# Computer Graphics (UCS505)

# Project on

# Aeroplane Animation

**Submitted By**

Priyal Singla 102103274

Sukhmanjot Kaur 102103291

# 3CO 10

**B.E. Third Year – COE**

**Submitted To:**

**Dr. Kuntal Chowdhury**



**Computer Science and Engineering Department Thapar Institute of Engineering and Technology Patiala – 147001**

**Table of Contents**

|  |  |  |
| --- | --- | --- |
| **Sr. No.** | **Description** | **Page No.** |
| 1. | Introduction to Project | 3-4 |
| 2. | Computer Graphics concepts used | 5-6 |
| 3. | User Defined Functions | 7-8 |
| 4. | Code | 9-26 |
| 5. | Output/ Screen shots | 27-28 |

# INTRODUCTION

In the realm of computer graphics, OpenGL (Open Graphics Library) stands as a fundamental tool for rendering 2D and 3D graphics. It provides a powerful set of functions for creating interactive applications, simulations, and visualizations. This project leverages the capabilities of OpenGL to animate a flying aeroplane in a simulated environment.

**Computer Graphics and OpenGL:**

* Computer graphics is the field of visual computing that deals with the generation, manipulation, and rendering of images and animations using computers. It encompasses a wide range of applications, from video games and virtual reality to scientific visualization and computer-aided design (CAD).
* OpenGL, an open-source graphics API (Application Programming Interface), serves as the cornerstone for many graphics applications. It provides developers with a cross-platform framework for creating high-performance graphics rendering pipelines. With OpenGL, developers can harness the computational power of modern GPUs (Graphics Processing Units) to create stunning visual effects and immersive experiences.

**Project Scope and Objectives:**

The primary objective of this project is to create an animated scene featuring a flying aeroplane within an OpenGL environment. The scene includes a runway, buildings, birds and a sky backdrop, all rendered using OpenGL primitives and transformations. The aeroplane undergoes a takeoff sequence, complete with moving parts and dynamic positioning. The animation is achieved through the manipulation of transformation matrices and the use of timer-based updates.

**Key Features and Components:**

* Aeroplane Model: The aeroplane model is constructed using basic geometric shapes such as rectangles and triangles. It consists of a body, wings, and tail sections, each rendered separately using OpenGL primitives.
* Runway and Buildings: The scene includes a textured runway and some buildings, birds represented as simple geometric structures. The buildings add depth and context to the environment, enhancing the overall realism of the animation.
* Animation Sequence: The aeroplane undergoes a series of movements, including taxiing along the runway, takeoff at an angle, and ascent into the sky and then the aeroplane is shown at a higher position, and buildings are rendered in the background. Finally, a blast effect is rendered near the buildings, potentially simulating an explosion or impact. These movements are managed using timer-based updates and transformation matrices to achieve smooth and realistic motion.

**Real-Time Rendering and Immersive Visual Experience :**

By combining the principles of computer graphics with the capabilities of OpenGL, this project demonstrates the versatility of real-time rendering techniques, and showcases the potential for crafting immersive visual experiences.

# COMPUTER GRAPHICS CONCEPTS USED

This project demonstrates the use of several computer graphics concepts from OpenGL (Open Graphics Library) :

1. Geometric Primitives:

The project utilizes geometric primitives such as polygons (e.g., `GL\_POLYGON`) and line loops (`GL\_LINE\_LOOP`) to construct the various objects in the scene. These primitives are defined by specifying a sequence of vertices using `glVertex2f()`, which are then rendered by OpenGL.

2. Transformations:

The code applies transformations to the objects being rendered using functions like glTranslated(). Translations are applied to the aeroplane to simulate its movement along the x and y axes during takeoff and the blast effect.

3. Matrix Operations:

OpenGL uses matrix operations to perform transformations and manage the model-view and projection matrices. The code uses functions like glPushMatrix() and glPopMatrix() to save and restore the current matrix state, allowing for hierarchical transformations.

4. Color and Shading:

The code sets the color of the objects being rendered using the glColor3f() function, which specifies the red, green, and blue (RGB) components of the color. No explicit lighting is used in this code, but different colors are employed to differentiate between the various components of the scene.

