VISVESVARAYA TECHNOLOGICAL UNIVERSITY

"Jnana Sangama", Belgaum-590018

Computer Graphics Mini Project On

"AIR SHOW"

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

BACHELOR OF ENGINEERING
IN
COMPUTER SCIENCE AND ENGINEERING

For The Academic Year
2021-2022

By

CHANDANA K
(1DB19CS031)

Under The Guidance Of
Dr. SHIVAKUMAR DALALI
Assistant 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 SCIENCE & ENGINEERING

CERTIFICATE

This is to certify that the mini project report entitled **"AIR SHOW"** is a bonafide work carried out by **CHANDANA K (1DB19CS031)** in partial fulfilment of award of degree of **Bachelor of Engineering in Computer Science and Engineering** of Visvesvaraya Technological University, Belagavi, during 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 with the degree mentioned.

Signature of Guide	**Signature of HOD**
……	…….
Dr. Shivakumar Dalali	**Dr. K. B. Shivakumar**
Associate Professor,	Head of Dept.,
Dept. of CSE,	Dept. of CSE,
DBIT, Bengaluru.	DBIT, Bengaluru.

DON BOSCO INSTITUTE OF TECHNOLOGY

Kumbalagodu, Bengaluru – 560 074.

DECLARATION

I, **CHANDANA K**, 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 **"AIR SHOW"** 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** duringthe academic year **2021-22**. 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 **CHANDANA K**

Date:20-07-2022 **1DB19CS031**

DON BOSCO INSTITUTE OF TECHNOLOGY

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

Kumbalagodu, Mysore Road, Bengaluru - 560074.

ACKNOWLEDGEMENT

Here by I am submitting the CG mini project report on **"AIR SHOW"**, 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 **Dr. Umashankar B. S., Principal, DBIT, Bangalore.**

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

I would like to express the deepest sense of gratitude to thank my Project Guide **Dr. Shivakumar Dalali**, Assoc. 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 report.

CHANDANA K

1DB19CS031

ABSTRACT

The aim of this project is to develop a graphics package which supports basic operation of creating menu, translation, rotation etc on the objects..This project uses the Conceptual framework model for an interactive graphics system. It mainly consists of Application Model, Application Program, and Graphics system and hardware components. Each of these systems interacts to produce an effective graphics display. It can be used to give a demo on **AIRSHOW** and give an overview of the surroundings to the pilots.It can also be used in arcade games.

The planes is controlled by pressing the left and right mouse buttons. The planes consists of three options. Any one option can be chosen at a time using mouse buttons.

The three options are:

- MOVE PLANES FORWARDS
- TURNS PLANES LEFT
- RESET SCENE

CONTENTS

CHAPTER 1: INTRODUCTION

1.1 : COMPUTER GRAPHICS

Computer graphics are graphics created using computers and, more generally, the representation and manipulation of image data by a computer.

The development of computer graphics has made computers easier to interact with, and better for understanding and interpreting many types of data. Developments in computer graphics have had a profound impact on many types of media and have revolutionized animation, movies and the video game industry.

Overview

The term computer graphics has been used in a broad sense to describe almost everything on computers that is not text or sound. Typically, the term computer graphics refers to several different things:

- the representation and manipulation of image data by a computer
- the various technologies used to create and manipulate images
- the images so produced, and
- the sub-field of computer science which studies methods for digitally synthesizing and manipulating visual content, see study of computer graphics

Today, computers and computer-generated images touch many aspects of daily life. Computer imagery is found on television, in newspapers, for example in weather reports, or in all kinds of medical investigation and surgical procedures. A well-constructed graph can present complex statistics in a form that is easier to understand and interpret. In the media such graphs are used to illustrate papers, reports, thesis, and other presentation material.

Many powerful tools have been developed to visualize data. Computer generated imagery can be categorized into several different types: 2D, 3D, 4D, 7D, and animated graphics. As technology has improved, 3D computer graphics have become more common, but 2D computer graphics are still widely used. Computer graphics has emerged as a sub-field of computer science which studies methods for digitally synthesizing and manipulating visual content. Over the past decade, other specialized fields have been developed like information visualization, and scientific visualization more concerned with

the visualization of three-dimensional phenomena (architectural, meteorological, medical, biological, etc.), where the emphasis is on realistic renderings of volumes, surfaces, illumination sources, and so forth, perhaps with a dynamic (time) component.

1.2 PIPELINE ARCHITECTURE:

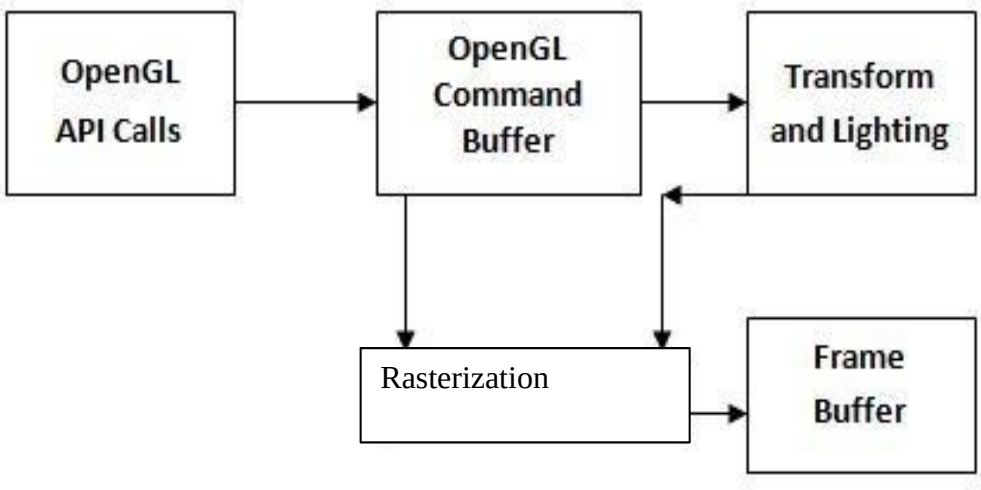


Fig. 1.2 Pipeline Architecture

The word pipeline is used to describe a process that can take two or more distinct stages or steps. The above figure shows a simplified version of the OpenGL pipeline. As an applicationmakes OpenGL API function call, the commands are placed in a command buffer. Thisbuffer eventually fills with commands, vertex data, texture data, and so on. When the bufferis flushed, either programmatically or by the driver's design, the commands and data arepassed to the next stage in the pipeline.

Image types

2D computer graphics are the computer-based generation of digital images—mostly from two-dimensional models, such as 2D geometric models, text, and digital images, and by techniques specific to them.

2D computer graphics are mainly used in applications that were originally developed upon traditional printing and drawing technologies, such as typography, cartography, technical drawing, advertising, etc. In those applications, the two-

dimensional image is not just a representation of a real-world object, but an independent artifact with added semantic value; two-dimensional models are therefore preferred, because they give more direct control of the image than 3D computer graphics, whose approach is more akin to photography than to typography.

Pixel art:

Pixel art is a form of digital art, created through the use of raster graphics software, where images are edited on the pixel level. Graphics in most old (or relatively limited) computer and video games, graphing calculator games, and many mobile phone games are mostly pixel art.

