CHAPTER 1

INTRODUCTION

• Overview of Computer Graphics

Computer Graphics is concerned with all aspects of producing pictures or images using computer. These are intentionally fashioned to achieve some visual effects. Computer Graphics also refers to the tools used to make such pictures. There are both hardware and software tools.

Hardware tools include video monitors and printers that display graphics, and input devices like a mouse or track ball. Software tools are editors, compilers, debugger found in any programming environment.

Interactive computer graphics is the most important means of producing pictures since the invention of photography and television. A picture is worth ten thousand words. Interactive computer graphics thus permits extensive, high-bandwidth user-computer interaction. Computer graphics concerns with the pictorial synthesis of real or imaginary objects from their computer based models, whereas the related field of image processing treats the converse process, the analysis of scenes, or the reconstruction of models of 2D or 3D objects from their pictures.

Computer Graphics has 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 Computer Graphics so widespread.

1.2 Software Requirements

- Operating System: Windows 2000, XP, Linux Versions, Windows 7/8
- Compiler: GCC Compiler integrated with glut package
- Graphics Library: (GL/glut.h) header files
- Library Files: glut32.dll and glut64.dll for running the application.

1.3 Hardware Requirements

- Processor – Intel 486/Pentium Processor
- RAM – Minimum 64 MB
- ROM (Storage space) – Minimum 2 MB

CHAPTER – 2

INTRODUCTION TO OPENGL

2.1 Introduction

OpenGL (Open Graphics Library) is a standard specification defining a cross-language, cross-platform API for writing applications that produce 2D and 3D computer graphics. OpenGL is not a programming language it is an API (Application Programming Interface). The interface consists of many different function calls which can be used for building application programs. OpenGL is widely used in CAD, virtual reality, scientific visualization, information visualization, and flight simulation.

OpenGL is portable to many platforms (windows, MAC, UNIX, LINUX) and callable from many programming languages (C/C++, JAVA, PEARL). OpenGL is primarily concerned with modeling and rendering operations such as, to specify geometric primitives (lines, pixels, polygons…), apply geometric transformations and specify camera light, color, texture information etc.

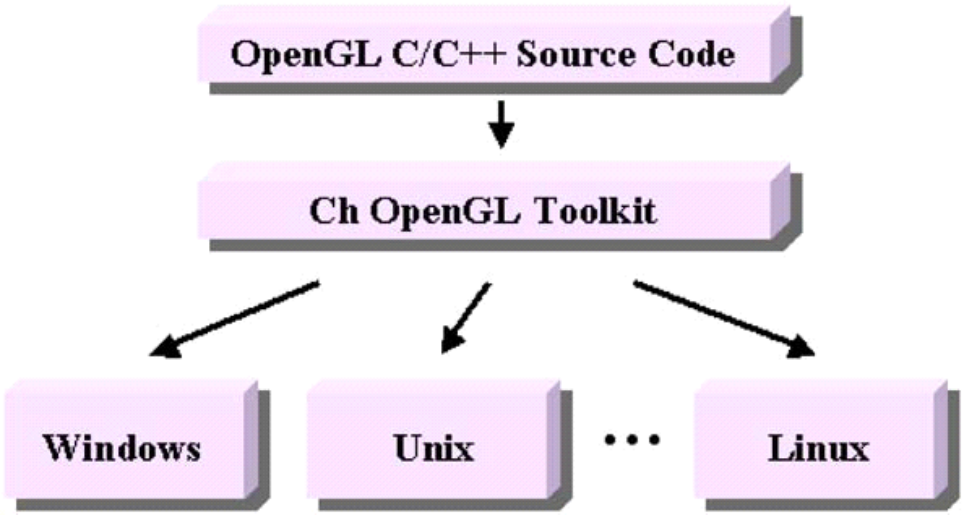


Fig 2.1: Displaying Compatibility of OpenGL in other Platforms

All functions in OpenGL library are device independent, many operations (windowing, I/O etc) are not included in basic core library, and so different auxiliary libraries are developed for

features not available in OpenGL. OpenGL libraries are OpenGL: GL (lib GL) and GLU (lib GLU).

These are windows native implementation (OpenGL32) and Mesa3D: free ware implementation for LINUX. GLUT: OpenGL utility tool kit (lib glut) the GLUT library is responsible for window and menu management, mouse and keyboard interactions.

Graphics libraries are GLUI: GLUT based user interface library (lib glut) -It controls OpenGL applications such as buttons, check boxes, radio buttons etc.

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 "glue" 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 agl. 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.

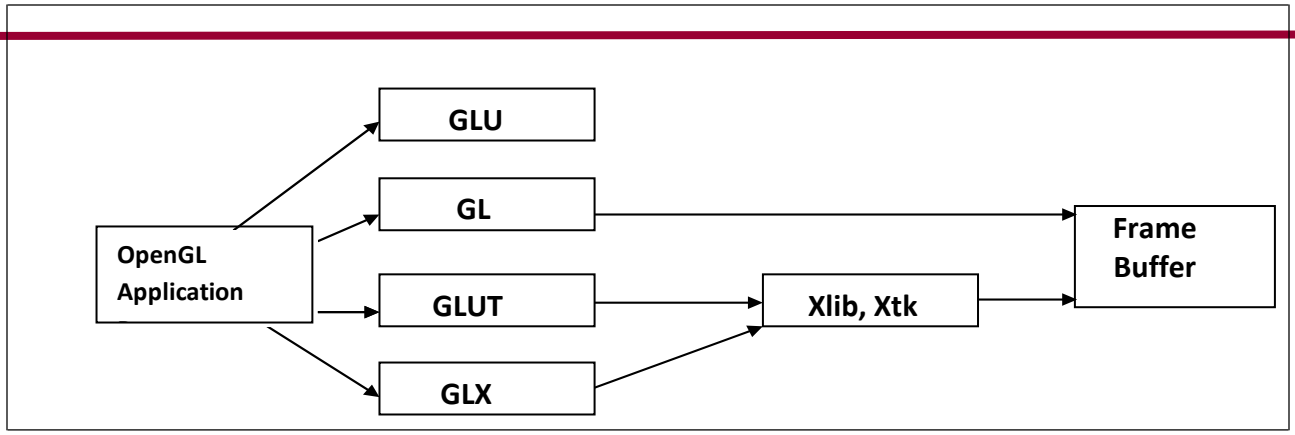


Fig 2.2: Library organization

Figure 2.2 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.

OpenGL makes heavy use of macros to increase code readability and avoid the use of magic numbers. Thus strings such as GL_FILL and GL_POIINTS are defined in header files. In most implementations, one of the include lines

#include<GL/glut.h>

Or

#include<GLUT/glut.h>

is sufficient to read in glut.h, gl.h, and glu.h.

2.2 Why OpenGL

The intention of this chapter is to give you basic concepts in OpenGL. OpenGL is a device and operating system independent library for 3D-graphics and rendering. OpenGL was originally developed by Silicon Graphics Inc (SGI) for use on their high end graphics workstations. Since then, OpenGL has become a widely accepted standard with implementations on many operating system and hardware platforms including Windows NT and Windows X operating systems. The purpose of the OpenGL library is to render two and three-dimensional objects into frame buffer. OpenGL is a library of high-quality three-dimensional graphics and rendering functions. The library's device- and platform-independence make it a library of choice for developing portable graphical applications.

OpenGL drawings are constructed from primitives; primitives are simple items such as lines or polygons, which in turn are composed of vertices.

