CHAPTER 1

INTRODUCTION

1.1 Computer Graphics

- Graphics provides one of the most natural means of communicating with a computer, since our highly developed 2D or 3D pattern-recognition abilities allow us to perceive and process pictorial data rapidly.

- Computers have become a powerful medium for the rapid and economical production of pictures.

- Graphics provide a so natural means of communicating with the computer that they have become widespread.

- Interactive graphics is the most important means of producing pictures since the invention of photography and television.

- We can make pictures of not only the real-world objects but also of abstract objects such as mathematical surfaces on 4D and of data that have no inherent geometry.

- A computer graphics system is a computer system with all the components of the general-purpose computer system. There are five major elements in system: input devices, processor, memory, frame buffer, output devices.

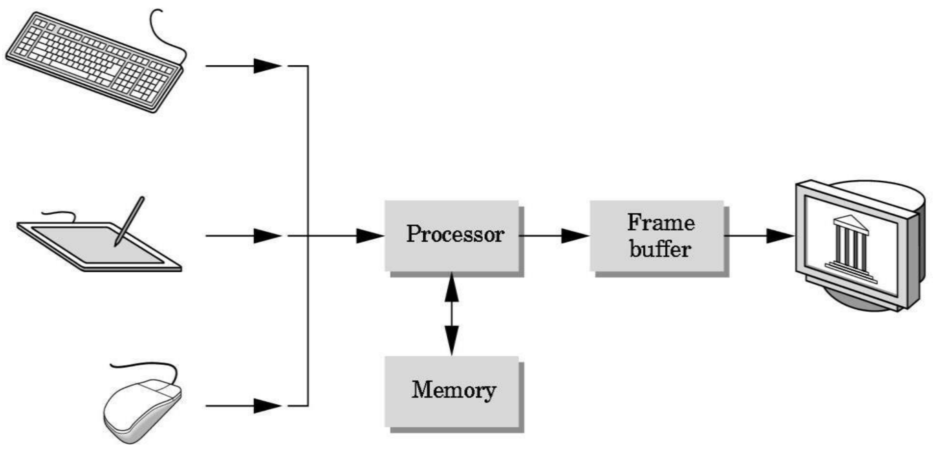

Fig 1.1: Graphic System

1.2 Areas of Application of Computer Graphics

- User interfaces and Process control
- Cartography
- Office automation and Desktop publishing
- Plotting of graphs and charts
- Computer aided Drafting and designs
- Simulation and Animation

1.3 Introduction to OpenGL

OpenGL is the premier environment for developing portable, interactive 2D and 3D graphics applications. Since its introduction in 1992, OpenGL has become the industry's most widely used and supported 2D and 3D graphics application programming interface (API), bringing thousands of applications to a wide variety of computer platforms.

OpenGL fosters innovation and speeds application development by incorporating a broad set of rendering, texture mapping, special effects, and other powerful visualization functions. Developers can leverage the power of OpenGL across all popular desktop and workstation platforms, ensuring wide application deployment.

OpenGL available Everywhere: Supported on all UNIX® workstations and shipped standard with every Windows 95/98/2000/NT and MacOS PC, no other graphics API operates on a wider range of hardware platforms and software environments.

OpenGL runs on every major operating system including Mac OS, OS/2, UNIX, Windows 95/98, Windows 2000, Windows NT, Linux, Open Step, and BeOS; it also works with every major windowing system, including Win32, MacOS, Presentation Manager, and X-Window System. OpenGL is callable from Ada, C, C++, Fortran, Python, Perl and Java and offers complete independence from network protocols and topologies.

1.3.1 The OpenGL interface

Our application will be designed to access OpenGL directly through functions in three libraries namely: gl , glu , glut.

