**Chapter 1: Introduction**

**1.1 COMPUTER GRAPHICS:**

Computer Graphics is concerned with all aspect of producing pictures or image using computer. The field began humble almost 50 years ago, with the display of few lines on the cathode-ray tube(CRT); now, we can create image using computer that are indistinguishable from photographs from the real objects. We routinely train pilots with simulated airplane, generating graphical display of the virtual environment in the real time. Feature length movies made entirely by computer have been successful, both critically and financially; massive multiplayer game can involve tens of thousands of concurrent participants.

Graphics is created using computers and, more generally, the representation and manipulation of pictorial data by a computer.The development of computer graphics has made computers easier to interact with and better for understanding and interpreting many types of data. Developments in computer graphics have had a profound impact on many types of media and have revolutionized the animation and video game industry. The phrase “Computer Graphics” was coined in 1960 by William Fetter, a graphic designer for Boeing. In today’s world advanced technology, interactive computer graphics has become a powerful tool for the production of realistic features. Today’s we find computer graphics used in various areas that include science, engineering, medicine, business, industry, art, entertainment etc. The main reason for effectiveness of the interactive computer graphics is the speed with which the user can understand the displayed information.

The graphics in OpenGL provides a wide variety of built-in function. The computer graphics remains one of the most exciting and rapidly growing computer fields. It has become a common element in user interface, data visualization, TV commercials, motion picture and many other applications. The current trend of computer graphics is to incorporate more physics principles into 3D graphics algorithm to better simulate the complex interactions between objects and lighting environment.

**1.2 Open-GL: History**

OpenGL was developed by **‘Silicon Graphics Inc‘**(SGI) on 1992 and is popular in the gaming industry where it competes with the Direct3D in the Microsoft Windows platform. OpenGL is broadly used in CAD (Computer Aided Design), virtual reality, scientific visualization, information visualization, flight simulation and video games development.

OpenGL is a standard specification that defines an API that is multi-language and multi-platform and that enables the codification of applications that output computerized graphics in 2D and 3D. The interface consists in more than 250 different functions, which can be used to draw complex tridimensional scenes with simple primitives. It consists of many functions that help to create a real world object and an particular existence for an object can be given.

**CHARECTERISTICS:**

* OpenGL is a better documented API.
* OpenGL is also a cleaner API and much easier to learn and program.
* OpenGL has the best demonstrated 3D performance for any API.

**Chapter 2: PROBLEM DEFINITION AND REQUIREMENTS**

**2.1 PROBLEM DEFINITION**

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.

**2.2 REQUIREMENT SPECIFICATION**

1. FUNCTIONAL REQUIREMENTS

Lists of standard library functions that are used are given below. A number of user defined functions also have been used and a summary of those functions follows this list of standard library 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(…);**
* **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:** glClearColor(0.0,0.0,0.0,0.0); glColor3f(0.0,0.0,0.0);

1. **NON-FUNCTIONAL REQUIREMENTS**

The various functions used to implement it are smoke(), drawSmoke(), cloud(), wings(), plane(), display(), forward().

* 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.

**2.3 SYSTEM REQUIREMENTS**

1. Hardware Requirements**:**

* Processor- Intel or AMD(Advanced Micro Devices)
* RAM- 1 GB(minimum)
* Hard Disk- 40MB(minimum)
* Graphics Memory- 128MB
* Mouse
* Keyboard
* Monitor

1. Software Requirements:

* Any Linux based OS.
* KDevelop
* Virtual Box

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.
* Easy to use user interface.

**Chapter 3: Design and Implementation**

**3.1 User Interface Design**

* **glutMouseFunc(…);**

This function handles mouse click events.

**Declaration:** glutMouseFunc(mouse);

* **glutKeyboardFunc(…);**

This function handles keyboard events.

**Declaration:** glutKeyboardFunc(keyboard);

**LOW LEVEL DESIGN**

Main()

Init()

Display()

Mouse()

Keyboard()

Bridge(), Plane(), Ground(), Cloud(), DrawSmoke(), translate(), popMatrix()

glutIdleFunc(straight)

pushMatrix(), translate(), popMatrix()

.