5. Animation:

The animation of the aeroplane’s takeoff and the blast effect is achieved through the use of the glutTimerFunc() function. This function registers a callback function (update) to be called repeatedly after a specified time interval. Within the update function, the positions and transformations of the objects are updated, and glutPostRedisplay() is called to trigger a redraw of the scene, creating the illusion of animation.

6. Coordinate Systems and Viewports:

The project sets up a 2D coordinate system and viewport using the `gluOrtho2D` function. This function defines the visible region of the scene and maps the coordinates to the window or viewport. The scene is drawn within this defined coordinate system and viewport.

7. Event-driven Programming:

The project uses GLUT (OpenGL Utility Toolkit) functions to handle user events and manage the rendering loop. The `glutDisplayFunc` function is used to specify the display callback function (`display`), which is called whenever the window needs to be redrawn.

8. Scene Management: The code separates the rendering of different components of the scene into separate functions, such as display(), display2(), display3(), building(), blast(),bird() and road(). This modularization makes it easier to manage and update different parts of the scene independently.

# USER DEFINED FUNCTIONS

The provided code contains several user-defined functions that handle different aspects of the graphics rendering and animation. Here's an explanation of each user-defined function:

1. update(int value): Updates the position and orientation of the aeroplane during takeoff animation.

2. display(void): Main rendering function that draws the scene for each frame, including the aeroplane, runway, and triggers the next phases of the animation.

3. building():

- This function renders the buildings in the scene.

- It uses GL\_POLYGON and GL\_LINES primitives to construct the buildings' shapes and floor separations.

- It calls the build\_outline() function to draw the outlines of the buildings.

4. build\_outline():

- This function draws the outlines of the buildings using GL\_LINE\_LOOP primitives.

5. blast(void):

- This function renders the blast effect using a complex GL\_POLYGON primitive.

- It applies a translation transformation (glTranslated(-10.0, -60.0, 0.0)) to position the blast relative to the buildings.

6. road():

- This function draws the runway using GL\_POLYGON primitives.

- It renders the black road surface and the white runway strips.

- It applies a translation transformation (glTranslated(b, 0.0, 0.0)) to simulate the movement of the runway strips during takeoff.

7. birds():

- This function renders the birds in the scene.

- It applies translation transformations glTranslated(bird1\_x, 450, 0),

glTranslated(bird2\_x, 470, 0), and glTranslated(bird3\_x, 430, 0)

to position the birds at different locations.

- It updates the positions of the birds (bird1\_x, bird2\_x, bird3\_x) to create the illusion of movement.

- It resets the bird positions if they go off-screen.

8. display2():

- This function renders the aeroplane components in the second scene, similar to the display() function.

- It applies a translation transformation (glTranslated(d, 300.0, 0.0)) to position the aeroplane in the second scene.

9. display3():

- This function renders the aeroplane components and buildings in the third scene.

- It calls the building() function twice to draw two sets of buildings.

- It applies a translation transformation (glTranslated(e, 300.0, 0.0)) to position the aeroplane in the third scene.

10. myinit():

- This function initializes the OpenGL rendering environment.

- It sets the clear color, point size, and projection matrix using gluOrtho2D() to define a 2D orthographic projection .

# CODE:

#include <stdio.h>

#include <GL/glut.h>

GLfloat a = 0, b = 0, c = 0, d = 0, e = 0;

GLfloat bird1\_x = 0, bird2\_x = 100, bird3\_x = 300;

void building();

void building1();

void outline();

void blast();

void road();

void display2();

void display3();

void birds();

void build\_outline();

void update(int value)

{

a += 20.0; //Plane position takeoff on x axis

b -= 10.0; //Road Strip backward movement

c += 15; //take off at certain angle on y axis

if (b <= -78.0) // moving of run way

b = 0.0;

glutPostRedisplay();

glutTimerFunc(150, update, 0); //delay

}

void birds() {

// Bird 1

glPushMatrix();

glTranslatef(bird1\_x, 450, 0);

glColor3f(0.0, 0.0, 0.0);

glBegin(GL\_POLYGON);

glVertex2f(0, 0);

glVertex2f(10, 10);

glVertex2f(20, 0);

glVertex2f(10, -5);

glEnd();

glPopMatrix();

