**Chapter 1**

**INTRODUCTION**

Computer Graphics is concerned with all aspects of producing pictures or images using a computer. The field began humbly almost 50 years ago, with the display of a few lines on a cathode-ray tube (CRT); now, we can create images by computer that are indistinguishable from photographs of real objects. We routinely train pilots with simulated airplanes, generating graphical displays of a virtual environment in real time. Feature-length movies made entirely by computer have been successful, both critically and financially. Massive multiplayer games can involve tens of thousands of concurrent participants

**1.1 Problem Statement**

The aim of this project is to design and implement the**“Different views of jet plane”**. The key challenges here are: (1) how to model the jet plane? (2) how to move the jet plane?(3) how to make jet plane visible in different views ?

In this project, this model is simulated by using OpenGL functions. This implementation

is achieved by using a standard user input interface.

**1.2 Objective of the Project**

1) The main objective of the project is to display jet movements using controls.

2) To implement the concepts of Computer Graphics we have learnt.

3) To implement movements of the objects.

4) To incorporate coloring effects.

**Chapter 2**

**COMPUTER GRAPHICS**

The representation and manipulation of image data by a computer.The subfield of computer science which studies methods for digitally synthesizing and manipulating visual content.Computer imagery is found on television, newspapers for example in weather reports, medical investigation and surgical procedures. A well-constructed graph can present the complex statistics in a form easier to understand and interpret.

Computer generated imagery can be categorized into several different types: 2D,3D,5D and animated graphics.3D computer graphics has become common ,but still 2D is widely used. Other specialized fields developed are information visualization and scientific visualization, where the emphasis is on realistic renderings of volumes,surfaces, illumination sources and so forth, perhaps with a dynamic component.An image or picture is an artifact that resembles a physical object. A pixel is represented as picture element. Rendering is the process of generating an image from a model by means of computer programs.

**2.1 History**

William fetter was credited with coning the term Computer Graphics in 1960, to describe his work at Boeng. One of the first displays of computer animation was future world (1976), which included an animation of a human face and hand-produced by Carmull and Fred Parkle at the University of Utah.

There are several international conferences and journals where the most significant results in computer-graphics are published. Among them are the SIGGRAPH and Euro graphics conferences and the association for computing machinery (ACM) transaction on Graphics journals.

**2.2Applications of computer graphics:**

* Simulation.
* User Interface.
* Display of Information
* Design.

**Display of Information:**

Computer Graphics has enabled researches to pictorially interpret the vast quantity of data in the field of fluid flow, molecular biology, mathematics etc. Images are generated by conducting data into visual representations. The process of visually representing the data has given the researchers new insight into the complex processes which previously was not possible.

**Design:**

Computer Graphics has helped professionals in the field of Engineering and Architecture who are concerned with design. After the professional generate a certain design, he uses graphics tools to analyze the design and display the analysis pictorially. This pictorial visualization has led to the generation of cost effective and efficient solution to design issues.

**Simulation and Animation:**

Computer Graphics system have evolved to such a stage where it is now possible generate sophisticated graphics images in real-time. This has led to the use of computer graphics for generation of simulators by Engineers and Researchers. One of the most important use of simulation has been in training pilots. Graphical flight simulators have proved to improve safety and reduce training expenses. The success of flight simulators led to the use of Computer Graphics for Animation in television motion pictures etc. This has further led to virtual reality simulation and video games.

**User Interfaces:**

Computer Graphics has led to the creation of GUI’s using which even naïve users are able to interact with a computer. Most of the operating systems such as Microsoft window, Machintosh OS etc provide GUI consisting of icons, menus etc which has greatly simplified the process of interaction with a computer

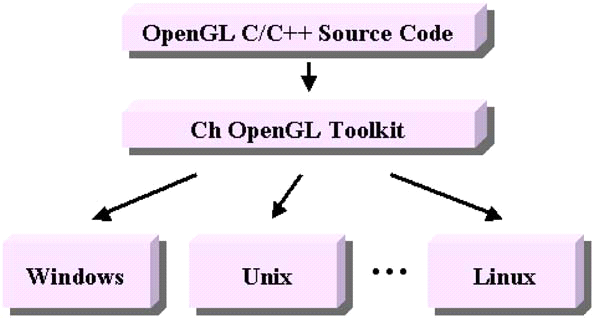
**Chapter 3**

**OPENGL**

**3.1 Introduction To OpenGL**

OpenGL (Open Graphics Library) is a standard specification defining a cross-language, cross-platform API for writing applications that produce 2D and 3D computer graphics. OpenGL is not a programming language it is an API (Application Programming Interface). The interface consists of many different function calls which can be used for building application programs. OpenGL is widely used in CAD, virtual reality, scientific visualization, information visualization, and flight simulation.