**3.2 Various features**

* 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.

**Chapter 4: SOURCE CODE**

#include<stdlib.h>

#include <GL/glut.h>

#include <math.h>

#include<string.h>

static double id = 0;

int s=0;

//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;

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]);

}

}

//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(0.0,0.128,0.0);

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){

bs = 0.75;

be = 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){

bs = 0.75;

be = 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 = 1.0,rb=1.0,bb=1.0;

} 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();

//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;

}

if(id==1)

spinz = (spinz + 10) % 360;

if(s==1)

forward(4);

glutPostRedisplay();

}

void up()

{

if (spinx >= 360)

{

spinx = 0;

}

spinx = (spinx -1) % 360;

forward(4);

glutPostRedisplay();

}

void down()

{

if (spinx >= 360) {

spinx = 0;

}

spinx = (spinx +1) % 360;

forward(4);

glutPostRedisplay();

}

void mouse(int btn, int state, int x, int y)

{

switch(btn)

{

case GLUT\_LEFT\_BUTTON:

if (state == GLUT\_DOWN)

{ id=1;

glutIdleFunc(straight);

break;

}

case GLUT\_RIGHT\_BUTTON:

if (state == GLUT\_DOWN)

{ id=0;

glutIdleFunc(straight);

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 front()

{

glColor3f(0.5,0.2,0.6);

output(365,130," GOGTE INSTITUTE OF TECHNOLOGY",fonts[3]);

glColor3f(0.3,0.5,0.8);

output(375,100,"DEPT. OF COMPUTER SCIENCE & ENGG.",fonts[0]);

glColor3f(0.8,0.1,0.2);

output(300,600,"GRAPHICAL IMPLEMENTATION OF AIRSHOW",fonts[2]);

glColor3f(1.0,0.0,1.0);

output(450,500,"SUBMITTED BY :",fonts[0]);

glColor3f(0.3,0.5,0.8);

output(225,450,"Akshay Iyer",fonts[3]);

output(670,450,"Akshay Kanagali",fonts[3]);

output(180,300,"",fonts[3]);

output(220,400,"(2GI13CS007)",fonts[0]);

output(680,400,"(2GI13CS009)",fonts[0]);

// glColor3f(0.6,0.25,0.0);

output(380,200,"[ PRESS ANY KEY TO CONTINUE ]",fonts[3]);

}

void menu()

{

glColor3f(0.8,0.1,0.2);

output(170,480,"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);

}

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 TO STOP THE ROTATIONAL TRANSLATION <=",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, GIT ]",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, GIT ]",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':

s=0;

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':s=1;

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;

