**VISVESVARAYA TECHNOLOGICAL UNIVERSITY**

BELGAUM – 590014



A CG LAB (17CSL68) MINI PROJECT REPORT ON

“AEROPLANE CRASH”

***Submitted in partial fulfillment of the requirement of 6th semester***

**BACHELOR OF ENGINEERING IN COMPUTER SCIENCE AND ENGINEERING BY**

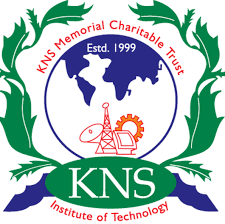
**ROSHINI P 1KN17CS070**

**MERCY RABECAL P 1KN17CS052**

***UNDER THE GUIDANCE OF***

**Ms.AMEENA ANJUM Mrs.AMBIKA.M.ALGUD**

Asst. Prof, Dept of CSE Asst. Prof, Dept of CSE

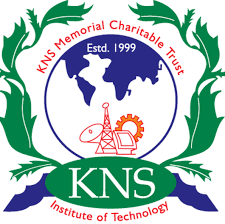
****

**K.N.S INSTITUTE OF TECHNOLOGY**

**Thirumenahalli, Yelahanka, Bangalore – 560064**

**K.N.S INSTITUTE OF TECHNOLOGY**

**Department Of Computer Science and Engineering**

****

**CERTIFICATE**

Certified that the **GRAPHICAL PACKAGE** project work entitled **“AEROPLANE CRASH”**  is a bonafide work carried out by **ROSHINI P (1KN17CS070) and MERCY RABECAL P (1KN17CS052)** in partial fulfillment of for the award of bachelor of Engineering In Information Science and Engineering of Visvesvaraya Technological University, Belgaum during the academic year 2019-2020.

It is certified that all correction/suggestions indicated for internal assessments have been incorporated in the report deposited in the department library. The project report has been approved as it satisfies the academic requirements in aspects of project work prescribed for the Bachelor Of Engineering degree.

………………………….. …………………………… …...……………………

**INTERNAL GUIDE INTERNAL GUIDE HOD**

Ms. Ameena Anjum Mrs. Ambika M Algud Mr.Mohamed Shakir

Asst Prof, Dept of CSE Asst Prof, Dept of CSE Dept of CSE

KNSIT Bangalore. KNSIT Bangalore. KNSIT Bangalore.

**External Viva:**

**Name of the Examiner** **Signature with Date**

1. **1.**

**2. 2.**

**ACKNOWLEDGEMENT**

The satisfaction and euphoria that accompanies the successful completion of any task would be incomplete without the mentioning the people who made it possible. If there is a driving force that kept use going on in doing this project, it is the constant support of our guide **Ms.Ameena Anjum ,Mr. Mohammed Ziaulla and Mrs.Ambika M Algud.** We present our sincere and heartiest thanks to him, for giving us a patienthearing and clearing our doubts.

We would like to express our profound thanks to our HOD **Mr. Mohamed Shakir** for his support and encouragement towards us at every stage in successful completion of our project.

We would like to express our gratitude to **Dr. S.M Prakash, Principal , KNSIT Bangalore,** for providing us congenial environment to work in.

We place our gratitude to our parents who always encouraged us to pursue our interest, who have led us onto the right path. We are undoubted for all the love and affection they bestowed upon us. Many are responsible for the knowledge and experience we have gained during our project and throughout the course. With profound sense of gratitude and regards we acknowledge for support extended by our friends for their support in completion of this project.

Finally, we would also like to thank all those who knowingly or unknowingly helped to us all throughout our project.

**ROSHINI P 1KN17CS070**

**MERCY RABECAL P 1KN17CS052**

# ABSTRACT

The main aim of Aeroplane Crash Computer Graphics Mini Project is to illustrate the concepts and usage of pre-built functions in OpenGL**.**The demolition of a building by aeroplane crash. The objects are building,aeroplane and road.The object aeroplane has the movement and the other objects are stationary.when the aeroplane starts moving from the run way and starts flying on the sky suddenly there is a building in the middle of the sky the aeroplane crashes the building.

Computer graphics remains one of the most exciting and rapidly growing computer fields.Computer graphics has now become a common element in user interfaces, data visualization,television commercial, motion pictures, and many-many other application. With the advantages of Graphics, modern subject, we can be able to design different applications and tools which can be beneficial in our daily life. Besides, some entertaining applications can also be designed usingGraphics such as Games, Photo Effects etc. We are trying to apply the Graphics applications on designing aeroplane crash.

**CONTENTS**

|  |  |  |
| --- | --- | --- |
| **SL.NO** | **TITLE** | **PAGE.NO** |
| **1.** | **INTRODUCTION**  1.1 COMPUTER GRAPHICS  1.2 USES OF COMPUTER GRAPHICS  1.3 INTRODUCTION TO OPENGL  1.4 HISTORY  1.5 FEATURES OF OPENGL  1.6 BASIC OPENGL OPERATION  1.7 OPENGL INTERFACE  1.8 OPENGL AS A STATE MACHINE  1.9 OPENGL RELATED LIBRARIES  1.10 ASSOCIATED UTILITY LIBRARIES  1.11 OPENGL:CAMERA  1.12 DATA TYPES  1.13 OBJECTIVES | **1**  **2**  **2-3**  **4** |
| **2.** | **SYSTEM REQUIREMENTS** | **12** |
| **3.** | **DESIGN**  3.1 INTRODUCTION  3.2 USER DEFINED FUNCTIONS  3.3 ROOT CAUSES OF ACCIDENTS IN REAL LIFE  3.4 GRAPHS RELATED TO AEROPLANE ACCIDENTS IN REAL LIFE | **13** |
| **4.** | **CODE** | **17-25** |
| **5.** | **IMPLEMENTATION**  5.1 BASIC FUNCTIONS  5.2 TRANSFORMATION FUNCTIONS  5.3 FUNCTIONS USED TO SET THE VIEWING VOLUME  5.4 CALL BACK FUNCTIONS  5.5 MAIN FUNCTIONS | **26-29** |
| **6.** | **SNAPSHOTS** | **29-30** |
| **7.** | **CONCLUSION AND FURTHER ENHANCEMENT** | **31-32** |
| **8.** | **REFERENCES & BIBLIOGRAPHY** | **33** |