// Bird 2

glPushMatrix();

glTranslatef(bird2\_x, 470, 0);

glColor3f(0.0, 0.0, 0.0);

glBegin(GL\_POLYGON);

glVertex2f(0, 0);

glVertex2f(10, 10);

glVertex2f(20, 0);

glVertex2f(10, -5);

glEnd();

glPopMatrix();

// Bird 3

glPushMatrix();

glTranslatef(bird3\_x, 430, 0);

glColor3f(0.0, 0.0, 0.0);

glBegin(GL\_POLYGON);

glVertex2f(0, 0);

glVertex2f(10, 10);

glVertex2f(20, 0);

glVertex2f(10, -5);

glEnd();

glPopMatrix();

// Move the birds

bird1\_x -= 2;

bird2\_x -= 3;

bird3\_x -= 4;

// Reset bird positions if they go off-screen

if (bird1\_x < -20) bird1\_x = 500;

if (bird2\_x < -20) bird2\_x = 500;

if (bird3\_x < -20) bird3\_x = 500;

}

void display(void)

{

glClear(GL\_COLOR\_BUFFER\_BIT);

road();

glPushMatrix();

glTranslated(a, c, 0.0);

glColor3f(1.0, 1.0, 1.0);

glBegin(GL\_POLYGON); //rectangular body

glVertex2f(0.0, 30.0);

glVertex2f(0.0, 55.0);

glVertex2f(135.0, 55.0);

glVertex2f(135.0, 30.0);

glEnd();

glPopMatrix();

glPushMatrix();

glTranslated(a, c, 0.0);

glColor3f(1.0, 1.0, 1.0);

glBegin(GL\_POLYGON); //upper triangle construction plane

glVertex2f(135.0, 55.0);

glVertex2f(150.0, 50.0);

glVertex2f(155.0, 45.0);

glVertex2f(160.0, 40.0);

glVertex2f(135.0, 40.0);

glEnd();

glPopMatrix();

glPushMatrix();

glTranslated(a, c, 0.0);

glColor3f(0.0, 0.0, 0.0);

glBegin(GL\_LINE\_LOOP); //outline of upper triangle plane

glVertex2f(135.0, 55.0);

glVertex2f(150.0, 50.0);

glVertex2f(155.0, 45.0);

glVertex2f(160.0, 40.0);

glVertex2f(135.0, 40.0);

glEnd();

glPopMatrix();

glPushMatrix();

glTranslated(a, c, 0.0);

glColor3f(1.0, 0.0, 0.0);

glBegin(GL\_POLYGON); //lower triangle

glVertex2f(135.0, 40.0);

glVertex2f(160.0, 40.0);

glVertex2f(160.0, 37.0);

glVertex2f(145.0, 30.0);

glVertex2f(135.0, 30.0);

glEnd();

glPopMatrix();

glPushMatrix();

glTranslated(a, c, 0.0);

glColor3f(1.0, 0.0, 0.0);

glBegin(GL\_POLYGON); //back wing

glVertex2f(0.0, 55.0);

glVertex2f(0.0, 80.0);

glVertex2f(10.0, 80.0);

glVertex2f(40.0, 55.0);

glEnd();

glPopMatrix();

glPushMatrix();

glTranslated(a, c, 0.0);

glColor3f(1.0, 0.0, 0.0);

glBegin(GL\_POLYGON); //left side wing

glVertex2f(65.0, 55.0);

glVertex2f(50.0, 70.0);

glVertex2f(75.0, 70.0);

glVertex2f(90.0, 55.0);

glEnd();

glPopMatrix();

glPushMatrix();

glTranslated(a, c, 0.0);

glColor3f(1.0, 0.0, 0.0);

glBegin(GL\_POLYGON); //rightside wing

glVertex2f(70.0, 40.0);

glVertex2f(100.0, 40.0);

glVertex2f(80.0, 15.0);

glVertex2f(50.0, 15.0);

glEnd();

glPopMatrix();

if (c > 360) //timer to jump to next display

{

display2();

d += 20; //plane takeoff on x in 2nd display

}

if (a > 500.0) //window position during take off

{

a = 0.0;

b = 0.0;

