VISVESVARAYA TECHNOLOGICAL UNIVERSITY

"Jnana Sangama", Belagavi, Karnataka, India.

A Computer Graphics Mini Project Report On

Aeroplane Crashing Simulation/Game

Submitted in partial fulfillment of the requirement for the award of the degree

of

Bachelor of Engineering
in
Computer Science and Engineering

Submitted By
Amrutha K
1GA18CS194

Under the guidance of
Mrs. Veena V Pattankar
Assistant Professor

Department of Computer Science and Engineering
(Accredited by NBA 2019-2022)
GLOBAL ACADEMY OF TECHNOLOGY
Rajarajeshwari Nagar, Bengaluru - 560 098

2020 – 2021

GLOBAL ACADEMY OF TECHNOLOGY

Department of Computer Science and Engineering

Bengaluru – 560098

CERTIFICATE

Certified that the Computer Graphics & Visualization Mini Project Entitled **"*Aeroplane Crashing Simulation/Game*"** carried out by **Amrutha K**, bearing USN **1GA18CS194,** bonafide student of Global Academy of Technology, is in partial fulfillment for the award of the **BACHELOR OF ENGINEERING** in Computer Science and Engineering from **Visvesvaraya Technological University, Belagavi** during the year 2020-2021. The report has been approved as it satisfies the academic requirements in respect of the Mini Project work prescribed for the said degree.

Mrs. Veena V Pattankar Assistant Professor Dept. of CSE GAT, Bengaluru.	Dr. Bhagyashri R Hanji Professor & Head Dept. of CSE GAT, Bengaluru.

ACKNOWLEDGEMENT

The satisfaction and the euphoria that accompany the successful completion of any task would be incomplete without the mention of the people who made it possible. The constant guidance of these persons and encouragement provided, crowned my efforts with success and glory. Although it is not possible to thank all the members who helped for the completion of the internship work individually, I take this opportunity to express my gratitude to one and all.

I am grateful to management and our institute **GLOBAL ACADEMY OF TECHNOLOGY** with its very ideals and inspiration for having provided me with the facilities, which made this, work a success.

I express my sincere gratitude to **Dr. N. Rana Pratap Reddy**, Principal, Global Academy of Technology for the support and encouragement.

I wish to place on record, my grateful thanks to **Dr. Bhagyashri R Hanji**, Head, Department of CSE, Global Academy of Technology, for the constant encouragement provided to me.

I am indebted with a deep sense of gratitude for the constant inspiration, encouragement, timely guidance and valid suggestion given to me by my guide **Mrs. Veena V Pattankar**, Assistant Professor, Department of CSE, Global Academy of Technology.

I am thankful to all the staff members of the department for providing relevant information and helped in different capacities in carrying out this project.

Last, but not least, I owe my debts to my parents, friends and also those who directly or indirectly have helped me to make the project work a success.

Date: 27-07-2021 **AMRUTHA K** **[1GA18CS194]**

ABSTRACT

The Project entitled ‘Aeroplane crashing simulation/Game’ is implemented using OpenGL. The aim of the project is to give a demo of a moving object with different colored light sources reflecting on it. The is on WTC PLANE CRASH, WTC stands for world trade center which was attacked by terrorist on the date of 9/11,in this project we tried to show the simulation of the attacked on WTC.

The software is developed for user convenience using the concept of GUI. Here the user is interactive with the simulation through menu and keyboard interface concepts containing multiple windows having the information of keyboard controls, object information and Simulation. This software is an excellent tool that can be used as a simulator to explain about the multi-lights.

TABLE OF CONTENTS

Chapter 1

INTRODUCTION

1.1 INTRODUCTION TO COMPUTER GRAPHICS

Computer Graphics is concerned with all aspects of producing pictures or images using a computer.