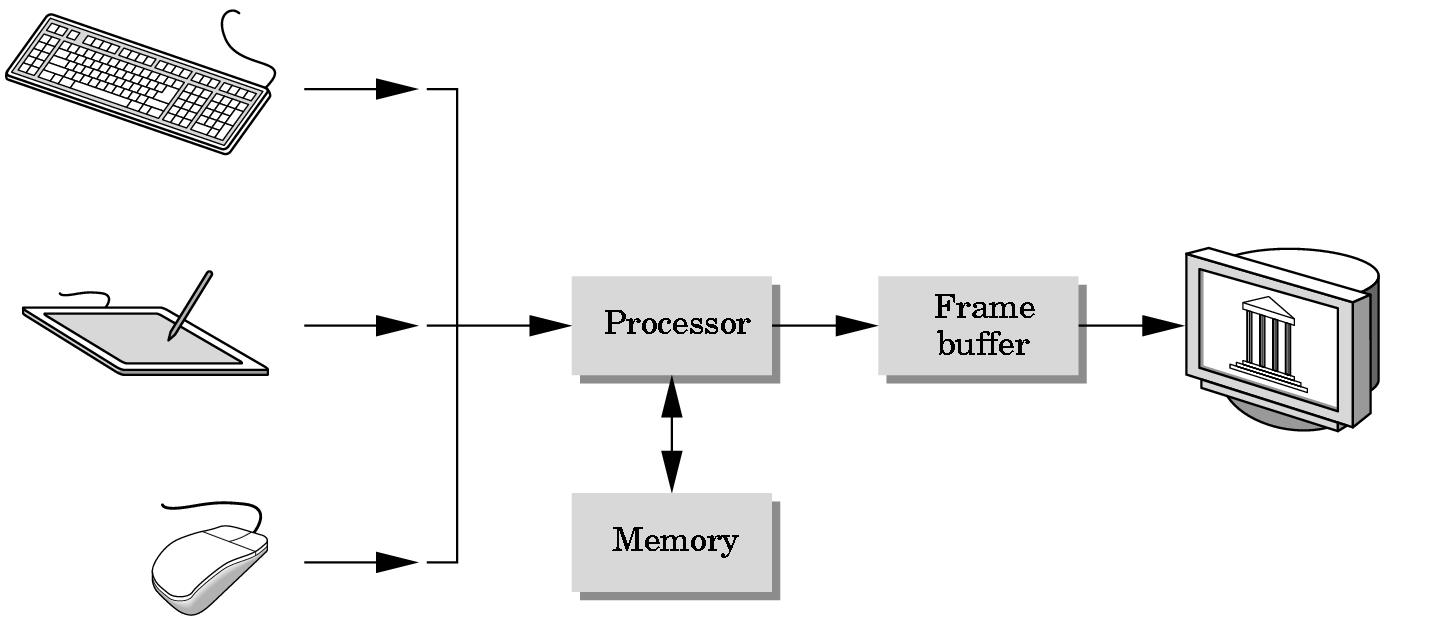
**CHAPTER 1**

**INTRODUCTION**

**1.1 COMPUTER GRAPHICS**

Computer graphics started with the display of data on hardcopy plotters and Cathode Ray Tube (CRT) screens soon after the Introduction of computers themselves. It has grown to include the Creation, Storage and Manipulation of Models and Images of objects. These models come from a diverse and expanding set of fields, and include physical, mathematical, engineering, architectural and even conceptual structures, natural phenomenon and so on. Computer graphics today is largely interactive: the user controls the contents, structure, and appearance of objects and of their displayed images by using input devices, such as a keyboard, mouse, or touch-sensitive panel on the screen. Because of the close relationship between the input devices and the display, the handling of such devices is included in the study of computer graphics.

A computer graphics system is a computer system with all the components of the general purpose computer system. There are five major elements in system: input devices, processor, memory, frame buffer, output devices.



**1.2 USES OF COMPUTER GRAPHICS:**

Computer graphics is used in many different areas of industry, business, government, education, entertainment etc.

* **User Interfaces**

Word-processing, spreadsheet and desktop-publishing programs are typical applications of such user-interface techniques.

* **Interactive Plotting in Business, Science and Technology**

The common use of graphics is to create 2D and 3D graphs of mathematical, physical and economic functions, histograms, and bar and pie charts.

* **Computer Aided Drafting and Design (CAD)**

In CAD, interactive graphics is used to design components and systems of mechanical, electrical and electronic devices including structures such as buildings, automobile bodies, aero planes, ship hulls etc.

* **Simulation and Animation for Scientific Visualization and Entertainment**

Computer-produced animated movies are becoming increasing popular for scientific and engineering visualization. Cartoon characters will increasingly be modeled in the computer as 3D shape descriptions whose movements are controlled by computer commands.

* **2D Graphics**

These editors are used to draw 2D pictures (line, rectangle, circle and ellipse) alter those with operations like cut, copy and paste. These may also support features like translation, rotation etc.

* **3D Graphics**

These editors are used to draw 3D pictures (line, rectangle, circle and ellipse).These may also support features like translation, rotation etc.

**1.3 INTRODUCTION TO OPENGL**

OpenGL is a software interface to graphics hardware. This interface consists of about 150 distinct commands that you use to specify the objects and operations needed to produce interactive three- dimensional applications.

OpenGL is 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, you must work through whatever windowing system controls the particular hardware you're using. Similarly, 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.