OpenGL is portable to many platforms (windows, MAC, UNIX, LINUX) and callable from many programming languages (C/C++, JAVA, PEARL). OpenGL is primarily concerned with modeling and rendering operations such as, to specify geometric primitives (lines, pixels, polygons…), apply geometric transformations and specify camera light, color, texture information etc.



**Fig. 3.1 Displaying Compatibility of OpenGL**

All functions in OpenGL library are device independent, many operations (windowing, I/O etc) are not included in basic core library, so different auxiliary libraries are developed for features not available in OpenGL. OpenGL libraries are OpenGL: GL (lib GL) and GLU (lib GLU). These are windows native implementation (OpenGL32) and Mesa3D: free ware implementation for LINUX. GLUT: OpenGL utility tool kit (lib glut) the GLUT library is responsible for window and menu management, mouse and keyboard interactions.

Graphics libraries are GLUI: GLUT based user interface library (lib glui) -It controls OpenGL applications such as buttons, check boxes, radio buttons etc.Mostof 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 devicesinto programs, user 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 the Macintosh, it is agl. Rather than using a different library for each system, user 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.

Fig 2.2 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.

The intention of this chapter is to give user basic concepts in OpenGL. OpenGL is a device and operating system independent library for 3D-graphics and rendering. OpenGL was originally developed by Silicon Graphics Inc (SGI) for use on their high end graphics workstations. Since then, OpenGL has become a widely accepted standard with implementations on many operating system and hardware platforms including Windows NT and Windows X operating systems. The purpose of the OpenGL library is to render two and three-dimensional objects into frame buffer. OpenGL is a library of high-quality three-dimensional graphics and rendering functions. The library's device- and platform-independence make it a library of choice for developing portable graphical applications.



**Fig. 3.2: Library organization of OpenGL**

OpenGL drawings are constructed from primitives; primitives are simple items such as lines or polygons, which in turn are composed of vertices.

The OpenGL Library assembles primitives from vertices while taking into account a variety of settings, such as color, lighting, and texture. Primitives are then processed in accordance with transformations, clipping settings, and other parameters; at the end of the rasterization process is pixel data deposited into a frame buffer.

Because of 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.

**Advantages of Using 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.
* 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.

**3.2 VC++**

Visual C++ is part of the Microsoft Developer’s workshop. As busy as the workshop frame seems when you first start it up, it contains only a portion of the toolbars, menus, and windows that are available to you while developing a program.

The workshop is not a static place. The screen changes according to what you are doing-whether editing a program file, debugging code, or building resources. On top of that, you can move toolbars and windows, placing them where they are most effective for you. You can even customize some of the individual windows and build your own toolbars. Microsoft truly designed this to be a developer’s workshop.

You can create, edit, or delete toolbars from the Visual C++ IDE to make the workshop reflect the way you work. Toolbars you create are custom toolbars. Any changes you make to a custom toolbar are permanent.

Creating a program is a repetitive process of writing code and testing it. During the writing phase you create files and classes, and variables and functions, and make your code handle the many windows messages. Errors are inevitable, but you can clean them up easily. A few methods of debugging include breakpoints, stepping through a program, and watches.

**Chapter 4**

**REQUIREMENT SPECIFICATON**

Specification must reflect the actual application to be handled by the system and must include the requirements.

**4.1System requirements**

Some of the basic requirements for the development of this project are as follows

### 4.1.1 Hardware requirement:

* + - * + Intel® Pentium® 4 CPU 2.40 GHz
        + 128 MB RAM
        + 40GB HDD
        + Mouse
        + Keyboard
        + Monitor

**4.1.2 Software requirement:**

* + Operating system Windows XP/Vista
    - * + Microsoft Visual studio 5.0
        + Open GL
        + Supporting files

**Chapter 5**

**SYSTEM DESIGN**

**5.1 Architectureofjet plane**

We have divided the project into three main modules as follows

1. Draw module
2. View module
3. Interface module

**APPLICATION**

**VIEWMODULE**

**DRAW MODULE**

**INTERFACE MODULE**

**5.1Architecture of jet plane .**

**5.1.1 Draw module**

The main objective of this module is to draw the different triangles,cones,cylinder required for viewing jet plane.

**5.1.2 View module**

The main objective of this module is to make the jet plane visible in different views like side, top, front.