Applications of Computer Graphics

1. Display of information
2. Design
3. Simulation and animation
4. User interfaces

The Graphics Architecture

Graphics Architecture can be made up of seven components:

1. Display processors
2. Pipeline architectures
3. The graphics pipeline
4. Vertex processing
5. Clipping and primitive assembly
6. Rasterization
7. Fragment processing

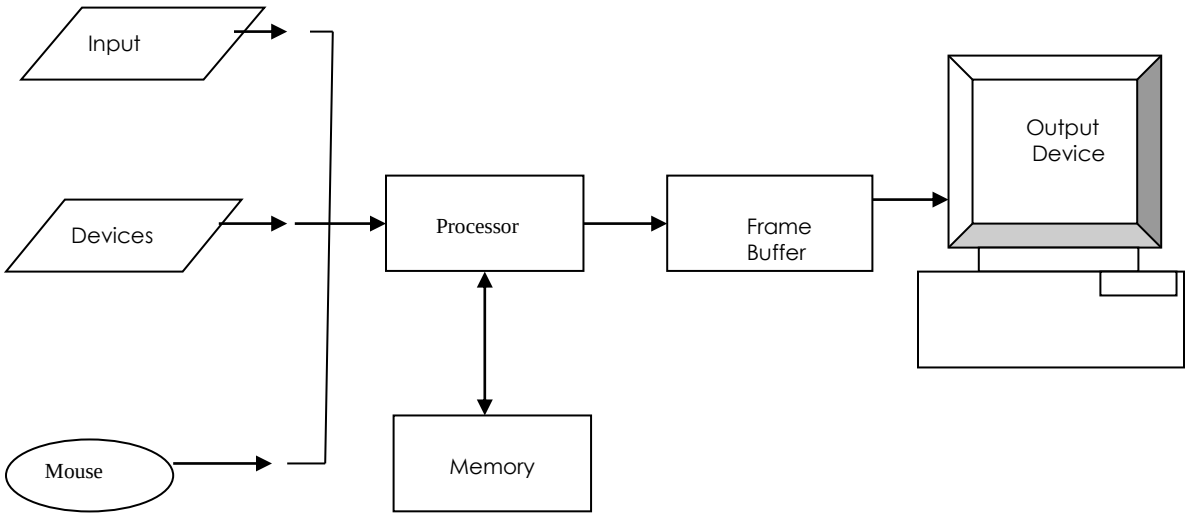


Figure 1.1: Components of Graphics Architecture and their working

1.2 INTRODUCTION TO OPENGL

OpenGL is software used to implement computer graphics. The structure of OpenGL is similar to that of most modern APIs including Java 3D and DirectX. OpenGL is easy to learn, compared with other.

APIs are nevertheless powerful. It supports the simple 2D and 3D programs. It also supports the advanced rendering techniques. OpenGL API explains following 3 components

1. Graphics functions
2. Graphics pipeline and state machines
3. The OpenGL interfaces

There are so many polygon types in OpenGL like triangles, quadrilaterals, strips and fans.