}

if (c > 750) //timer to jump to 3rd display

{

display3();

e += 20; //plane takeoff on x in 3rd display

if (e > 250) //timer to call blast function

{

blast();

e = 250;

}

}

birds();

glFlush();

}

void building()

{

glColor3f(0.60, 0.40, 0.70);

glBegin(GL\_POLYGON);

glVertex2f(350.0, 80.0);

glVertex2f(350.0, 480.0);

glVertex2f(400.0, 400.0);

glVertex2f(400.0, 0.0);

glEnd();

glColor3f(0.75, 0.75, 0.75);

glBegin(GL\_POLYGON);

glVertex2f(400.0, 0.0);

glVertex2f(400.0, 400.0);

glVertex2f(450.0, 400.0);

glVertex2f(450.0, 0.0);

glEnd();

glColor3f(1.0, 1.0, 1.0);

glBegin(GL\_POLYGON);

glVertex2f(400.0, 400.0);

glVertex2f(350.0, 480.0);

glVertex2f(400.0, 480.0);

glVertex2f(450.0, 400.0);

glEnd();

glColor3f(0.60, 0.40, 0.70);

glBegin(GL\_POLYGON); //upper triangle of building

glVertex2f(400.0, 400.0);

glVertex2f(350.0, 480.0);

glVertex2f(400.0, 480.0);

glEnd();

glColor3f(0.0, 0.0, 0.0);

glBegin(GL\_LINES); //separation line of floors

glVertex2f(350.0, 180);

glVertex2f(400.0, 100);

glEnd();

glColor3f(0.0, 0.0, 0.0);

glBegin(GL\_LINES);

glVertex2f(350.0, 280);

glVertex2f(400.0, 200);

glEnd();

glColor3f(0.0, 0.0, 0.0);

glBegin(GL\_LINES);

glVertex2f(350.0, 380);

glVertex2f(400.0, 300);

glEnd();

glColor3f(0.0, 0.0, 0.0);

glBegin(GL\_LINES);

glVertex2f(450.0, 100);

glVertex2f(400.0, 100);

glEnd();

glColor3f(0.0, 0.0, 0.0);

glBegin(GL\_LINES);

glVertex2f(450.0, 200);

glVertex2f(400.0, 200);

glEnd();

glColor3f(0.0, 0.0, 0.0);

glBegin(GL\_LINES);

glVertex2f(450.0, 300);

glVertex2f(400.0, 300);

glColor3f(0.0, 0.0, 0.0);

glBegin(GL\_LINES);

glVertex2f(350.0, 180);

glEnd();

//2nd

glColor3f(0.60, 0.40, 0.70);

glBegin(GL\_POLYGON);

glVertex2f(250.0, 80.0);

glVertex2f(250.0, 380.0);

glVertex2f(300.0, 300.0);

glVertex2f(300.0, 0.0);

glEnd();

glColor3f(0.75, 0.75, 0.75);

glBegin(GL\_POLYGON);

glVertex2f(300.0, 0.0);

glVertex2f(300.0, 300.0);

glVertex2f(350.0, 300.0);

glVertex2f(350.0, 0.0);

glEnd();

glColor3f(1.0, 1.0, 1.0);

glBegin(GL\_POLYGON);

glVertex2f(300.0, 300.0);

glVertex2f(250.0, 380.0);

glVertex2f(300.0, 380.0);

glVertex2f(350.0, 300.0);

glEnd();

glColor3f(0.60, 0.40, 0.70);

glBegin(GL\_POLYGON); //upper triangle of building

glVertex2f(300.0, 300.0);

glVertex2f(250.0, 380.0);

glVertex2f(300.0, 380.0);

glEnd();

glColor3f(0.0, 0.0, 0.0);

glBegin(GL\_LINES); //separation line of floors

glVertex2f(250.0, 80);

glVertex2f(300.0, 0.0);

glEnd();

glColor3f(0.0, 0.0, 0.0);

glBegin(GL\_LINES);

glVertex2f(250.0, 180);

glVertex2f(300.0, 100);

glEnd();

