VISVESVARAYA TECHNOLOGICAL UNIVERSITY
"Jnana Sangama", Belgaum-590018.

Computer Graphics Mini Project On
"Aeroplane Crash"

Submitted in partial fulfillment for the requirements of the VI Semester degree of

BACHELOR OF ENGINEERING
IN
COMPUTER SCIENCE AND ENGINEERING

For The Academic Year
2021-22
By

NEHA R RAO
(1DB19CS095)

Under The Guidance Of

Prof. Vishesh J
Asst. Professor
Dept. of CSE, DBIT

Department of Computer Science and Engineering

DON BOSCO INSTITUTE OF TECHNOLOGY

Kumbalagodu, Mysore Road, Bengaluru - 560 074.

DON BOSCO INSTITUTE OF TECHNOLOGY

Kumbalagodu, Bengaluru – 560 074.

DEPARTMENT OF COMPUTER SCINECE AND ENGINEERING

CERTIFICATE

This is to certify that the mini project report entitled **"Aeroplane Crash"** is a bonafide work carried out by **NEHA R RAO (1DB19CS095)** in partial fulfillment of award of Degree of **Bachelor of Engineering in Computer Science and Engineering** of Visvesvaraya Technological University, Belagavi, during the academic year 2021-2022. It is certified that all corrections/suggestions indicated for Internal Assessment have been incorporated. The mini project has been approved as it satisfies the academic requirements associated with the degree mentioned.

Signature of guide

……

Prof. Vishesh J

Asst. Professor,
Dept. of CSE,
DBIT, Bengaluru.

Signature of HOD

…….

Dr K B Shivakumar

Head of Dept.,
Dept. of CSE,
DBIT, Bengaluru.

DON BOSCO INSTITUTE OF TECHNOLOGY

Kumbalagodu, Bangalore – 560 074.

DECLARATION

I, **NEHA R RAO**, student of sixth semester B.E, Department of Computer Science and Engineering, Don Bosco Institute of Technology, Kumbalagodu, Bangalore, declare that the mini project work entitled **"Aeroplane Crash "** has been carried out by me and submitted in partial fulfillment of the course requirements for the award of degree in **Bachelor of Engineering** in **Computer Science and Engineering** of **Visvesvaraya Technological University, Belgaum** during the academic year **2021-2022**. The matter embodied in this report has not been submitted to any other university or institution for the award of any other degree or diploma.

Place: Bangalore
Date: 07/07/2022

NEHA R RAO
1DB19CS095

ACKNOWLEDGEMENT

Here by I am submitting the CG mini project report on **"Aeroplane Crash"**, as per the scheme of Visvesvaraya Technological University, Belgaum.

In this connection, I would like to express my deep sense of gratitude to my beloved institution Don Bosco Institute of Engineering and also, I like to express my sincere gratitude and indebtedness to **Prof. B.S. Umashankar, Principal, DBIT, Bangalore.**

I would like to express my sincere gratitude to **Dr. K.B. Shivakumar**, **Head of Department of Computer Science and Engineering,** for providing a congenial environment to work in and carryout my mini project.

I would like to express the deepest sense of gratitude to thank my Project Guide **Prof. Vishesh. J**, **Assistant Professor, Department of Computer Science and Engineering, DBIT, Bangalore** for his constant help and support extended towards me during the course of the project.

Finally, I am very much thankful to all the teaching and non-teaching members of the Department of Computer Science and Engineering, my seniors, friends and my parents for their constant encouragement, support and help throughout completion of mini project.

NEHA R RAO

(1DB19CS095)

ABSTRACT

Computer Graphics has grown into a very important topic in the branch of Computer Science. This is due to an effective and rapid communication formed between man and the machine. Human eye can absorb the information in a displayed diagram or perspective diagram much faster than it can scan a page or a table of contents.

The goal of a DEMOLITION OF A BUILDING BY AEROPLANE CRASH is to start with some randomized, shuffled, messy configuration. I make use of different concepts such as pushmatrix(),translate() ,popmatrix(),timer function

The code implemented makes use of various OpenGL functions for translation, rotation and keyboard callback function, built-in functions for solids and many more.