A sophisticated library that provides these features could certainly be built on top of OpenGL. The OpenGL Utility Library (GLU) provides many of the modeling features, such as quadric surfaces and NURBS (Non-Uniform Rational B-Splines) curves and surfaces. GLU is a standard part of every OpenGL implementation. Also, there is a higher-level, object-oriented toolkit, Open Inventor, which is built atop OpenGL, and is available separately for many implementations of OpenGL.

OpenGL provides a set of commands to render a three dimensional scene. OpenGL is a hardware- and system-independent interface. An OpenGL-application will work on every platform, as long as there is an installed GLUT library.

GLUT is a complete API written by Mark Kilgard which allows us to create windows and render the 2D or 3D scenes. It exists for several platforms, that means that a program which uses GLUT can be compiled on many platforms without (or at least with very few) changes in the code.

**1.4 HISTORY**

As a result, SGI released the OpenGL standard In the 1980s, developing software that could function with a wide range of graphics hardware was a real challenge. Software developers wrote custom interfaces and drivers for each piece of hardware. This was expensive and resulted in much duplication of effort.

By the early 1990s, Silicon Graphics (SGI) was a leader in 3D graphics for workstations. Their IRIS GL API was considered the state of the art and became the de facto industry standard, overshadowing the open standards-based PHIGS. This was because IRIS GL was considered easier to use, and because it supported immediate mode rendering. By contrast, PHIGS was considered difficult to use and outdated in terms of functionality.

SGI's competitors (including Sun Microsystems, Hewlett-Packard and IBM) were also able to bring to market 3D hardware, supported by extensions made to the PHIGS standard. This in turn caused SGI market share to weaken as more 3D graphics hardware suppliers entered the market. In an effort to influence the market, SGI decided to turn the Iris GL API into an open standard.

SGI considered that the Iris GL API itself wasn't suitable for opening due to licensing and patent issues. Also, the Iris GL had API functions that were not relevant to 3D graphics. For example, it included a windowing, keyboard and mouse API, in part because it was developed before the X Window System and Sun's NEWS systems were developed.

### 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. Meanwhile, SGI would continue to try to maintain their customers tied to SGI hardware by developing the advanced and proprietary Iris Inventor and Iris Performer programming APIs.

### 1.5 FEATURES OF OPENGL

#### INDUSTRY STANDARD

An independent consortium, the OpenGL Architecture Review Board, guides the OpenGL specification. With broad industry support, OpenGL is the only truly open, vendor-neutral, multiplatform graphics standard.

#### STABLE

OpenGL implementations have been available for more than seven years on a wide variety of platforms. Additions to the specification are well controlled, and proposed updates are announced in time for developers to adopt changes. Backward compatibility requirements ensure that existing applications do not become obsolete.

#### RELIABLE AND PORTABLE

All OpenGL applications produce consistent visual display results on any OpenGL API- compliant hardware, regardless of operating system or windowing system.

#### SCALABLE

OpenGL API-based applications can run on systems ranging from consumer electronics to PCs, workstations, and supercomputers. As a result, applications can scale to any class of machine that the developer chooses to target.

#### EVOLVING

Because of its thorough and forward-looking design, OpenGL allows new hardware innovations to be accessible through the API via the OpenGL extension mechanism. In this way, innovations appear in the API in a timely fashion, letting application developers and hardware vendors incorporate new features into their normal product release cycles.

#### EASY TO USE

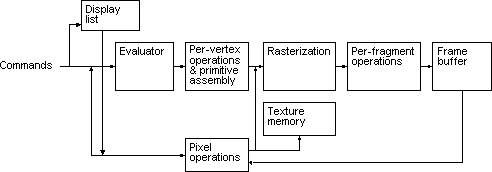
OpenGL is well structured with an intuitive design and logical commands. Efficient OpenGL routines typically result in applications with fewer lines of code than those that make up programs generated using other graphics libraries or packages. In addition, OpenGL drivers encapsulate information about the underlying hardware, freeing the application developer from having to design for specific hardware features.

#### WELL-DOCUMENTED

Numerous books have been published about OpenGL, and a great deal of sample code is readily available, making information about OpenGL inexpensive and easy to obtain.

### 1.6 BASIC OPENGL OPERATION

The following diagram illustrates how OpenGL processes data. As shown, commands enter from the left and proceed through a processing pipeline. Some commands specify geometric objects to be drawn, and others control how the objects are handled during various processing stages.

 Fig: 1.4 OpenGL Block Diagram

The processing stages in basic OpenGL operation are as follows:

#### DISPLAY LIST

Rather than having all commands proceed immediately through the pipeline, you can choose to accumulate some of them in a display list for processing later.

#### EVALUATOR

The evaluator stage of processing provides an efficient way to approximate curve and surface geometry by evaluating polynomial commands of input values.

#### PER-VERTEX OPERATIONS AND PRIMITIVE ASSEMBLY

OpenGL processes geometric primitives - points, line segments, and polygons all of which are described by vertices. Vertices are transformed, and primitives are clipped to the viewport in preparation for rasterization.

#### RASTERIZATION

The rasterization stage produces a series of frame-buffer addresses and associated values using a two-dimensional description of a point, line segment, or polygon. Each so produced is fed into the last stage, per-fragment operations.

#### PER-FRAGMENT OPERATIONS

These are the final operations performed on the data before it is stored as pixels in the frame buffer Per-fragment operations include conditional updates to the frame buffer based on incoming and previously stored z values (for z buffering) and blending of incoming pixel colors with stored colors, as well as masking and other logical operations on pixel values.

#### PIXEL OPERATION

Input data can be in the form of pixels rather than vertices. Such data which might describe an image for texture mapping skips the first stage of processing and instead processed as pixels in the pixel operation stage.