glColor3f(0.0, 0.0, 0.0);

glBegin(GL\_LINES);

glVertex2f(250.0, 280);

glVertex2f(300.0, 200);

glEnd();

glColor3f(0.0, 0.0, 0.0);

glBegin(GL\_LINES);

glVertex2f(350.0, 0.0);

glVertex2f(300.0, 0.0);

glEnd();

glColor3f(0.0, 0.0, 0.0);

glBegin(GL\_LINES);

glVertex2f(250.0, 100);

glVertex2f(300.0, 100);

glEnd();

glColor3f(0.0, 0.0, 0.0);

glBegin(GL\_LINES);

glVertex2f(350.0, 200);

glVertex2f(300.0, 200);

glColor3f(0.0, 0.0, 0.0);

glBegin(GL\_LINES);

glVertex2f(250.0, 80);

glEnd();

build\_outline();

}

void build\_outline() //building out lines

{

glColor3f(0.0, 0.0, 0.0);

glBegin(GL\_LINE\_LOOP);

glVertex2f(350.0, 80.0);

glVertex2f(350.0, 480.0);

glVertex2f(400.0, 400.0);

glVertex2f(400.0, 0.0);

glEnd();

glColor3f(0.0, 0.0, 0.0);

glBegin(GL\_LINE\_LOOP);

glVertex2f(400.0, 0.0);

glVertex2f(400.0, 400.0);

glVertex2f(450.0, 400.0);

glVertex2f(450.0, 0.0);

glEnd();

glColor3f(0.0, 0.0, 0.0);

glBegin(GL\_LINE\_LOOP);

glVertex2f(400.0, 400.0);

glVertex2f(350.0, 480.0);

glVertex2f(400.0, 480.0);

glVertex2f(450.0, 400.0);

glEnd();

glColor3f(0.0, 0.0, 0.0);

glBegin(GL\_LINE\_LOOP);

glVertex2f(250.0, 80.0);

glVertex2f(250.0, 380.0);

glVertex2f(300.0, 300.0);

glVertex2f(300.0, 0.0);

glEnd();

glColor3f(0.0, 0.0, 0.0);

glBegin(GL\_LINE\_LOOP);

glVertex2f(300.0, 0.0);

glVertex2f(300.0, 300.0);

glVertex2f(350.0, 300.0);

glVertex2f(350.0, 0.0);

glEnd();

glColor3f(0.0, 0.0, 0.0);

glBegin(GL\_LINE\_LOOP);

glVertex2f(300.0, 300.0);

glVertex2f(250.0, 380.0);

glVertex2f(300.0, 380.0);

glVertex2f(350.0, 300.0);

glEnd();

}

void blast(void) //blast polygon construction

{

glPushMatrix();

glTranslated(-10.0, -60.0, 0.0);

glColor3f(1.0, 0.0, 0.0);

glBegin(GL\_POLYGON);

glVertex2f(404.4, 320.0);

glVertex2f(384.0, 285.0);

glVertex2f(368.0, 344.5);

glVertex2f(344.0, 355.0);

glVertex2f(347.2, 414.5);

glVertex2f(332.8, 442.5);

glVertex2f(347.2, 477.5);

glVertex2f(352.0, 530.0);

glVertex2f(379.2, 519.5);

glVertex2f(396.8, 565.0);

glVertex2f(416.0, 530.0);

glVertex2f(440.0, 547.5);

glVertex2f(452.8, 512.5);

glVertex2f(472.0, 512.5);

glVertex2f(475.2, 470.5);

glVertex2f(488.0, 442.5);

glVertex2f(488.0, 404.0);

glVertex2f(470.0, 372.5);

glVertex2f(475.2, 337.5);

glVertex2f(464.0, 306.0);

glVertex2f(444.8, 320.0);

glVertex2f(425.6, 285.0);

glVertex2f(404.8, 320.0);

glEnd();

glPopMatrix();

}

void road()