The OpenGL Library assembles primitives from vertices while taking into account a variety of settings, such as color, lighting, and texture. Primitives are then processed in accordance with transformations, clipping settings, and other parameters; at the end of the rasterization, process is pixel data deposited into a frame buffer.

Because of high visual quality and performance, any visual computing application requiring maximum performance-from 3D animation to CAD to visual simulation-can exploit high-quality, high-performance OpenGL capabilities. These capabilities allow developers in diverse markets such as broadcasting, CAD/CAM/CAE, entertainment, medical imaging, and virtual reality to produce and display incredibly compelling 2D and 3D graphics.

2.3 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

BASIC CONCEPTS OF 3-D TRANSFORMATION

3.1 Overview

Transformation are fundamental part of the computer graphics. Transformation are the movement of the object in Cartesian plane.

Types of 3D transformation:

- Translation
- Rotation
- Scaling
- Shearing
- Mirror Reflection

Transformation are used to position objects, to shape objects, to change viewing positions, and even how something is viewed.

When the transformation takes place on a 3D plane, it is called 3D transformation. Generalize from 2D by including Z-coordinate. Straight forward for translation and scale, rotation more difficult.

In 3D translation, we transfer the Z coordinate along with the X and Y coordinates. The process for translation in 3D is similar to 2D translation. A translation moves an object into a different position on the screen.

3D rotation is not same as 2D rotation. In 3D rotation, we have to specify the angle of rotation along with the axis of rotation. We can perform 3D rotation about X, Y, and Z axes.

You can change the size of an object using scaling transformation. In the scaling process, you either expand or compress the dimensions of the object. Scaling can be achieved by multiplying the original coordinates of the object with the scaling factor to get the desired result.

A transformation that slants the shape of an object is called the shear transformation. Like in 2D shear, we can shear an object along the X-axis, Y-axis, or Z-axis in 3D.

3.2 Working

The project AIRSHOW is created to demonstrate OpenGL's concepts. It encompasses some of the skills learnt in our OpenGL classes such as pushmatrix(), translate(), popmatrix(), scale(). The scope is to use the basic primitives defined in OpenGL library creating complex objects. Expected Input will be the keyboard and mouse click events and expected output is the movement and rotational motion of the airplanes.

The project designed has been tested for its working, and is found to be working properly to meet all its requirements.

The project has been found to be giving correct outputs to the inputs that were given, like pressing of left mouse button will rotate the planes, pressing of right mouse button will stop the rotation of the plane and keyboard button R will reset the movement of planes.
The designed project has been tested for errors and has been found to be meeting all the requirements of design with which it was started. Thus the project is declared to be working properly.

CHAPTER – 4

DESIGN AND IMPLEMENTATION

4.1 Built-in Functions

- **glutInit(…);**

  This function defines the interaction between the windowing system and the OpenGL.

  Declaration: glutInit(&argc,argv);

- **glutInitDisplayMode(…);**

  Here we specify RGB color system and also double buffering.

  Declaration: glutInitDisplayMode(GLUT_DOUBLE|GLUT_RGB|GLUT_DEPTH);

- **glutInitWindowSize(…);**

  This function specifies a window in the top left corner of the display.

  Declaration: glutInitWindowSize(600,600);

- **glutCreateWindow("…");**

  This creates an OpenGL window using the glut function where the title at the top of the window is given by the string inside the parameter of the above function.

  Declaration: glutCreateWindow("AIRSHOW");

- **glutReshapeFunc(…);**

  The reshape event is generated whenever the window is resized, such as user interaction.

  Declaration: glutReshapeFunc(reshape);

- **glutDisplayFunc(…);**

Graphics are sent to the screen through a function called the display call back, here the function named ‘func’ will be called whenever the windowing system determines that OpenGL window needs to be redisplayed.

Declaration: glutDisplayFunc(display);

- **glutIdleFunc(…);**

  glutIdle function checks idle call back i.e. it can perform a background processing task or continuous animation when windows system events are not being received.

  Declaration: glutIdleFunc(straight);

- **glutMouseFunc(…);**

  This function handles mouse click events.

  Declaration: glutMouseFunc(mouse);

- **glutKeyboardFunc(…);**

  This function handles keyboard events.

  Declaration: glutKeyboardFunc(keyboard);

- **glViewport(…);**

  Specifies within to height viewport in pixels whose lower left corner is (x,y) measured from origin of the window.

  Declaration: glViewport (0,0,width,height);

- **glMatrixMode(…);**

  This function specifies which matrix will be affected by subsequent transformation in GL_MODELVIEW and GL_PROJECTION.

  Declaration: glMatrixMode (…);

- **glLoadIdentity (…);**

This function sets the current transformation matrix to an identity matrix.

Declaration: glLoadIdentity( );

- **glClearColor(…);**

  Declaration: glClearColor(0.0,0.0,0.0,0.0);

- **glColor3f(…);**

  In RGB color we have three attributes to set. The first is the clear color, which is set to black and we can select the rendering color for points by setting the state of the variable to black by using the following function call.

  Declaration:glColor3f(0.0,0.0,0.0)