There are 2 control functions, which will explain OpenGL through,

1. Interaction with window system
2. Aspect ratio and view ports

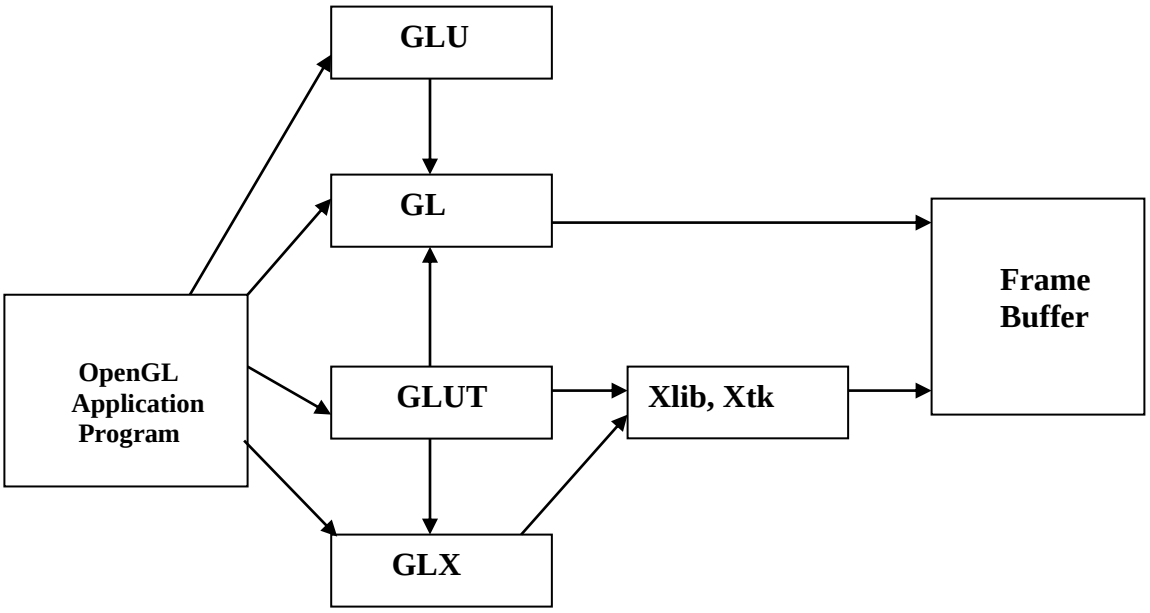


Figure 1.2: OpenGL Library organization

Most implementations of OpenGL have a similar order of operations, a series of processing stages called the OpenGL rendering pipeline. This ordering, as shown in Figure 1.2, is not a strict rule of how OpenGL is implemented but provides a reliable guide for predicting what OpenGL will do. The following diagram shows the assembly line approach, which OpenGL takes to process data. Geometric data (vertices, lines, and polygons) follow the path through the row of boxes that includes evaluators and per-vertex operations, while pixel data (pixels, images, and bitmaps) are treated differently for part of the process. Both types of data undergo the same final steps (rasterization and per-fragment operations) before the final pixel data is written into the frame buffer.

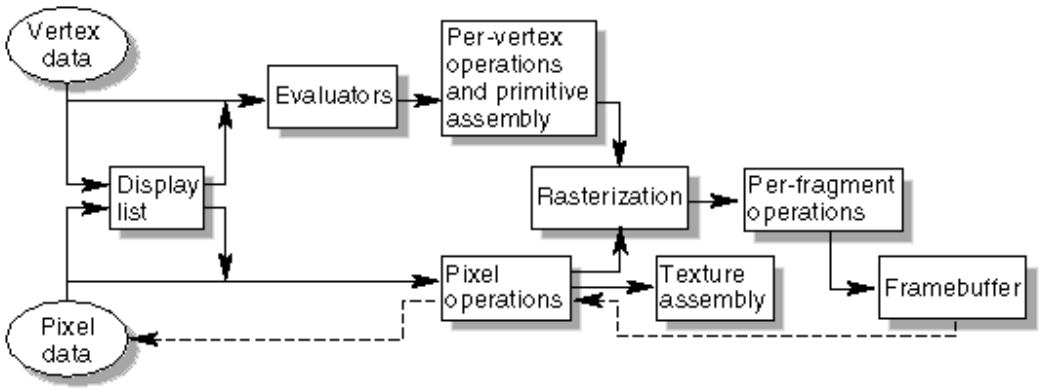


Figure 1.3: OpenGL Order of Operations

Chapter 2

REQUIREMENTS SPECIFICATION

2.1 SOFTWARE REQUIREMENTS

- This project runs using Visual Studio 2013
- OPENGL software
- Language used is C/C++

2.2 HARDWARE REQUIREMENTS

- There are no rigorous restrictions on the machine configurations. Since OPENGL software is portable, that is, it is platform independent or architecturally neutral it can run on any machine with any configurations.

2.3 MISCELLANEOUS REQUIREMENTS

- All the required library files and the header files should be available in the include directory.
- The library files are GL, GLU, and GLUT.