{

glColor3f(0.0, 0.0, 0.0);

glBegin(GL\_POLYGON); //black road

glVertex2f(0.0, 0.0);

glVertex2f(0.0, 100.0);

glVertex2f(500.0, 100.0);

glVertex2f(500.0, 0.0);

glEnd();

glPopMatrix();

glPushMatrix();

glTranslated(b, 0.0, 0.0);

glColor3f(1.0, 1.0, 1.0);

glBegin(GL\_POLYGON); //white strips on roadglVertex2f(0.0,40.0);

glVertex2f(8.0, 60.0);

glVertex2f(58.0, 60.0);

glVertex2f(50.0, 40.0);

glEnd();

glPopMatrix();

glPushMatrix();

glTranslated(b, 0.0, 0.0);

glColor3f(1.0, 1.0, 1.0);

glBegin(GL\_POLYGON);

glVertex2f(100.0, 40.0);

glVertex2f(108.0, 60.0);

glVertex2f(158.0, 60.0);

glVertex2f(150.0, 40.0);

glEnd();

glPopMatrix();

glPushMatrix();

glTranslated(b, 0.0, 0.0);

glColor3f(1.0, 1.0, 1.0);

glBegin(GL\_POLYGON);

glVertex2f(200.0, 40.0);

glVertex2f(208.0, 60.0);

glVertex2f(258.0, 60.0);

glVertex2f(250.0, 40.0);

glEnd();

glPopMatrix();

glPushMatrix();

glTranslated(b, 0.0, 0.0);

glColor3f(1.0, 1.0, 1.0);

glBegin(GL\_POLYGON);

glVertex2f(300.0, 40.0);

glVertex2f(308.0, 60.0);

glVertex2f(358.0, 60.0);

glVertex2f(350.0, 40.0);

glEnd();

glPopMatrix();

glPushMatrix();

glTranslated(b, 0.0, 0.0);

glColor3f(1.0, 1.0, 1.0);

glBegin(GL\_POLYGON);

glVertex2f(400.0, 40.0);

glVertex2f(408.0, 60.0);

glVertex2f(458.0, 60.0);

glVertex2f(450.0, 40.0);

glEnd();

glPopMatrix();

}

void display2()

{

glClear(GL\_COLOR\_BUFFER\_BIT);

glPushMatrix();

glTranslated(d, 300.0, 0.0);

glColor3f(1.0, 1.0, 1.0);

glBegin(GL\_POLYGON);

glVertex2f(0.0, 30.0); //rectangular body

glVertex2f(0.0, 55.0);

glVertex2f(135.0, 55.0);

glVertex2f(135.0, 30.0);

glEnd();

glPopMatrix();

glPushMatrix();

glTranslated(d, 300.0, 0.0);

glColor3f(1.0, 1.0, 1.0);

glBegin(GL\_POLYGON);

glVertex2f(135.0, 55.0); //upper triangle construction plane

glVertex2f(150.0, 50.0);

glVertex2f(155.0, 45.0);

glVertex2f(160.0, 40.0);

glVertex2f(135.0, 40.0);

glEnd();

glPopMatrix();

glPushMatrix();

glTranslated(d, 300.0, 0.0);

glColor3f(0.0, 0.0, 0.0);

glBegin(GL\_LINE\_LOOP);

glVertex2f(135.0, 55.0); //upper triangle construction plane

glVertex2f(150.0, 50.0);

glVertex2f(155.0, 45.0);

glVertex2f(160.0, 40.0);

glVertex2f(135.0, 40.0);

glEnd();

glPopMatrix();

glPushMatrix();

glTranslated(d, 300.0, 0.0);

glColor3f(1.0, 0.0, 0.0);

glBegin(GL\_POLYGON); //lower triangle

glVertex2f(135.0, 40.0);

glVertex2f(160.0, 40.0);

glVertex2f(160.0, 37.0);

glVertex2f(145.0, 30.0);

glVertex2f(135.0, 30.0);

glEnd();

glPopMatrix();

glPushMatrix();

glTranslated(d, 300.0, 0.0);

glColor3f(1.0, 0.0, 0.0);

glBegin(GL\_POLYGON); //back wing

glVertex2f(0.0, 55.0);

glVertex2f(0.0, 80.0);

glVertex2f(10.0, 80.0);

glVertex2f(40.0, 55.0);