4.2 PROJECT CODE

```
#include<windows.h>
#include<stdlib.h>
#include <GL/glut.h>
#include <math.h>
#include<string.h>

//Camera position
static double cx = 100;
static double cy = 50;
static double cz = 100;
//Rotation
static int spinx = 0;
static int spiny = 0;
static int spinz = 0;
```

```
//Position
static double x = 0.0;
static double y = 0.0;
static double z = 0.0;
//Smoke particle rotation
static int spinxs[150];
static int spinys[150];
static int spinzs[150];
//smoke  particle postion
float sx[150];
float sy[150];
float sz[150];
float sa[150];
float ss[150];
//Count



int i = 0,f=0;



//Sphere
void sphere(float r, float g, float b, float a)
{
        glColor4f(r,g,b,a);
        glutSolidSphere(4,32,32);
```

```
}



void smoke(float size, float alpha, float R, float G, float B)
{
        glPushMatrix();
        //Colour and transparency of Smoke
        glColor4f(R,G,B,alpha);
        glTranslatef(0,0,-15);
        glutSolidSphere((1 + size),16,16);
        glPopMatrix();
}

void drawSmoke(float R, float G, float B, float x, float y, int reflect)
{
        // Calculate each position, size and transparency of smoke
        for (int xi = 0; xi < 150; xi++)
        {
                sa[xi] = sa[xi] - 0.0011;
                ss[xi] = ss[xi] + 0.005;
                glPushMatrix();
                glScalef(reflect*0.5,reflect*0.5,0.5);
                glTranslatef((reflect*sx[xi])- x, sy[xi] - y,sz[xi]);
                glRotatef(spinxs[xi],1,0,0);
                glRotatef(reflect*spinys[xi],0,1,0);
```

```
                glRotatef(reflect*spinzs[xi],0,0,1);
                smoke(ss[xi], sa[xi],R,G,B);
                glPopMatrix();
        }
}


void slab(float r, float g, float b)
{
        glColor3f(r,g,b);
        glutSolidCube(1);
}
void bridge()
{
        glPushMatrix();
        glScalef(20,1,10);
        slab(0.5,0.5,0.5);
        glPopMatrix();

        glPushMatrix();
        glScalef(20,2,1);
        glTranslatef(0,0.75,4.5);
        slab(0.3,0.3,0.3);
        glPopMatrix();
```

```
        glPushMatrix();
        glScalef(20,2,1);
        glTranslatef(0,0.75,-4.5);
        slab(0.3,0.3,0.3);
        glPopMatrix();

        glPushMatrix();
        glScalef(1,4,1);
        glTranslatef(0,-0.65,4.5);
        slab(0.3,0.3,0.3);
        glPopMatrix();

        glPushMatrix();
        glScalef(1,4,1);
        glTranslatef(0,-0.65,-4.5);
        slab(0.3,0.3,0.3);
        glPopMatrix();
}
void ground()
{
        glBegin(GL_POLYGON);
        glColor3f(0,0.128,0.0);
        glVertex3f(-100,0,100);
        glVertex3f(0,0,100);
        glVertex3f(-30,0,50);
```

```
glVertex3f(0,0,0);
glVertex3f(-50,0,-100);
glVertex3f(-100,0,-100);
glEnd();

glBegin(GL_POLYGON);
glColor3f(0,0.128,0.0);
glVertex3f(10,0,0);
glVertex3f(-20,0,50);
glVertex3f(10,0,100);
glVertex3f(100,0,100);
glVertex3f(100,0,-100);
glVertex3f(-40,0,-100);
glEnd();

glBegin(GL_POLYGON);
glColor3f(0.0,0.128,0.0);
glVertex3f(0,-10,100);
glVertex3f(-30,-10,50);
glVertex3f(-30,0,50);
glVertex3f(0,0,100);
glEnd();

glBegin(GL_POLYGON);
glColor3f(0.0,0.128,0.0);
```

```
glVertex3f(-30,-10,50);
glVertex3f(0,-10,0);
glVertex3f(0,0,0);
glVertex3f(-30,0,50);
glEnd();

glBegin(GL_POLYGON);
glColor3f(0.4,0.2,0.16);
glVertex3f(0,-10,0);
glVertex3f(-50,-10,-100);
glVertex3f(-50,0,-100);
glVertex3f(0,0,0);
glEnd();

glBegin(GL_POLYGON);
glColor3f(0.4,0.25,0.16);
glVertex3f(-40,-10,-100);
glVertex3f(10,-10,0);
glVertex3f(10,0,0);
glVertex3f(-40,0,-100);
glEnd();

glBegin(GL_POLYGON);
glColor3f(0.4,0.25,0.16);
glVertex3f(10,-10,0);
```

```
        glVertex3f(-20,-10,50);
        glVertex3f(-20,0,50);
        glVertex3f(10,0,0);
        glEnd();

        glBegin(GL_POLYGON);
        glColor3f(0.4,0.25,0.16);
        glVertex3f(-20,-10,50);
        glVertex3f(10,-10,100);
        glVertex3f(10,0,100);
        glVertex3f(-20,0,50);
        glEnd();

//Water
        glBegin(GL_POLYGON);
        glColor4f(0,0.6,0.6,0.0);
        glVertex3f(-100,-2,100);
        glVertex3f(100,-2,100);
        glVertex3f(100,-2,-100);
        glVertex3f(-100,-2,-100);
        glEnd();
}
```

```
//Wings
void wing(int Colour)
{
	float rs=0; //Side red
	float re=0;	//Edge red
	float bs=0;	//Side blue
	float be=0;	//Edge blue
	float gs=0;	//Side green
	float ge=0;	//Edge green

	if (Colour == 1){
		rs = 0.75;
		re = 0.5;
	} else if (Colour == 2){
		gs = 0.75;
		ge = 0.5;
	} else if (Colour == 3){
		bs = 0.75;
		be = 0.5;
	}
//Front
	glBegin(GL_POLYGON);
		glColor3f(re,ge,be);
		glVertex3f(1.5,-1,10);
		glVertex3f(25,-0.25,0);
```

```
                glVertex3f(25,0.25,0);
                glVertex3f(1.5,1,10);
        glEnd();
//Back
        glBegin(GL_POLYGON);
                glColor3f(re,ge,be);
                glVertex3f(1.5,1,-1);
                glVertex3f(25,0.25,-7);
                glVertex3f(25,-0.25,-7);
                glVertex3f(1.5,-1,-1);
        glEnd();
//Top
        glBegin(GL_POLYGON);
                glColor3f(rs,gs,bs);
                glVertex3f(1.5,1,10);
                glVertex3f(25,0.25,0);
                glVertex3f(25,0.25,-7);
                glVertex3f(1.5,1,-1);
        glEnd();
//Bottom
        glBegin(GL_POLYGON);
                glColor3f(rs,gs,bs);
                glVertex3f(1.5,-1,-1);
                glVertex3f(25,-0.25,-7);
                glVertex3f(25,-0.25,0);
```

```
                glVertex3f(1.5,-1,10);
        glEnd();
//End
        glBegin(GL_POLYGON);
                glColor3f(re,ge,be);
                glVertex3f(25,-0.25,0);
                glVertex3f(25,-0.25,-7);
                glVertex3f(25,0.25,-7);
                glVertex3f(25,0.25,0);
        glEnd();
}
//Fin
void fin(int Colour)
{
        float rs=0;
        float re=0;
        float bs=0;
        float be=0;
        float gs=0;
        float ge=0;

        if (Colour == 1){
                rs = 0.75;
                re = 0.5;
        } else if (Colour == 2){
```

```
                gs = 0.75;
                ge = 0.5;
        } else if (Colour == 3){
                bs = 0.75;
                be = 0.5;
        }
//Front
        glBegin(GL_POLYGON);
                glColor3f(re,ge,be);
                glVertex3f(-0.5,2,-7);
                glVertex3f(0.5,2,-7);
                glVertex3f(0,9.5,-10);
        glEnd();
//Back
        glBegin(GL_POLYGON);
                glColor3f(re,ge,be);
                glVertex3f(0,9.5,-12);
                glVertex3f(0.5,2,-11);
                glVertex3f(-0.5,2,-11);
        glEnd();
//Side A
        glBegin(GL_POLYGON);
                glColor3f(rs,gs,bs);
                glVertex3f(0.5,2,-7);
                glVertex3f(0.5,2,-11);
```

```
                glVertex3f(0,9.5,-12);
                glVertex3f(0,9.5,-10);
        glEnd();
//Side B
        glBegin(GL_POLYGON);
                glColor3f(rs,gs,bs);
                glVertex3f(0,9.5,-10);
                glVertex3f(0,9.5,-12);
                glVertex3f(-0.5,2,-11);
                glVertex3f(-0.5,2,-7);
        glEnd();
}
void plane(int Colour)
{
        float rb=0;
        float gb=0;
        float bb=0;
        //Selects which colour plane to display
        if (Colour == 1){
                rb = 0.65;
        } else if (Colour == 2){
                gb = 0.65;
        } else if (Colour == 3){
                bb = 0.65;
        }
```

```
//Body
glPushMatrix();
glTranslatef(0,0,4);
glScaled(1,0.8,5);
sphere(rb,gb,bb,1);
glPopMatrix();
//Windscreen
glPushMatrix();
glTranslatef(0,2,16);
