Computer Graphics (UCS505)

Project on

Aeroplane Crash

Submitted By

Tushar Jain	102003391
Shivam Sharma	102003382

3CO15

B.E. Third Year – COE

Submitted To:

Dr. Rajkumar Tekchandani

THAPAR INSTITUTE
OF ENGINEERING & TECHNOLOGY
(Deemed to be University)

Computer Science and Engineering Department

Thapar Institute of Engineering and Technology

Patiala – 147001

Table of Contents

INTRODUCTION TO PROJECT

In the project, Aeroplane Crash Animation, created using OpenGL utilizes the power of computer graphics to recreate a realistic and dynamic simulation of an aeroplane crash. Using OpenGL, a powerful graphics library, the details of an aeroplane crash are bought to life , including the flight dynamics, physics, and visual effects.

The project showcases the realistic rendering of aeroplane model , and environmental elements, such as the sun and its rays, hills, trees, river and mountain. For the animation to feel real, the focus is given on the two essential elements that the animation needs to get right are the "Aeroplane" and the "Mountain". We made the project to show the best that we have learned. The main objectives of the project are:

1. Make it user friendly
2. To provide an easy interface (menu driven: to change to aeroplane colour).
3. To entertain people in the leisure time.

COMPUTER GRAPHICS CONCEPT USED

1. **Graphics Primitives:** The code uses various graphics primitives like points, circles, and polygons to draw graphical elements on the screen.

2. **Coordinate System:** The code uses a Cartesian coordinate system to specify the positions of points, circles, and polygons on the screen. It uses functions like glVertex2f() to specify the coordinates of points and vertices of polygons.

3. **Color:** The code uses color functions like glColor3f() to specify the RGB values of colors for drawing graphical elements with different colors.

4. **Callback Functions:** The code uses callback functions like glutDisplayFunc(), glutMouseFunc(), and glutTimerFunc() to handle various events like display, keyboard input, mouse input, and timer events respectively.

5. **Circle Drawing Algorithm**: It uses Bresenham's circle drawing algorithm and midpoint circle drawing algorithm to draw circles, which involve drawing lines between points on the circumference of the circle.

6. **Transformation:** It updates the positions of graphical elements like the plane using the update() function, which involves changing the values of global variables representing the positions.

7. **Rendering Pipeline:** It uses the OpenGL rendering pipeline to perform rendering operations like clearing the screen, specifying the color of graphical elements, and drawing graphical elements on the screen.

8. **Animation:** It uses the glutTimerFunc() function to create animation effects by repeatedly calling the update() function to update the positions of graphical elements and redraw the screen, creating a sense of animation.

9. **Basic Drawing Operations**: It uses basic drawing operations like drawing points, circles, and polygons using OpenGL functions like glBegin(), glEnd(), and glVertex2f() to create graphical elements on the screen.

10. The **glPushMatrix() and glPopMatrix()** functions are used in the given code, which are part of the transformation stack in OpenGL. These functions allow for saving and restoring

the current transformation state, which can be useful when applying transformations to graphical elements

USER DEFINED FUNCTIONS

1. **myinit() -** This function initializes light source for ambient, diffuse and specular types.

2. **color(int id) –** This function is used as a callback function to change the color of the aeroplane based on the 'id' selected by from the menu, and changes it to 'red', 'green', 'purple' respectively.

3. **display():** This is the main display function that is called by GLUT to redraw the window. It clears the screen using glClear() function, and then calls various functions to draw graphical elements like circles, sun rays, and mountains

4. **main():** The execution of the program starts from this function. It initializes the graphics system and includes many callback functions.

5. **update() –** This function is used as a timer callback function to update the position of the aeroplane by changing the global variables 'a' and 'c' which represent the plane's position on x and y axis. It is called repeatedly with a delay of 150 milliseconds using glutTimerFunc() function. The updated positions are used to redraw the plane's position using glutPostRedisplay() function..

6. **plot(int x, int y) –** This function is used to draw a point at the specified coordinates (x, y). It uses glBegin(GL_POINTS) and glVertex2f() functions to draw a single point.

7. **hill() –** This function is meant to draw a big mountain responsible for the plane crash.