The concepts of computer graphics stand a backbone to achieve the aforementioned idea. Primitive drawing, event driven interactions and basic animation have been the important concepts brought out by this application.

LIST OF FIGURES

FIGURE: **PAGE:**

LIST OF ABBREVATIONS

- OpenGl: Open Graphics Library
- GLU: OpenGL Utility Library
- GLUT : OpenGL Utility Toolkit

CONTENTS

CHAPTER 1
INTRODUCTION

Graphics provides one of the natural means of communicating with a computer. Graphics has also become a key technology for communicating ideas, data and trends in most areas of commerce, science, engineering and education. So, graphics refers to pictures, sketch of building, flowcharts, control flow diagrams, bar charts, and pie charts.

Computer graphics is the creation, manipulation and storage of models and images of picture objects by the aid of computers. This was started with the display of plotters and CRT. Computer graphics is also defined as the study of techniques to improve the communication between user and machine. Thus, computer graphics is one of the effective media of communication between machine and user.

Classification:

Computer graphics is broadly classified into three categories:

- Based on type of Object
  - 2-Dimensional Graphics (Ex. Pixel, line, circle etc...)
  - 3-Dimensional Graphics (Ex. Cube, polyhedron etc...)
- Based on User Interaction
  - Interactive Computer Graphics
  - Non-Interactive Computer Graphics
- Based on Application
  - Business or presentation Graphics
  - Scientific Graphics
  - Scaled Drawings
  - Cartoons, games, and artwork

CHAPTER 2

LITERATURE SURVEY

Computer graphics started with the display of data on hardcopy plotters and cathode ray tube (CRT) screens soon after the introduction of computers.

Computer graphics today largely interactive, the user controls the contents, structure, and appearance of objects and of displayed images by using input devices, such as keyboard, mouse, or touch-sensitive panel on the screen. Graphics based user interfaces allow millions of new users to control simple, low-cost application programs, such as spreadsheets, word processors, and drawing programs.

OpenGL (Open Graphics Library) is a standard specification defining a cross-language, cross-platform API for writing applications that produce 2D and 3D computer graphics. The interface consists of over 250 different function calls which can be used to draw complex three-dimensional scenes from simple primitives. OpenGL was developed by Silicon Graphics Inc. (SGI) in 1992 and is widely used in CAD, virtual reality, scientific visualization, information visualization, and flight simulation. It is also used in video games, where it competes with Direct3D on Microsoft Windows platforms (see Direct3D vs. OpenGL). OpenGL is managed by the non-profit technology consortium, the Chromos Group.

In the 1980s, developing software that could function with a wide range of graphics hardware was a real challenge. By the early 1990s, Silicon Graphics (SGI) was a leader in 3D graphics for workstations. SGI's competitors (including Sun Microsystems, Hewlett-Packard, and IBM) were also able.

In addition, SGI had a large number of software customers; by changing to the OpenGL API, they planned to keep their customers locked onto SGI (and IBM) hardware for a few years while market support for OpenGL matured to bring to market 3D hardware, supported by extensions made to the PHIGS standard. In 1992, SGI led the creation of the OpenGL architectural review board (OpenGL ARB), the group of companies that would maintain and expand the OpenGL specification took for years to come.

On 17 December 1997, Microsoft and SGI initiated the Fahrenheit project, which was a joint effort with the goal of unifying the OpenGL and Direct3D interfaces (and adding a scene-graph API too). In 1998 Hewlett-Packard joined the project. [Donald D Hearn and M.

Pauline Baker, "Computer Graphics with OpenGL", 3rd Edition]. It initially showed some promise of bringing order to the world of interactive 3D computer graphics APIs, but on account of financial constraints at SGI, strategic reasons at Microsoft, and general lack of industry support, it was abandoned in 1999.

Many OpenGL functions are used for rendering and transformation purposes. Transformations functions like glRotate (), glTranslate (), glScaled () can be used.

OpenGL provides a powerful but primitive set of rendering command, and all higher-level drawing must be done in terms of these commands. There are several libraries that allow you to simplify your programming tasks, including the following:

OpenGL Utility Library (GLU) contains several routines that use lower-level OpenGL commands to perform such tasks as setting up matrices for specific viewing orientations and projections and rendering surfaces.