**5.1.3 Interface module**

The main objective of this module is to make use of the keyboard to change the view of the jet plane.

## 5.2Module description

**5.2.1 Draw module**

This module uses different function for drawing picture of a jet plane which consists of different movements of jet planes.

1. Function to draw a triangle

glColor3ub(205,0,0);

glVertex3f(0.0f,0.0f,60.0f);

glVertex3f(-15.0f,0.0f,30.0f);

glVertex3f(15.0f,0.0f,30.0f);

1. Function to draw body of the plane

glColor3ub(128,128,128);

glVertex3f(-15.0f,0.0f,30.0f);

glVertex3f(0.0f, 15.0f, 30.0f)

glVertex3f(0.0f, 0.0f, -56.0f);

1. Function to draw left wing

glColor3ub(128,128,128);

glVertex3f(0.0f,2.0f,27.0f);

glVertex3f(-60.0f, 2.0f, -8.0f);

glVertex3f(60.0f, 2.0f, -8.0f);

1. Function to draw other wing in top section

glColor3ub(64,64,64);

glVertex3f(0.0f,2.0f,27.0f);

glVertex3f(0.0f, 7.0f, -8.0f);

glVertex3f(-60.0f, 2.0f, -8.0f);

1. Function to draw tail section

(1) bottom of back fin

glColor3ub(128,128,128);

glVertex3f(-30.0f, -0.50f, -57.0f);

glVertex3f(30.0f, -0.50f, -57.0f);

glVertex3f(0.0f,-0.50f,-40.0f);

(2) top of left side

glColor3ub(128,128,0);

glVertex3f(0.0f,-0.5f,-40.0f);

glVertex3f(30.0f, -0.5f, -57.0f);

glVertex3f(0.0f, 4.0f, -57.0f);

(3) top of right side

glColor3ub(128,128,0);

glVertex3f(0.0f, 4.0f, -57.0f);

glVertex3f(-30.0f, -0.5f, -57.0f);

glVertex3f(0.0f,-0.5f,-40.0f);

(4) back of bottom of tail

glColor3ub(128,0,0);

glVertex3f(30.0f,-0.5f,-57.0f);

glVertex3f(-30.0f, -0.5f, -57.0f);

glVertex3f(0.0f, 4.0f, -57.0f);

(5)Top of Tail section left

glColor3ub(128,0,0);

glVertex3f(0.0f,0.5f,-40.0f);

glVertex3f(3.0f, 0.5f, -57.0f);

glVertex3f(0.0f, 25.0f, -65.0f);

(6) back of horizontal section

glColor3ub(128,0,0);

glVertex3f(0.0f,0.5f,-40.0f);

glVertex3f(3.0f, 0.5f, -57.0f);

glVertex3f(0.0f, 25.0f, -65.0f);

**5.2.2 View module**

This module uses function to change the different views of the jet plane

1)Function that changes views based on the keyboard value

view()

**5.2.3 Interface module**

This module uses function to make interaction from keyword

char string[]="z--->Move Left x--->Move Right n--->Original Position";

char string1[]="Arrow Keys:";

char string2[]="UP --->Clockwise UP Rotation";

char string3[]="DOWN--->Anticlockwise DOWN Rotation";

char string4[]="LEFT--->Clockwise LEFT Rotation";

char string5[]="RIGHT-->Anticlockwise RIGHT Rotation";

**Chapter 6**

**IMPLEMENTATION**

## 6.1 Basic Data types

|  |  |  |  |
| --- | --- | --- | --- |
| Name | Description | Size in bytes | Values/Range |
| Int | Signed integer | 4 | -3.4e38 to +3.4e38 |
| Float | Signed floating-point number | 4 | -32768 to +32767 |

**Table 6.1 Data types**

**6.2 Macros used**

**6.2.1 In-built macros**

GLUT\_DOUBLE 2

GLUT\_RGB 0

GLUT\_DEPTH 16

**6.2.2 User-defined macros**

C 3.14/180

PI 3.14

TWO\_PI 2.0 \* PI

RAD\_TO\_DEG 180.0 / PI

**6.3User-defined functions for modules**

**6.3.1 Draw module**

glBegin(GL\_TRIANGLES);

//Red Color

glColor3ub(205,0,0);

glVertex3f(0.0f, 0.0f, 60.0f);

glVertex3f(-15.0f, 0.0f, 30.0f);

glVertex3f(15.0f,0.0f,30.0f);