Chapter 3

SYSTEM DEFINITION

3.1 Project Description

The scope is to use the basic primitives defined in openGL library creating complex objects. Users make use of different concepts such as pushmatrix(), translate(), popmatrix(), timer function. When we press 'S' the flight takes off and when we press 'Q' the program quits. The final goal of the project is that aeroplane crashes , when the aeroplane hits there is a 'BOOM!' effect at the end. If the aeroplane crosses the small building it can easily sail, but when it hits a long building there is a 'BOOM' effect. The project was a pictorial representation of WTC attack stands for world trade centre attack, which was attacked by terrorists on the date of 9/11.

3.2 User Defined Functions

- **myinit() :** This function initializes light source for ambient, diffuse and specular types.
- **display() :** This function creates and translates all the objects in a specified location in a particular order and also rotates the objects in different axes.

glClear(GL_COLOR_BUFFER_BIT);

glFlush();

- **timerfunc() :** This function starts a timer in the event loop that delays the event loop for delay miliseconds.
- **MainLoop() :** This function whose execution will cause the program to begin an event processing loop.
- **PushMatrix() :** Save the present values of attributes and matrices placing , or pushing on the top of the stack.
- **PopMatrix() :** We can recover them by removing them from stack
- **Translated() :** In translate func the variables are components of the displacement vector.
- **main() :** The execution of the program starts from this function. It initializes the graphics system and includes many callback functions.
- **PostRedisplay() :** It ensures that the display will be drawn only once each time the program goes through the event loop.