8. **mountain() –** This function is meant to display mountains in the background scenery.

9. **trees() –** This function displays trees along the mountains in the background

10. **bresenham_circle(int r) -** This function implements the Bresenham's circle drawing algorithm to draw a circle with radius r using points. It calculates the points on the circumference of the circle using Bresenham's algorithm and calls plot() function to draw the points.

11. **midPointCircleAlgo(int r) -** This function implements the midpoint circle drawing algorithm to draw a circle with radius r using points. It calculates the points on the circumference of the circle using the midpoint algorithm and calls plot() function to draw the points.

12. **fillCircle(int xc, int yc, int radius) -** This function is used to draw a filled circle with center coordinates (xc, yc) and radius ‘radius’ using OpenGL's GL_TRIANGLE_FAN primitive. It calculates the coordinates of points on the circumference of the circle using trigonometric functions and calls glVertex2f() function to draw the points.

13. **drawSunRays(int xc, int yc, int numRays, int rayLength) –** This function is used to draw sun rays originating from a center point (xc, yc) with a specified number of rays numRays and length of rays rayLength.

14. **blast() –** This function is meant to depict a blast or explosion effect.

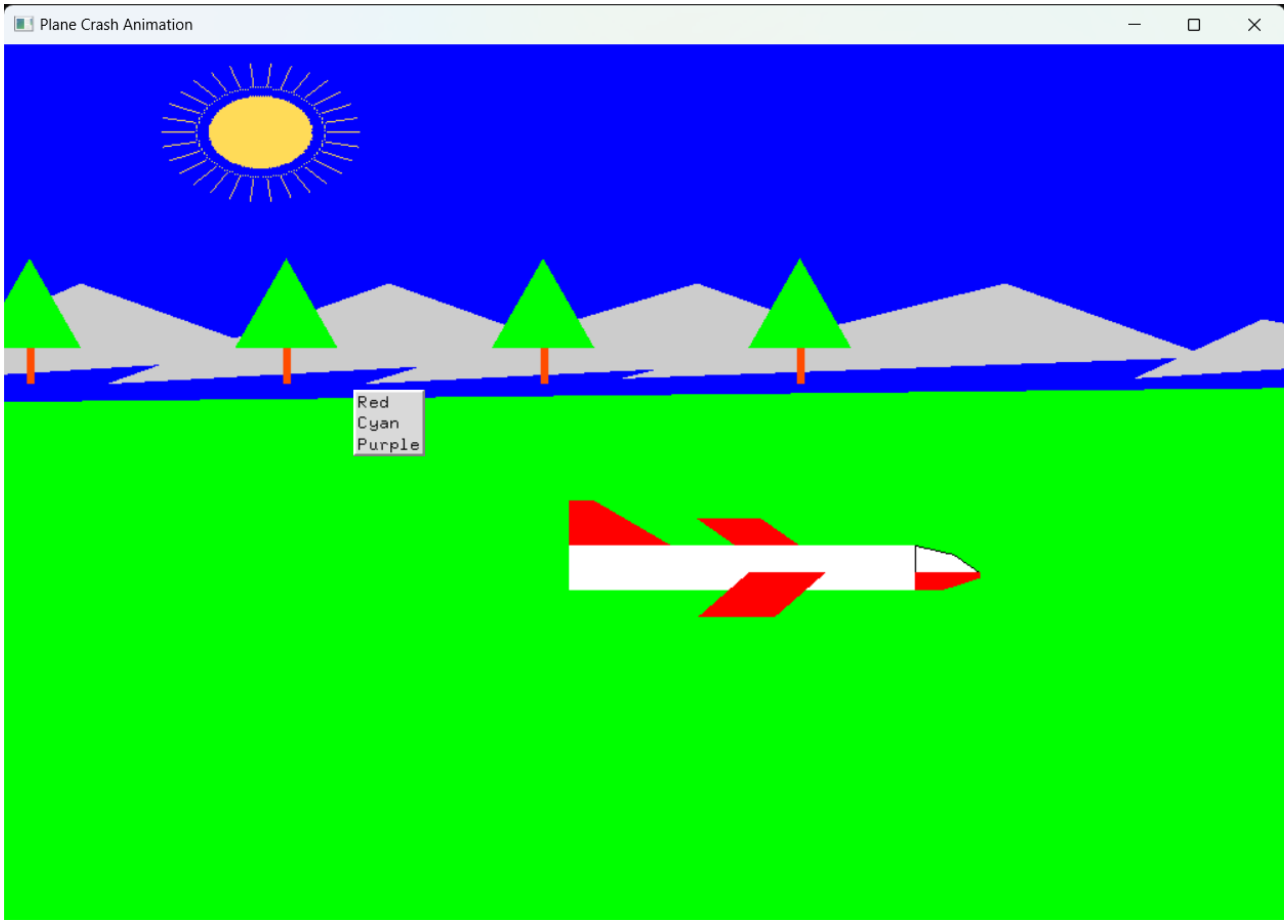
15. display2() – This function is an additional display function to show the aeroplane crash scene