//Red Color

glColor3ub(245,0,0);

glVertex3f(15.0f,0.0f,30.0f);

glVertex3f(0.0f, 15.0f, 30.0f);

glVertex3f(0.0f, 0.0f, 60.0f);

//Red Color

glColor3ub(255,0,0);

glVertex3f(0.0f, 0.0f, 60.0f);

glVertex3f(0.0f, 15.0f, 30.0f);

glVertex3f(-15.0f,0.0f,30.0f);

// Body of the Plane ////////////////////////

glColor3ub(128,128,128);

glVertex3f(-15.0f,0.0f,30.0f);

glVertex3f(0.0f, 15.0f, 30.0f);

glVertex3f(0.0f, 0.0f, -56.0f);

glColor3ub(140,140,140);

glVertex3f(0.0f, 0.0f, -56.0f);

glVertex3f(0.0f, 15.0f, 30.0f);

glVertex3f(15.0f,0.0f,30.0f);

glColor3ub(130, 130,130);

glVertex3f(15.0f,0.0f,30.0f);

glVertex3f(-15.0f, 0.0f, 30.0f);

glVertex3f(0.0f, 0.0f, -56.0f);

///////////////////////////////////////////////

// Left wing

// Large triangle for bottom of wing

glColor3ub(128,128,128);

glVertex3f(0.0f,2.0f,27.0f);

glVertex3f(-60.0f, 2.0f, -8.0f);

glVertex3f(60.0f, 2.0f, -8.0f);

glColor3ub(64,64,64);

glVertex3f(60.0f, 2.0f, -8.0f);

glVertex3f(0.0f, 7.0f, -8.0f);

glVertex3f(0.0f,2.0f,27.0f);

glColor3ub(192,192,192);

glVertex3f(60.0f, 2.0f, -8.0f);

glVertex3f(-60.0f, 2.0f, -8.0f);

glVertex3f(0.0f,7.0f,-8.0f);

// Other wing top section

glColor3ub(64,64,64);

glVertex3f(0.0f,2.0f,27.0f);

glVertex3f(0.0f, 7.0f, -8.0f);

glVertex3f(-60.0f, 2.0f, -8.0f);

// Tail section///////////////////////////////

// Bottom of back fin

glColor3ub(128,128,128);

glVertex3f(-30.0f, -0.50f, -57.0f);

glVertex3f(30.0f, -0.50f, -57.0f);

glVertex3f(0.0f,-0.50f,-40.0f);

// top of left side

glColor3ub(128,128,0);

glVertex3f(0.0f,-0.5f,-40.0f);

glVertex3f(30.0f, -0.5f, -57.0f);

glVertex3f(0.0f, 4.0f, -57.0f);

// top of right side

glColor3ub(128,128,0);

glVertex3f(0.0f, 4.0f, -57.0f);

glVertex3f(-30.0f, -0.5f, -57.0f);

glVertex3f(0.0f,-0.5f,-40.0f);

// back of bottom of tail

glColor3ub(128,0,0);

glVertex3f(30.0f,-0.5f,-57.0f);

glVertex3f(-30.0f, -0.5f, -57.0f);

glVertex3f(0.0f, 4.0f, -57.0f);

// Top of Tail section left

glColor3ub(128,0,0);

glVertex3f(0.0f,0.5f,-40.0f);

glVertex3f(3.0f, 0.5f, -57.0f);

glVertex3f(0.0f, 25.0f, -65.0f);

glColor3ub(117,0,0);

glVertex3f(0.0f, 25.0f, -65.0f);

glVertex3f(-3.0f, 0.5f, -57.0f);

glVertex3f(0.0f,0.5f,-40.0f);

// Back of horizontal section

glColor3ub(128,128,128);

glVertex3f(3.0f,0.5f,-57.0f);

glVertex3f(-3.0f, 0.5f, -57.0f);

glVertex3f(0.0f, 25.0f, -65.0f);

**6.3.2 Interface module**

char string[]="z--->Move Left x--->Move Right n--->Original Position";

char string1[]="Arrow Keys:";

char string2[]="UP --->Clockwise UP Rotation";

char string3[]="DOWN--->Anticlockwise DOWN Rotation";

char string4[]="LEFT--->Clockwise LEFT Rotation";

char string5[]="RIGHT-->Anticlockwise RIGHT Rotation";

void SpecialKeys(int key, int x, int y)

{

if(key == GLUT\_KEY\_UP)

{

xRot-= 2.0f;flag=0;

}

if(key == GLUT\_KEY\_DOWN)