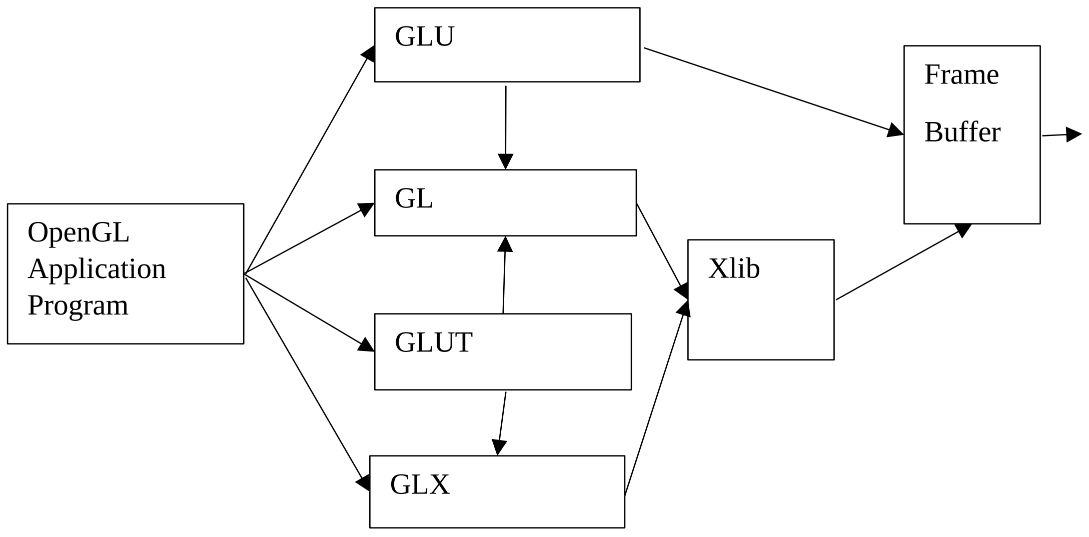

Fig 1.2 Library Organization of OpenGL

1.3.2 Graphics Functions

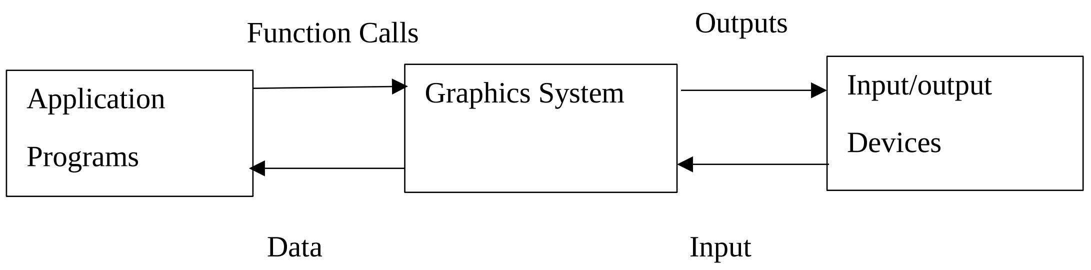

Fig 1.3: Graphics system as a black box.

Our basic model of a graphics package is a black box, a term that engineers use to denote a system whose properties are described only by its inputs and outputs. We describe an API through the functions in its library. Some of the functions are:

- The primitive functions define the low-level objects or atomic entities that our system can display.
- Attribute functions allow us to perform operations ranging from choosing the color with which we display a line segment, to picking a pattern with which to fill the inside of a polygon, to selecting a typeface for the titles of a graph.
- Transformation function allows carrying out transformations of objects, such as rotation, translation, and scaling.
- A set of input functions allow us to deal with the diverse forms of input that characterize modern graphics systems.
- The control functions enable us to communicate with the window systems, to initialize our programs, and to deal with any errors that take place during the execution of programs.

CHAPTER 2

REQUIREMENTS SPECIFICATION

2.1 Purpose of the requirements document

The software requirement specification is the official statement of what is required for development of particular project. It includes both user requirements and system requirements. This requirement document is utilized by variety of users starting from project manager who gives project to the engineer responsible for development of project.

It should give details of how to maintain, test, verify and what all the actions to be carried out through life cycle of project.

2.1.1 Scope of the project

The scope is to use the basic primitives defined in OpenGL library creating complex objects. We make use of different concepts such as Transformation, state change, scaling.

2.1.2 Definition

The project **AEROPLANE CRASH** is created to demonstrate the usage of openGL in games, using many of the known techniques in game development such as state updates, frame drawing, collision detecting, timed updates.

.

2.2 Specific Requirements

2.2.1 User Requirements:

- Easy to understand and should be simple.
- The built-in functions should be utilized to maximum extent.
- OpenGL library facilities should be used.

2.2.2 Software Requirements:

- Platform used: Visual Studio Code, MSVC, Windows 10, C-Make.
- Technology used: freeglut-3.
- Language: C++

2.2.3 Hardware Requirements:

- Processor: 2.4Ghz or greater.
- RAM: 2GB or greater.
- Hard Disk: 1 GB or grater.
- Mouse
- Keyboard
- Monitor

CHAPTER 3

DESIGN

3.1Functions Used:

- **glutInitDisplayMode():**
  - Definition: void glutInitDisplayMode(unsigned int mode);
  - Remarks: Initializes a new Display mode for the created screen. We are using GLUT_DOUBLE for double buffer.
- **glHint():**
  - Definition: void glhint(int hint, int mode);
  - Remarks: Hints to the openGL implementation on rendering the polygons to a certain smoothness.
- **glBlendFunc():**
  - Definition: void glBlendFunc (GLenum sfactor,GLenum dfactor);
  - Remarks: Used to Indicate the blend function that allow the openGL implementation to use interesting effects while rendering. This is used for transparency rendering.
- **glOrtho():**
  - Definition: void glOrtho(GLdouble left, GLdouble right, GLdouble bottom,GLdouble top,GLdouble nearVal, GLdouble farVal);
  - Remarks: Multiplies the current matrix with an orthographic Matrix.
- **glMatrixMode():**
  - Definition: void glMatrixMode(GLenum mode);
  - Remarks: Specify which matrix is the current matrix.
- **glViewport():**
  - Definition: void glViewport(GLint x, GLint y, GLsizei width, GLsizei, height);
  - Remarks: This function sets the view port of the screen. Sets the width and height of the screen, from the lower left corner.
- **glutMouseFunc():**
  - Definition: void glutMouseFunc(void (func*) (int action, int state, int x, int y);
  - Remarks: Sets the function that handles the mouse movements inside the window. The state and action represent the event that happened and the x and y co-ordinates that it happened in.

- **glTranslatef():**
  - Definition: void glTranslatef(float x, float y, float z);
  - Remarks: Translates from the current position to the given position on the viewport.
- **glColor4f():**
  - Definition: void glColor4f(float red, float green, float blue, float alpha);
  - Remarks: Used to set the color and transparency of the object being drawn.
- **glRectf():**

  Definition: void Rectf(float x1, float y1, float x2, float y2);

  Remarks: A convenience function that draws a rectangle from (x1, y1) to (x2, y2);