OpenGL Utility Toolkit (GLUT) is a window-system-independent toolkit, written by Mark Kill guard, to hide the complexities of differing window APIs.

To achieve the objective of the project, information related to the light sources is required with OpenGL we can manipulate the lighting and objects in a scene to create many different kinds of effects. It explains how to control the lighting in a scene, discusses the OpenGL conceptual model of lighting, and describes in detail how to set the numerous illumination parameters to achieve certain effects.

To demonstrate the transformation and lightening, effects, different polygons have to be used. Polygons are typically drawn by filling in all the pixels enclosed within the boundary, but we can also draw them as outlined polygons or simply as points at the vertices.

The properties of a light source like its material, diffuse, emissive, has to mention in the project. So, to design the light source and the objects, programming guide of an OpenGL is used.

CHAPTER 3

PROBLEM DEFINITION

3.1 PROPOSED SYSTEM

In the proposed system, OpenGL is a graphic software system designed as a streamlined, hardware-independent interface to be implemented on many different hardware platforms. To achieve these qualities, no commands for performing windowing tasks or obtaining user input are included in OpenGL; instead, we must work through whatever windowing system controls the hardware you're using.

OpenGL doesn't provide high-level commands for describing models of three-dimensional objects. Such commands might allow you to specify relatively complicated shapes such as automobiles, parts of the body, airplanes, or molecules. With OpenGL, you must build up your desired model from a small set of geometric primitives – points, lines, and polygons.

CHAPTER 4

SYSTEM REQUIREMENTS

4.1 SOFTWARE REQUIREMENTS

- Operating System : Windows 10
- Language : C++
- Compiler : Microsoft Visual Studio Code 2010
- OpenGL : library Files And dll files
- Editor : Visual C++

4.2 HARDWARE REQUIREMENTS

- Processor : Intel Core i3
- RAM : 8GB
- Hard Disk : 20 GB(approx.)
- Keyboard : Standard qwerty serial or PS/2 keyboard
- Mouse : Standard serial or PS/2 mouse
- Monitor : VGA color Monitor

CHAPTER 5
SYSTEM DESIGN

5.1 FLOWCHART/ARCHECTURE

The system is designed with the aeroplane crash with the following system design description. The program starts with start with the main function. There are init, display, keyboard function. Then the control flows to the init function then the control flows to display function then followed by keyboard function these are used to operate step by step process of transmission