//glVertex2f(165.0,40.0);

glEnd();

glPopMatrix();

glPushMatrix();

glTranslated(d, 300.0, 0.0);

glColor3f(1.0, 0.0, 0.0);

glBegin(GL\_POLYGON); //left side wing

glVertex2f(65.0, 55.0);

glVertex2f(50.0, 70.0);

glVertex2f(75.0, 70.0);

glVertex2f(90.0, 55.0);

glEnd();

glPopMatrix();

glPushMatrix();

glTranslated(d, 300.0, 0.0);

glColor3f(1.0, 0.0, 0.0);

glBegin(GL\_POLYGON);

glVertex2f(70.0, 40.0);

glVertex2f(100.0, 40.0);

glVertex2f(80.0, 15.0);

glVertex2f(50.0, 15.0);

glEnd();

glPopMatrix();

birds();

}

void display3()

{

glClear(GL\_COLOR\_BUFFER\_BIT);

building();

building();

glPushMatrix();

glTranslated(e, 300.0, 0.0);

glColor3f(1.0, 1.0, 1.0);

glBegin(GL\_POLYGON);

glVertex2f(0.0, 30.0); //rectangular body

glVertex2f(0.0, 55.0);

glVertex2f(135.0, 55.0);

glVertex2f(135.0, 30.0);

glEnd();

glPopMatrix();

glPushMatrix();

glTranslated(e, 300.0, 0.0);

glColor3f(1.0, 1.0, 1.0);

glBegin(GL\_POLYGON);

glVertex2f(135.0, 55.0);

glVertex2f(150.0, 50.0);

glVertex2f(155.0, 45.0);

glVertex2f(160.0, 40.0);

glVertex2f(135.0, 40.0);

glEnd();

glPopMatrix();

glPushMatrix();

glTranslated(e, 300.0, 0.0);

glColor3f(0.0, 0.0, 0.0);

glBegin(GL\_LINE\_LOOP);

glVertex2f(135.0, 55.0);

glVertex2f(150.0, 50.0);

glVertex2f(155.0, 45.0);

glVertex2f(160.0, 40.0);

glVertex2f(135.0, 40.0);

glEnd();

glPopMatrix();

glPushMatrix();

glTranslated(e, 300.0, 0.0);

glColor3f(1.0, 0.0, 0.0);

glBegin(GL\_POLYGON); //lower triangle

glVertex2f(135.0, 40.0);

glVertex2f(160.0, 40.0);

glVertex2f(160.0, 37.0);

glVertex2f(145.0, 30.0);

glVertex2f(135.0, 30.0);

glEnd();

glPopMatrix();

glPushMatrix();

glTranslated(e, 300.0, 0.0);

glColor3f(1.0, 0.0, 0.0);

glBegin(GL\_POLYGON); //back wing

glVertex2f(0.0, 55.0);

glVertex2f(0.0, 80.0);

glVertex2f(10.0, 80.0);

glVertex2f(40.0, 55.0);

//glVertex2f(165.0,40.0);

glEnd();

glPopMatrix();

glPushMatrix();

glTranslated(e, 300.0, 0.0);

glColor3f(1.0, 0.0, 0.0);

glBegin(GL\_POLYGON);

glVertex2f(65.0, 55.0);

glVertex2f(50.0, 70.0);

glVertex2f(75.0, 70.0);

glVertex2f(90.0, 55.0);

//glVertex2f(165.0,40.0);

glEnd();

glPopMatrix();

glPushMatrix();

glTranslated(e, 300.0, 0.0);

glColor3f(1.0, 0.0, 0.0);

glBegin(GL\_POLYGON);

glVertex2f(70.0, 40.0);

glVertex2f(100.0, 40.0);

glVertex2f(80.0, 15.0);

glVertex2f(50.0, 15.0);

glEnd();

glPopMatrix();

birds();

}

void myinit()

{

glClearColor(0.9f, 0.6f, 0.0f, 0.0f);

glColor3f(1.0, 0.0, 0.0);

glPointSize(1.0);

glMatrixMode(GL\_PROJECTION);

glLoadIdentity();

gluOrtho2D(0.0, 499.0, 0.0, 499.0);

}

void main(int argc, char \*argv[])