Vector Graphics:

Vector graphics formats are complementary to raster graphics, which is the representation of images as an array of pixels, as it is typically used for the representation of photographic images Vector graphics consists of encoding information about shapes and colors that comprise the image, which can allow for more flexibility in rendering. There are instances when working with vector tools and formats is best practice, and instances when working with raster tools and formats is best practice. There are times when both formats come together. An understanding of the advantages and limitations of each technology and the relationship between them is most likely to result in efficient and effective use of tools.

Three-Dimensional:

3D computer graphics in contrast to 2D computer graphics are graphics that use a three-dimensional representation of geometric data that is stored in the computer for the purposes of performing calculations and rendering 2D images. Such images may be for later display or for real-time viewing.

Despite these differences, 3D computer graphics rely on many of the same algorithms as 2D computer vector graphics in the wire frame model and 2D computer raster graphics in the final rendered display. In computer graphics software, the distinction between 2D and 3D is occasionally blurred; 2D applications may use 3D techniques to achieve effects such as lighting, and primarily 3D may use 2D rendering techniques.

3D computer graphics are often referred to as 3D models. Apart from the rendered graphic, the model is contained within the graphical data file. However, there are differences. A 3D model is the mathematical representation of any three-dimensional object. A model is not technically a graphic until it is visually displayed. Due to 3D printing, 3D models are not confined to virtual space. A model can be displayed visually as a two-dimensional image through a process called 3D rendering, or used in non-graphical computer simulations and calculations. There are some 3D computer graphics software for users to create 3D images.

Applications of Computer Graphics

- Computer graphics user interfaces - GUIs.
- A graphic, mouse-oriented paradigm which allows the user to interact with a computer.
- Business presentation graphics - "A picture is worth a thousand words".
- Cartography - drawing maps
- Weather Maps - real time mapping, symbolic representations
- Satellite Imaging - geodesic images
- Photo Enhancement - sharpening blurred photos
- Medical imaging - MRIs, CAT scans, etc. - non-invasive internal examination.
- Engineering drawings - mechanical, electrical, civil, etc. - replacing the blueprints of the past.
- Typography - the use of character images in publishing - replacing the hard type of the past.
- Architecture - construction plans, exterior sketches - replacing the blueprints and hand drawings of the past.
- Art - computers provide a new medium for artists.
- Computational biology
- Computational physics
- Computer-aided design
- Computer simulation
- Digital art
- Education
- Graphic design
- Info graphics
- Information visualization
- Rational drug design
- Scientific visualization
- Video Games
- Virtual reality
- Web design

CHAPTER 2:

LITERATURE SURVEY

I found the code of the other games that were coded using OpenGL library like Tetris, chess.I studied these codes thoroughly and got the following conclusions:

- Using OpenGL library, we can create various graphics related structure.
- Different functions defined in the OpenGL can be used to give different colors and textures to the ghost and Pacman.
- Glut (Toolkit of OpenGL library) can be used to create window, translate and rotate the matrix of the object, to track the position of mouse, to show the render screen etc.

Example:

- To create the window Function used: glutCreateWindow ("window name")
- To define the position of window Function used: glutInitWindowposition (int x, int y)X and Y are the co-ordinates of top left part of window in pixel.
- Define the size of window Function used: glutInitWindowsize (int x, int y) x and y are the width and height of the window.

Likewise, many user-defined functions are used to manipulate the game flow. The works are commented in the source code. In this way I gathered most of my knowledge from the previously made similar projects and got familiar with Open GL library and its toolkit.

CHAPTER 3:

SYSTEM DESIGN

3.1 OPENGL TECHNOLOGY

What is OpenGL?

OpenGL is the premier environment for developing portable, interactive 2D and 3D graphics applications. Since its introduction in 1992, OpenGL has become the industry's most widely used and supported 2D and 3D graphics application programming interface (API), bringing thousands of applications to a wide variety of computer platforms. OpenGL fosters innovation and speeds application development by incorporating a broad set of rendering, texture mapping, special effects, and other powerful visualization functions. Developers can leverage the power of OpenGL across all popular desktop and workstation platforms, ensuring wide application deployment.

High Visual Quality and Performance

Any visual computing application requiring maximum performance-from 3D animation to CAD to visual simulation-can exploit high-quality, high-performance OpenGL capabilities. These capabilities allow developers in diverse markets such as broadcasting, CAD/CAM/CAE, entertainment, medical imaging, and virtual reality to produce and display incredibly compelling 2D and 3D graphics.

Developer-Driven Advantages:

- 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.
- 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.

- 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.
- 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.
- Governance

The OpenGL Architecture Review Board (ARB), was an independent consortium formed in 1992 that governed the future of OpenGL, proposing and approving changes to the specification, new releases, and conformance testing. In Sept 2006, the ARB became the OpenGL Working Group under the Khronos Group consortium for open standard APIs.

The OpenGL Performance Characterization Committee, another independent organization, creates and maintains OpenGL benchmarks and publishes the results of those benchmarks on its Web site: www.specbench.org/gpc/opc.static/index.html.

Continued Innovation:

The OpenGL standard is constantly evolving. Formal revisions occur at periodic intervals, and extensions allowing application developers to access the latest hardware advances through OpenGL are continuously being developed. As extensions become widely accepted, they are considered for inclusion into the core OpenGL standard. This process allows OpenGL to evolve in a controlled yet innovative manner.

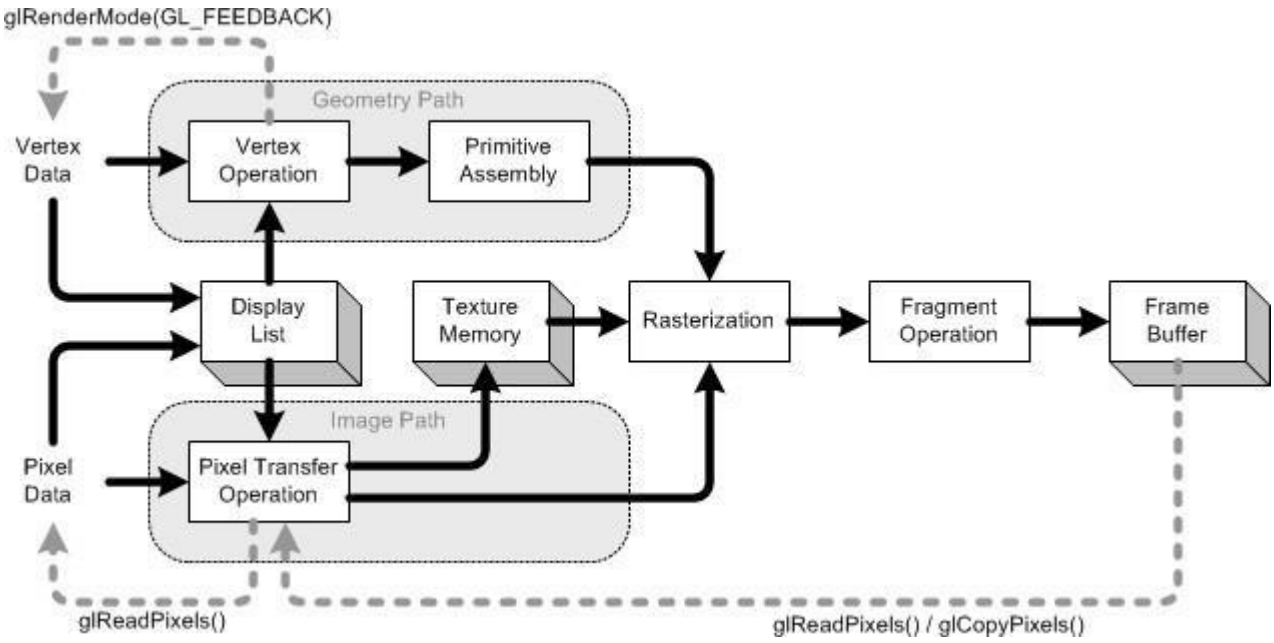
Licensing:

ARB-approved OpenGL specifications and source code are available to licensed hardware platform vendors. End users, independent software vendors, and others writing code based on the OpenGL API are free from licensing requirements. See SGI'slicensing web site for more information.