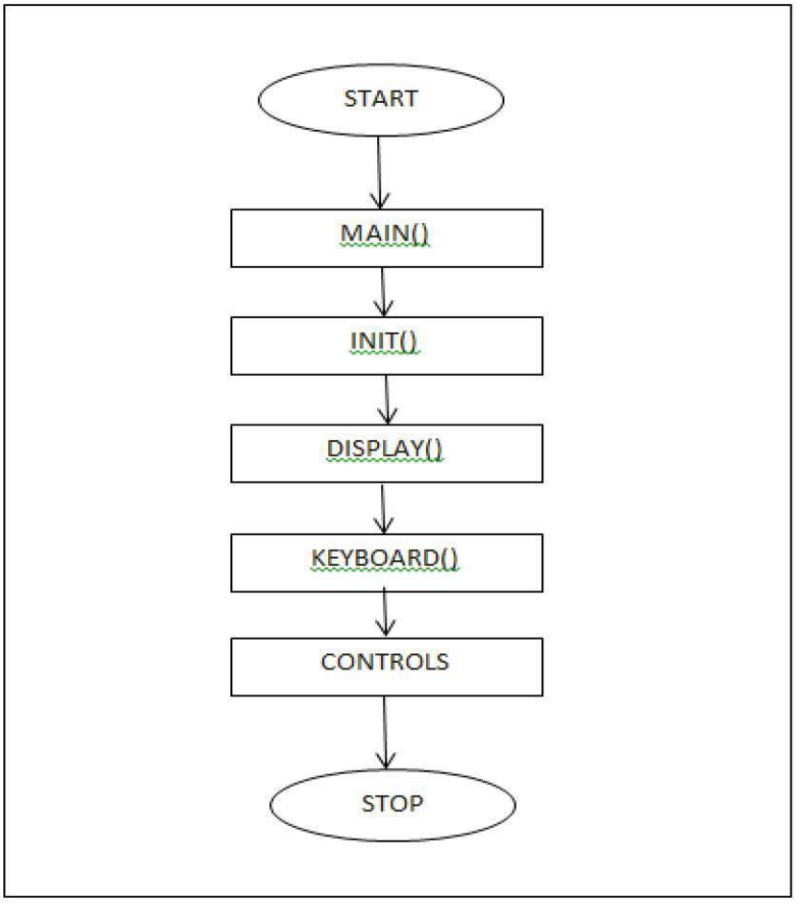

Figure 5.1: Flow Diagram

5.2 CONCEPTS AND PRINCIPLES OF OPENGL

5.2.1 THE OPENGL MODEL:

Figure relationships between an application program, the graphics system, input and output devices, and the user.

5.2.2 THE GRAPHICAL INTERFACE:

The application program has its own internal model. It draws the graphics using the facilities of the graphics system. The user views the graphics, and uses input devices, such as a mouse, to interact. Information about the users is sent back to the applications are sent back to the application, which decides what action to take. Typically, it will make changes to its internal model, which will cause the graphics to be updated, and so another loop in the interaction cycle begins.

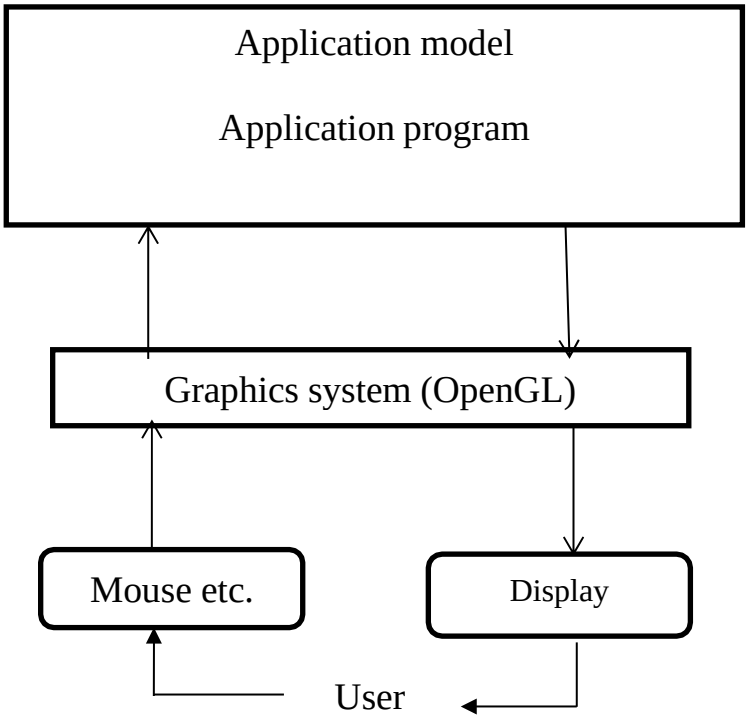

Fig 5.2: Graphical interaction loop

5.3 OPENGL related libraries

5.3.1 LIBRARIES:

OpenGL provides a powerful but primitive set of rendering command, and all the higher-level drawing must be done in terms of these commands. There are several libraries that allow you to simplify your programming tasks, including the following:

- OpenGL Utility Library (GLU) contains several routines that use lower-level OpenGL commands to perform such tasks as setting up matrices for specific viewing orientations and projections and rendering surfaces.
- OpenGL Utility Toolkit (GLUT) is a window-system-independent toolkit, written by Mark Kilgard, to hide the complexities of differing window APIs.

5.3.2 INCLUDE FILES:

For all OpenGL applications, you want to include the gl.h header file in every file. Almost all OpenGL applications use GLU, which also requires inclusion of the glu.h header file. So almost every OpenGL source file begins with:

#include<GL/gl.h>; #include<GL/glut.h>

5.4 WINDOWS MANAGEMENT IN OPENGL

Five routines perform tasks necessary to initialize a window.