#### TEXTURE MEMORY

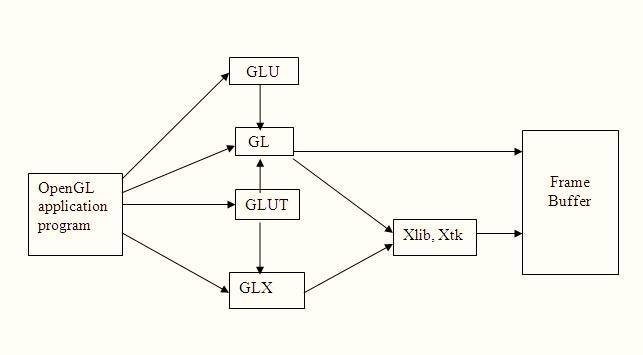
The result of pixel operation stage is either stored as texture memory for use in rasterization stage or rasterised and resulting fragment merged into the frame buffer just as they were generated from the geometric data.

**1.7 The OpenGL interface**

Most of our application will be designed to access OpenGL directly through functions in three libraries. Functions in the main GL (or OpenGL in windows) library have names that begin with the letters gl and are stored in a library usually referred to as GL (or OpenGL in windows).

 The second is the **OpenGL Utility Library** (GLU). This library uses only GL functions but contains code for creating common objects and simplifying viewing. All functions in GLU can be created from the core GL library but application programmers prefer not to write the code repeatedly. The GLU library is available in all OpenGL implementations; functions in the GLU library begin with letters glu.

 To interface with the window system and to get input from external devices into our programs, we need at least one more system-specific library that provides the “glue” between the window system and OpenGL. For the X window system, this library is functionality that should be expected in any modern windowing system.

Below Figure shows the organization of the libraries for an X Window System environment. For this window system, GLUT will use GLX and the X libraries. The application program, however, can use only GLUT functions and thus can be recompiled with the GLUT library for other window systems.

**Fig:Library Organization**

**1.8 OpenGL as a State Machine**

OpenGL is a state machine. It is called a state machine because it can be put into various states until you change them. As you've already seen, the current color is a state variable. You can set the current color to white, red, or any other color, and thereafter every object is drawn with that color until you set the current color to something else.

The current color is only one of many state variables that OpenGL maintains. Others control such things as the current viewing and projection transformations; line and polygon stipple patterns, polygon drawing modes, pixel-packing conventions, positions and characteristics of lights, and material properties of the objects being drawn. Many state variables refer to modes that are enabled or disabled with the command **glEnable()** or **glDisable()**. Each state variable or mode has a default value, and at any point you can query the system for each variable's current value.

**1.9 OpenGL-Related Libraries**

OpenGL provides a powerful but primitive set of rendering commands, and all higher-level drawing must be done in terms of these commands. Also, OpenGL programs have to use the underlying mechanisms of the windowing system. A number of libraries exist to allow you to simplify your programming tasks, including the following:

* The 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, performing polygon tessellation, and rendering surfaces. This library is provided as part of every OpenGL implementation. GLU routines use the prefix **glu**.
* The OpenGL Utility Toolkit (GLUT) is a window system-independent toolkit. It contains rendering commands but is designed to be independent of any window system or operating system. Consequently, it contains no commands for opening windows or reading events from the keyboard or mouse. Since OpenGL drawing commands are limited to those that generate simple geometric primitives (points, lines, and polygons), GLUT includes several routines that create more complicated three-dimensional objects such as a sphere, a torus, and a teapot. GLUT may not be satisfactory for full-featured OpenGL applications, but you may find it a useful starting point for learning OpenGL.

**1.10 Associated utility libraries**

Several libraries are built on top of or beside OpenGL to provide features not available in OpenGL itself. Libraries such as GLU can be found with most OpenGL implementations, and others such as GLUT and SDL have grown over time and provide rudimentary cross-platform windowing and mouse functionality, and if unavailable can easily be downloaded and added to a development environment. Simple graphical user interface functionality can be found in libraries like GLUI or FLTK. Still other libraries like GLAux (OpenGL Auxiliary Library) are deprecated and have been superseded by functionality commonly available in more popular libraries.

Other libraries have been created to provide OpenGL application developers a simple means of managing OpenGL extensions and versioning. Examples of these libraries include GLEW (the OpenGL Extension Wrangler Library) and GLEE (the OpenGL Easy Extension Library). In addition to the aforementioned simple libraries, other higher-level object-oriented scene graph retained mode libraries exist such as PLIB, OpenSG, OpenSceneGraph, and OpenGL Performer. These are available as cross-platform free/open source or proprietary programming interfaces written on top of OpenGL and systems libraries to enable the creation of real-time visual simulation applications.

Comprises several libraries with varying levels of abstraction: GL, GLU, and GLUT

• Software Interface to Graphics Hardware

• Consists of about 150 Distinct Commands

* Hardware-independent Interface
* no command for windows or user input handling
* does not include low-level I/O management

• Mid-level, device-independent, portable graphics subroutine package

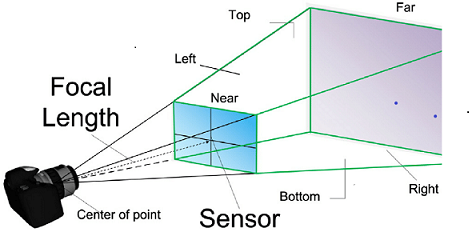
• Developed primarily by SGI

• 2D/3D graphics, lower-level primitives (polygons)

• Basis for higher-level libraries/toolkits

**1.11 OpenGL: Camera**

* Two things to specify:
* Physical location of camera in the scene (MODELVIEW matrix in OpenGL).
* Projection properties of the camera (PROJECTION matrix in OpenGL):





void glFrustum(GLdouble left, GLdouble right, GLdouble bottom,GLdouble top, GLdouble near, GLdouble far);

### 1.12 DATA TYPES

OpenGL supports different data types. A list of data types supported by OpenGL is given in the following table.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Sl no.** | **Suffix** | **Data type** | **C type** | **OpenGL type** |
| 1. | B | 8 bit int | signed int | GLbyte |
| 2. | S | 1 bit int | Short | GLshort |
| 3. | I | 32 bit int | Long | GLint , GLsizei |
| 4. | F | 32 bit float | Float | GLfloat ,GLclampf |
| 5. | D | 64 bit float | Double | GLdouble, GLclampd |
| 6. | Ub | 8 bit unsigned | unsigned char | GLubyte, GLboolean |
| 7. | Us | 16 bit unsigned | unsigned short | GLushort |
| 8. | Ui | 32 bit unsigned | unsigned int | GLuint, GLenum, GLbitfield |

#### Table 1.1: DATA TYPES LIST

**1.13 OBJECTIVES**

The objectives of this study are summarized below:

* + - To develop a Open GL software called “AEROPLANE CRASH”.
    - To implement the features of graphics.
    - To interface the applications of graphics to the real world.
    - To give some benefits to the disability.
    - To make the life easier.
    - To build the environment to improve his quick thinking/accuracy.
    - To become familiarization with Graphics and its logical coding.