glScaled(0.5,0.4,0.75);
sphere(0,0,0,0.75);
glPopMatrix();
//Left Wing
wing(Colour);
//Right Wing
glPushMatrix();
glScalef(-1,-1,1);
wing(Colour);
glPopMatrix();
//Left mini wing
glPushMatrix();
glScalef(0.4,0.4,0.4);
glTranslatef(0,3,-25);
wing(Colour);
glPopMatrix();
```

```
        //Right mini wing
        glPushMatrix();
        glScalef(-0.4,-0.4,0.4);
        glTranslatef(0,-3,-25);
        wing(Colour);
        glPopMatrix();
        //Fin
        fin(Colour);
}

void cloud()
{
        glPushMatrix();
        glScalef(1.5,1,1.25);
        glTranslatef(0,12,0);
        sphere(1,1,1,0.9);
        glPopMatrix();

        glPushMatrix();
        glScalef(1.5,1,1.25);
        glTranslatef(0,5,3);
        sphere(1,1,1,0.9);
        glPopMatrix();

        glPushMatrix();
```

```
        glScalef(1.5,1,1.25);
        glTranslatef(4,7,0);
        sphere(1,1,1,0.9);
        glPopMatrix();

        glPushMatrix();
        glScalef(1.5,1,1.25);
        glTranslatef(-4,7,0);
        sphere(1,1,1,0.9);
        glPopMatrix();

        glPushMatrix();
        glScalef(2,1.5,2);
        glTranslatef(0,5,0);
        sphere(1,1,1,0.5);
        glPopMatrix();
}

/* reshape callback function
  executed when window is moved or resized */
void reshape(int width, int height)
{
  glViewport(0, 0, width, height);
  glMatrixMode(GL_PROJECTION);
  glLoadIdentity();
```

```
  gluPerspective(50.0,1.0,15.0,600.0);  //Perspective
  glMatrixMode(GL_MODELVIEW);
  glLoadIdentity();
}



/* display routine this where the drawing takes place  */
void display1(void)
{

  glClear (GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);   /* clear window */
  glMatrixMode(GL_MODELVIEW);
  glLoadIdentity();

 gluLookAt(cx, cy, cz, x/2, y/2, z/2, 0.0, 1.0, 0.0);  //position of the eye
//Calculate position and rotation of current smoke particle
  sx[i] = x;
  sy[i] = y;
  sz[i] = z;
  spinxs[i] = spinx;
  spinys[i] = spiny;
  spinzs[i] = spinz;
  sa[i] = 1;
  ss[i] = 0;
```

```
 //Bridge
for (int b = 0; b < 600; b=b+20)
{
glPushMatrix();
glTranslatef(-300+b,-47,0);
bridge();
glPopMatrix();
}
//Bridge Reflection
glPushMatrix();
glScalef(-1,-1,1);
glTranslatef(0,56,0);
bridge();
glPopMatrix();
glPushMatrix();
glScalef(-1,-1,1);
glTranslatef(-20,56,0);
bridge();
glPopMatrix();


//Red plane reflection
glPushMatrix();
glScalef(-0.5,-0.5,0.5);
glTranslatef(-x+60,y+130,z);
glRotatef(spinx,1,0,0);
```

```
glRotatef(-spiny,0,1,0);
glRotatef(-spinz,0,0,1);
plane(1);
glPopMatrix();

glPushMatrix();
glScalef(-0.5,-0.5,0.5);
glTranslatef(-x+120,y+130,z);
glRotatef(spinx,1,0,0);
glRotatef(-spiny,0,1,0);
glRotatef(-spinz,0,0,1);
plane(1);
glPopMatrix();

glPushMatrix();
glScalef(-0.5,-0.5,0.5);
glTranslatef(-x-120,y+130,z);
glRotatef(spinx,1,0,0);
glRotatef(-spiny,0,1,0);
glRotatef(-spinz,0,0,1);
plane(1);
glPopMatrix();

//Green plane reflection
glPushMatrix();
```

```
glScalef(-0.5,-0.5,0.5);
glTranslatef(-x,y+130,z);
glRotatef(spinx,1,0,0);
glRotatef(-spiny,0,1,0);
glRotatef(-spinz,0,0,1);
plane(1);
glPopMatrix();

//Blue plane reflection
glPushMatrix();
glScalef(-0.5,-0.5,0.5);
glTranslatef(-x-60,y+130,z);
glRotatef(spinx,1,0,0);
glRotatef(-spiny,0,1,0);
glRotatef(-spinz,0,0,1);
plane(1);
glPopMatrix();
//Smoke trails for reflective planes
drawSmoke(1,0.5,0.2,-60,-130,-1);
drawSmoke(1,0.5,0.2,-120,-130,-1);
drawSmoke(1,1,1,0,-130,-1);
drawSmoke(0,0.64,0.0,60,-130,-1);
drawSmoke(0,0.64,0.0,120,-130,-1);
```

```
//Ground
glPushMatrix();
glScalef(3,1,3);
glTranslatef(0,-50,0);
ground();
glPopMatrix();
//yellow plane
glPushMatrix();
glScalef(0.5,0.5,0.5);
glTranslatef(x-120,y,z);
glRotatef(spinx,1,0,0);
glRotatef(spiny,0,1,0);
glRotatef(spinz,0,0,1);
plane(1);
glPopMatrix();

glPushMatrix();
glScalef(0.5,0.5,0.5);
glTranslatef(x+120,y,z);
glRotatef(spinx,1,0,0);
glRotatef(spiny,0,1,0);
glRotatef(spinz,0,0,1);
plane(1);
glPopMatrix();
```

```
 //Red plane
 glPushMatrix();
 glScalef(0.5,0.5,0.5);
 glTranslatef(x-60,y,z);
 glRotatef(spinx,1,0,0);
 glRotatef(spiny,0,1,0);
 glRotatef(spinz,0,0,1);
 plane(1);
 glPopMatrix();

//Green plane
 glPushMatrix();
 glScalef(0.5,0.5,0.5);
 glTranslatef(x,y,z);
 glRotatef(spinx,1,0,0);
 glRotatef(spiny,0,1,0);
 glRotatef(spinz,0,0,1);
 plane(1);
 glPopMatrix();

//Blue Plane
 glPushMatrix();
 glScalef(0.5,0.5,0.5);
```

```
 glTranslatef(x+60,y,z);
 glRotatef(spinx,1,0,0);
 glRotatef(spiny,0,1,0);
 glRotatef(spinz,0,0,1);
 plane(1);
 glPopMatrix();
//Smoke trails for planes
 drawSmoke(1,0.5,0.2,120,0,1);
 drawSmoke(1,0.5,0.2,60,0,1);
 drawSmoke(1,1,1,0,0,1);
 drawSmoke(0,0.64,0,-60,0,1);
 drawSmoke(0,0.64,0,-120,0,1);
//Increase count
        i++;
//Reset count
        if (i > 149) i = 0;



//Clouds
 glPushMatrix();
 glTranslatef(-40,10,0);
 glScalef(2,1.5,2);
 cloud();
 glPopMatrix();
```

```
glPushMatrix();
glTranslatef(40,10,0);
glScalef(1.5,1,1.5);
cloud();
glPopMatrix();

glPushMatrix();
glTranslatef(0,20,-70);
glRotatef(45,0,1,0);
glScalef(1,1,1);
cloud();
glPopMatrix();

glPushMatrix();
glTranslatef(30,20,70);
glRotatef(45,1,0,0);
glScalef(0.5,0.5,0.5);
cloud();
glPopMatrix();

glPushMatrix();
glTranslatef(-70,20,70);
glRotatef(35,0,1,0);
glScalef(2,1,2);
cloud();
```

```
 glPopMatrix();
 glutSwapBuffers();
glFlush();
}

/* graphics initialisation */
void init(void)
{
  glClearColor(0.45,0.8,0.88,0);   /* window will be cleared to sky blue
*/
        glEnable(GL_DEPTH_TEST);
        //Enable Alpha channel
        glEnable(GL_BLEND);
        glBlendFunc (GL_SRC_ALPHA, GL_ONE_MINUS_SRC_ALPHA);
        glAlphaFunc(GL_GREATER, 0./255.);
        glEnable(GL_CULL_FACE);  // Enable back  culling
        glCullFace(GL_BACK);     // Cull back faces

}

void forward(int dir)
{
        float xrad; float yrad; float zrad;
                xrad = (spinx * 3.141592654)/180;
                yrad = (spiny * 3.141592654)/180;
```

```
zrad = (spinz * 3.141592654)/180;
if (spinx <90)
{
        z = z +(dir*(cos(yrad)));
} else if (spinx == 90 || spinx == 270)
{
        z = z;
}else if(spinx <270)
{
        z = z - (dir*(cos(yrad)));
}else
{
        z = z + (dir*(cos(yrad)));
}
x = x + (dir*(sin(yrad)));
if (spiny <90)
{
        y = y - (dir*(sin(xrad)));
} else if (spiny == 90 || spiny == 270)
{
        y = y;
}else if(spiny <270)
{
        y = y + (dir*(sin(xrad)));
}else
```

```
                {
                        y = y - (dir*(sin(xrad)));
                }
}
void straight()
{
        if (spinz >= 360) {
                spinz = 0;
          }
      spinz = (spinz + 10) % 360;
        forward(1);
        glutPostRedisplay();
}

void curve()
{
        if (spiny >= 360) {
                spiny = 0;
          }
      spiny = (spiny + 1) % 360;
        forward(1);
        glutPostRedisplay();

}
void up()
```

```
{
        if (spinx >= 360) {
                spinx = 0;
          }
      spinx = (spinx -1) % 360;
        forward(1);
        glutPostRedisplay();

}
void down()

{
        if (spinx >= 360) {
                spinx = 0;
          }
      spinx = (spinx +1) % 360;
        forward(1);
        glutPostRedisplay();

}

void mouse(int btn, int state, int x, int y)
{
        switch(btn)
```

```
        {
        case GLUT_LEFT_BUTTON:
                if (state == GLUT_DOWN)
                {
                        glutIdleFunc(straight);
                        break;
                }

        case GLUT_RIGHT_BUTTON:
                if (state == GLUT_DOWN)
                {
                        glutIdleFunc(curve);