- **glutInit(**int **argc*, char ***argv*) initializes GLUT and processes any command line arguments (for X, this would be options like –display and -geometry). It should be called before any other GLUT routine.
- **glutInitDisplayMode**(*unsigned int mode*) specifies whether to use an RGBA or color-index color model. We can also specify whether we want a single or double buffered window. GLUT SINGLE: selects a single-buffered window which is the default if isn't called;GLUT DOUBLE: selects a double-buffered window.
- **glutInitWindowPosition**(int*x*,int*y*) specifies the screen location for the upper-left corner of your window.
- **glutInitWindowSize** (int*width*,int*size*) specifies the size, pixels, of window.
- **glutCreateWindow** (char **string*) creates a window with an OpenGL context. It returns a unique identifier for the new window.

5.5 CALLBACK FUNCTIONS

All callback function more often just called a callback, in C function, written by the application programmer. But there's one important difference between a callback function and an ordinary C function: application never calls the callback function directly. Instead, the callback function is called by OpenGL.

5.5.1 THE DISPLAY CALLBACK:

glutDisplayFunc (void (**func*)(void)) is the first and most important event callback function. Whenever GLUT determines the contains of the window to be redisplayed.

- The callback function registered by glutDisplayFunc () is executed. Therefore, we should put all the routines you need to redraw the scene in the display callback function.
- Void glutDisplayFunc(void(**func*)(void));

glutDisplayFunc () registers the name of the callback function to be invoked when OpenGL needs to redisplay the contents of the window. The application must register a display function – it is not optional.

If your program changes the contents of the window, sometimes you will have to call glutPostRedisplay(*void*), which gives glutMainLoop () a nudge to call the registered display callback at its next opportunity.

5.5.2 THE KEYBOARD CALLBACK:

- void **glutKeyboardFunc**(void(**func*) (unsigned char *key*, intx, inty));
- **glutKeyboardFunc ()** registers the application function to call when OpenGL detects a key press generating an ASCII character. This can only occur when the mouse focus is inside the OpenGL window. It expects a function func () which returns void, and has three arguments key, x and y. So, it's function is:

```
void keyboard (unsigned char key, intx, inty)
{
/* called when a key is pressed */
}
```

Three values are passed to the callback function: key is the ASCII code of the key pressed; x and y give the pixel position of the mouse at the time. Inside the keyboard() callback, we look at the value of key. If it's 27 we call the standard C function exit () to terminate the program cleanly.

5.5.3 THE MOUSE CALLBACK

- void **glutMouseFunc** (void (***func*) (int*button*, int*state*, int*x*, int*y*));

glutMouseFunc () register an application callback function which GLUT will call when the user presses a mouse button within the window. The following values are passed to the callback function:

- button records which button was pressed, and can be
  -GLUT LEFT BUTTON
  -GLUT MIDDLE BUTTON
  -GLUT RIGHT BUTTON
- state records whether the event was generated by pressing the button (GLUT DOWM), or releasing it (GLUT UP).
- x,y give the current mouse position in pixels. Note: when using OpenGL with X, the mouse y position is measured from the top of the window.

5.5.4 HANDLING INPUT EVENTS:

You can use these routines to register callback commands that are invoked when specified events occur.

- **glutReshapeFunc**(void(***func*) (int*w*, inth)) indicates what action should be taken when the window is resized.
- **glutKeyboardFunc**(void(***func*)(unsignedchar*key*,int*x*,int*y*))and **glutMouseFunc**(void(**func)(int*button*,int*state*,int*x*,ints)) allow you to link a keyboardkey or a mouse button with routine that is invoked when the key or mouse button is pressed or released.
- **glutMotionFunc** (void (***func*) (int*x*,int*y*)) registers a routine to call back when the mouse is moved while a mouse button is also pressed.

5.5.5 TRANSLATION:

- void **glTranslatef**(GLfloat*x*,GLfloat*y*,GLfloat*z*);
  glTranslatef () creates a matrix M which performs a translation by (x,y,z) and then post-multiplies the current matrix by M.

5.5.6 SCALING:

- void **glScalef** (GLfloat*x*, GLfloaty, GLfloatz);
  glScalef() creates a matrix M which performs a scale by (x,y,z) and then post-multiplies the current matrix by M.