**CHAPTER 2**

**SYSTEM REQUIREMENTS**

System requirements are intended to communicate in precise way, the functions that the system must provide. To reduce ambiguity, they may be written in a structured form of natural language supplemented by tables and system models.

**2.1 Purpose**

          Purpose of this project is to learn and understand computer graphics using opengl and to implement opengl functions. The aim is to simulate the truck movements.

**2.2 Scope**

The scope of the project is to portray a 3D environment that provides the user with the example of various basic transformations and viewing techniques available in OpenGL. It provides most of the features that a 3-D graphics editor should have. It is developed in C.

**2.3   Functional Requirements**

* mouse();
* glTranslate();
* glRotate();
* glScale();
* menu();
* main();
* display();
* reshape();

**2.4   Non Functional Requirements**

**MINIMUM HARDWARE REQUIREMENTS:**

The physical components required are:

* Processor - Pentium Pro
* Memory - 128MB RAM
* 40GB Hard Disk Drive

**MINIMUM** **SOFTWARE REQUIREMENTS:**

The software used in building this program are as specified:-

* Operating system –Windows (XP, Vista,7), Linux(Ubuntu 10.4)
* Tools: Microsoft Visual C++ 2008
* Graphics Library – glut. h
* OpenGL 2.0

**CHAPTER 3**

**DESIGN**

**3.1 INTRODUCTION**

The main goal of the project is to represent the concepts learned in the OpenGL. The demolition of a building by aeroplane crash. The objects are building,aeroplane and road.The object aeroplane has the movement and the other objects are stationary. The aeroplane has mouse function in GLUT\_RIGHT\_BUTTON and it has two menus that is start and quit.

At beginning the aeroplane is stationary in the x-axis .when the start menu is pressed the aeroplane starts moving from the runway in x-axis and after some time its starts flying in the sky at certain angle in y-axis.suddenly there is a obstruction in the middle of the sky that is the building the plane does not get proper control and crashes the building.



FIGURE:AEROPLANE

**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.

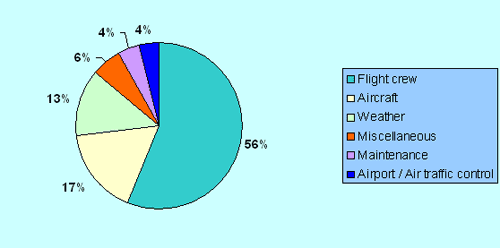
glClear(GL\_COLOR\_BUFFER\_BIT);

glFlush();

* **timerfunc():** This function starts a timer in the event loop that delays the event loop for delay miiliseconds.
* **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,or pop the current matrix 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 Root causes of accidents in real life

It is quite rare for an accident to be explained by one single cause. Almost every mishap is the consequence of a chain of events and accident reports usually discriminate between the main cause and a number of contributing factors. The following graph shows the distribution of main causes identified in plane crashes.

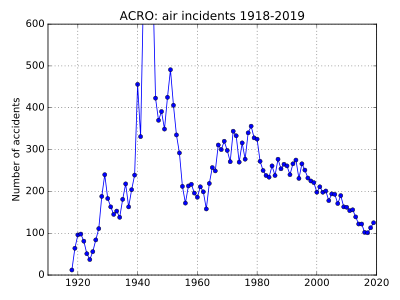


**Root causes of plane accidents**

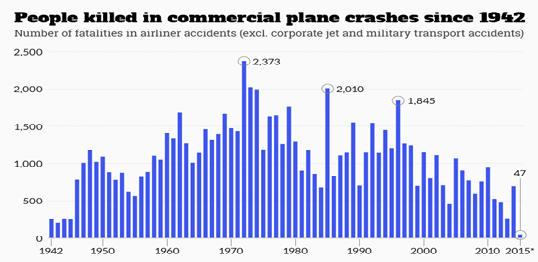
The main root cause is human error. In order to try and eliminate this as a source of accidents, crews are requested to follow a strict training routine. Next come aircraft failures, but these are less likely when it comes to modern aircraft.

**3.4 Graphs Related to aeroplane accidents in real life**

The below graph shows the air accident years from 1918-2019 in the x-axis and the number of accidents that took place in y-axis.



From above graph it is clearly mentioned that in the year between 1940-1960 more number of accidents has been occurred upto 600 and above.later on from year 1960 onwards there is an reduction in the accidents taking place.



In the above scenario we can come across the years and number of people died in according to x-axis and y-axis.

In the year 1942the first crash in the approximately around 250 people were passed away and later if we check for number of people killed where reached to a peak of around 2373 by the year 1972 and in the year 1985 around 2010 people were passed away.If we go on seeinglike this in the year 2014 it was 47 people and it was a drastic reduction and finally year 2015 it was still reduced to below 500.