Chapter 4

IMPLEMENTATION

```
#include<GL/freeglut.h>
#include<time.h>
#include<stdio.h>
#include<random>
#include"Utility.h"
#define BLOCKSPEED 0.04f

int SCREENH = 600, SCREENW = 800;


typedef struct building
{
	float block_x, block_y;
	bool state;
	int no_floors;
}building;

typedef struct Cloud
{
	float block_x, block_y;
	bool state;
}Cloud;


//-------------------declarations---------------
float bspd = BLOCKSPEED;  // block speed
bool pause = false, lflag = true, welcome_flag = true, gameEndStatus = false,  start = false;
//flags
float plane_mvmt = 0.0;//jet movement up or down
char score_Str[20], slevel[20];   //score string and levelstring

float score = 1;
int level = 1; //Level and Score Counter

color buildColor(randomGenF(), randomGenF(), randomGenF());
building building_state;  // building struct
Cloud cloud_state;     // cloud struct

//-------------function prototypes------------------
void keyPressed(unsigned char, int, int);
void mouse(int button, int state, int x, int y);
void buildingBlock();
void CloudBlock();
void init();
void drawJet();
void gameEnd();
void drawBg();
void welcome();
void drawBuilding();
void drawCloud();
bool cloudHit();
bool buildingHit();
void printScore();
void display();
void moveJetU();
void moveJetD();
void drawScreenBg();


//------------------ Object Makers:

void buildingBlock()
{
	building_state.block_x = 50.0;
	srand(time(0));
	building_state.no_floors = rand() % 3 + 4;
	buildColor = color(randomGenF(), randomGenF(), randomGenF());
	building_state.block_y = building_state.no_floors * 10 + 15;	// generate block y
cordinate depending on no of floors
	building_state.state = true;
	cloud_state.state = false;

}

void CloudBlock()
{
	cloud_state.block_x = 50.0;
	srand(time(0));
	cloud_state.block_y = (rand() % 30) + 50;	//randomly generate block y cordinate
	cloud_state.state = true;
	building_state.state = false;
}

//------------------- Drawing Functions:
void drawBg()
{
	color darkGrass(0.0, 0.48, 0.047), lightGrass(0.0, 0.3, 0.03);
	color lightGrey(44.0 / 255, 62.0 / 255, 80.0 / 255), sunset(253.0 / 255, 116.0 / 255,
108.0 / 255);
```

```
	drawRectGrad(0.0, 0.9, 100.0, 10.0, darkGrass, lightGrass); //Grass
	drawRectGrad(0.0, 100.0, 100.0, 10.0, lightGrey, sunset); //Skybox.
}

void drawScreenBg() {
	color lightGrey(44.0 / 255, 62.0 / 255, 80.0 / 255), sunset(253.0 / 255, 116.0 / 255,
108.0 / 255);
	drawRectGrad(0.0, 0.0, 100.0, 100.0, sunset, lightGrey); //Skybox.
}

void drawBuilding()
{

	color window(1.0, 1.0, 1.0);
	for (int i = 1; i <= building_state.no_floors; i++)
	{
		glPushMatrix();
		glTranslatef(building_state.block_x, 10.0 * i, 0.0);   //base
			drawRectGrad(0.0, 0.0, 15.0, 10.0, buildColor, buildColor); //The
Building Itself.
			drawRectGrad(2.5, 5.0, 5.5, 8.0, window, window); //Left Window
			drawRectGrad(12.5, 5.0, 9.5, 8.0, window, window); //Right Window
		glPopMatrix();
	}

	glPushMatrix();
		glTranslatef(building_state.block_x, 10.0, 0.0);   //base
		drawRectGrad(0.0, 0.0, 15.0, 10.0, buildColor, buildColor);
		drawRectGrad(5.5, 0.0, 9.5, 6.0, window, window);
	glPopMatrix();
}

void drawCloud()
{
	glColor4f(1.0, 1.0, 1.0, 1.0);
	glTranslatef(cloud_state.block_x, cloud_state.block_y, 0.0);
	glutSolidSphere(5, 100, 10);
	glTranslatef(6, -3.0, 0.0);
	glutSolidSphere(5, 100, 10);
	glTranslatef(0, 6.0, 0.0);
	glutSolidSphere(5, 100, 10);
	glTranslatef(6, -3.0, 0.0);
	glutSolidSphere(5, 100, 10);
}

void drawJet()
{
	color aeroGrey(0.8, 0.8, 0.8);
	color dome(0.0, 0.0, 0.0);

	std::vector<float> leftTailX({ 13.0, 17.0, 13.0, 11.0});
	std::vector<float> leftTailY({ 47.0, 47.0, 49.0, 49.0});
	drawPoly(leftTailX, leftTailY, aeroGrey);

	std::vector<float> tailX({ 2.7, 3.0, 5.0, 7.0, });
	std::vector<float> tailY({ 47.0, 51.0, 51.0, 47.0, });
	drawPoly(tailX, tailY, aeroGrey);


	//body -- Special, Hence, Use Normal Primitives.
	glColor3f(3/255, 0.0, 30/255);
	glBegin(GL_POLYGON);
	glVertex2f(2.5, 48.0);
	glVertex2f(6.0, 47.8);
	glColor3f(115 / 255, 3/255, 192 / 255);
	glVertex2f(7.0, 47.6);
	glVertex2f(8.0, 47.4);
	glColor3f(236 / 255, 56 / 255, 188/ 255);
	glVertex2f(11.0, 48.0);
	glVertex2f(22.0, 46.5);
	glColor3f(253 / 239, 56 / 255, 249 / 255);
	glVertex2f(22.0, 45.0);
	glColor3f(3 / 255, 0.0, 30 / 255);
	glVertex2f(2.5, 46.0);
	glEnd();