                        break;
                }
        }
}

void *fonts[]=
{
GLUT_BITMAP_9_BY_15,
GLUT_BITMAP_TIMES_ROMAN_10,
GLUT_BITMAP_TIMES_ROMAN_24,
GLUT_BITMAP_HELVETICA_18,
GLUT_BITMAP_HELVETICA_12
};
```

```
void output(int x, int y, char *string,void *font)
{
  int len, i;

  glRasterPos2f(x, y);
  len = (int) strlen(string);
  for (i = 0; i < len; i++) {
    glutBitmapCharacter(font, string[i]);
  }
}

void front()
{
        glColor3f(0.5,0.2,0.6);
        output(305,130,"BTL INSTITUTE OF TECHNOLOGY AND MANAGEMENT",fonts[3]);
        glColor3f(0.3,0.5,0.8);
        output(369,100,"DEPT. OF COMPUTER SCIENCE & ENGG.",fonts[0]);
        output(369,100,"DEPT. OF COMPUTER SCIENCE & ENGG.",fonts[0]);
        glColor3f(0.8,0.1,0.2);
        output(290,600,"GRAPHICAL IMPLEMENTATION OF AIRSHOW",fonts[2]);
        glColor3f(1.0,0.0,1.0);
        output(430,500,"SUBMITTED BY :",fonts[0]);
        output(400,350,"UNDER THE GUIDANCE OF :",fonts[0]);
        glColor3f(0.3,0.5,0.8);
```

```
        output(180,450,"VINODHINI SKS",fonts[3]);
        output(680,450,"SHANKRAMMA",fonts[3]);
        output(389,300,"Mr. MANJUNATHA REDDY",fonts[3]);
        glColor3f(0.0,0.6,0.3);
        output(180,400,"(1BT17CS053)",fonts[0]);
        output(680,400,"(1BT17CS041)",fonts[0]);
        output(379,250,"Asst. Professor,Dept. of CSE",fonts[0]);
        glColor3f(0.6,0.25,0.0);
        output(367,200,"[ PRESS ANY KEY TO CONTINUE ]",fonts[3]);
}
void menu()
{
        glColor3f(0.8,0.1,0.2);
        output(220,480,"GRAPHICAL IMPLEMENTATION OF AIRSHOW",fonts[2]);
        glColor3f(0.0,0.6,0.3);
        output(300,400,"SELECT AN OPTION",fonts[2]);
        output(300,380,"-----------------",fonts[2]);
        glColor3f(0.3,0.5,0.8);
        output(300,340,"[1] PROCEED",fonts[3]);
        output(300,300,"[2] HELP",fonts[3]);
        output(300,260,"[3] INTRODUCTION",fonts[3]);
        output(300,220,"[b] BACK",fonts[3]);
        output(300,180,"[q] QUIT",fonts[3]);
        glColor3f(0.5,0.2,0.6);
        output(500,60,"[ Dept. of CS&E, BTLIT ]",fonts[0]);
```

```
}
void help()
{
        glColor3f(0.8,0.1,0.2);
        output(170,600,"GRAPHICAL IMPLEMENTATION OF AIRSHOW",fonts[2]);
        glColor3f(0.0,0.6,0.3);
        output(180,560,"=> RIGHT CLICK MOUSE FOR TURN <=",fonts[3]);
        output(180,520,"=> LEFT CLICK MOUSE FOR ROTATION TRANSLATION <=",fonts[3]);
        output(180,480,"=> [U] FOR MOVEUP <=",fonts[3]);
        output(180,440,"=> [D] FOR MOVEDOWN <=",fonts[3]);
        output(180,400,"=> [P] FOR PAUSE <=",fonts[3]);
        output(180,360,"=> [S] FOR START <=",fonts[3]);
        output(180,320,"=> [R] FOR RESET <=",fonts[3]);

        glColor3f(0.3,0.5,0.8);
        output(400,280,"SELECT AN OPTION",fonts[2]);
        output(400,265,"-----------------",fonts[2]);
        output(400,230,"[h] HOME",fonts[3]);
        output(400,190,"[b] BACK",fonts[3]);
        output(400,150,"[q] QUIT",fonts[3]);
        glColor3f(0.5,0.2,0.6);
        output(600,60,"[ Dept. of CS&E, BTLIT ]",fonts[0]);
}
void intro()
{
```

```
        glColor3f(0.8,0.1,0.2);
        output(170,480,"GRAPHICAL IMPLEMENTATION OF AIRSHOW",fonts[2]);
        glColor3f(0.0,0.6,0.3);
        output(160,430,"IN THIS AIRSHOW A GROUP OF AIRPLANE EMITS COLORFUL
SMOKE",fonts[0]);


        output(160,400,"    WHICH PAINTS THE SKY IN TRICOLOR(INDIAN FLAG).",fonts[0]);
        glColor3f(0.3,0.5,0.8);
        output(400,300,"SELECT AN OPTION",fonts[2]);
        output(400,270,"-----------------",fonts[2]);
        output(400,230,"[h] HOME",fonts[3]);
        output(400,190,"[b] BACK",fonts[3]);
        output(400,150,"[q] QUIT",fonts[3]);


        glColor3f(0.5,0.2,0.6);
        output(600,60,"[ Dept. of CS&E, BTLIT ]",fonts[0]);
}
void menuset()
{
        glMatrixMode(GL_PROJECTION);
        glLoadIdentity();
        glOrtho(0, 1000, 0.0, 750,-2000,1500);
        glMatrixMode(GL_MODELVIEW);
        glClear( GL_DEPTH_BUFFER_BIT | GL_COLOR_BUFFER_BIT);
}
void display()
```

```
{


        if(f==0)
        {
                menuset();
                front();


                glutSwapBuffers();
        }
        else if(f==1)
        {
           menuset();
                menu();
                glutSwapBuffers();
        }
        else if(f==3)
        {
                menuset();
                help();
                glutSwapBuffers();
        }
        else if(f==4)
        {
                menuset();
                intro();
```

```
            glutSwapBuffers();
        }
        else
        {
    glClearColor(0.45,0.8,0.88,0.0);
                display1();

        }


}
void keyboardFunc( unsigned char key, int x, int y )
{
        if(f==0)
                f=1;
        else if(f==1)
        {
                switch(key)
                {
                case '1':f=2;break;
                case '2':f=3;break;
                case '3':f=4;break;
                case 'b':
                case 'B':f=0;break;
                case 'q':
```

```
            case 'Q':exit(0);
            }
}
else if(f==2)
{
            switch(key)
            {
            case 'q':
            case 'Q':exit(0);break;
            case 'b':
            case 'B':f=1;break;
            case 'h':
            case 'H':f=0;break;
            case 'r':
            case 'R':
      spinx=0.0;
                  spiny=0.0;
                  spinz=0.0;
      x = 0.0;
         y = 0.0;
                  z = 0.0;
                  for (i = 0; i<150; i++)
                  {
                        sx[i] = x;
                        sy[i] = y;
```

```
                    sz[i] = z;
                    spinxs[i] = spinx;
                    spinys[i] = spiny;
                    spinzs[i] = spinz;
                    sa[i] = 1;
                    ss[i] = 0;
                }
                glutIdleFunc(NULL);
            glutPostRedisplay();
break;
                case 'p':
                case 'P':

        spinx=0.0;
                spiny=0.0;
                spinz=0.0;



            glutIdleFunc(NULL);
            glutPostRedisplay();
break;
            case'u':
                case 'U':
  up();
```

```
                    break;
                    case 'D':
                    case 'd':
                    down();

                    break;
                    case 'S':
                    case 's':
                        glutIdleFunc(straight);
                        break;

            }
}
else if(f==3)
{
            switch(key)
            {
            case 'b':
            case 'B':f=1;break;
            case 'h':
            case 'H':f=0;break;
            case 'q':
            case 'Q':exit(0);
            }
```

```
        }
        else
        {

                switch(key)
                {
                case 'b':
                case 'B':f=1;break;
                case 'h':
                case 'H':f=0;break;
                case 'q':
                case 'Q':exit(0);
                }
        }

  reshape( 1400,700 );
   glutPostRedisplay( );
}


int main(int argc, char** argv)
{

  glutInit(&argc, argv);
  glutInitDisplayMode (GLUT_DOUBLE | GLUT_RGB | GLUT_DEPTH);
```

```
  glEnable(GL_DEPTH_TEST);
  glutInitWindowSize (1400, 700);
  glutInitWindowPosition (0, 0);
  glutCreateWindow ("AIRSHOW");
  init();
  glutDisplayFunc(display);
  glutReshapeFunc(reshape);
  glutMouseFunc(mouse);
  glutKeyboardFunc(keyboardFunc);
  glutMainLoop();
  return 0;
}
```