5.5.7 ROTATION:

- void **glRotatef**(GLfloat*angle*,GLfloat*x*,GLfloat*y*,GLfloat*z*);
  glRotate() creates a matrix M which performs a counter-clockwise rotation of angle in degrees. The axis about which the rotation occurs in the vector from the origin (0, 0, 0) to the point (x,y,z),and then post-multiplies the current matrix by M.

5.5.8 PUSH AND POP OPERATIONS:

There are two functions which operate on the current matrix stack:

glPushMatrix()

glPopMatrix()

void **glPushMatrix(*void*);** Pushes the current matrix stack down one level. The matrix on the top of the stack is copied into the next-to-top position.

void **glPopMatrix(*void*);** Pops the current matrix stack, moving each matrix in the stack one position towards the top of the stack.

The current matrix stack is determined by the most recent call to **glMatrixMode().**

5.6 RUNNING THE PROGRAM

After all the setup is completed, GLUT programs enter an event processing loop, **glutMainLoop().**

void **glutMainLoop**(void) enters the GLUT processing loop, never to return. Registered callback functions will be called when the corresponding events instigate them.

CHAPTER 6

IMPLEMENTATION

```
#include<stdio.h>
#include<GL/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);//outlineof 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 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);
}
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();
glutTimerFunc(100,update,0);
glutMainLoop();
}
```