{

xRot += 2.0f;flag=0;

}

if(key == GLUT\_KEY\_LEFT)

{

yRot -= 2.0f;flag=0;

}

if(key == GLUT\_KEY\_RIGHT)

{

yRot += 2.0f;flag=0;

}

if(key > 358.0f)

xRot = 0.0f;

if(key < -1.0f)

xRot = 358.0f;

if(key > 358.0f)

yRot = 0.0f;

if(key < -1.0f)

yRot = 358.0f;

// Refresh the Window

glutPostRedisplay();

}

void Mykeys(unsigned char key, int x, int y)

{

if(key == 'x' )

{

pxTra -=1.0;flag=0;

}

if(key == 'z' )

{

nxTra +=1.0;flag=0;

}

if(key == 'n' )

{

pxTra =0; nxTra=0;flag=0;

}

RenderScene();

}

**6.3.3 View module**

void SetupRC()

{

// Light values and coordinates

GLfloat ambientLight[] = { 0.3f, 0.3f, 0.3f, 1.0f };

GLfloat diffuseLight[] = { 0.7f, 0.7f, 0.7f, 1.0f };

GLfloat specular[] = { 1.0f, 1.0f, 1.0f, 1.0f};

GLfloat specref[] = { 1.0f, 1.0f, 1.0f, 1.0f };

glEnable(GL\_DEPTH\_TEST); // Hidden surface removal

glFrontFace(GL\_CCW); // Counter clock-wise polygons face out

glEnable(GL\_CULL\_FACE); // Do not calculate inside of jet

// Enable lighting

glEnable(GL\_LIGHTING);

// Setup and enable light 0

glLightfv(GL\_LIGHT0,GL\_AMBIENT,ambientLight);

glLightfv(GL\_LIGHT0,GL\_DIFFUSE,diffuseLight);

glLightfv(GL\_LIGHT0,GL\_SPECULAR,specular);

glEnable(GL\_LIGHT0);

// Enable color tracking

glEnable(GL\_COLOR\_MATERIAL);

// Set Material properties to follow glColor values

glColorMaterial(GL\_FRONT, GL\_AMBIENT\_AND\_DIFFUSE);

// All materials hereafter have full specular reflectivity

// with a high shine

glMaterialfv(GL\_FRONT, GL\_SPECULAR,specref);

glMateriali(GL\_FRONT,GL\_SHININESS,128);

// Light blue background

glClearColor(0.0f, 0.0f, 1.0f, 1.0f );

}

**6.4 In-built functions used**

glutInitDisplayMode()

Used to define the display mode. GLUT\_RGB specifies *RGB* colour mode.GLUT\_DOUBLE specifies a double buffered window. GLUT\_DEPTH is used for hidden surface removal using depth buffer.

glutInitWindowPosition()

specifies the screen location for the upper-left corner of the window.

glutInitWindowSize()

specifies the size, in pixels, of the window.

glutCreateWindow()

creates a window with an OpenGL context. It returns a unique identifier for the new window.

glutDisplayFunc()

registers the display call-back

glutKeyboardFunc()

registers keyboard call-back.

glutIdleFunc()

registers the idle call-back.

glutReshapeFunc()

indicates what action should be taken when the window is resized.The name of function to be called when window is resized is passed as parameter.

glutMainLoop()

gets the application in a never ending loop, always waiting for next event to process.

glBegin()

initiates a new primitive and begins collection of vertices.

glEnd()

terminates a list of vertices.

glVertex3f()

specifies the position of a vertex in 3 dimensions.

glClearColor()

sets the present RGBA clear color used when clearing the color buffer.

glClear()

indicates that the appropriate buffer is to be cleared for redisplaying purpose. GL\_COLOR\_BUFFER\_BIT specifies color buffer has to be cleared. GL\_DEPTH\_BUFFER\_BIT specifies depth buffer has to be cleared.

glMatrixMode ()

specifies which matrix will be affected by subsequent transformations. Mode can be GL\_PROJECTION or GL\_MODELVIEW.

glLoadIdentity() sets the current transformation matrix to the identity matrix.

gluPerspective()

defines a perspective viewing volume using y direction field of view measured in degrees, aspect ratio of front clipping plane, and near and far distances.

gluLookAt()

post multiplies the current matrix by a matrix determined by a viewer at eye point looking at the at point with a specified up direction.

glSwapBuffers()

swaps front and back buffers

glColor3f()

sets the present RGB colors.

glViewport()

specifies a width x height viewport in pixels .