User Defined Functions

There are eight user defined functions in the source code of <TITLE>

- The **smoke()** function is used to show the color and transparency of smoke that is liberated out of the planes.
- The **wings()** function is used to render the front, back, top, bottom portions of the wings and providing colors to them.
- The **plane()** function is used to form the body of the plane by calling different functions like wing(), fin().
- The **display()** function consists of gluLookAt() which calculates the position of eye and rotation of current smoke particles.
- The **cloud()** function uses functions like PushMatrix() ,Scalef() ,Translatef() , sphere() to render the clouds.
- The **forward()** function is used to calculate the direction of plane by checking for Z-coordinate and Y-coordinate.
- The **up()** function will move the plane up.

- The **down()** function will move the plane down.

4.3 Flow Control of Code

The flow of control in the below flow chart is respected to the Texture Package. For any of the program flow chart is compulsory to understand the program. We consider the flow chart for the texture project in which the flow starts from start and proceeds to the main function after which it comes to the initialization of call back functions and further it proceeds to mouse and keyboard functions, input and calculation functions. Finally, it comes to quit, the end of flow chart.

CALCULATE

READING INPUT STRING AND ANIMATE MOUSE CLICK

MOUSE INTERUPT

KEYBOARD

PRINT OUTPUT

END OF INPUT

MAIN SCREEN DISPLAYED

INITIALIZE CALLBACK FUNCTIONS

MAIN

CHAPTER – 5

Snapshots and Result

Fig 5.1: Start Menu

The snapshot above shows the initial start menu.

Fig 5.2: Menu options

The snapshot above shows the menu containing different options.

Fig 5.3: Help menu

The snapshot above shows help menu in the project.

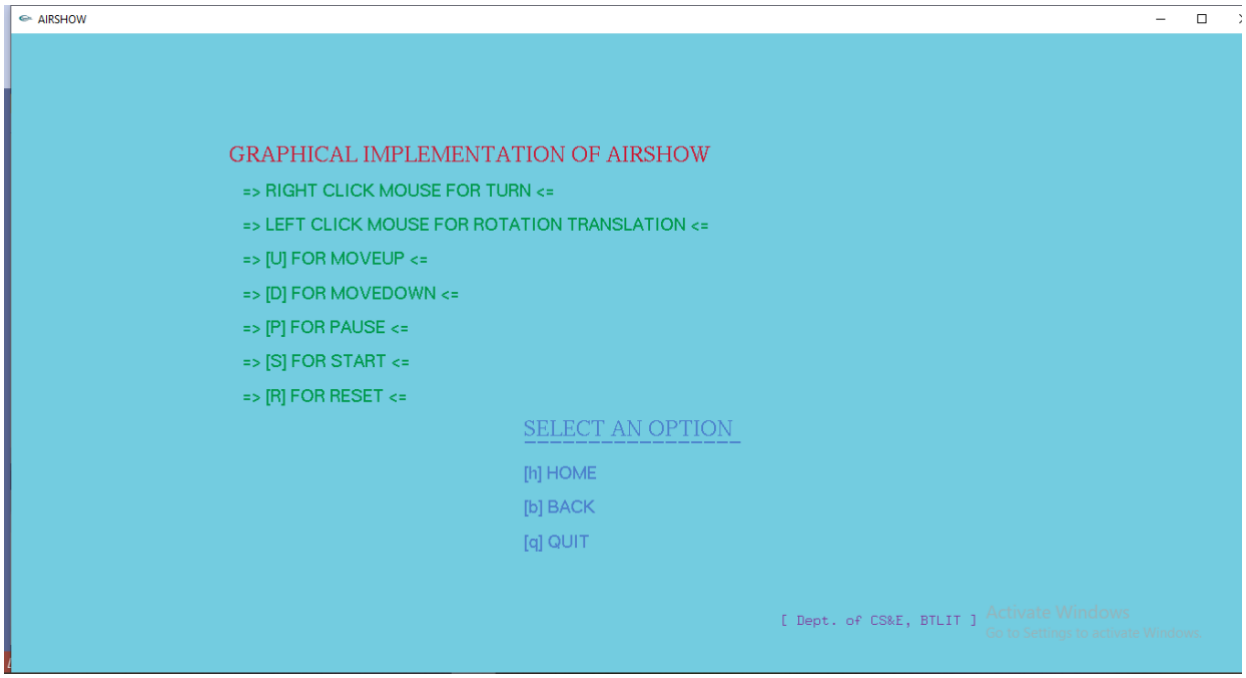


Fig 5.4: Airplanes moving upwards

The snapshot above shows the Airplanes moving upwards. The user is able to toggle this movement by use of [U] key.