3.3 Flow diagram

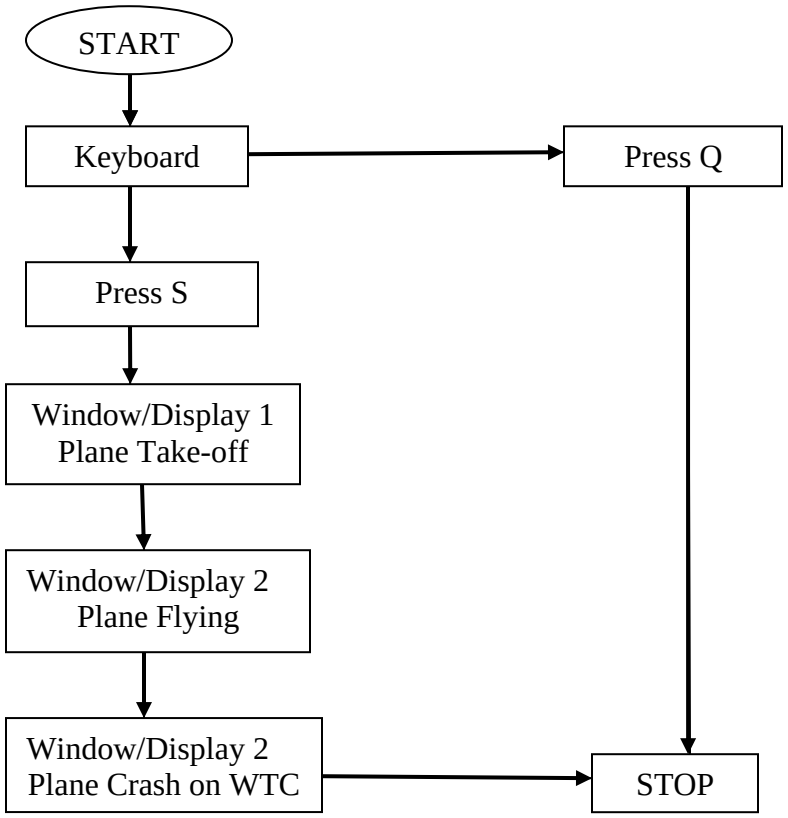


Fig 3.3.1 : System Architecture

Chapter 4

IMPLEMENTATION

4.1 Source Code

```
#include <stdio.h>
#include <glut.h>
GLfloat a = 0, b = 0, c = 0, d = 0, e = 0;
void building();
void building1();
void outline();
void blast();
void road();
void display2();
void display3();
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 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;
        }
    }
    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); //seperation 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); //seperation 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);
    glColor3f(0.0, 0.0, 1.0);
```

```
    glVertex2f(368.0, 344.5);
    glVertex2f(344.0, 355.0);
    glVertex2f(347.2, 414.5);
    glColor3f(0.0, 1.0, 0.0);
    glVertex2f(332.8, 442.5);
    glVertex2f(347.2, 477.5);
    glVertex2f(352.0, 530.0);
    glVertex2f(379.2, 519.5);
    glColor3f(0.0, 0.0, 1.0);
    glVertex2f(396.8, 565.0);
    glVertex2f(416.0, 530.0);
    glVertex2f(440.0, 547.5);
    glColor3f(1.0, 1.0, 0.0);
    glVertex2f(452.8, 512.5);
    glVertex2f(472.0, 512.5);
    glVertex2f(475.2, 470.5);
    glColor3f(0.0, 1.0, 1.0);
    glVertex2f(488.0, 442.5);
    glVertex2f(488.0, 404.0);
    glVertex2f(470.0, 372.5);
    glColor3f(0.3, 1.0, 0.6);
    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); //rectangulr 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();


//3

glColor3f(0.60, 0.40, 0.70);
glBegin(GL_POLYGON);
glVertex2f(150.0, 80.0);
glVertex2f(150.0, 280.0);
glVertex2f(200.0, 200.0);
glVertex2f(200.0, 0.0);
glEnd();
glColor3f(0.75, 0.75, 0.75);
glBegin(GL_POLYGON);
glVertex2f(200.0, 0.0);
glVertex2f(200.0, 200.0);
glVertex2f(250.0, 200.0);
glVertex2f(250.0, 0.0);
glEnd();
glColor3f(1.0, 1.0, 1.0);
glBegin(GL_POLYGON);
glVertex2f(200.0, 200.0);
glVertex2f(150.0, 280.0);
glVertex2f(200.0, 280.0);
glVertex2f(250.0, 200.0);
glEnd();
```

```
glColor3f(0.60, 0.40, 0.70);
glBegin(GL_POLYGON); //upper triangle of building
glVertex2f(200.0, 200.0);
glVertex2f(150.0, 280.0);
glVertex2f(200.0, 280.0);
glEnd();
glColor3f(0.0, 0.0, 0.0);
glBegin(GL_LINES); //seperation line of floors
glVertex2f(150.0, 80);
glVertex2f(200.0, 0.0);
glEnd();
glColor3f(0.0, 0.0, 0.0);
glBegin(GL_LINES);
glVertex2f(150.0, 180);
glVertex2f(200.0, 100);
glEnd();
glColor3f(0.0, 0.0, 0.0);
glBegin(GL_LINES);
glVertex2f(150.0, 280);
glVertex2f(200.0, 200);
glEnd();
glColor3f(0.0, 0.0, 0.0);
glBegin(GL_LINES);
glVertex2f(250.0, 0.0);
glVertex2f(200.0, 0.0);
glEnd();
glColor3f(0.0, 0.0, 0.0);
glBegin(GL_LINES);
glVertex2f(150.0, 100);
glVertex2f(200.0, 100);
glEnd();
glColor3f(0.0, 0.0, 0.0);
glBegin(GL_LINES);
glVertex2f(250.0, 200);
glVertex2f(200.0, 200);
glColor3f(0.0, 0.0, 0.0);
glBegin(GL_LINES);
glVertex2f(150.0, 80);
```

```
    glEnd();
}

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();
}
void myinit(){
    glClearColor(0.196078, 0.6, 0.8, 1);
    glColor3f(1.0, 0.0, 0.0);
    glPointSize(1.0);
    glMatrixMode(GL_PROJECTION);
    glLoadIdentity();
    gluOrtho2D(0.0, 499.0, 0.0, 499.0);
}
void keys(unsigned char key, int x, int y){
    if (key == 's'){
        glutTimerFunc(100, update, 0);
    }
    if (key == 'q') exit(0);
}
void main(int argc, char* argv[]){
    glutInit(&argc, argv);
    glutInitDisplayMode(GLUT_SINGLE | GLUT_RGB);
    glutInitWindowSize(500.0, 500.0);
    glutInitWindowPosition(0, 0);
    glutCreateWindow("AERO");
    glutDisplayFunc(display);
    myinit();
    glutKeyboardFunc(keys);
    //glutTimerFunc(100, update, 0);
    glutMainLoop();
}
```

Chapter 5

TESTING AND RESULTS

5.1 Different types of testing

1. Unit Testing

Individual components are tested to ensure that they operate correctly. Each component is tested independently, without other system components.