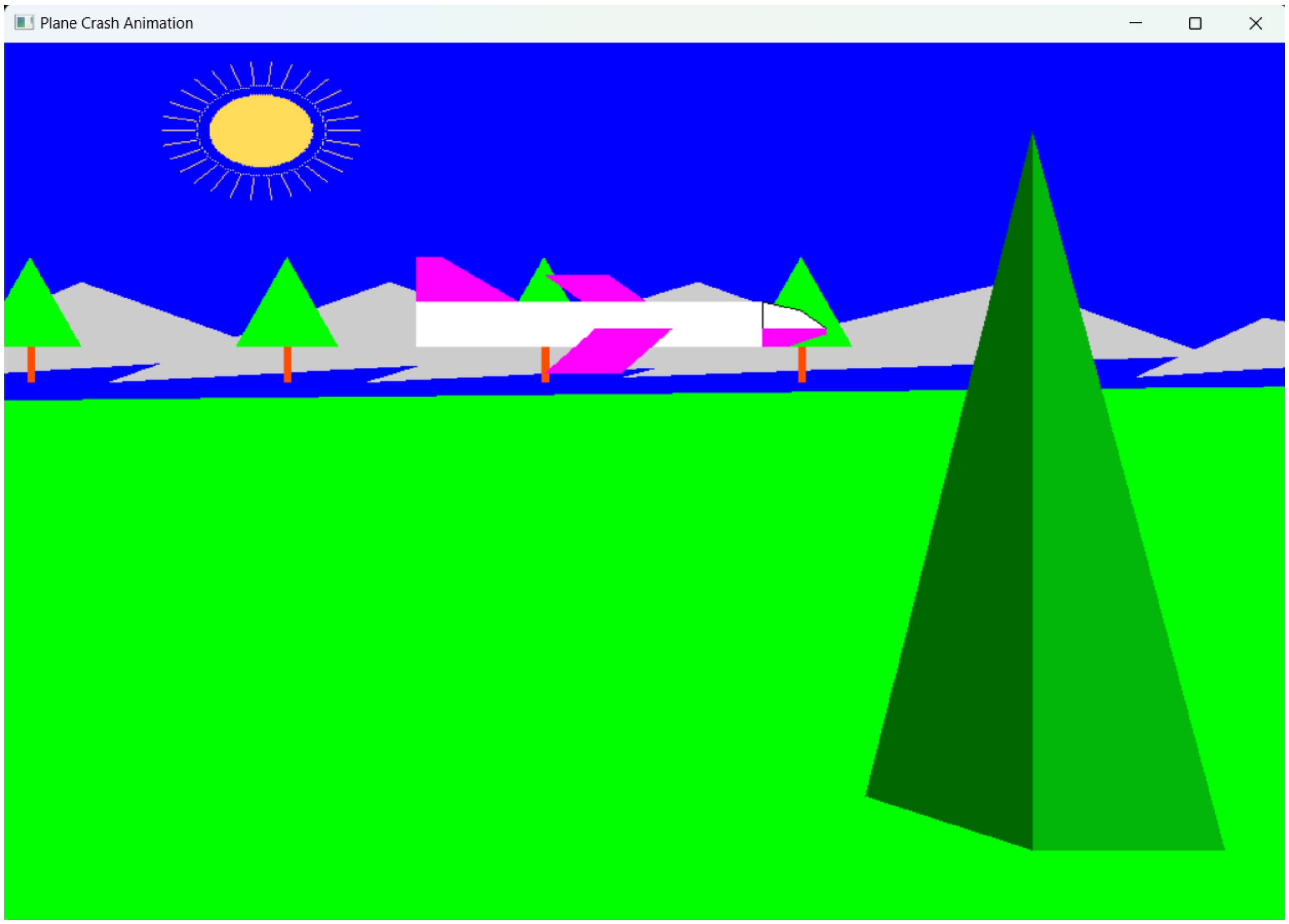
C++ CODE

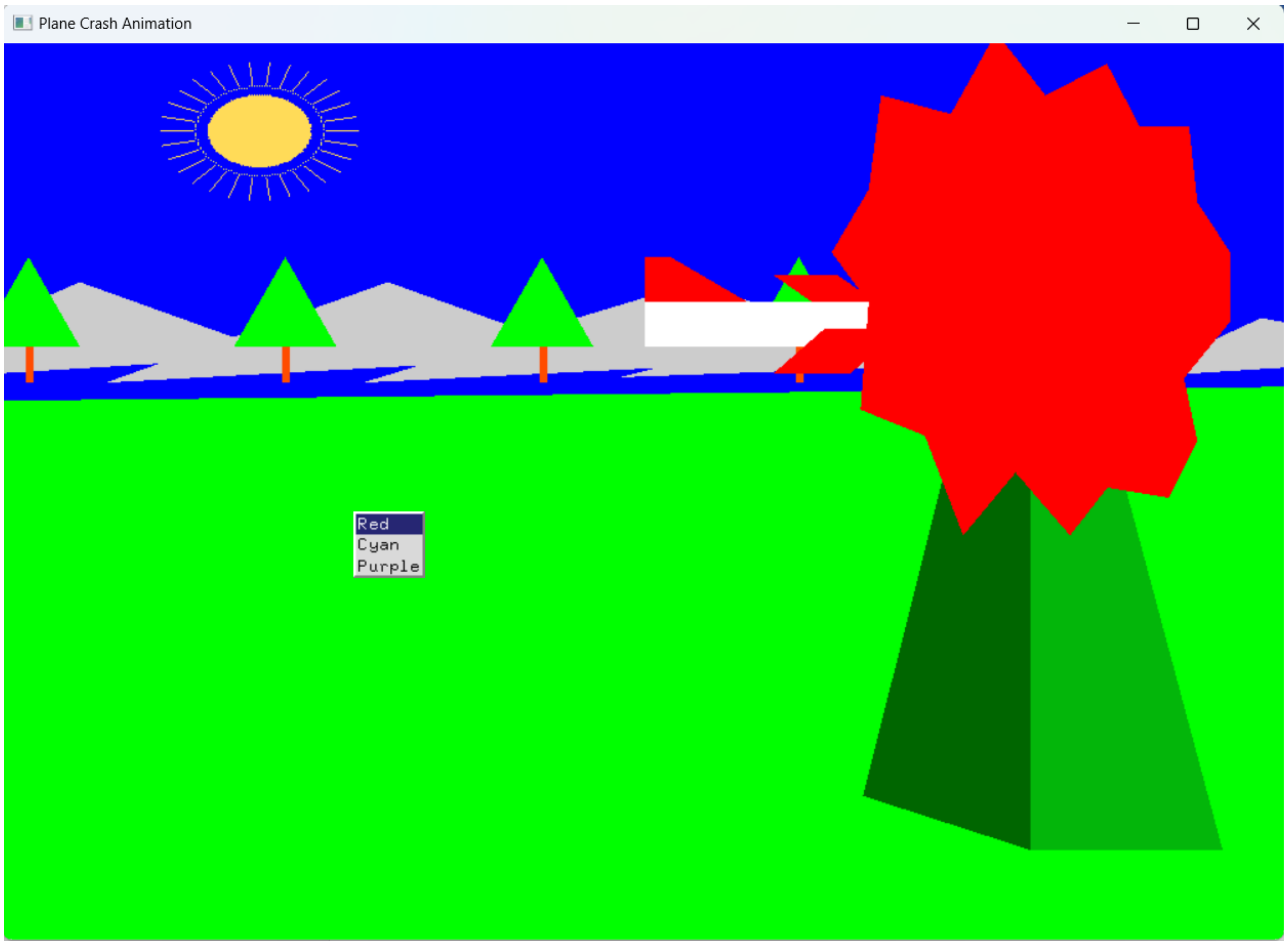
```
#include <stdio.h>
#include <GL/glut.h>
#include <cmath>
GLfloat a = 0, c = 0, e = 0;

void blast();
void display2();
void build_outline();
GLfloat R = 1.0, G = 0.0, B = 0.0;
void hill()
{
    glColor3f(0, 0.4, 0);
    glBegin(GL_POLYGON);
    glVertex2f(400.0, 450.0);
    glVertex2f(335.0, 80.0);
    glVertex2f(400.0, 50.0);
    glEnd();
    glColor3f (0.0117, 0.7137, 0.0392);
    glBegin(GL_TRIANGLES);
    glVertex2f(400.0, 450.0);
    glVertex2f(400.0, 50.0);
    glVertex2f(475.0, 50.0);
    glEnd();


}
void mountain(){
    glBegin(GL_POLYGON);
    glColor3f(0.8, 0.8, 0.8);
    glVertex2i(-60, 310);
    glVertex2i(110, 325);
    glVertex2i(30, 366);
    glEnd();
    glBegin(GL_POLYGON);
    glColor3f(0.8, 0.8, 0.8);
    glVertex2i(40, 310);
    glVertex2i(230, 325);
    glVertex2i(150, 366);
    glEnd();
    glBegin(GL_POLYGON);
    glColor3f(0.8, 0.8, 0.8);
    glVertex2i(140, 310);
    glVertex2i(350, 325);
    glVertex2i(270, 366);
    glEnd();
    glBegin(GL_POLYGON);
    glColor3f(0.8, 0.8, 0.8);
    glVertex2i(240, 313);
    glVertex2i(470, 325);
    glVertex2i(390, 366);
    glEnd();
    glBegin(GL_POLYGON);
    glColor3f(0.8, 0.8, 0.8);
    glVertex2i(440, 313);
    glVertex2i(570, 325);
    glVertex2i(490, 346);
    glEnd();
}

void trees(){
    glBegin(GL_POLYGON);
    glColor3f(1.0, 0.3, 0.0);
    glVertex2i(312, 310);
    glVertex2i(312, 333);
    glVertex2i(309, 333);
    glVertex2i(309, 310);
    glEnd();
    glBegin(GL_POLYGON);
    glColor3f(0.0, 1.0, 0.0);
    glVertex2i(290, 330);
    glVertex2i(330, 330);
    glVertex2i(310, 380);
    glEnd();
    glBegin(GL_POLYGON);
    glColor3f(1.0, 0.3, 0.0);
    glVertex2i(212, 310);
    glVertex2i(212, 333);
    glVertex2i(209, 333);
    glVertex2i(209, 310);
    glEnd();
    glBegin(GL_POLYGON);
    glColor3f(0.0, 1.0, 0.0);
    glVertex2i(190, 330);