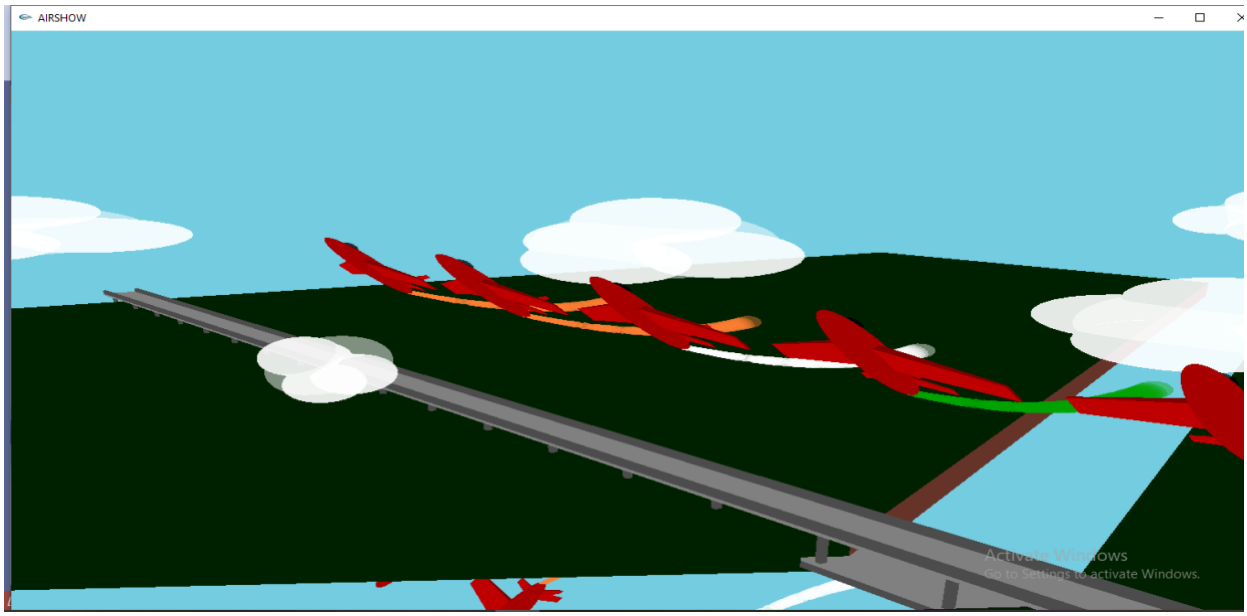


Fig 5.5: Airplanes moving downwards

The snapshot above shows the Airplanes moving downwards. The user is able to toggle this movement by use of [D] key.

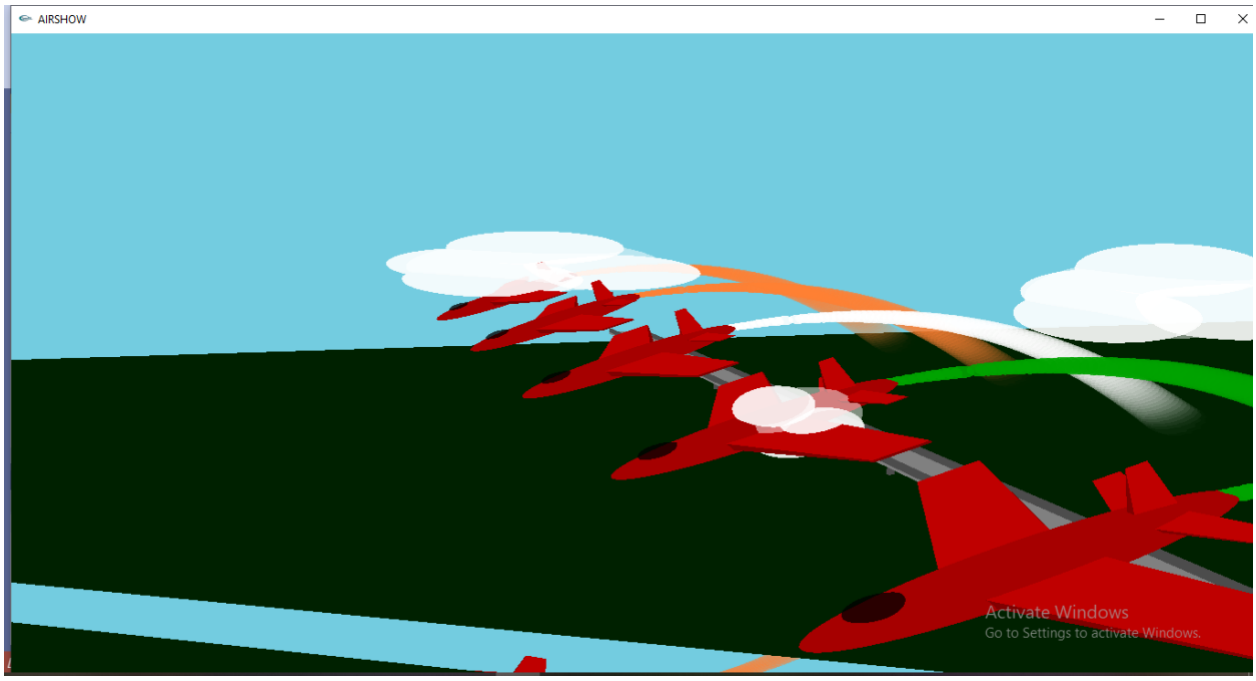


Fig 5.6: Rotation Translation

The snapshot above shows the Airplanes rotation translation. The user is able to toggle this movement by use of mouse left button.