{

glutInit(&argc, argv);

glutInitDisplayMode(GLUT\_SINGLE | GLUT\_RGB);

glutInitWindowSize(500.0, 500.0);

glutInitWindowPosition(0, 0);

glutCreateWindow("AEROPLANE ANIMATION");

glutDisplayFunc(display);

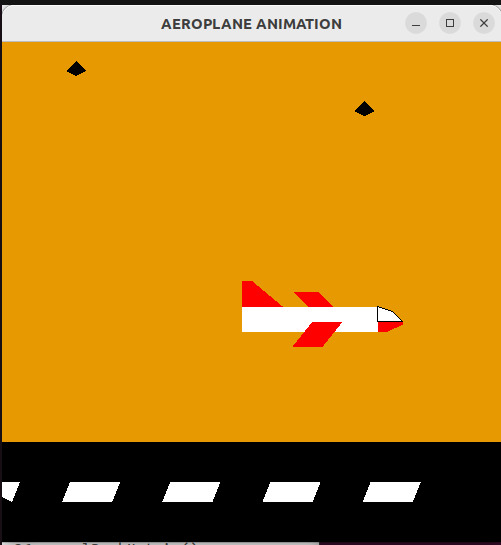
myinit();

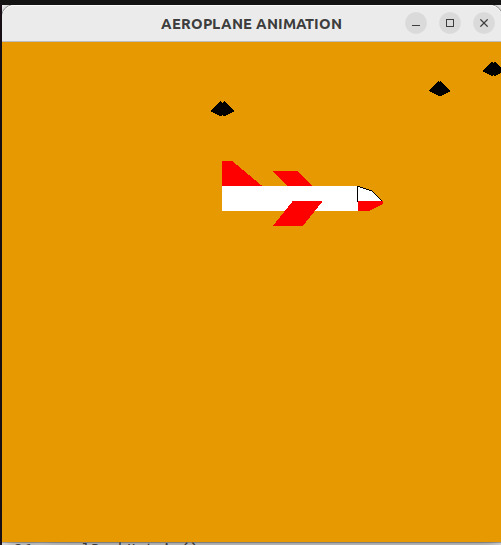
glutTimerFunc(100, update, 0);

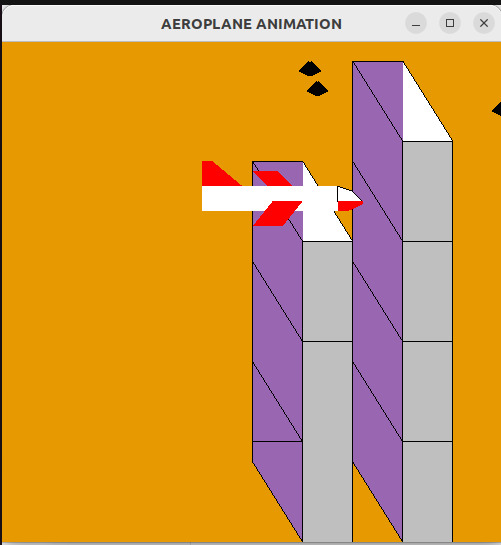
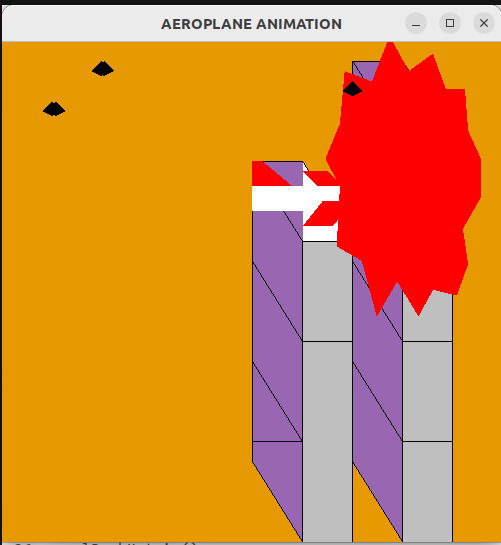
glutMainLoop();

}

# OUTPUT/ SCREENSHOTS :

  
Fig 1. Display 1

  
Fig 2. Display 2

    
Fig 3. Display 3 before and after the blast