glutPostRedisplay()

requests that the display callback be executed after the current callback returns.

glPushMatrix() and glPopMatrix()

pushes to and pops from the matrix stack corresponding to the current matrix mode.

glRotatef()

alters the current matrix by a rotation of specified angle about the specified axis.

glTranslatef()

alters the current matrix by a specified displacement.

glLightf()

sets light properties.

glMaterialf()

sets material properties.

glutSolidSphere()

generates a solid sphere.

glutReshapeFunc(changesSize)

changes the size.

glutSpecialFunc(Special Keys)

sets the special keyboard callback for the current window.

glutAddMenuEntry( )

append an item to the current menu.

glutAttachMenu( )

attaches mouse button for the current window to the identifier of the current menu.

glutBitMapCharacter(

renders the character at the required position and advances the current raster position by the width of the character.

glRasterPos3f( )

specifies the rater position for pixel operation.

**Chapter 7**

**CONCLUSION AND FUTURE ENHANCEMENT**

**Conclusion**

This project has been a successful learning experience to practically use the functions defined in OpenGL and to understand a variety of features and options present in it. This project clearly demonstrated the richness of graphics library and its ability to make complex 3D images in a very simple way. We intended to make maximum use of the OpenGL functions which we have done to our satisfaction.

This project showed us a jet plane with different views. The inclusion of textures, complex shading and lighting models can make the planes more clear and colourful. The project can be developed into a game by allowing the user to move the jet plane using direction keys. This project is also helpful for students of computer science to study the OpenGL functions in a better way.