	std::vector<float> rightFrontWingX({ 13.0, 18.0, 13.0, 11.0, });
	std::vector<float> rightFrontWingY({ 46.0, 46.0, 44.0, 44.0, });
	drawPoly(rightFrontWingX, rightFrontWingY, aeroGrey);

	std::vector<float> domeX({ 13.0, 15.0, 17.0, 19.0, 21.0, 17.0, 15.0, 13.0, });
	std::vector<float> domeY({ 47.0, 48.5, 49.0, 48.0, 46.0, 46.0, 47.5, 47.0, });
	drawPoly(domeX, domeY, dome, 0.3);


	std::vector<float> rightTailX({ 2.5, 5.5, 2.5, 1.5, });
	std::vector<float> rightTailY({ 47.0, 47.0, 45.0, 45.0, });
	drawPoly(rightTailX, rightTailY, aeroGrey);

}

void gameEnd()
{
	drawScreenBg();
	color white(1.0, 1.0, 1.0);
```

```
        glColor3f(0.196, 0.196, 0.8);  // disp box
        glRectf(20.0, 20.0, 80.0, 80.0);
        glColor3f(0.0, 0.0, 0.0);
        glRectf(21.0, 21.0, 79.0, 79.0);

        buttonMake(40, 5, 60, 10, "RESTART", white, SCREENW);

        drawString(41, 71, 0, GLUT_BITMAP_HELVETICA_18, "GAME OVER");
        drawString(23, 61, 0, GLUT_BITMAP_HELVETICA_18, "DISTANCE :" +
std::string(score_Str));
        drawString(23, 56, 0, GLUT_BITMAP_HELVETICA_18,"LEVEL :" +
std::string(slevel));
        glutPostRedisplay();
}

void welcome()
{
        color white(1.0, 1.0, 1.0);

        drawScreenBg();
        buttonMake(39.5, 39.5, 60.5, 45.5, "PLAY", white, SCREENW);
        buttonMake(40, 10, 60, 15, "EXIT", white, SCREENW);

        drawString(29.5, 92, 0, GLUT_BITMAP_HELVETICA_18, "COMPUTER
GRAPHICS PROJECT");
        drawString(35.5, 80, 0, GLUT_BITMAP_HELVETICA_18, "AIRPLANE CRASH");
}

void PlayScreen() {
        if ((int)score % 50 == 0 && lflag == true)// l-level
        {
                lflag = false;
                level++;
        }
        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();
```

```
	score += 0.01;
	if ((building_state.state == true && building_state.block_x < -10) || (cloud_state.state
== true && cloud_state.block_x < -10)) // If Building or Could has gone outside the area.
	{
		srand(time(NULL));
		int random = rand() % 2;//for random building or cloud
		if (random == 0)
		{
			buildingBlock();
		}
		else
		{
			CloudBlock();
		}
	}

	else if (building_state.state == true)
	{
		building_state.block_x -= bspd;
		glTranslatef(building_state.block_x, 0.0, 0.0);
		drawBuilding();
	}
	else if (cloud_state.state == true)
	{
		cloud_state.block_x -= bspd;
		glTranslatef(cloud_state.block_x, 0.0, 0.0);
		drawCloud();
	}

	glPopMatrix();

	printScore();
}

void printScore()
{
	sprintf_s(slevel, "%d", (int)level);
	sprintf_s(score_Str, "%d m", (int)score);


	glColor3f(1.0, 1.0, 1.0);//score
	drawString(58, 1.8, 0, GLUT_BITMAP_HELVETICA_10,
const_cast<char*>("Level"));
	drawString(58, 3.5, 0, GLUT_BITMAP_HELVETICA_18, slevel);
```

```
        drawString(38, 1.5, 0, GLUT_BITMAP_HELVETICA_10,
const_cast<char*>("Distance"));
        drawString(38, 3, 0, GLUT_BITMAP_HELVETICA_18, score_Str);

}

//--------------------Calculating Functions.

bool cloudHit()
{
        if (cloud_state.block_x < 13 && cloud_state.block_x> -5)
                if (plane_mvmt - 3 + 50 > cloud_state.block_y - 3 && plane_mvmt - 3 + 50 <
cloud_state.block_y + 3)   // plane front to cloud mid box1
                        return true;

        if (cloud_state.block_x < 12 && cloud_state.block_x> -4)
                if (plane_mvmt - 3 + 50 > cloud_state.block_y - 5 && plane_mvmt - 3 + 50 <
cloud_state.block_y + 5)   // plane front to cloud mib box2
                        return true;

        if (cloud_state.block_x < 10 && cloud_state.block_x> -1)
                if (plane_mvmt - 3 + 50 > cloud_state.block_y - 6 && plane_mvmt - 3 + 50 <
cloud_state.block_y - 2)
                        return true;

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

        return false;
}

bool buildingHit()
{
        if (((int)building_state.block_x <= 8 && (int)building_state.block_x >= -7 &&
((int)plane_mvmt + 50) - building_state.block_y <= 3))   //buildin back  body to tail
                return true;
        else if (((int)building_state.block_x <= 10 && (int)building_state.block_x >= -5 &&
((int)plane_mvmt + 50) - building_state.block_y <= 0))   //body to tail
                return true;
        else if (((int)building_state.block_x <= 6 && (int)building_state.block_x >= -3 &&
((int)plane_mvmt + 47) - building_state.block_y <= 0))  //right wing