OpenGL Applications & Games

OpenGL is the pervasive standard for 3D consumer and professional applications across all major OS platforms. A partial list for Window, Linux and MacOS are available in the products section.

Rendering Pipeline



OpenGL Pipeline

OpenGL Pipeline has a series of processing stages in order. Two graphical information, vertex-based data and pixel-based data, are processed through the pipeline, combined together then written into the frame buffer. Notice that OpenGL can send the processed data back to your application. (See the grey color lines)

Vertex Operation:

Each vertex and normal coordinates are transformed by GL_MODELVIEW matrix (from object coordinates to eye coordinates). Also, if lighting is enabled, the lighting calculation per vertex is performed using the transformed vertex and normal data. This lighting calculation updates new color of the vertex.

Primitive Assembly:

After vertex operation, the primitives (point, line, and polygon) are transformed once again by projection matrix then clipped by viewing volume clipping planes; from eye coordinates to clip coordinates. After that, perspective division by w occurs and viewport transform is applied in order to map 3D scene to window space coordinates. Last thing to do in Primitive Assembly is culling test if culling is enabled.

Pixel Transfer Operation:

After the pixels from client's memory are unpacked(read), the data are performed scaling, bias, mapping and clamping. These operations are called Pixel Transfer Operation. The transferred data are either stored in texture memory or rasterized directly to fragments.

Texture Memory:

Texture images are loaded into texture memory to be applied onto geometric objects.

Rasterization:

Rasterization is the conversion of both geometric and pixel data into fragment. Fragments are a rectangular array containing color, depth, line width, point size and antialiasing calculations (GL_POINT_SMOOTH, GL_LINE_SMOOTH, GL_POLYGON_SMOOTH). If shading mode is GL_FILL, then the interior pixels (area) of polygon will be filled at this stage. Each fragment corresponds to a pixel in the frame buffer.

Fragment Operation:

It is the last process to convert fragments to pixels onto frame buffer. The first process in this stage is texture element generation; a texture element is generated from texture memory and it is applied to each fragment. Then fog calculations are applied. After that, there are several fragment tests follow in order; Scissor Test ⇒ Alpha Test ⇒ Stencil Test ⇒ Depth Test. Finally, blending, dithering, logical operation and masking by bitmask are performed and actual pixel data are stored in frame buffer.

Libraries

Most of the applications will be designed to access OpenGL directly through functions in the 3 libraries. Functions in the main GL (Graphics Library) library have names that begin with the letters *gl* and are stored in a library usually referred to as GL. 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 library. For each major window system there is a system-specific library that provides the 'glue' between the window system and OpenGL. For the X Window System, this library is called GLX, for Windows, it is wgl, and for Macintosh, it is agl. Rather than using a different library for each system, we use a readily available library called the OpenGL Utility Toolkit (GLUT), which provides the minimum functionality that should be expected in any modern windowing system.

As we know, GLUT is the OpenGL Utility Toolkit, a window system independent toolkit for writing OpenGL programs. It implements a simple windowing application

programming interface (API) for OpenGL. GLUT makes it considerably easier to learn about and explore OpenGL programming. GLUT provides a portable API so you can write a single OpenGL program that works on both Win32 Pcs and X11 workstations.

GLUT is designed for constructing small to medium sized OpenGL programs. While GLUT is well-suited to learning OpenGL and developing simple OpenGL applications, GLUT is not a full-featured toolkit so large applications requiring sophisticated user interfaces are better off using native window system toolkits like Motif. GLUT is simple, easy, and small.

The GLUT library supports the following functionality:

- Multiple windows for OpenGL rendering.
- Call-back driven event processing.
- An `idle' routine and timers.
- Utility routines to generate various solid and wire frame objects.
- Support for bitmap and stroke fonts.

The GLUT library has both C, C++, FORTRAN, and Ada programming bindings. The GLUT source code distribution is portable to nearly all OpenGL implementations for the X Window System and Windows 95 and NT. GLUT also works well with Brian Paul's Mesa, a freely available implementation of the OpenGL API.

Because GLUT is window system independent, GLUT can be implemented for window systems other than X. The current version of the GLUT API is 3. The current source code distribution is GLUT 3.7.

Header files

For all OpenGL applications, gl.h header has to be included in every file. Almost all OpenGL applications use GLU, the OpenGL Utility Library, which also requires inclusion of the glu.h header file. So almost every OpenGL source file begins with:

#include<GL/gl.h>

#include<GL/glu.h>

If OpenGL Utility Toolkit (GLUT) is being used for managing the window manager tasks,glut.h needs to be included:

#include <GL/glut.h>

Including all the three files is redundant. To make the GLUT programs portable, includeglut.h and gl.h or glu.h need not be included explicitly.

3.2 PROJECT DESCRIPTION

This project uses the Conceptual framework model for an interactive graphics system. It mainly consists of Application Model, Application Program, and Graphics system and hardware components. Each of these systems interacts to produce an effective graphics display. It can be used to give a demo on air show and give an overview of the surroundings to the pilots. It can also be used in arcade games

WORKING

The planes is controlled by pressing the left and right mouse buttons. The planes consists of three options. Any one option can be chosen at a time using mouse buttons. The three options are:

- PLANES FORWARDS MOVE
- TURNS PLANES LEFT
- RESET SCENE

The actual implementations are explained in detail along with the functions in this project.

3.3 FUNCTIONS USED

➢ Header Files Used

#include<stdlib.h>

stdlib.h is the header of the general-purpose standard library of C programming language which includes functions involving memory allocation, process control, conversions and others. It is compatible with C++ also.

Functions supported by the header:

int rand(void) : generates a pseudo-random number

void srand(unsigned int seed): set the rand() pseudo-random generator seed [common convention uses time() to seed]

#include<GL/glut.h>

GLUT is the OpenGL Utility Toolkit. It implements a simple windowing application programming interface (API) for OpenGL.

#include<time>

This header file contains definitions of functions to get and manipulate date and time information**.**

Functions supported by the header:
time() : Get current time (function). Time is used as seed for 'srand' function in this project.