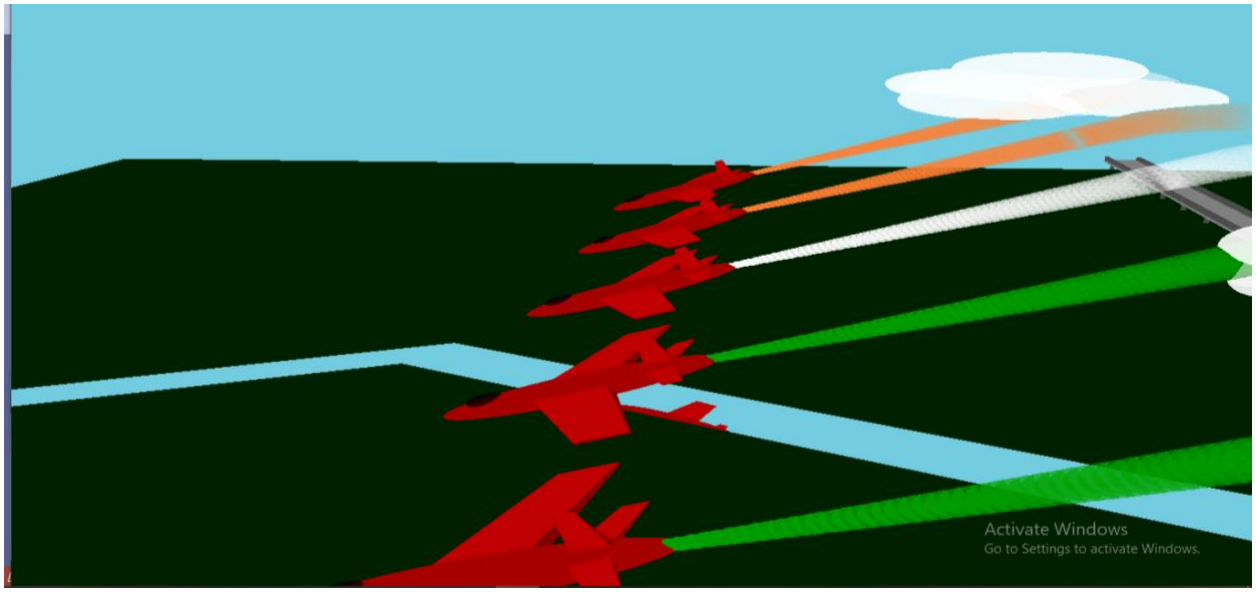
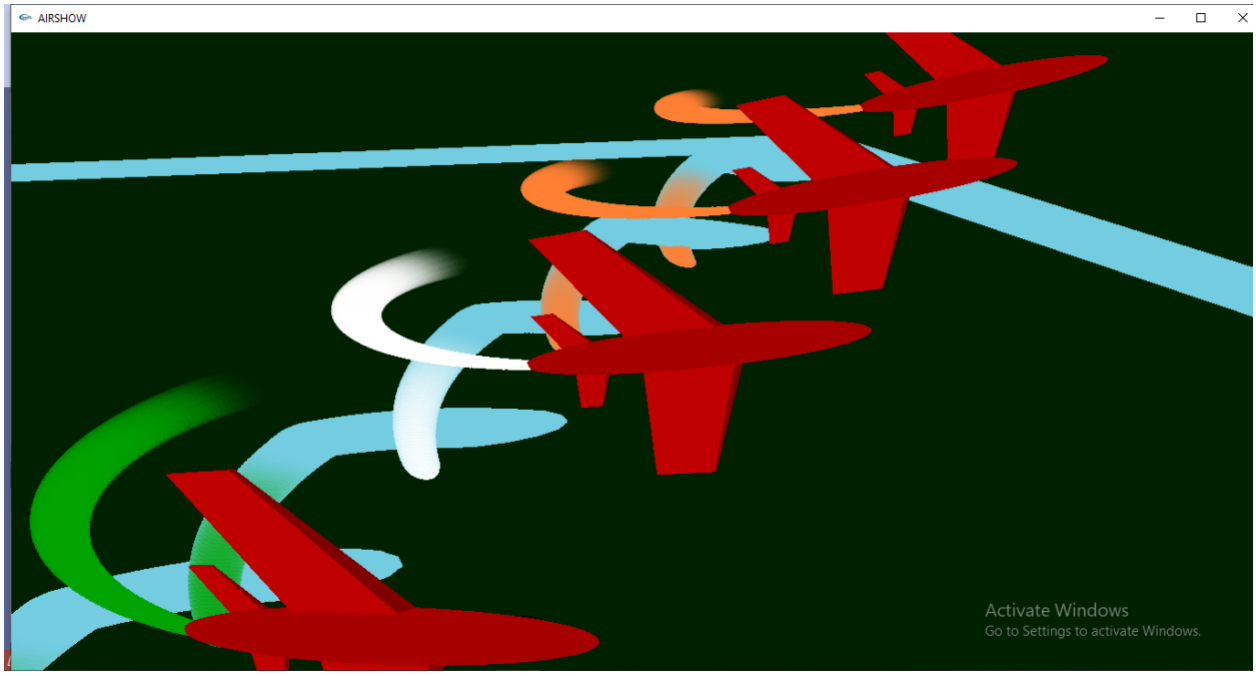
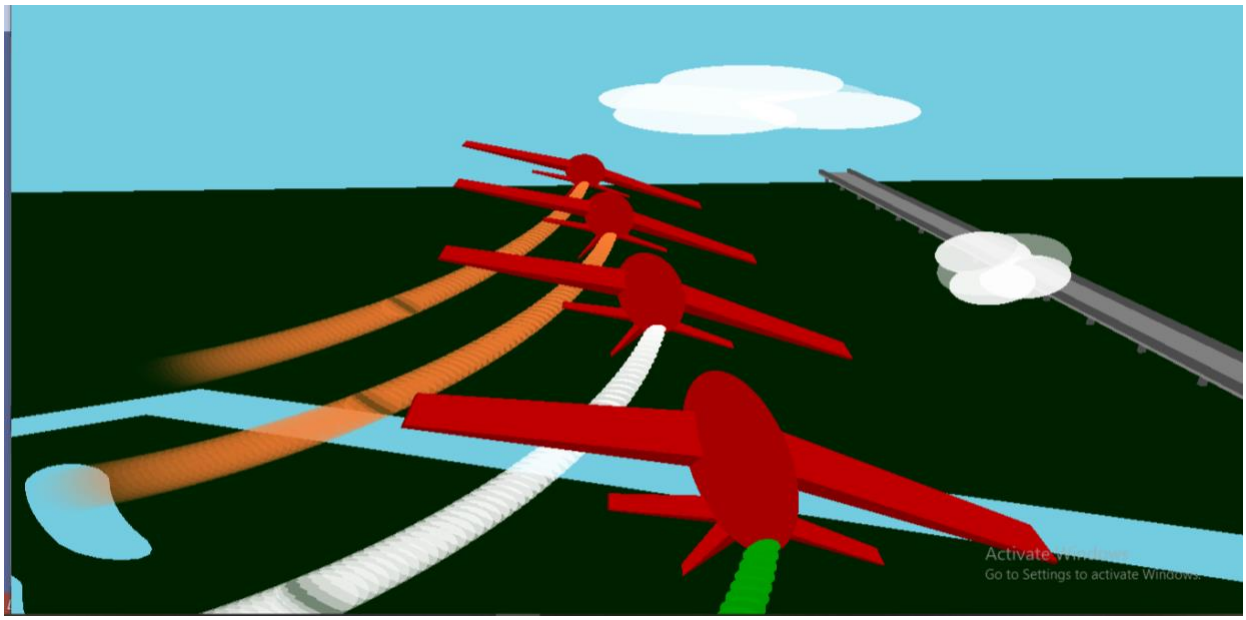


Fig 5.7: Airplane turning

The snapshot above shows the Airplanes turning. The user is able to toggle this movement by use of mouse right button

.





CHAPTER – 6

CONCLUSION AND FUTURE SCOPE

6.1 Conclusion

This mini project on Airshow using OpenGL is a reliable graphics package that provides the user with the basic working of transformation properties in Computer Graphics and the concepts of them as well. By implementing a project using Open GL we came to know how to use the functions like menu‘s, rotation, translation and scaling. Also it allows the user to read a short description of what the project is about. The user-friendly interface allows the user to interact with it very effectively.

6.2 Future Scope

This project can be used for simulation of Republic Day air shows to avoid accidents happening during training by scaling the airplanes in a screen of 1366x768 and using a multiplying factor to check the pattern of air show in real life.

REFERENCES

- Edward Angel –**Interactive Computer Graphics- A Top –Down Approach Using OpenGL,** Addison Wesley Publications 5th Edition in C-2009
- **Computer Graphics Using OpenGL-** F.S. Hill, Jr. 2nd Edition, Pearson Education, 2001.
- Herbert Schildt- C - **The Complete Reference,** Tata McGraw Hill Publications 6th Edition
- **The Official Reference Document for OpenGL** -Addison-Wesley Publishing Company
- www.wikipedia.com
- www.nehe.com
- www.opengl.org
- www.basic4gl.wikispaces.com