                return true;
        else if (((int)building_state.block_x <= 4 && (int)building_state.block_x >= -4 &&
((int)plane_mvmt + 47) - building_state.block_y <= 3))  //   building back right wing
                return true;
        else
                return false;
}

bool boundHit()
{
        if (plane_mvmt + 50 >= 100 || plane_mvmt + 50 <= 18)   // top and bottom boundary
                return true;
        else
                return false;
}

void moveJetU()      // jet moving up
{
        if (pause == false) {
                plane_mvmt += 0.05;
        }
        glutPostRedisplay();
}

void moveJetD()         // jet moving down
{
        if (pause == false && gameEndStatus == false && welcome_flag == false)  {
                plane_mvmt -= 0.05;
        }
        glutPostRedisplay();
}

void mouse(int button, int state, int x, int y)           // takes input from mouse
{
        int mx = x * 100 / SCREENW, my = (SCREENH - y) * 100 / SCREENH;            // m
= mouse cordinate to graphics

        /*                  mouse calculation//converting to screen coordinates-ortho values
        SCREENSIZE  ---->  ORTHO
        x(reqd val) ---->  ???
        */
        if (gameEndStatus)
        {
                if (mx > 40 && mx < 60 && my > 5 && my < 10) //Co-ordinates for the
Button.
                {
                        welcome_flag = true;
                        gameEndStatus = false;
                        plane_mvmt = 0;
                        start = false;
                        init();
                        bspd = BLOCKSPEED;//restarting the game
                        score = 1;
                        level = 1;
                        glutPostRedisplay();

                }
        }
        else if (welcome_flag == true)
        {
                if (mx > 40 && mx < 60) //Line of Options:
                {
                        if (my > 40 && my < 45) //Start
                        {
                                start = true;
                                welcome_flag = false;
                        }
                        else if (my > 10 && my < 15) //Exit
                        {
                                glutLeaveMainLoop();
                        }

                }
        }
        else // For Playing the game.
        {
                if (button == GLUT_LEFT_BUTTON)
                {
                        if (state == GLUT_DOWN)
                                glutIdleFunc(moveJetU);

                        else if (state == GLUT_UP)
                                glutIdleFunc(moveJetD);
                }
        }
}

void keyPressed(unsigned char key, int x, int y) // int x and y are mouse pos at time of press
{
        if (key == 27)
        {
                exit(0);
        }
        glutPostRedisplay();
}

void myReshape(int w, int h)
{
        SCREENH = h, SCREENW = w;
        printf("width = %d\theight= %d", w, h);
```

```
	glViewport(0, 0, w, h);
	glMatrixMode(GL_PROJECTION);
	glLoadIdentity();
	glOrtho(0.0, 100.0, 0.0, 100.0, -5.0, 10.0);
	glMatrixMode(GL_MODELVIEW);
}

//------------------------ Main Stuff
void display()
{
	glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);

	if ((gameEndStatus == true) ||
		(building_state.state == true && buildingHit() == true) || (boundHit() == true)
||
		(cloud_state.state == true && cloudHit() == true)
		)
	{
		gameEndStatus = true;
		gameEnd();
	}
	else if (welcome_flag == true)//Welcome Screen
	{
		welcome();
	}
	else
	{
		PlayScreen();
	}
	//glFlush();
	glutSwapBuffers();
}

void init()
{
	srand(time(0));
	int random = rand() % 2;
	if (random == 0)
	{
		buildingBlock();
	}
	else
	{
		CloudBlock();
	}

}
```

```
int main(int argc, char** argv)
{
	glutInit(&argc, argv);
	glutInitDisplayMode(GLUT_DOUBLE | GLUT_RGB | GLUT_DEPTH);
	glutInitWindowSize(SCREENW, SCREENH);
	glutInitWindowPosition(50, 50);
	glutCreateWindow("Airplane Crash");
	glHint(GL_POLYGON_SMOOTH_HINT, GL_NICEST);
	glHint(GL_FOG, GL_NICEST);
	glEnable(GL_LINE_SMOOTH);
	glBlendFunc(GL_SRC_ALPHA, GL_ONE_MINUS_SRC_ALPHA);
	glEnable(GL_BLEND);
	init();

	glutDisplayFunc(display);
	glutReshapeFunc(myReshape);
	glutMouseFunc(mouse);
	glutKeyboardFunc(keyPressed);
	glutMainLoop();
	return 0;

}
```