➢ Functions used

- **glOrtho**

Syntax: void glOrtho(GLdouble left, GLdouble right, GLdouble bottom, GLdouble top, Gldouble near, GLdouble far);

Usage: creates a matrix for an orthographic parallel viewing volume and multiplies the current matrix by it. (Left, bottom,-near) and (right, top, near) are points on near clipping plane that are mapped to the lower left and upper right of the viewport window. (Left, bottom, -far) and (right, top, -far) are points far clipping plane that are mapped to the same respective corners of the viewport.

- **glTranslatef**

Syntax: void glTranslate [fd](TYPE x, TYPE y, TYPE z)
Usage: alters the current matrix by a displacement of (x, y, z), where TYPE is either GLfloat or GLdouble.

- **glColor**

Syntax: void glColor[3 4] [b i f ub us ui] (TYPE r,TYPE g,TYPE b)
Usage: sets the present RGB or RGBA colors. Valid types are byte(b), int(i), float(f), double(d), unsigned byte(ub), unsigned short(us), and unsigned int(ui). The maximum and minimum values of the floating-point types are 1.0 and 0.0 respectively, whereas the maximum and minimum values of the discrete types are those of the type, for example, 255 and 0 for unsigned bytes.

- **glutSolidSphere**

Syntax: void glutSolidSphere(GLdouble radius, GLint slices, GLint stacks);Usage:
Creates a solid sphere of the specified radius.

- **glLineWidth**

Syntax: void glLineWidth(GLfloat width);

Usage: Sets the width, in pixels, for rendered lines; width must be greater than 0.0and by default is 1.0**.**

- **glVertex2f**

Syntax: glVertex2f (int, int);
Usage: It locates the two coordinates in the display window. Here 2 indicate number of arguments. Vertex is the basic command. gl is the gl library function name.

- **glEnd**

Syntax: void glEnd(void);
Usage: Marks the end of a vertex data list.

- **glBegin**

Syntax: void glBegin(GLenum mode);
Usage: Marks the beginning of a vertex-data list that describes a geometricprimitive. The type of primitive is indicated by mode, which can be any of the values

- **glClearColor**

Syntax: void glClearColor(GLclampf red, GLclampf green, GLclampf blue, GLclampf alpha);
Usage: Sets the current clearing color for use in clearing color buffers in RGBA mode. The red, green, blue, and alpha values are clamped if necessary to the range[0, 1]. The default clearing color is (0, 0, 0, 0), which is black.

- **glClear**

Syntax: void glClear(GLbitfield mask);
Usage: To clear the entire window to background color. Clears the specified buffers. The value of mask is the bitwise logical OR of some combination of GL_COLOR_BUFFER_BIT,GL_DEPTH_BUFFER_BIT, GL_STENCIL_BUFFER_BIT, and GL_ACCUM_BUFFER_BIT to identify which buffers are to be cleared. GL_COLOR_BUFFER_BIT clears either the RGBA color buffer or the color-index buffer, depending on the mode of the systemat the time. When you clear the color or color-index buffer, all the color buffers that are enabled for writing (see the

next section) are cleared. The pixel ownershiptest, scissor test, and dithering, if enabled, are applied to the clearing operation. Masking operations, such as glColorMask() and glIndexMask(), are also effective.The alpha test, stencil test, and depth test do not affect the operation of glClear().

- **glFlush**

Syntax: void glFlush (void);

Usage: forces previously issued openGL commands to begin executing, thusguaranteeing that they complete in finite time. This command flushes the networkas glFlush() does and then waits for notification from the graphics hardware or network indicating that the drawing is complete in the frame buffer.

CHAPTER 4:

REQUIREMENT SPECIFICATION

4.1 Hardware Requirements:

- Intel Pentium D CPU 2.66 GHz
- 768 MB RAM
- Mouse
- Keyboard 108 Standard
- Minimum Monitor Resolution 800 x 800

4.2 Software Requirements:

- The graphics package has been designed for OpenGL; hence the machine must Have Dev C++.
- GLUT libraries, Glut utility toolkit must be available.
- Operating System**:** Windows
- Version of Operating System**:** Windows 7,Windows XP, Windows NT and Higher
- Language**:** C

CHAPTER 5:

INTERFACE AND ARCHITECTURE

5.1 Interface

Interface is to communicate between the user and the computer. It provides an environment to communicate with the software using the hardware devices like mouse, joysticks, keyboard etc.

1. **glutKeyboardFunc Function**

glutKeyboardFunc allow us to give input using keyboard.

SYNTAX:

void glutKeyboardFunc(unsigned char key, int x, int y);

PARAMETERS:

- key
  The pressed key ASCII value is stored in it.
- X and Y gives the position of pointer.

2. **glutCreateMenu**
   glutCreateMenu creates a new pop-up menu.

USAGE

int glutCreateMenu(void (*func)(int value));

PARAMETERS:

Func:

The callback function for the menu that is called when a menu entry from the menu is selected. The value passed to the callback is determined by the value for the selected menu entry.

3. glutAttachMenu

glutAttachMenu attaches a mouse button for the current window to the identifier of the current menu; glutDetachMenu detaches an attached mouse button from the current window.

Usage

```
void glutAttachMenu(int button);
void glutDetachMenu(int button);
```

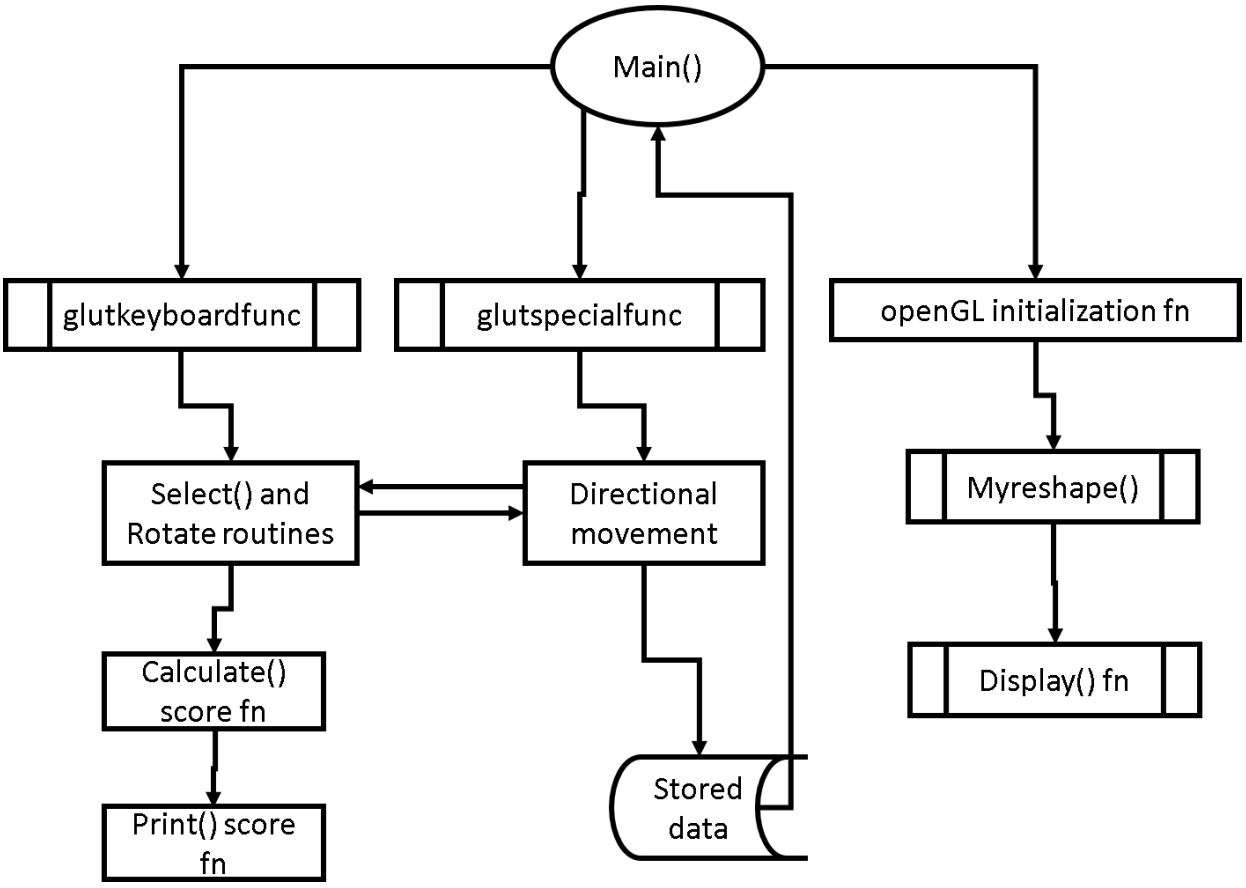
PARAMETERS:

button
The button to attach a menu or detach a menu,

DESCRIPTION:

glutAttachMenu attaches a mouse button for the current window to the identifier of the current menu; By attaching a menu identifier to a button, the named menu will be popped up when the user presses the specified button. button should be one of GLUT_LEFT_BUTTON, GLUT_MIDDLE_BUTTON, and GLUT_RIGHT_BUTTON. Note that the menu is attached to the button by identifier, not by reference.

5.2 Architecture



Fig(5.2) Flow diagram of program

CHAPTER 6:

IMPLEMENTATION

OpenGL is a software tool for developing the graphics objects. OpenGL library called GLUT i.e., Graphics Library Utility toolkit supports graphics system with the necessary modelling and rendering techniques. The Lighting system is a technique for displaying graphic objects on the monitor and displaying the light effects. It provides the following functionalities.

6.1 Initialization

This function is the initial stage of the system where the system initializes the various aspects of the graphics system based on the user requirements, which include Command line processing, window system initialization and also the initial window creation state is controlled by these routines. This also includes the window and menu management functions which are routines to create and control windows and menus.

6.2 Event Processing

This routine enters GLUT's event processing loop. This routine never returns, and it continuously calls GLUT call-back as and when necessary. This can be achieved with the help of the call-back registration functions. These routines register call-backs to be called by the GLUT event processing loop.

6.3 GLUT functions used:

- **glutPostRedisplay**

  Syntax: void glutPostRedisplay(void);
  Usage: Marks the current window as needing to be redrawn. At the next opportunity, the callback function registered by glutDisplayFunc() will be called.

- **glutInit**

  Syntax: init(int argc, char ** argv)
  Usage: initializes GLUT and processes any commands line arguments (For example, this would be options such as display and geometry). **glutinit ()** should be called before any other **GLUT** routine.

- **glutInitDisplayMode**

  Syntax: glutInitDisplayMode (unsigned int mode);
  Usage: specifies whether to use an RGBA or color –index model. Can also set a number of other options for the display window such as buffering and a choice of color modes with the function. Double frame buffer and RGBA color buffer specifies a double refresh buffer is used for display window and RGBA color mode is to be used for selecting color values.

- **glutInitWindowSize**

  Syntax: glutInitWindowSize (int width , int size);
  Usage: specifies the size, in pixels, of the window. The function is used to set the initial pixel and height of the display window.

- **glutCreateWindow**

  Syntax: int glutCreateWindow (char * string)
  Usage: creates a window with an openGL context. It returns a unique identifier for the new window.

- **glutDisplayFunc**

  Syntax: void glutDisplayFunc (void (*func) (void))
  Usage: is the first and most important event callback function. Whenever GLUT determines that the contents of the window need to be redisplayed, the callback function registered by glutDisplayFunc() is executed.

- **glutSpecialFunc**

  Syntax: void glutSpecialFunc (void (*func)(int key, int x, int y))
  Usage: glutSpecialFunc sets the special keyboard callback for the current window. The special keyboard callback is triggered when keyboard function or directional keys are pressed. The key callback parameter is a GLUT_KEY_* constant for the special key pressed. The x and y callback parameters indicate the mouse in window relative coordinates when the key was pressed. When a new window is created, no special callback is initially registered and special key strokes

in the window are ignored. Passing NULL to glutSpecialFunc disables the generation of special callbacks.
GLUT_KEY_LEFT: Left directional key
GLUT_KEY_UP: Up directional key
GLUT_KEY_RIGHT: Right directional key
GLUT_KEY_DOWN: Down directional key

- **glutMainLoop**

Syntax: glutMainLoop (void)
Usage: the very last thing we must do is call glutMainLoop (void). All windows that have been created are now shown, and rendering to those windows are now effective. Event processing begins, and registered display is triggered. Once this loop is entered, it is never exited.