**CHAPTER 4**

**CODE**

#include<stdio.h>

#include<GL/glut.h>

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

void road();

void display2();

void display3();

void blast();

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<=0.0)// moving of run way

b=0.0;

glutTimerFunc(150,update,0);

//delay

glutPostRedisplay();

}

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();

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();

}

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 road

glVertex2f(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);

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);

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);

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);

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);

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);

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();

}

void display3()

{

glClear(GL\_COLOR\_BUFFER\_BIT);

building();

glPushMatrix();

glTranslated(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();

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);

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);

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(0.0f,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\_menu(int ch)

{

switch(ch)

{

case 1:glutPostRedisplay();

glutTimerFunc(150,update,0);

break;

case 2:exit(0);

break;

}

glutPostRedisplay();

}

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

{

glutInit(&argc, argv);

glutInitDisplayMode(GLUT\_SINGLE | GLUT\_RGB);

glutInitWindowSize(500.0,500.0);

glutInitWindowPosition(0,0);

glutCreateWindow("AERO");

glutCreateMenu(main\_menu);

glutAddMenuEntry("start",1);

glutAddMenuEntry("Quit",2);

glutAttachMenu(GLUT\_RIGHT\_BUTTON);

glutDisplayFunc(display);

myinit();

glutMainLoop();

}

**CHAPTER 5**

**IMPLEMENTATION**

**5.1 Basic Functions**

* **GL\_LINES** - Treats each pair of vertices as an independent line segment. Vertices 2n - 1 and 2n define line n. N/2 lines are drawn.
* **GL\_LINE\_LOOP** - Draws a connected group of line segments from the first vertex to the last, then back to the first. Vertices n and n + 1 define line n. The last line, however, is defined by vertices N and N lines are drawn.
* **glPushMatrix, glPopMatrix Function**

The glPushMatrix and glPopMatrix functions push and pop the current matrix stack.

**SYNTAX:** void glPushMatrix();

void glPopMatrix(void);

* **glBegin, glEnd Function**

The glBegin and glEnd functions delimit the vertices of a primitive or a group of like primitives.

**SYNTAX:**void glBegin, glEnd(GLenum mode);

**PARAMETERS:**

mode - The primitive or primitives that will be created from vertices presented between glBegin and the subsequent glEnd. The following are accepted symbolic constants and their meanings

**5.2 Transformation Functions**

* **glTranslate Function**

The glTranslated and glTranslatef functions multiply the current matrix by a translation matrix.

**SYNTAX:** void glTranslate( x, y, z);

**PARAMETERS:**

x, y, z - The x, y, and z coordinates of a translation vector.

**Funtions used to display**

* **glMatrixMode Function:** The glMatrixMode function specifies which matrix is the current matrix.

**SYNTAX:** void glMatrixMode(GLenum mode);

**PARAMETERS:**

mode - The matrix stack that is the target for subsequent matrix operations. The mode parameter can assume one of three values:

Value Meaning

GL\_MODELVIEW Applies subsequent matrix operations to the modelview matrix Stack.

* **glLoadIdentity Function:** The glLoadIdentity function replaces the current matrix with the identity matrix.

**SYNTAX:** void glLoadIdentity(void);

**5.3 Functions used to set the viewing volume**

* **glOrtho:** This function defines orthographic viewing volume with all parameters measured from the centre of projection. multiply the current matrix by a perspective matrix.

**SYNTAX:** void glOrtho( GLdouble left, GLdouble right, GLdouble bottom, GLdouble top, GLdouble near, GLdouble far);

**PARAMETERS:**

* left, right - Specify the coordinates for the left and right vertical clipping planes.
* bottom, top - Specify the coordinates for the bottom and top horizontal clipping planes.
* nearVal, farVal - Specify the distances to the nearer and farther depth clipping planes. These values are negative if the plane is to be behind the viewer.

**5.4 Call back functions**

* **glutDisplayFunc Function:** glutDisplayFunc sets the display callback for the current window

**SYNTAX:** void glutDisplayFunc(void (\*func)(void));

* **glutReshapeFunc Function:** glutReshapeFunc sets the reshape callback for the current window.

**SYNTAX:** void glutReshapeFunc(void (\*func)(int width, int height));

**5.5 Main function**

* **glutInit Function:** glutInit is used to initialize the GLUT library.

**SYNTAX:** glutInit(int \*argcp, char \*\*argv);

**PARAMETERS:**

* argcp - A pointer to the program's unmodified argc variable from main. Upon return, the value pointed to by argcp will be updated, because glutInit extracts any command line options intended for the GLUT library.
* Argv - The program's unmodified argv variable from main. Like argcp, the data for argv will be updated because glutInit extracts any command line options understood by the GLUT library.

• glutInit(&argc,argv);

* **glutInitDisplayMode Function:** glutInitDisplayMode sets the initial display mode.

**SYNTAX:** void glutInitDisplayMode(unsigned int mode);

**PARAMETERS:**

mode – Display mode, normally the bitwise OR-ing of GLUT display mode bit masks.

See values below:

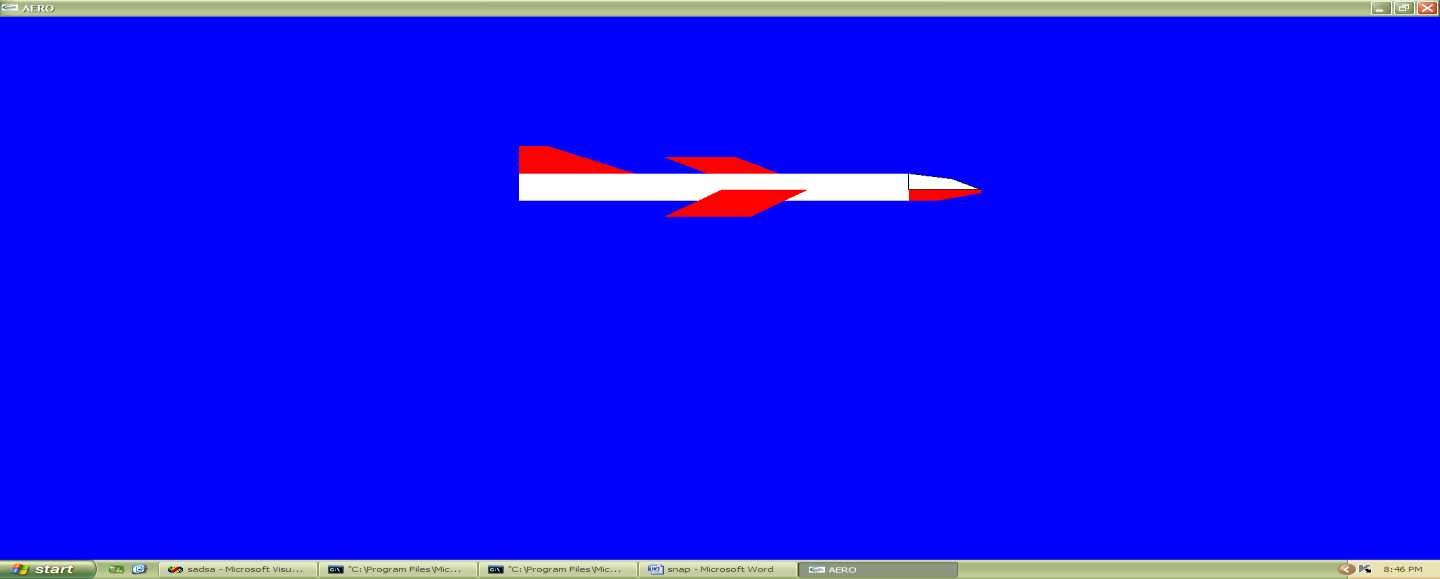
* GLUT\_RGB: An alias for GLUT\_RGBA.
* GLUT\_DOUBLE:Bit mask to select a double buffered window. This overrides GLUT\_SINGLE.
* GLUT\_DEPTH: Bit mask to select a window with a depth buffer.
* **glutMainLoop Function:** glutMainLoop enters the GLUT event processing loop.