2. Module Testing

A module is a collection of dependent components such as a object class, an abstract data type or some looser collection of procedures and functions. A module related components, so can be tested without other system modules.

3. System Testing

This is concerned with finding errors that result from unanticipated interaction between sub-system interface problems.

4. Acceptance Testing

The system is tested with data supplied by the system customer rather than simulated test data.

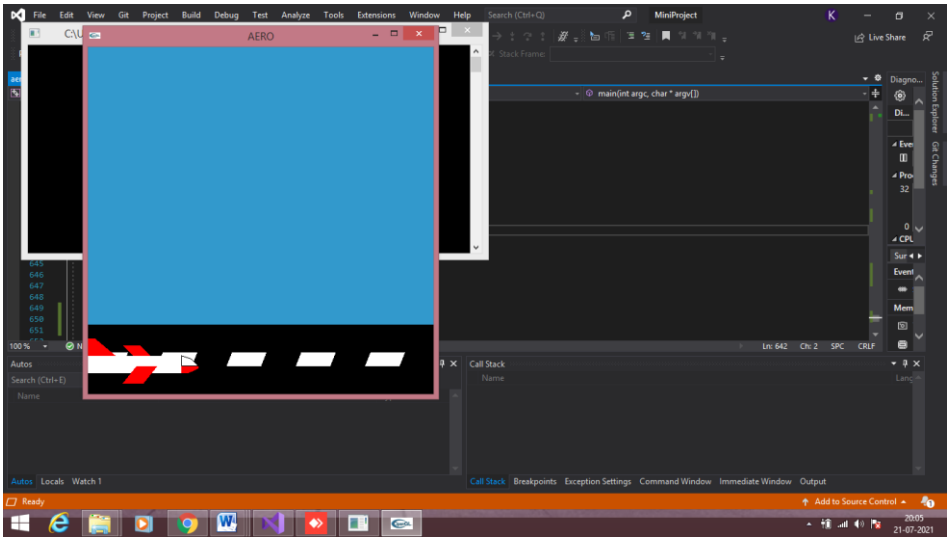
5.2 TEST CASES FOR THE PROJECT

SLNO	TEST INPUT	EXPECTED RESULTS	OBSERVED RESULTS	REMARKS
1.	Press S	Plane take off.	Plane takes off.	PASS
2.	Press Q	Quit Program	Quits program	PASS