    glVertex2i(230, 330);
    glVertex2i(210, 380);
    glEnd();
    glBegin(GL_POLYGON);
    glColor3f(1.0, 0.3, 0.0);
    glVertex2i(112, 310);
    glVertex2i(112, 333);
    glVertex2i(109, 333);
    glVertex2i(109, 310);
    glEnd();
    glBegin(GL_POLYGON);
    glColor3f(0.0, 1.0, 0.0);
    glVertex2i(90, 330);
    glVertex2i(130, 330);
    glVertex2i(110, 380);
    glEnd();
    glBegin(GL_POLYGON);
    glColor3f(1.0, 0.3, 0.0);
    glVertex2i(12, 310);
    glVertex2i(12, 333);
    glVertex2i(9, 333);
    glVertex2i(9, 310);
    glEnd();
    glBegin(GL_POLYGON);
    glColor3f(0.0, 1.0, 0.0);
    glVertex2i(-10, 330);
    glVertex2i(30, 330);
    glVertex2i(10, 380);
    glEnd();
}

void color(int id)
{
    switch (id)
    {
    case 1:
        R = 1.0, G = 0.0, B = 0.0;
        glutPostRedisplay();
        break;
    case 2:
        R = 0.0, G = 1.0, B = 1.0;
        glutPostRedisplay();
        break;
    case 3:
        R = 1.0, G = 0.0, B = 1.0;
        glutPostRedisplay();
        break;
    }
}
int pntX1, pntY1;
void update(int value)
{
    a += 20.0; // Plane position takeoff on x axis
    c += 15;   // take off at certain angle on y axis
    glutPostRedisplay();
    glutTimerFunc(150, update, 0); // delay
}
void plot(int x, int y)
{
    glBegin(GL_POINTS);
    glVertex2f(x + pntX1, y + pntY1);
    glEnd();
}

void bresenham_circle(int r)
{
    glColor3f(1, 1, 0.34);
    pntX1 = 100;
    pntY1 = 450;
    int x = 0;
    int y = r;
    float d = 3 - 2 * r;
    plot(x, y);

    while (y > x)
    {
        if (d < 0)
        {
            d += 4 * x + 6;
            x++;
        }
        else
        {
            d += 4 * (x - y) + 10;
            y--;
            x++;
        }
        plot(x, y);
        plot(x, -y);
        plot(-x, y);
        plot(-x, -y);
        plot(y, x);
        plot(-y, x);
        plot(y, -x);
        plot(-y, -x);
    }
}
void midPointCircleAlgo(int r)
```

```
{
    glColor3f(1, 0.858, 0.345);