}

**Chapter 5: EXPERIMENTAL RESULTS AND ANALYSIS**

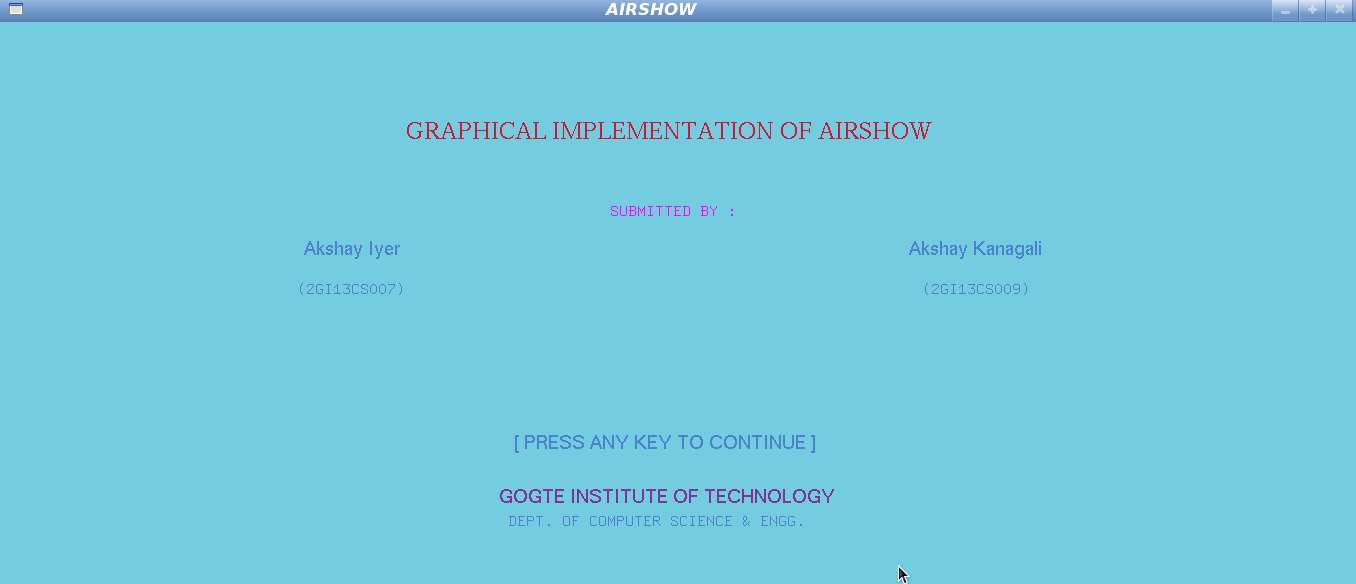
The project designed has been tested for its working, and is found to be working properly to meet all its requirements.

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.

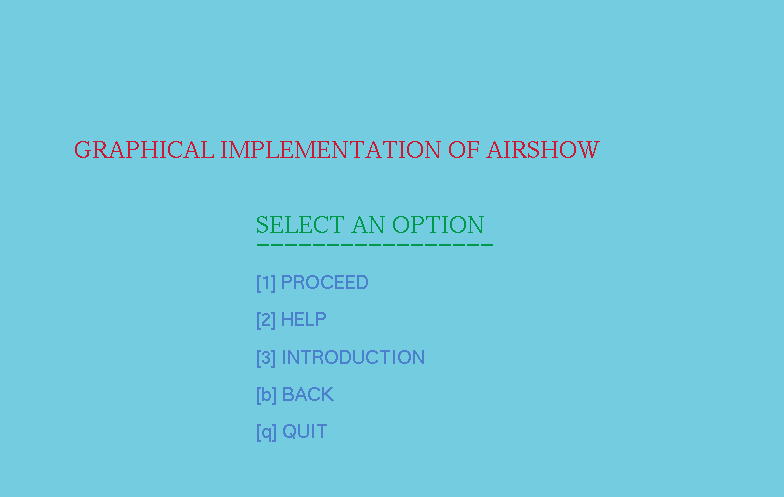
The designed project has been tested for errors and has been found to be meeting all the requirements of design with which it was started. Thus the project is declared to be working properly.

Working of the project is explained through snapshots as follows.