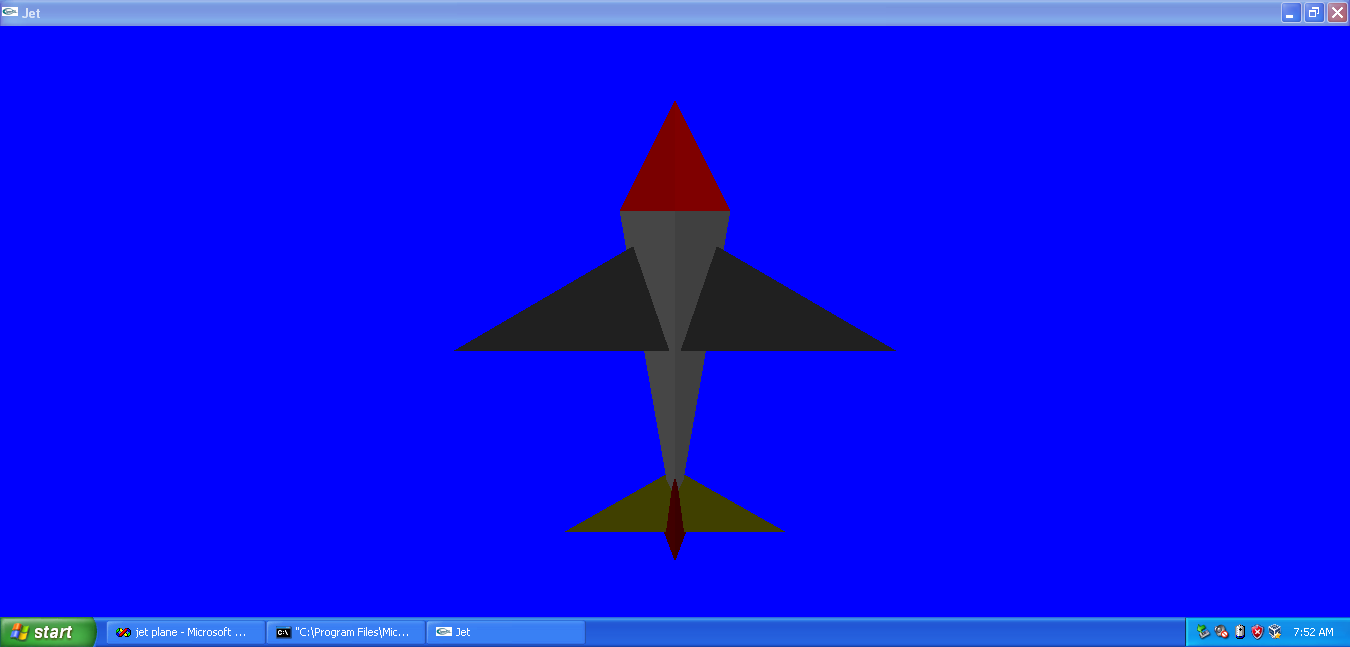
**Future enhancement**

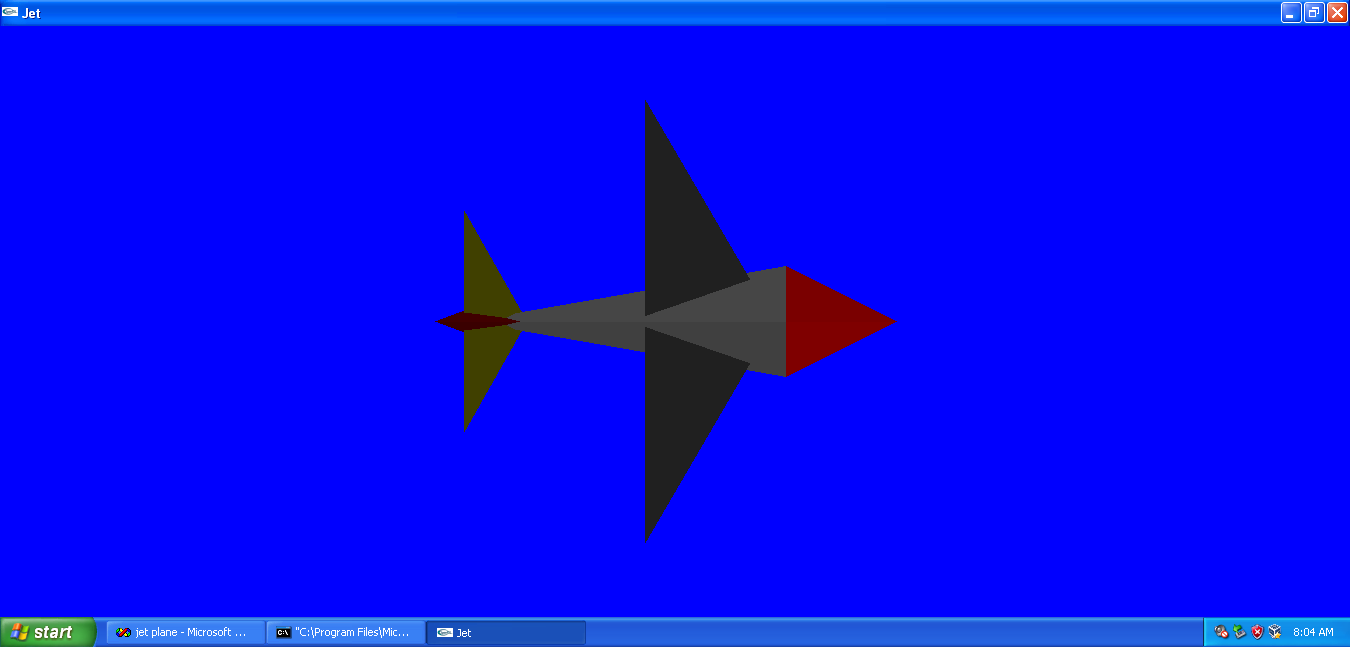
This project can be enhanced to make the jet plane flying at constant speed and altitude and can be viewed in different views.