    int x = 0;
    int y = r;
    pntX1 = 100;
    pntY1 = 450;
    float d = 5 / 4 - r;
    plot(x, y);

    while (y > x)
    {
        if (d < 0)
        {
            d += 2 * x + 3;
            x++;
        }
        else
        {
            d += 2 * (x - y) + 5;
            y--;
            x++;
        }
        plot(x, y);
        plot(x, -y);
        plot(-x, y);
        plot(-x, -y);
        plot(y, x);
        plot(-y, x);
        plot(y, -x);
        plot(-y, -x);
    }
}
void fillCircle(int xc, int yc, int radius)
{
    float xb1 = xc, yb11 = yc, xb2, yb2;
    float angle;

    glColor3f(1, 0.858, 0.345);
    glBegin(GL_TRIANGLE_FAN);
    glVertex2f(xb1, yb11);
    for (angle = 1.0f; angle < 361.0f; angle += 0.2)
    {
        xb2 = xb1 + sin(angle) * radius;
        yb2 = yb11 + cos(angle) * radius;
        glVertex2f(xb2, yb2);
    }
    glEnd();
    glColor3f(0, 0.27, 0.34);
}
void drawSunRays(int xc, int yc, int numRays, int rayLength)
{
    float angleIncrement = 360.0f / numRays; // Angle between each ray in degrees
    float angle = 0.0f;                      // Initial angle in degrees

    glBegin(GL_LINES);
    glColor3f(1, 0.858, 0.345);
    for (int i = 0; i < numRays; i++)
    {
        float startX = xc + rayLength * cos(angle * M_PI / 180.0f);      // Calculate
start point x-coordinate
        float startY = yc + rayLength * sin(angle * M_PI / 180.0f);      // Calculate
start point y-coordinate
        float endX = xc + rayLength * 1.5f * cos(angle * M_PI / 180.0f); // Calculate
end point x-coordinate
        float endY = yc + rayLength * 1.5f * sin(angle * M_PI / 180.0f); // Calculate
end point y-coordinate

        glVertex2f(startX, startY); // Set start point
        glVertex2f(endX, endY);     // Set end point

        angle += angleIncrement; // Increment angle for the next ray
    }
    glEnd();
}

void display(void)
{
    glClear(GL_COLOR_BUFFER_BIT);
    // glColor3f(1, 0.858, 0.345);