CHAPTER 7

SNAPSHOTS

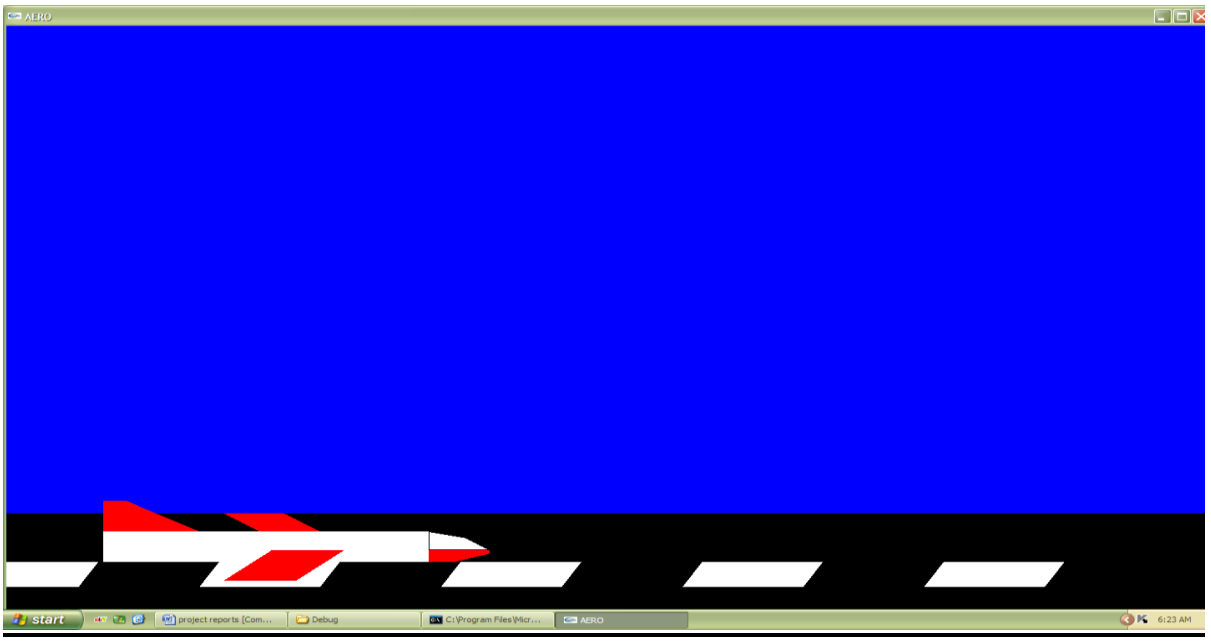

Fig 7.1 Ready state

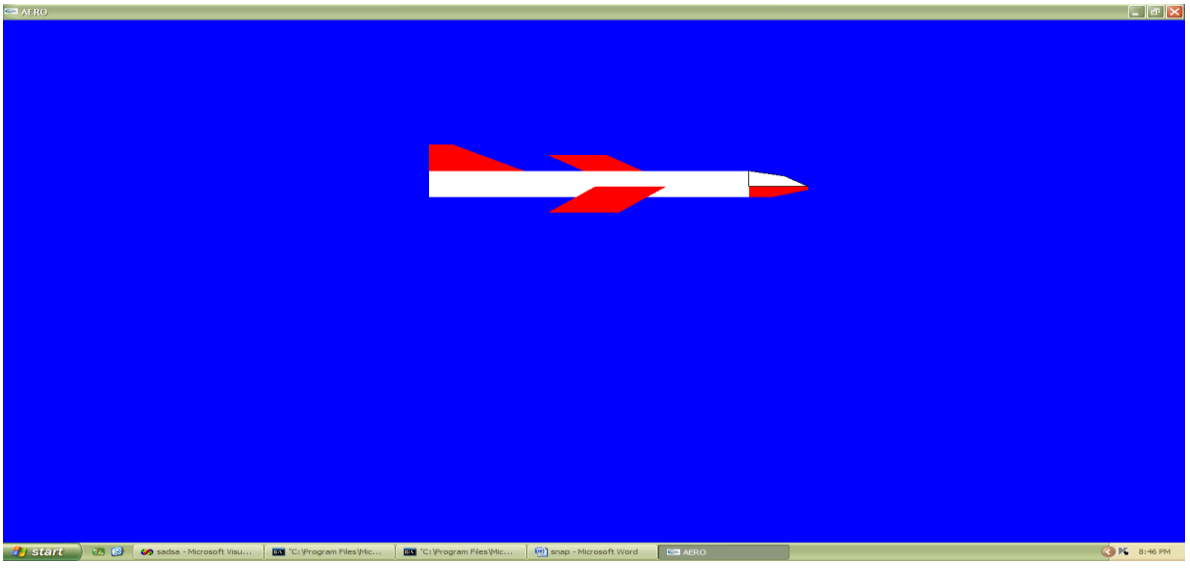

Fig 7.2 Aeroplane take off

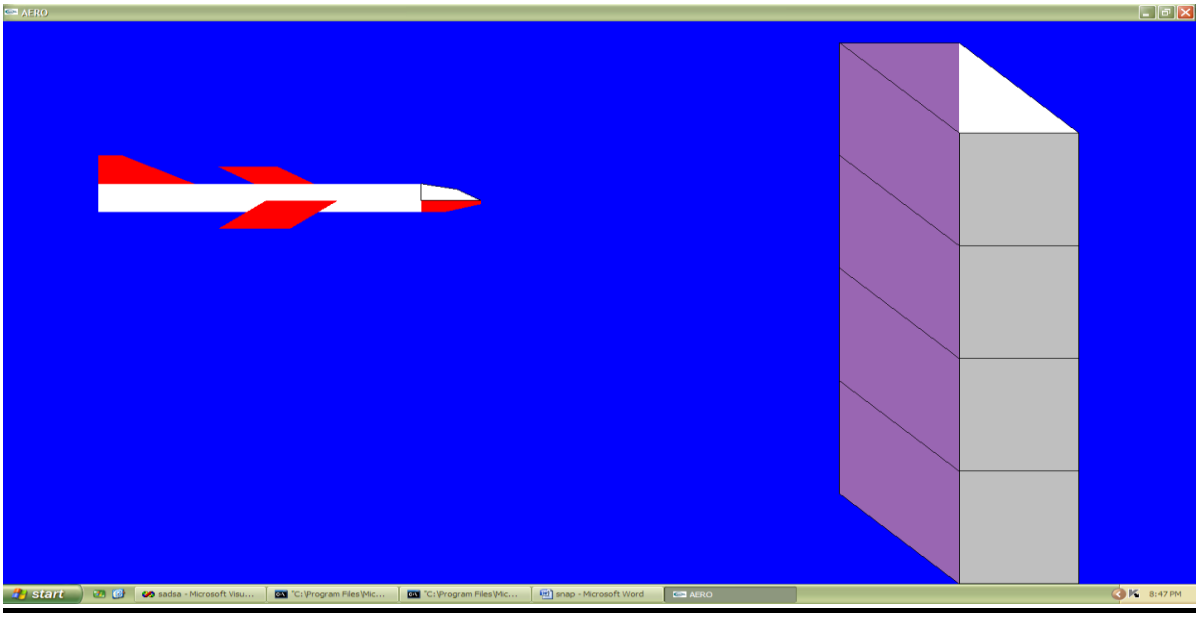

Fig 7.3Aeroplane flying

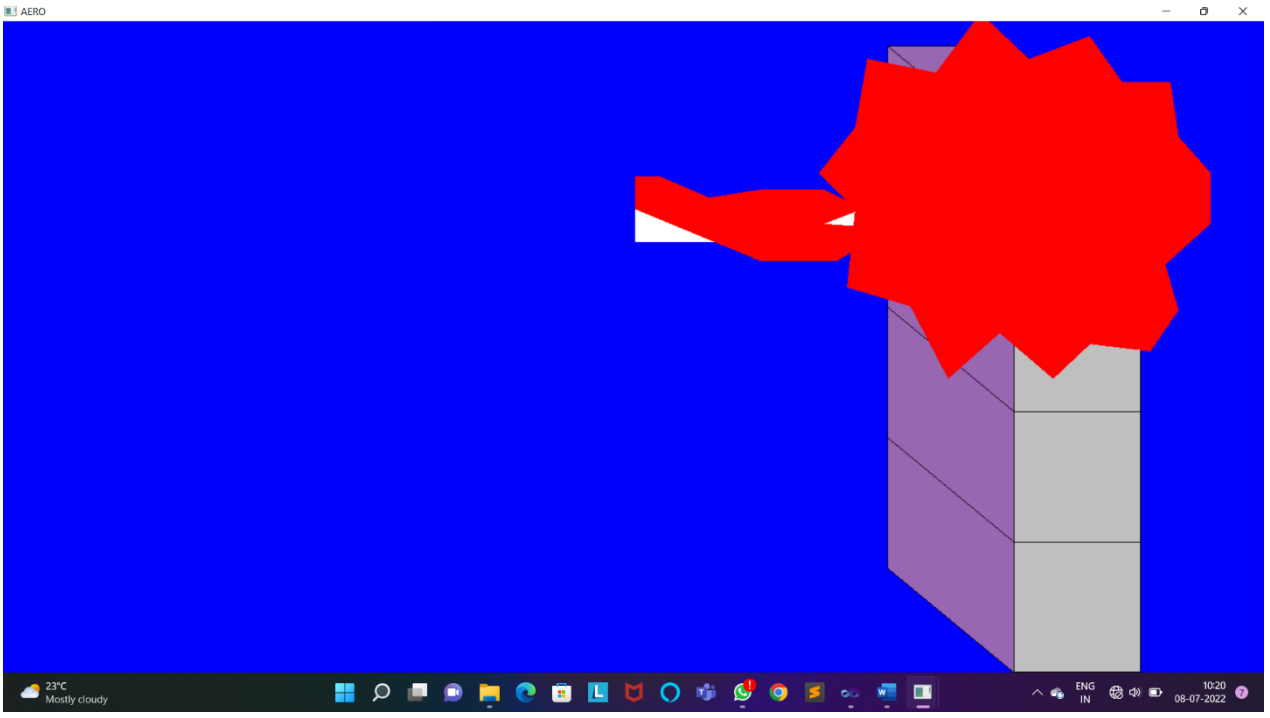

Fig 7.4 Aeroplane crash

CONCLUSION

The aim of project is to illustrate the working of Aeroplane crash and working it using various functions in OpenGL. The code has been written in OpenGL. The code contains the functions that accomplish all the tasks required for the project. It can be further improved upon to provide better facilities and user interface.

One major improvement which can be made to this package is the inclusion of the Pop Up Menus which we can use with the mouse to create sophisticated interactive application

REFERENCES

Books:

Edward Angel, "Interactive Computer Graphics", 5th edition, Pearson Education, 2005

Donald D Hearn and **M. Pauline Baker,** "Computer Graphics with OpenGL", 3rd Edition

Websites:

- **http://www.opengl.org**
- **http://glprogramming.com/red**
- **http://jerome.jouvie.free.fr/OpenGl**
- **https://www.3dgep.com/introduction-to-opengl-and-glsl/**