**SYNTAX:** void glutMainLoop(void);

**CHAPTER 6**

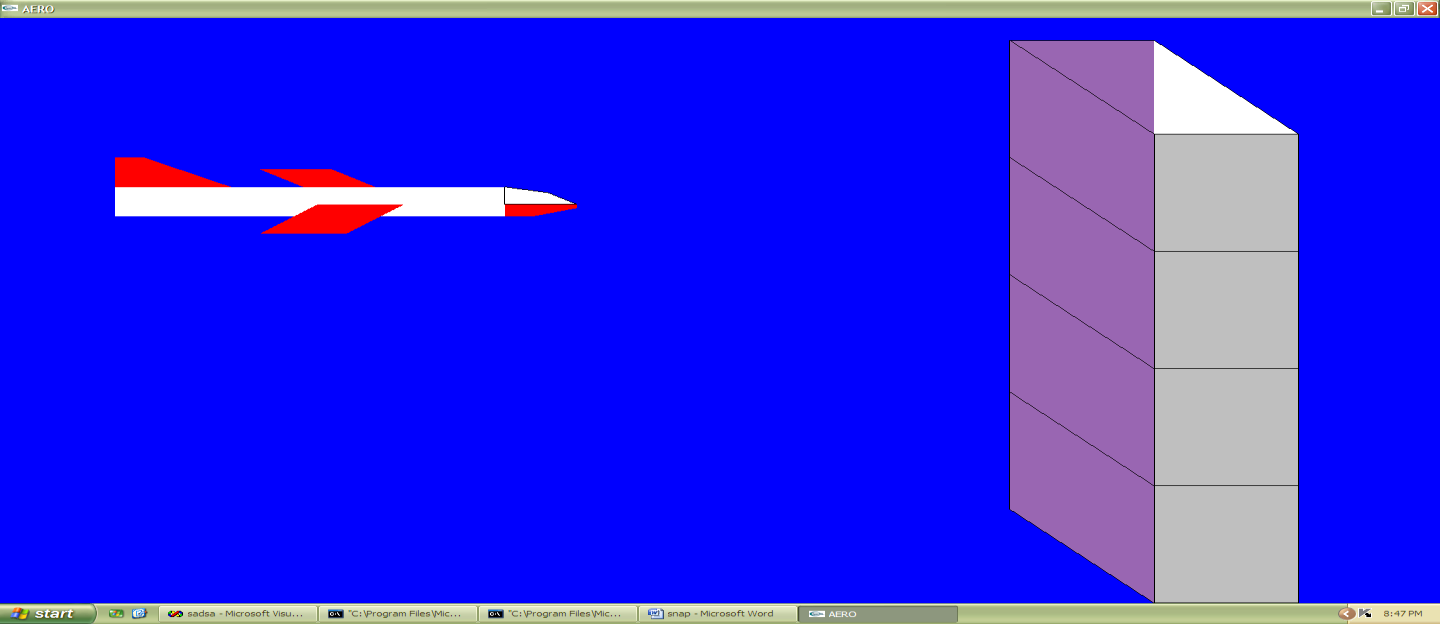
**SNAPSHOTS**

**Fig. 6.1**

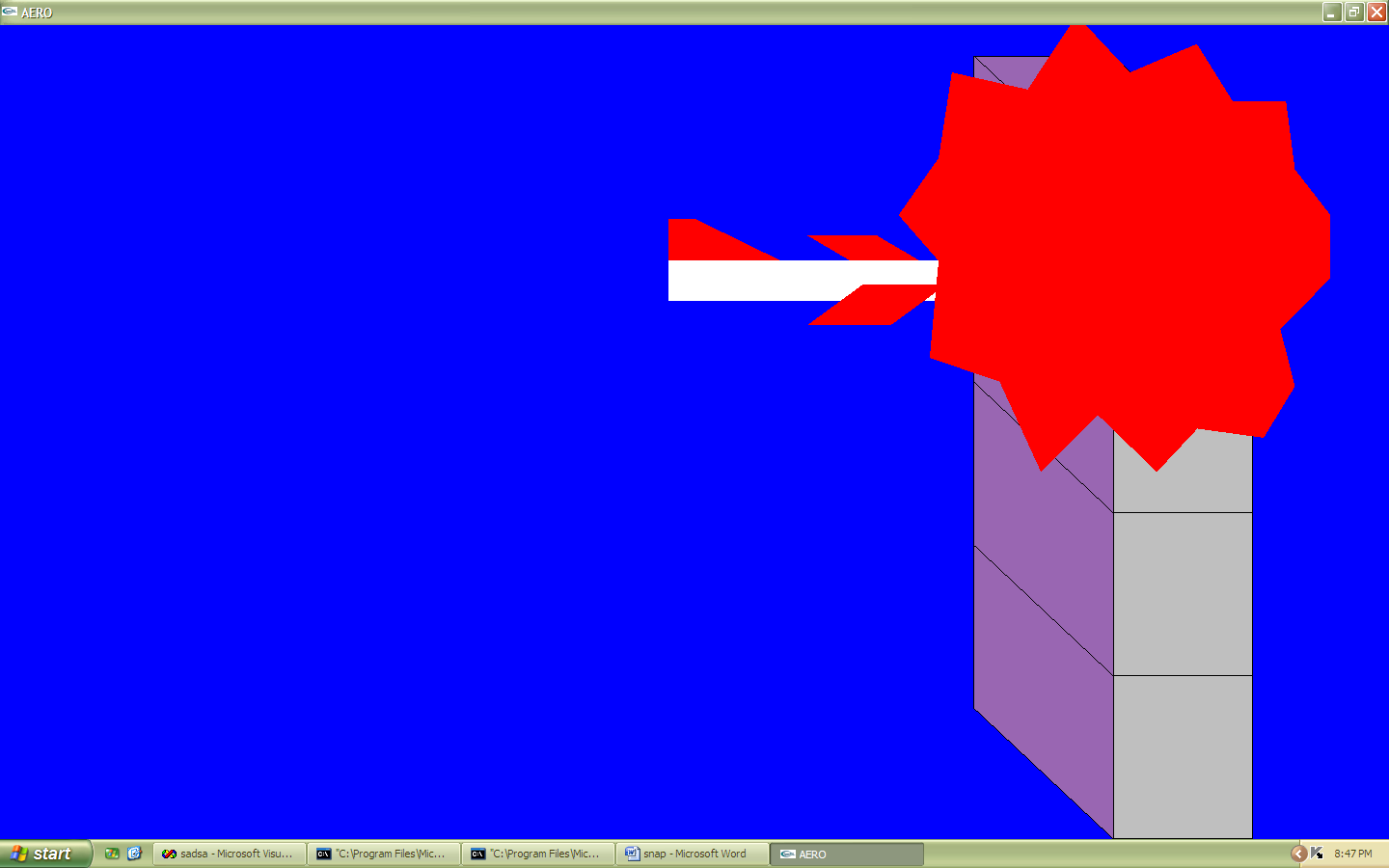


**Fig. 6.2**





**Fig 6.3**



**Fig 6.4**

**CONCLUSION AND FURTHER ENHANCEMENT**

Aeroplane crash is designed and implemented using a graphics software system called Open GL which became a widely accepted standard for developing graphics application. OpenGL provides various commands to create different aspects and visual effects that will enable programmer to deliver intended visuals to the audience, for complex puzzles like Tower of Hanoi having more number disks makes it difficult for humans to evaluate moves, with graphical demonstration using OpenGL libraries we can use computing power to simplify solution and give graphical representation to the problem for better perception of solution and reduce pressure on human brain imposed due to complex problem.

Usage of Open GL functions and primitives are well understood and henceforth can be applied for real time applications.

This project is both informative and entertaining. This project provided an opportunity to learn the various concepts of the subject in detail and provided a platform to express creativity and imagination come true. Further animation can be included to enhance the project’s look and feel, we can also use gaming approach in this design and produce it as a gaming software meant to discover ones ability in solving complex problems.

Computer Graphics has revolutionized almost every computer-based application in science and technology. Information technology is a trend today. As the volume of information increases, problem of storage arises. As time is money, in the 21st century people doesn’t have the time to read huge number of pages. So this problem is solved by Computer Graphics. Picture can represent a huge database like bar charts, pie charts etc. suppose, we have to show the performance of some factory related with profit since 1980. One requires large number of pages to store this huge information related with financial, numerical and statistical information. A common man requires a lot of time to understand it. There is an alternative to show or represent this information with the help of graphical tools such as bar chart or pie chart i.e. we can express this data in pictorial forms. The importance of computer graphics lies in its applications. In engineering applications (e.g. automotive and aerospace) the ability to quickly visualize newly designed shapes is indispensible. Computer graphics has also expanded the boundaries of art and entertainment. Movies such as Jurassic Park make extensive use of computer graphics to create images that test the bounds of imagination. The development of computer graphics has made possible virtual reality, a synthetic reality that exists only inside a computer. Virtual reality is fast becoming an indispensable tool in education. Flight simulators are used to train pilot for extreme conditions. Surgical simulators are used to train novice surgeons without endangering patients. Our project is one of the examples for the above mentation features and scope of computer graphics and its applications. Applications like our projects can be further improved with more features and can be implemented in our day-to-day life. We can take many advantages of this subject in the future. It is probable that at present situation also, it is anyhow utilized in modern technical gadgets.

**BIBLIOGRAPHY & REFERENCES**

##### Interactive Computer Graphics – Edward Angel

* Official OPENGL Documentation at [https://www.**opengl**.org/**documentation**/](https://www.opengl.org/documentation/)

##### OpenGL Programming Guide: The Official Guide to Learning OpenGL- Dave Shreiner.

* Computer Graphics with OpenGL Version,4th Edition : Donald Hearn & Pauline Baker
* Computer graphics with OpenGL: James D Foley, Andries Van Dam, Steven K Feiner, John F Huges
* Computer Graphics : Xiang, Plastock
* Interactive Computer Graphics, concepts and applications :Kelvin Sung, Peter Shirley, Steven Baer
* Computer Graphics using OpenGL: M M Raiker