    midPointCircleAlgo(20);
    bresenham_circle(25);
    drawSunRays(pntX1, pntY1, 30, 26);
    fillCircle(pntX1, pntY1, 20);
    mountain();
    trees();

    glFlush();
    glBegin(GL_POLYGON);
    glColor3f(0.0, 1.0, 0.0); // green ground

    glVertex2i(0, 300);
    glVertex2i(645, 310);

    glVertex2i(940, 0);

    glVertex2i(0, 0);
    glEnd();
    glFlush();
    glPushMatrix();
    glTranslatef(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();
    glTranslatef(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();
    glTranslatef(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();
    glTranslatef(a, c, 0.0);
    glColor3f(R, G, B);
    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();
    glTranslatef(a, c, 0.0);
    glColor3f(R, G, B);
    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();
    glTranslatef(a, c, 0.0);
    glColor3f(R, G, B);
    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();
    glTranslatef(a, c, 0.0);
    glColor3f(R, G, B);
    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 (a > 500.0) // window position during take off
    {
        a = 0.0;
    }
    if (c > 300) // timer to jump to 2nd display
    {
        display2();
        e += 20;     // plane takeoff on x in 2nd display
        if (e > 250) // timer to call blast function
        {
            blast();
            e = 250;
        }
    }
    glFlush();
}

void blast(void) // blast polygon construction
{
    glPushMatrix();
    glTranslatef(-10.0, -60.0, 0.0);
    glColor3f(R, G, B);
    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 display2()
{
    glClear(GL_COLOR_BUFFER_BIT);
    midPointCircleAlgo(20);
    bresenham_circle(25);
    drawSunRays(pntX1, pntY1, 30, 26);
    fillCircle(pntX1, pntY1, 20);
    mountain();
    trees();
    glFlush();
    glBegin(GL_POLYGON);
    glColor3f(0.0, 1.0, 0.0); // green ground
    glVertex2i(0, 300);
    glVertex2i(645, 310);