6.4: Program Logical Functions:

```
7       #include<GL/glut.h>
8       #include <cmath>
9       //Camera position
10      static double cx = 100;
11      static double cy = 50;
12      static double cz = 100;
13      //Rotation
14      static int spinx = 0;
15      static int spiny = 0;
16      static int spinz = 0;
17      //Position
18      static double x = 0.0;
19      static double y = 0.0;
20      static double z = 0.0;
21      //Smoke particle rotation
22      static int spinxs[150];
23      static int spinys[150];
24      static int spinzs[150];
25      //smoke  particle postion
26      float sx[150];
27      float sy[150];
28      float sz[150];
29      float sa[150];
30      float ss[150];
31      //Count
32      int i = 0;
33
34      //Sphere
35      void sphere(float r, float g, float b, float a)
36      {
37      glColor4f(r,g,b,a);
```

```
38      glutSolidSphere(4,32,32);
39      }
40
41
42      void smoke(float size, float alpha, float R, float G, float B)
43      {
44      glPushMatrix();
45
46      //Colour of and transparency of Smoke
47      glColor4f(R,G,B,alpha);
48      glTranslatef(0,0,-15);
49      glutSolidSphere((1 + size),16,16);
50      glPopMatrix();
51      }
52
53      void drawSmoke(float R, float G, float B, float x, float y,
54      int reflect)
55      {
56      // Calculate each position, size and transparency of smoke
57      for (int xi = 0; xi < 150; xi++)
58      {
59      sa[xi] = sa[xi] - 0.0011;
60      ss[xi] = ss[xi] + 0.005;
61      glPushMatrix();
62      glScalef(reflect*0.5,reflect*0.5,0.5);
63      glTranslatef((reflect*sx[xi])- x, sy[xi] - y,sz[xi]);
64      glRotatef(spinxs[xi],1,0,0);
65      glRotatef(reflect*spinys[xi],0,1,0);
66      glRotatef(reflect*spinzs[xi],0,0,1);
67      smoke(ss[xi], sa[xi],R,G,B);
68      glPopMatrix();
69      }
70      }
71
72
73      void slab(float r, float g, float b)
74      {
75      glColor3f(r,g,b);
76      glutSolidCube(1);
77      }
78      void wing(int Colour)
79      {
80      float rs=0; //Side red
81      float re=0;      //Edge red
82      float bs=0;      //Side blue
83      float be=0;      //Edge blue
84      float gs=0;      //Side green
85      float ge=0;      //Edge green
86
87      if (Colour == 1){
88      rs = 0.75;
89      re = 0.5;
90      } else if (Colour == 2){
91      gs = 0.75;
92      ge = 0.5;
93      } else if (Colour == 3){
94      bs = 0.75;
95      be = 0.5;
```

```
96      }
97      //Front
98      glBegin(GL_POLYGON);
99      glColor3f(re,ge,be);
100     glVertex3f(1.5,-1,10);
101     glVertex3f(25,-0.25,0);
102     glVertex3f(25,0.25,0);
103     glVertex3f(1.5,1,10);
104     glEnd();
105     //Back
106
107
108     glBegin(GL_POLYGON);
109     glColor3f(re,ge,be);
110     glVertex3f(1.5,1,-1);
111     glVertex3f(25,0.25,-7);
112     glVertex3f(25,-0.25,-7);
113     glVertex3f(1.5,-1,-1);
114     glEnd();
115     //Top
116     glBegin(GL_POLYGON);
117     glColor3f(rs,gs,bs);
118     glVertex3f(1.5,1,10);
119     glVertex3f(25,0.25,0);
120     glVertex3f(25,0.25,-7);
121     glVertex3f(1.5,1,-1);
122     glEnd();
123     //Bottom
124     glBegin(GL_POLYGON);
125     glColor3f(rs,gs,bs);
126     glVertex3f(1.5,-1,-1);
127     glVertex3f(25,-0.25,-7);
128     glVertex3f(25,-0.25,0);
129     glVertex3f(1.5,-1,10);
130     glEnd();
131     //End
132     glBegin(GL_POLYGON);
133     glColor3f(re,ge,be);
134     glVertex3f(25,-0.25,0);
135     glVertex3f(25,-0.25,-7);
136     glVertex3f(25,0.25,-7);
137     glVertex3f(25,0.25,0);
138     glEnd();
139     }
140     //Fin
141     Void fin(int Colour)
142     {
143     float rs=0;
144     float re=0;
145     float bs=0;
146     float be=0;
147     float gs=0;
148     float ge=0;
149
150     if (Colour == 1){
151     rs = 0.75;
152     re = 0.5;
153     } else if (Colour == 2){
```

```
154      gs = 0.75;
155      ge = 0.5;
156      } else if (Colour == 3){
157      bs = 0.75;
158      be = 0.5;
159      }
160      //Front
161      glBegin(GL_POLYGON);
162      glColor3f(re,ge,be);
163      glVertex3f(-0.5,2,-7);
164      glVertex3f(0.5,2,-7);
165      glVertex3f(0,9.5,-10);
166      glEnd();
167
168
169      //Back
170      glBegin(GL_POLYGON);
171      glColor3f(re,ge,be);
172      glVertex3f(0,9.5,-12);
173      glVertex3f(0.5,2,-11);
174      glVertex3f(-0.5,2,-11);
175      glEnd();
176      //Side A
177      glBegin(GL_POLYGON);
178      glColor3f(rs,gs,bs);
179      glVertex3f(0.5,2,-7);
180      glVertex3f(0.5,2,-11);
181      glVertex3f(0,9.5,-12);
182      glVertex3f(0,9.5,-10);
183      glEnd();
184      //Side B
185      glBegin(GL_POLYGON);
186      glColor3f(rs,gs,bs);
187      glVertex3f(0,9.5,-10);
188      glVertex3f(0,9.5,-12);
189      glVertex3f(-0.5,2,-11);
190      glVertex3f(-0.5,2,-7);
191      glEnd();
192      }
193      void plane(int Colour)
194      {
195      float rb=0;
196      float gb=0;
197      float bb=0;
198      //Selects which colour plane to display
199      if (Colour == 1){
200      rb = 0.65;
201      } else if (Colour == 2){
202      gb = 0.65;
203      } else if (Colour == 3){
204      bb = 0.65;
205      }
206      //Body
207      glPushMatrix();
208      glTranslatef(0,0,4);
209      glScaled(1,0.8,5);
210      sphere(rb,gb,bb,1);
211      glPopMatrix();
```

```
212     //Windscreen
213     glPushMatrix();
214     glTranslatef(0,2,16);
215     glScaled(0.5,0.4,0.75);
216     sphere(0,0,0,0.75);
217     glPopMatrix();
218     //Left Wing
219     wing(Colour);
220     //Right Wing
221     glPushMatrix();
222     glScalef(-1,-1,1);
223     wing(Colour);
224     glPopMatrix();
225     //Left mini wing
226     glPushMatrix();
227     glScalef(0.4,0.4,0.4);
228
229
230     glTranslatef(0,3,-25);
231     wing(Colour);
232     glPopMatrix();
233     //Right mini wing
234     glPushMatrix();
235     glScalef(-0.4,-0.4,0.4);
236     glTranslatef(0,-3,-25);
237     wing(Colour);
238     glPopMatrix();
239     //Fin
240     fin(Colour);
241     }
242
243     void cloud()
244     {
245     glPushMatrix();
246     glScalef(1.5,1,1.25);
247     glTranslatef(0,12,0);
248     sphere(1,1,1,0.9);
249     glPopMatrix();
250     glPushMatrix();
251     glScalef(1.5,1,1.25);
252     glTranslatef(0,5,3);
253     sphere(1,1,1,0.9);
254     glPopMatrix();
255
256     glPushMatrix();
257     glScalef(1.5,1,1.25);
258     glTranslatef(4,7,0);
259     sphere(1,1,1,0.9);
260     glPopMatrix();
261
262     glPushMatrix();
263     glScalef(1.5,1,1.25);
264     glTranslatef(-4,7,0);
265     sphere(1,1,1,0.9);
266     glPopMatrix();
267
268     glPushMatrix();
269     glScalef(2,1.5,2);
```

```
270     glTranslatef(0,5,0);
271     sphere(1,1,1,0.5);
272     glPopMatrix();
273     }
274
275     /* reshape callback function
276     executed when window is moved or resized */
277     void reshape(int width, int height)
278     {
279     glViewport(0, 0, width, height);
280     glMatrixMode(GL_PROJECTION);
281     glLoadIdentity();
282     gluPerspective(50.0,1.0,15.0,600.0);  //Perspective
283     glMatrixMode(GL_MODELVIEW);
284     glLoadIdentity();
285     }
286
287
288     /* display routine this where the drawing takes place  */
289
290
291     void display(void)
292     {
293
294     glClear (GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
295     /* clear window */
296     glMatrixMode(GL_MODELVIEW);
297     glLoadIdentity();
298     gluLookAt(cx, cy, cz, x/2, y/2, z/2, 0.0, 1.0, 0.0);
299     //position of the eye
300     //Calculate position and rotation of current smoke particle
301     sx[i] = x;
302     sy[i] = y;
303     sz[i] = z;
304     spinxs[i] = spinx;
305     spinys[i] = spiny;
306     spinzs[i] = spinz;
307     sa[i] = 1;
308     ss[i] = 0;
309
310     //Bridge
311     for (int b = 0; b < 600; b=b+20)
312     {
313     glPushMatrix();
314     glTranslatef(-300+b,-47,0);
315     //bridge();
316     glPopMatrix();
317     }
318     //Bridge Reflection
319     glPushMatrix();
320     glScalef(-1,-1,1);
321     glTranslatef(0,56,0);
322     //bridge();
323     glPopMatrix();
324     glPushMatrix();
325     glScalef(-1,-1,1);
326     glTranslatef(-20,56,0);
327     //bridge();
```

```
328     glPopMatrix();
329
330     //Red plane reflection
331     glPushMatrix();
332     glScalef(-0.5,-0.5,0.5);
333     glTranslatef(-x+60,y+130,z);
334     glRotatef(spinx,1,0,0);
335     glRotatef(-spiny,0,1,0);
336     glRotatef(-spinz,0,0,1);
337     plane(1);
338     glPopMatrix();
339
340     //Green plane reflection
341     glPushMatrix();
342     glScalef(-0.5,-0.5,0.5);
343     glTranslatef(-x,y+130,z);
344     glRotatef(spinx,1,0,0);
345     glRotatef(-spiny,0,1,0);
346     glRotatef(-spinz,0,0,1);
347     plane(2);
348     glPopMatrix();
349
350
351
352     //Blue plane reflection
353     glPushMatrix();
354     glScalef(-0.5,-0.5,0.5);
355     glTranslatef(-x-60,y+130,z);
356     glRotatef(spinx,1,0,0);
357     glRotatef(-spiny,0,1,0);
358     glRotatef(-spinz,0,0,1);
359     plane(3);
360     glPopMatrix();
361     //Smoke trails for reflective planes
362     drawSmoke(1,0.5,0.2,-60,-130,-1);
363     drawSmoke(1,1,1,0,-130,-1);
364     drawSmoke(0,0.64,0,60,-130,-1);
365
366
367     glRotatef(spinz,0,0,1);
```

