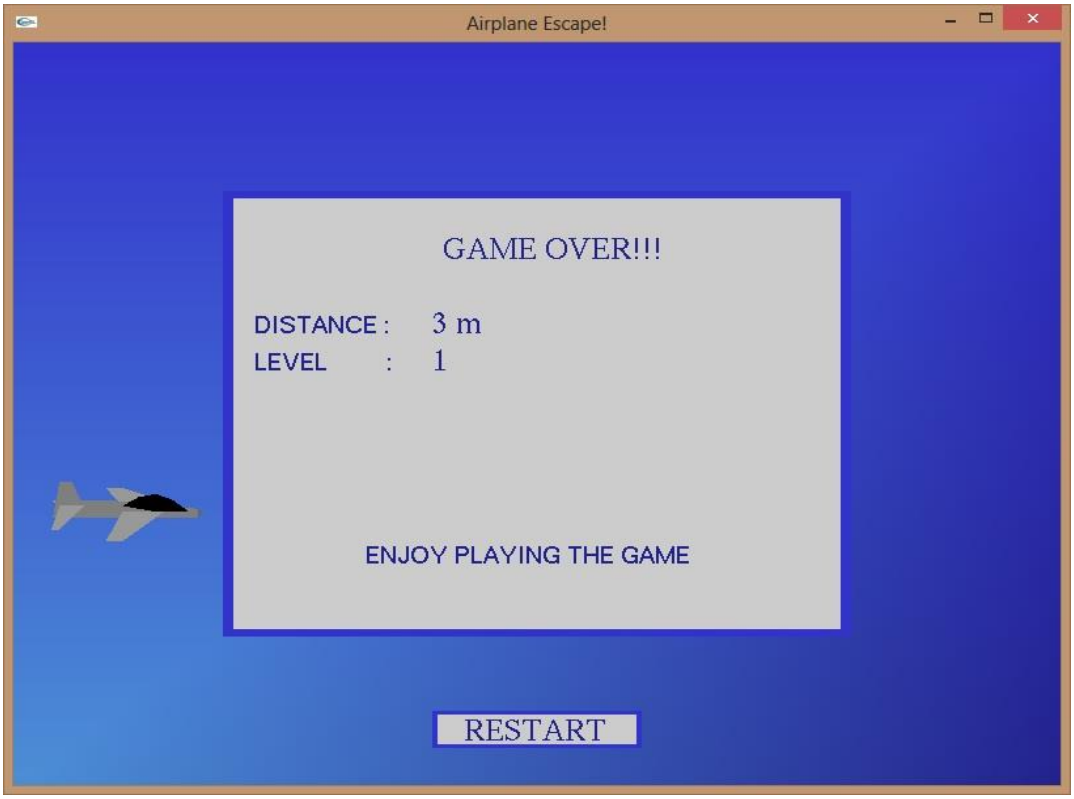

Fig 5.4 Game over Screen

When the plane collides with the cloud/building, the game ends and the game over screen appears. This screen shows the distance travelled by the plane (in meters) and the levels completed.

By clicking on the Restart button, it goes back to the start screen and you can choose among the four options.

CHAPTER 6

FUTURE ENHANCEMENT

The enhancements I would like to make for my project are as follows:

- ✓ Adding sound effects – for movement of the plane, collision of the plane with cloud/building, level up, background music, plane crashing the boundaries etc.
- ✓ Additional obstacles – like birds, hot air balloons, kites etc.
- ✓ Adding collectables – like coins and boosters to buy new parts for the plane and other boosts.
- ✓ Adding missions and challenges – daily and weekly challenges, completing missions for upgrades and power-ups.
- ✓ Customizing the plane – with the coins and boosts collected; customize the plane with new parts for greater speed and control.
- ✓ Adding various backgrounds for differ levels – new backgrounds for each new level unlocked.
- ✓ Adding more lives for the plane – each player can have a maximum of 3 lives before the game ends.
- ✓ Option to disable sound effects – an option to disable sound effects during or before/after the game.

CHAPTER 7

CONCLUSION

The project ‘Airplane game’ implemented on Windows platform using C was completed successfully. It involves various translating, scaling and rotating effects effectively bringing out a simple animation. The utilities and functions provided by the OpenGL have been used in an optimal way to achieve a completely working project.

The work done during the process of completion of this project has helped me understand the various functionalities available in a better and a more practical environment and I realize that the scope of OpenGL platform has a premier game developing launch pad. Hence it has indeed been useful in developing many games. OpenGL in its own right is good for low cost and simple game development. It serves as an important stepping stone for venturing into other fields of Computer Graphics design and applications. 3D animation, modeling, gaming etc. are the next big things, and working on this project has helped me learn the basics required for it. I have implemented this project for entertainment purpose and this game is very easy to play.

BIBLIOGRAPHY

[1]. Edward Angel-Interactive Computer Graphics: A Top-Down Approach using OpenGL Fifth Edition, Published by Pearson Education, 2009

[2]. The OpenGL Programming Guide, 5th Edition. The Official guide to learning OpenGL Version 2.1 by OpenGL Architecture Review Board.

[3]. Donald P Leach, Albert Paul Malvino & GoutamSaha -Digital Principles and Applications, 6th Edition, TMH, 2006.

[4]. www.opengl.org