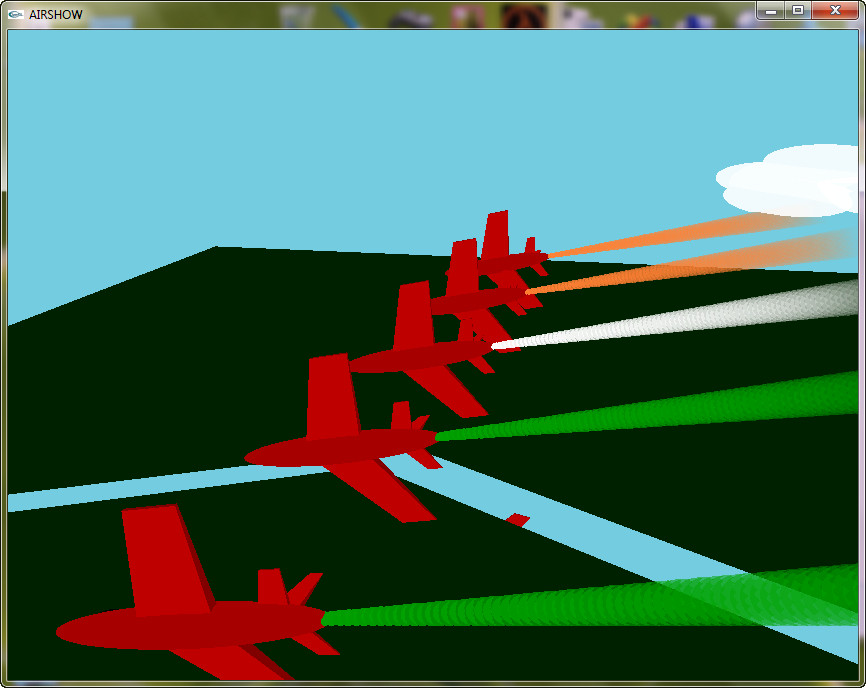
**5.1 SNAPSHOT**



**The Start menu**



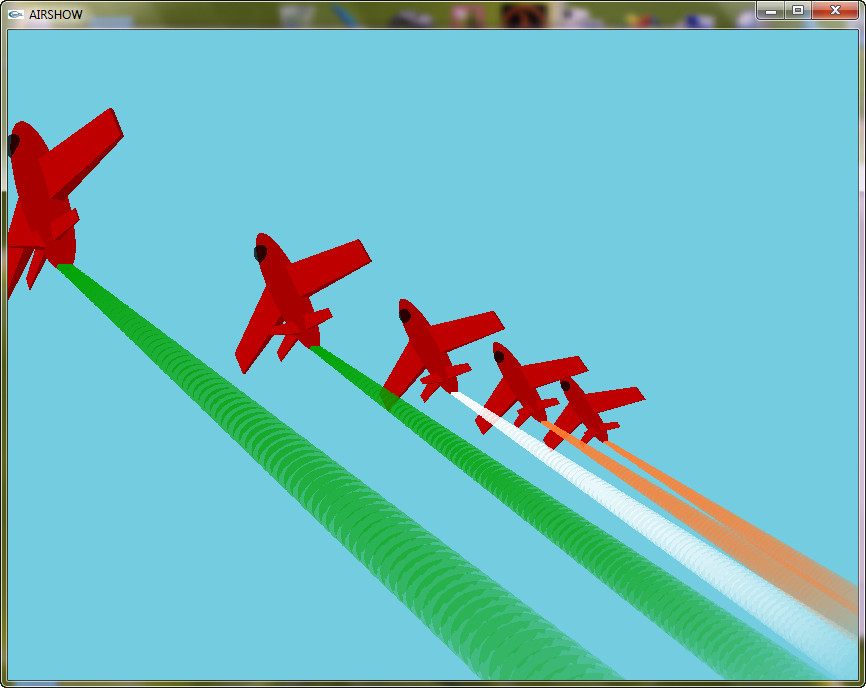
**When you press S or s from the keyboard**

****

**When you press up U or u button on keyboard**



**Tricolor is being displayed through smoke function.**



**Chapter 6: CONCLUSION AND FUTURE SCOPE**

After the completion of this project we came to know how we can implement a project using an Open source OpenGL tool kit. By implementing a project using Open GL we came to know how to use the functions like menu‘s, rotation, translation and scaling. With the completion of this project we have achieved a sense of happiness and we want to thank all those who helped us directly or indirectly to make this idea come true.

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**

1. Edward Angel, Interactive Computer Graphics A Top-Down Approach with OpenGL, 5th Edition, Addison-Wesley, 2008.
2. F.S. Hill,Jr, Computer Graphics Using OpenGL. 2nd Edition, Pearson Education, 2001.
3. James D Foley, Andries Van Dam, Steven K Feiner, John F Hughes, Computer Graphics –Addison-Wesley 1997.
4. Donald Hearn and Pauline Baker, Computer Graphics - OpenGL Version –2nd Edition, Pearson Education, 2003.
5. <http://www.opengl.com>