    glVertex2i(940, 0);
    glVertex2i(0, 0);
    glEnd();
    glFlush();

    hill();
    glPushMatrix();
    glTranslatef(e, 300.0, 0.0);
    glColor3f(1.0, 1.0, 1.0);
    glBegin(GL_POLYGON);
    glVertex2f(0.0, 30.0);
    glVertex2f(0.0, 55.0);
    glVertex2f(135.0, 55.0);
    glVertex2f(135.0, 30.0);
    glEnd();
    glPopMatrix();
    glPushMatrix();
    glTranslatef(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();
    glTranslatef(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();
    glTranslatef(e, 300.0, 0.0);
    glColor3f(R, G, B);
    glBegin(GL_POLYGON);
    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();
    glTranslatef(e, 300.0, 0.0);
    glColor3f(R, G, B);
    glBegin(GL_POLYGON);
    glVertex2f(0.0, 55.0);
    glVertex2f(0.0, 80.0);
    glVertex2f(10.0, 80.0);
    glVertex2f(40.0, 55.0);
    glEnd();
    glPopMatrix();
    glPushMatrix();
    glTranslatef(e, 300.0, 0.0);
    glColor3f(R, G, B);
    glBegin(GL_POLYGON);
    glVertex2f(65.0, 55.0);
    glVertex2f(50.0, 70.0);
    glVertex2f(75.0, 70.0);
    glVertex2f(90.0, 55.0);
    glEnd();
    glPopMatrix();
    glPushMatrix();
    glTranslatef(e, 300.0, 0.0);
    glColor3f(R, G, B);
    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();
}
void myinit()
{
    glClearColor(0.0f, 0.0f, 1.0f, 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);
}
int main(int argc, char **argv){
    glutInit(&argc, argv);
    glutInitDisplayMode(GLUT_SINGLE | GLUT_RGB);
    glutInitWindowSize(1000, 700);
    glutInitWindowPosition(100, 100);
    glutCreateWindow("Plane Crash Animation");
    glutDisplayFunc(display);
    glutCreateMenu(color);
    glutAddMenuEntry("Red", 1);
    glutAddMenuEntry("Cyan", 2);
    glutAddMenuEntry("Purple", 3);
    glutAttachMenu(GLUT_RIGHT_BUTTON);
    myinit();
    glutTimerFunc(100, update, 0);
    glutMainLoop();
}
```

OUTPUT/ SCREENSHOTS




Plane Crash Animation


Plane Crash Animation
Red
Cyan
Purple