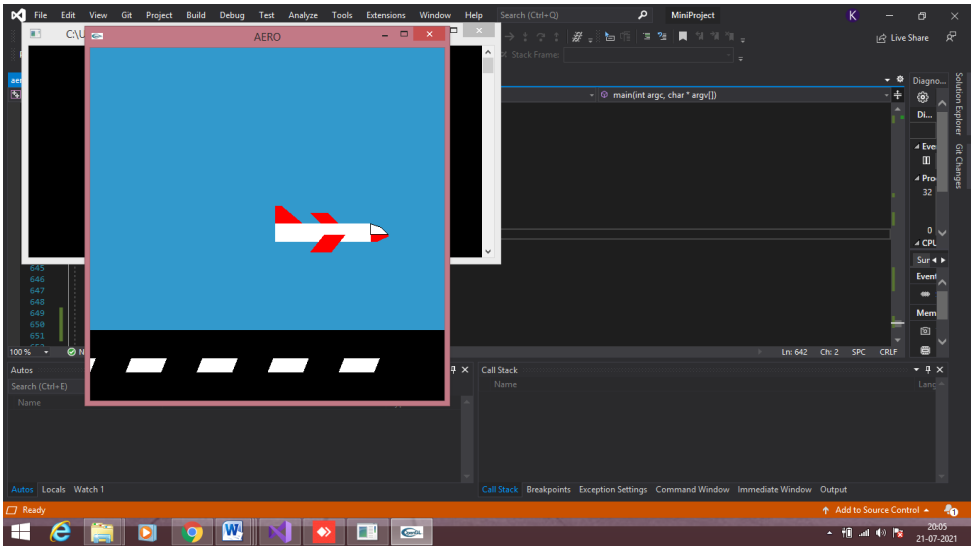
Table 5.2: Test cases

Chapter 6

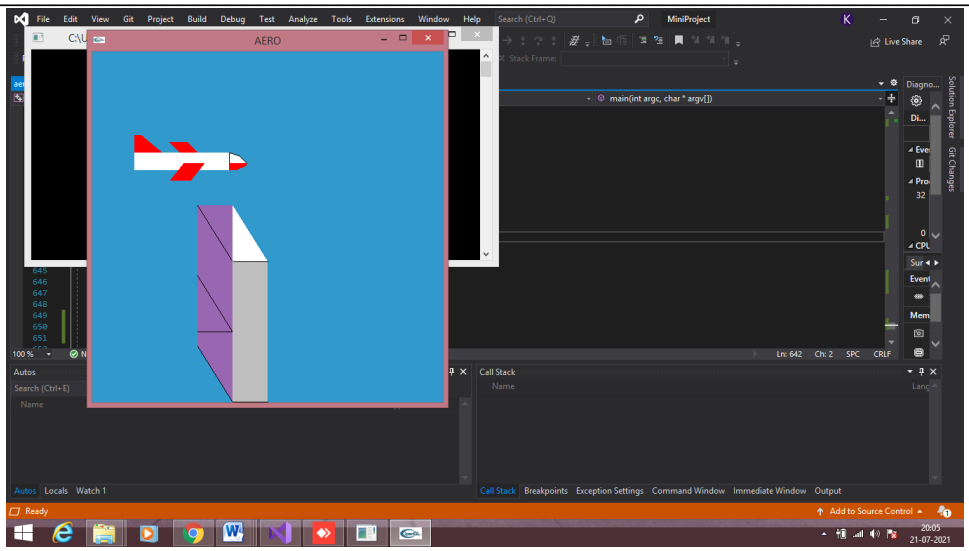
Screenshots



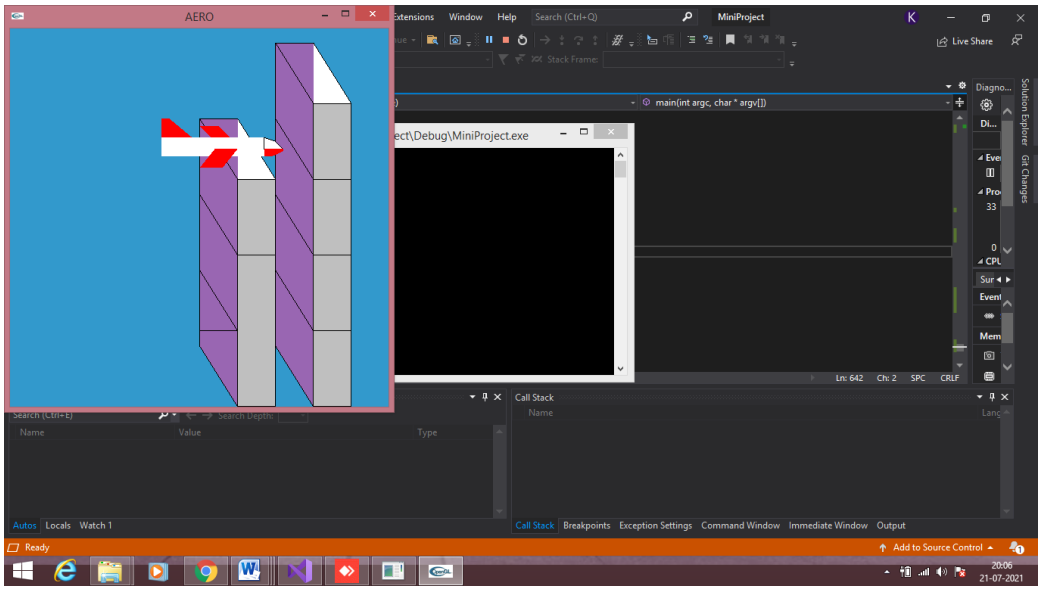
Screenshot 6.1. Flight on run-way



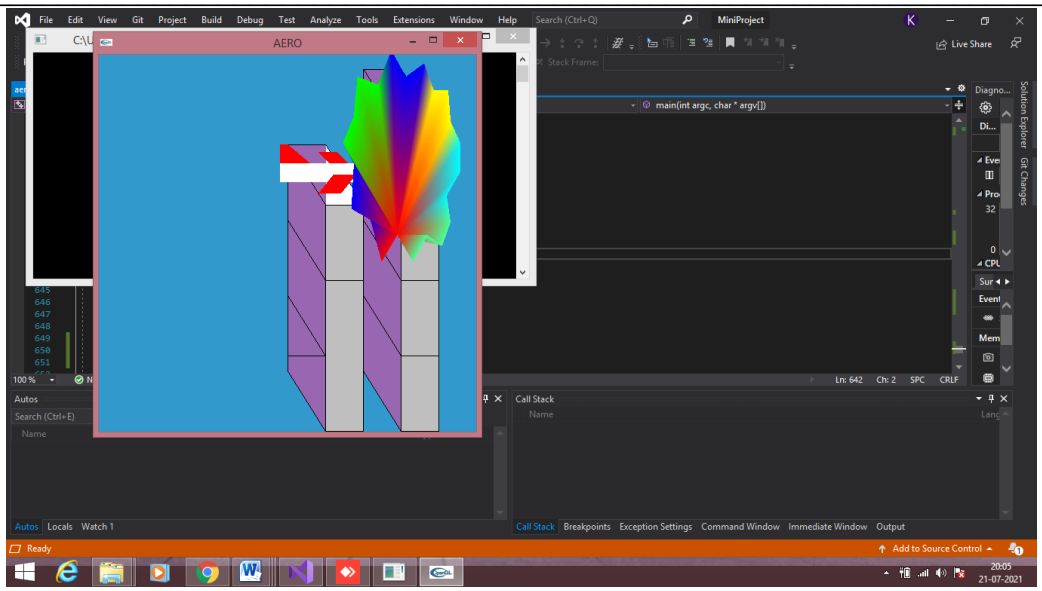
Screenshot 6.2. Flight when take off



Screenshot 6.3: Display 2



Screenshot 6.4: Display 3



Screenshot 6.5: Crashing simulation (Display 3)

Chapter 7

CONCLUSION

The project has been successful in demonstrating Aeroplane crashing simulation/game using a variety of features and options present in OpenGL. The project combines the richness of the graphics library and programming skills. The development of this project was very helpful to demonstrate the crashing simulation of aeroplane. Designing this Project, gave a good learning experience. It helped a lot to learn about computer graphics and design of graphical interfaces. The project thought how the inbuilt and user defined functions can be integrated to produce an efficient working code. The development of the mini project has given a good exposure to OpenGL by which some of the techniques which help in development of animated pictures and gaming were learnt.

Chapter 8

BIBLIOGRAPHY

Books

[1] Computer Graphics with OpenGL(4th ed.) [Hearn, Baker & Carithers 2013]

[2] Edward Angel: Interactive Computer Graphics- A Top Down approach with OpenGL,5th edition. Pearson Education, 2008

Websites

[3] https://www.opengl.org/

[4] https://www.glprogramming.com/red/

[5] https://www.codeproject.com/