CHAPTER 5

DISCUSSION AND SNAPSHOTS

5.1 DISCUSSION

- The Object of the game is to fly as far as possible without colliding with any objects on the given plane (Such as the ground, skybox, buildings and clouds)
- Holding the Mouse button Makes the plane rise up and leaving it makes the plane go down.
- If the plane collides with any obstacle, then the game is over, and a score is displayed.
- The Project can be illustrated.
- Collision detection is done via a simple model of proximity detection and model co-ordinates being known during state transition.
- There is no timer function used, this is the same method used in NES and older games that depended on the processor speed to emulate the game. Likewise, this is also processor speed dependent.

5.2 SCREEN SNAPSHOTS

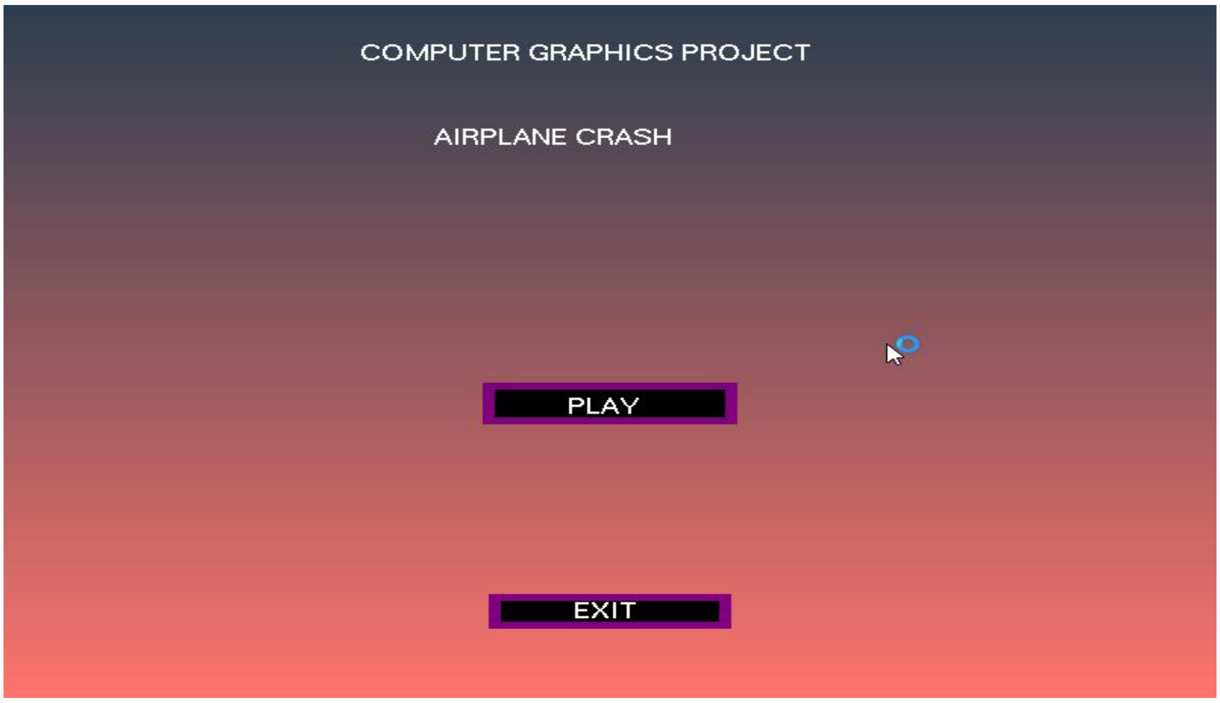

Fig 4.1 The Welcome Screen

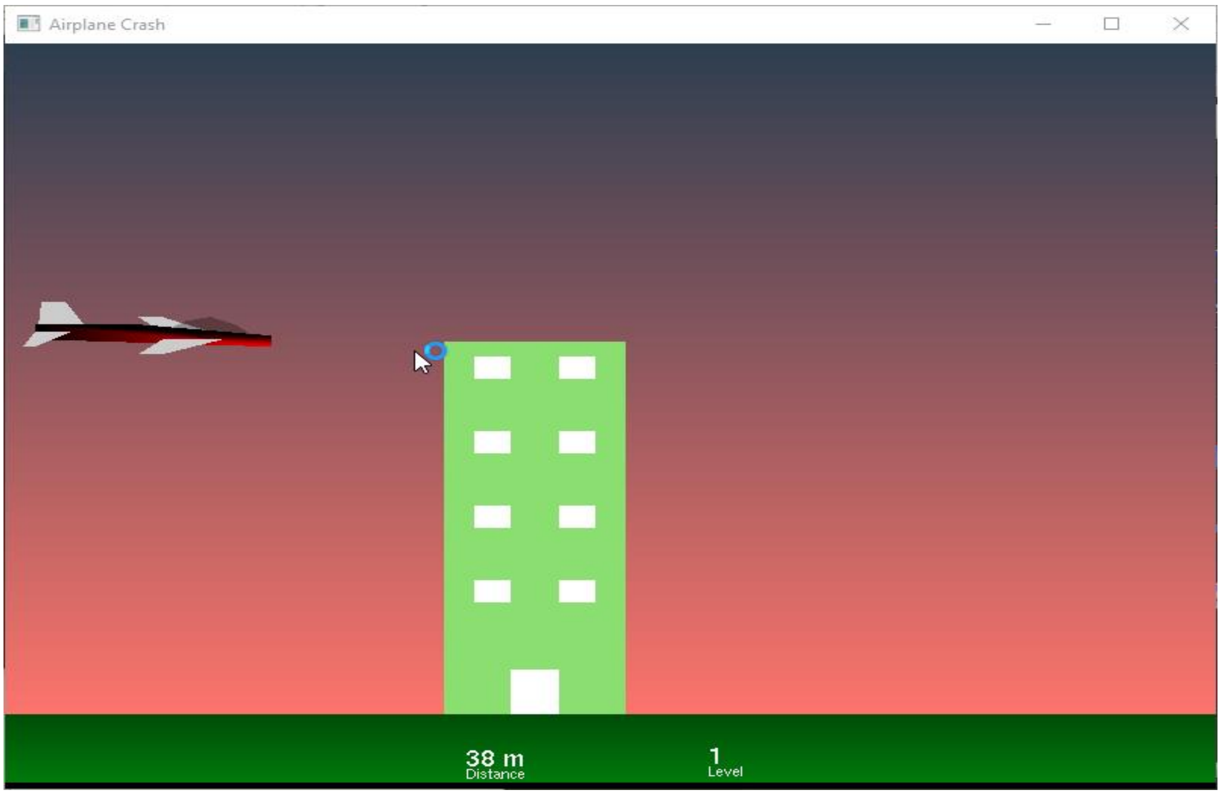

Fig 4.2 Playing Screen

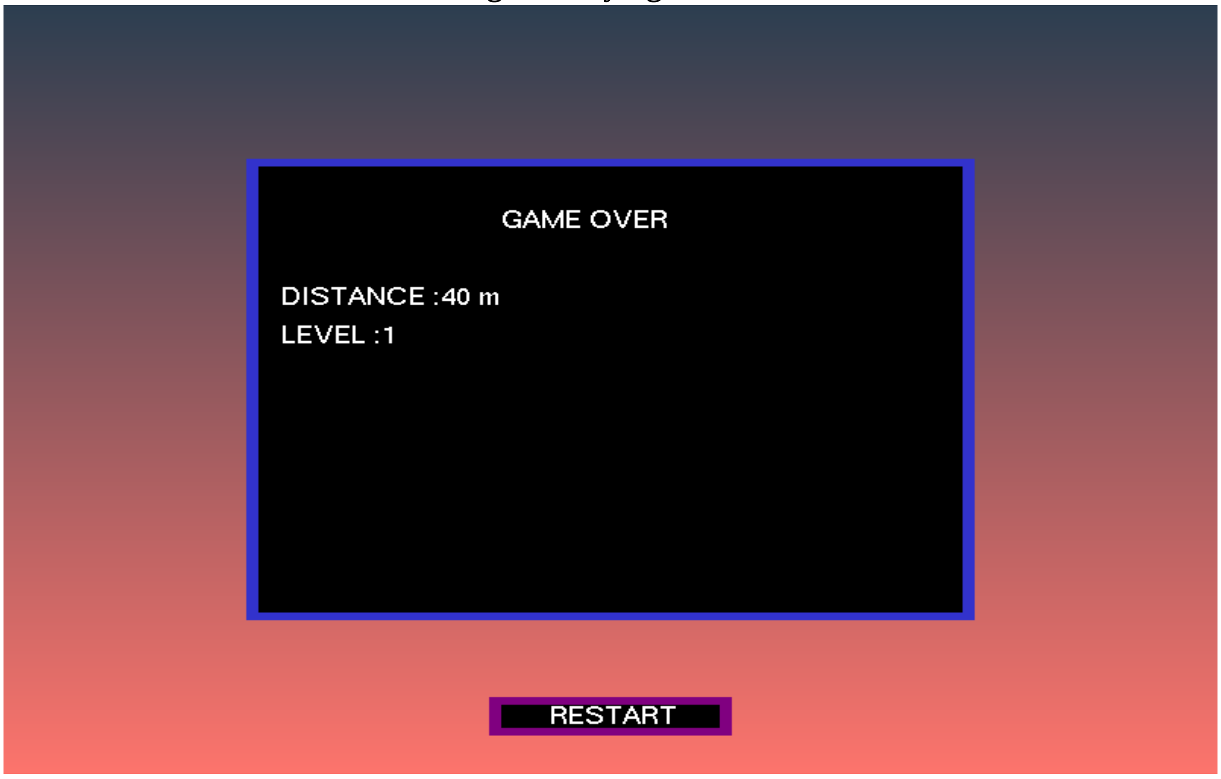

Fig 4.3 Game Over Screen

CHAPTER 6

CONCLUSION AND FUTURE SCOPE

The OpenGL libraries create a lot of future scope for the project, since, being a state machine means that the project can be expanded. In the future, as we learn more of the world that graphics has to offer, I believe we can improve on this project.

A lot of the original scope of this project involved mechanisms for fuel tracking, picks and a lot more, inspired by the original NES game from ’89 era, but due to the constraints on limited options to pick and draw each frame of code, we are inevitably thankful that we were presented this opportunity to learn.

The original NES game, also contained smoother controls, even in that era, 24 frames per second were cleanly animated and maintained to have a consistent frame-rate. The original NES titles were designed with careful consideration of state updating, even in those 24 frames, and this is the pinnacle of graphic design, in our eyes. We would like to eventually reach this state of perfection within our little game created here today.

We would like to conclude by saying that openGL enabled us to do a lot of work in understanding the world of graphics.