CHAPTER 7:

RESULT AND SNAPSHOTS

7.1 RESULTS

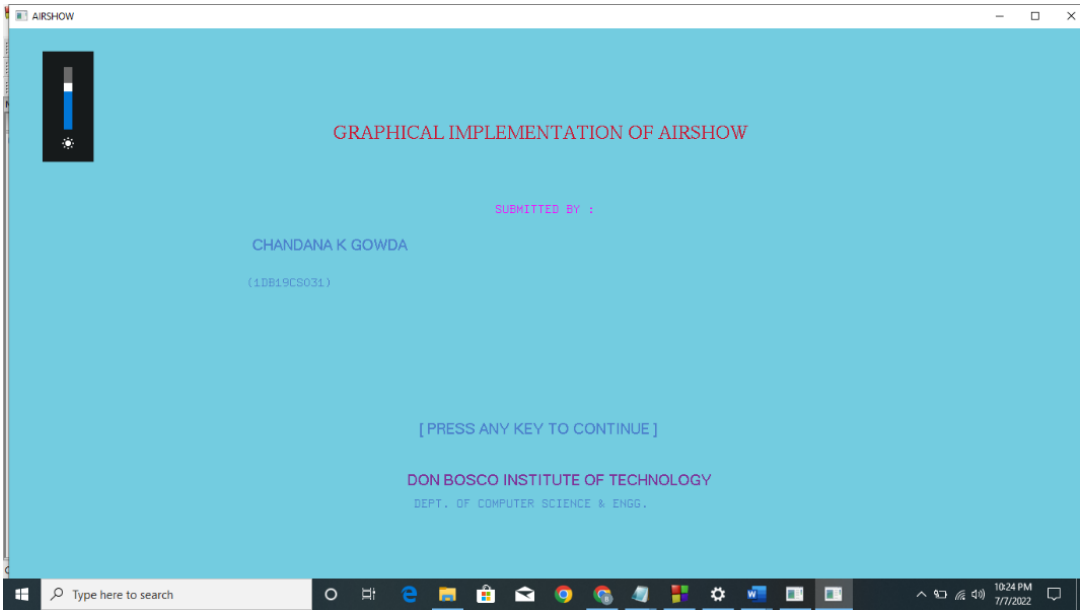
Testing is the where the program is executed to check for its proper working.

- The Air show program is tested and verified on Windows Platform.
- The program is tested to verify the positioning and color of the smoke emitted by the planes.
- Movement of the planes has been verified. .
- The cascading of the planes has been verified to see to it that they are moving towards their correct position.

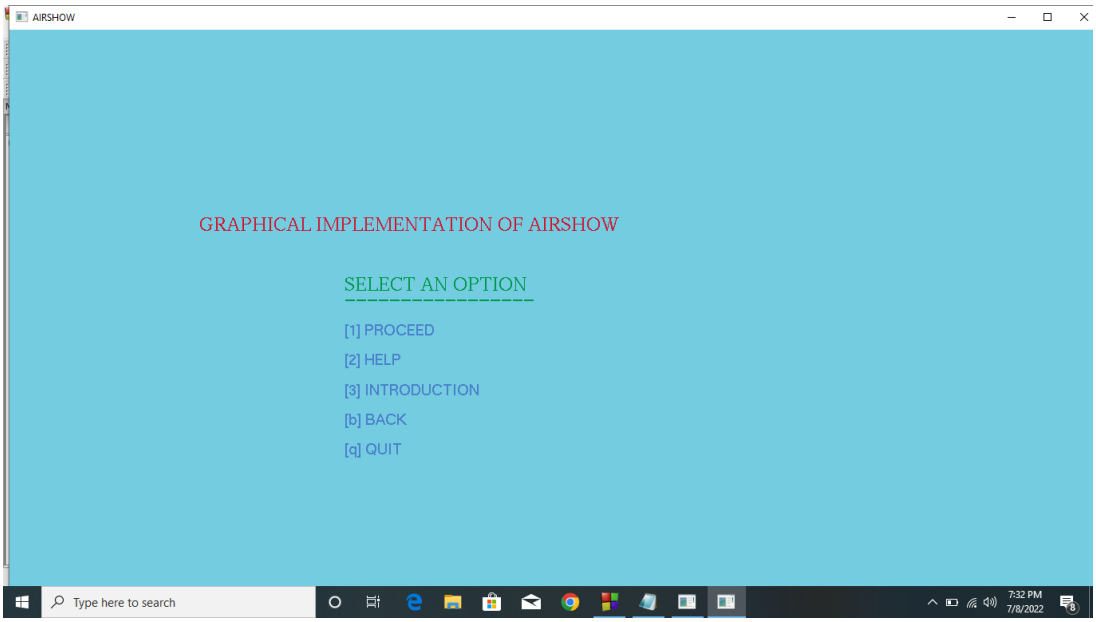
An example of a test case is as given below:

Left-click	ROTATIONAL TRANSLATION
Right-click	TO STOP ROTATIONAL TRASLATION
U	MOVEUP
D	MOVEDOWN
S	START
R	RESET

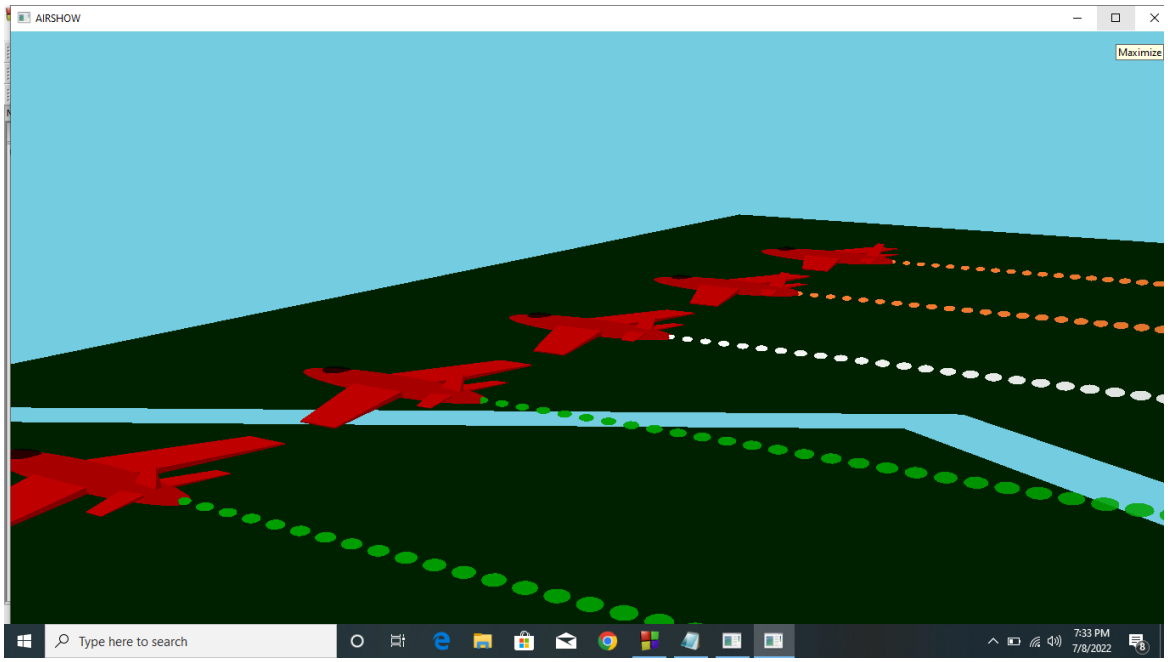
7.2 SNAPSHOTS



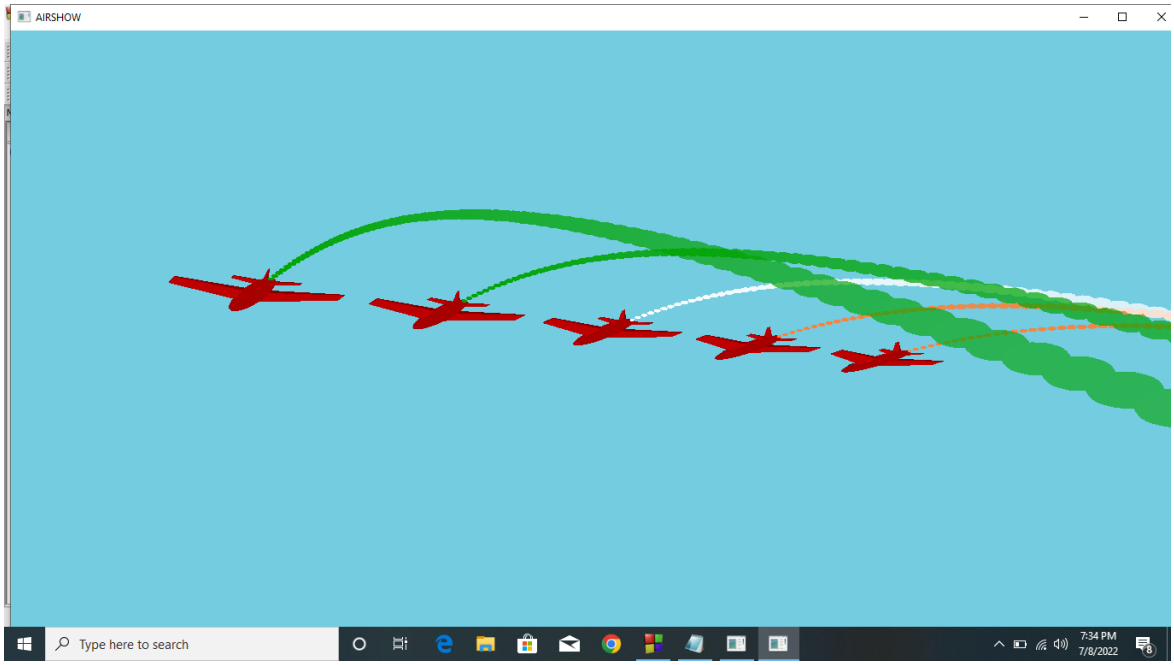
1.TITLE CARD



2.INSTRUCTIONS



3.PLANES IN MOTION



4.PLANES MOVING AND ROTATING

CHAPTER 8:

CONCLUSION & FUTURE ENHANCEMENT

8.1 Conclusion

An attempt has been made to develop an OpenGL package which meets necessary requirements of the user successfully. Since it is user friendly it enables the user to interact efficiently and easily. The development of mini project has given us a good exposure to OpenGL by which we have learnt some of the techniques which help in the development of animation picture and gaming.
Hence it is helpful for us even to take this field as our career too and develop some other features in OpenGL and provide as token of contribution to the graphics world.

8.2 Future Enhancement

These are the features that are planned to be supported in the future

- Support for multiple canvases.
- Support for pattern filling.
- Support for 3d transformations.
- Support for transparency of layers.

CHAPTER 9:

BIBLIOGRAPHY

Books:

[1] Edward Angel's Interactive Computer Graphics Pearson Education 5th Edition

[2] Interactive computer Graphics --A top down approach using open GL--by Edward Angle

[3] Jackie .L. Neider, Mark Warhol, Tom.R.Davis, "OpenGL Red Book", Second Revised Edition, 2005.

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

Websites:

[1] http://www.opengl.org/documentation

[2] http://www.nehe.gamedev.net/lesson.asp?index=01

[3] http://en.wikipedia.org/wiki/OpenGL#Documentation

[4] http://www.google.com

CHAPTER 10:

APPENDIX

User Manual

Creating the Project

Step 1: To create an empty console project in Code Blocks, do the following:

1. Create a new project(File→New→Project)
2. In the Project Types: pane, select Glut Project, Win32. Then select Win32 Console Application in Templates: pane. Name your project, select the location for the project and click OK.
   Click the Application Settings tab on the left, and check the Empty Project box. Then click Finish button.

Step 2: Add source code

1. Select Project.
2. In the categories pane, select OpenGL, code. Then select C File(.cpp) in the Templates: pane. Name your file, and then click Add.

Steps for execution

1. Compile From the Visual Studio's menu Build option(Build→Build Solution).
2. Execute the program from the Codeblocks's menu Run option. (Build → Run).