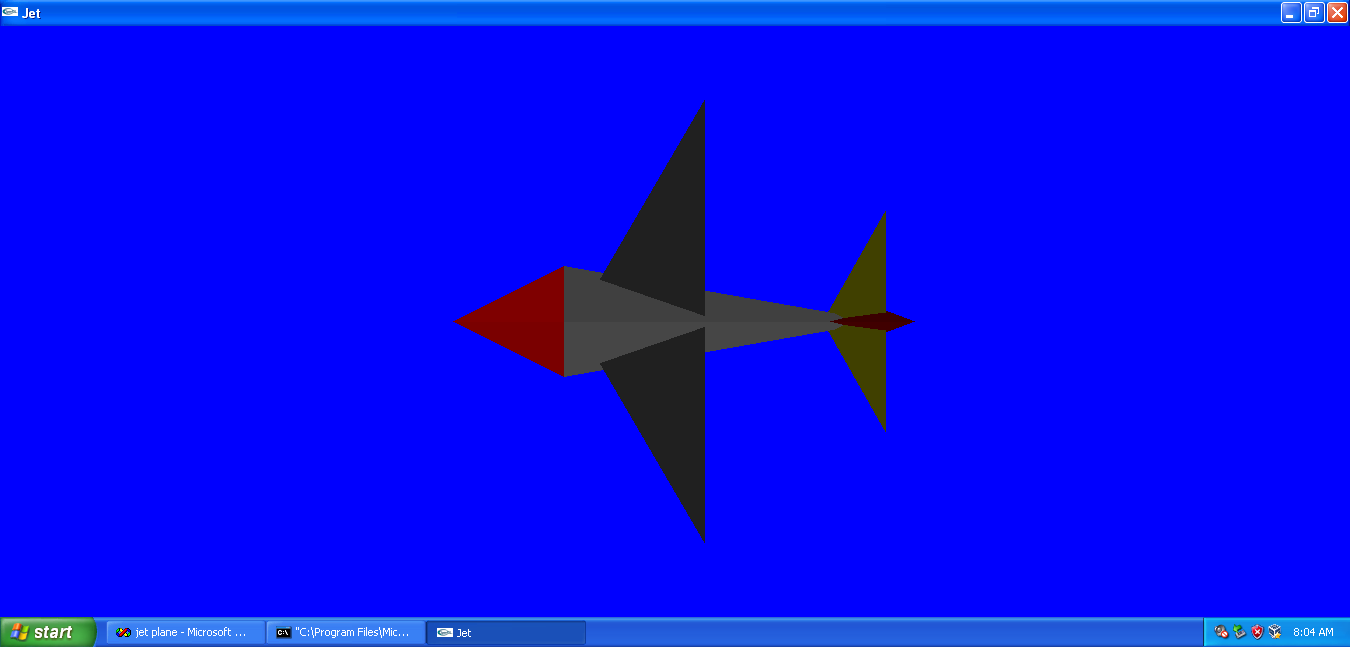
**Chapter 8**

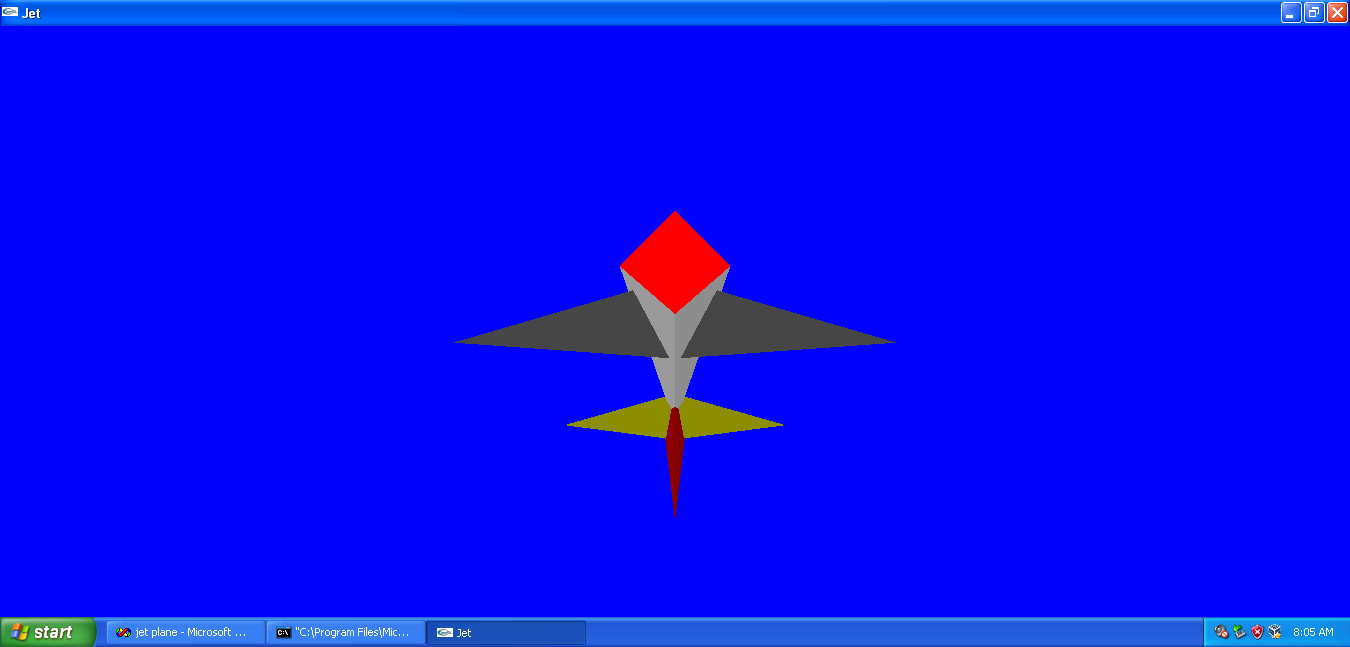
**RESULTS**

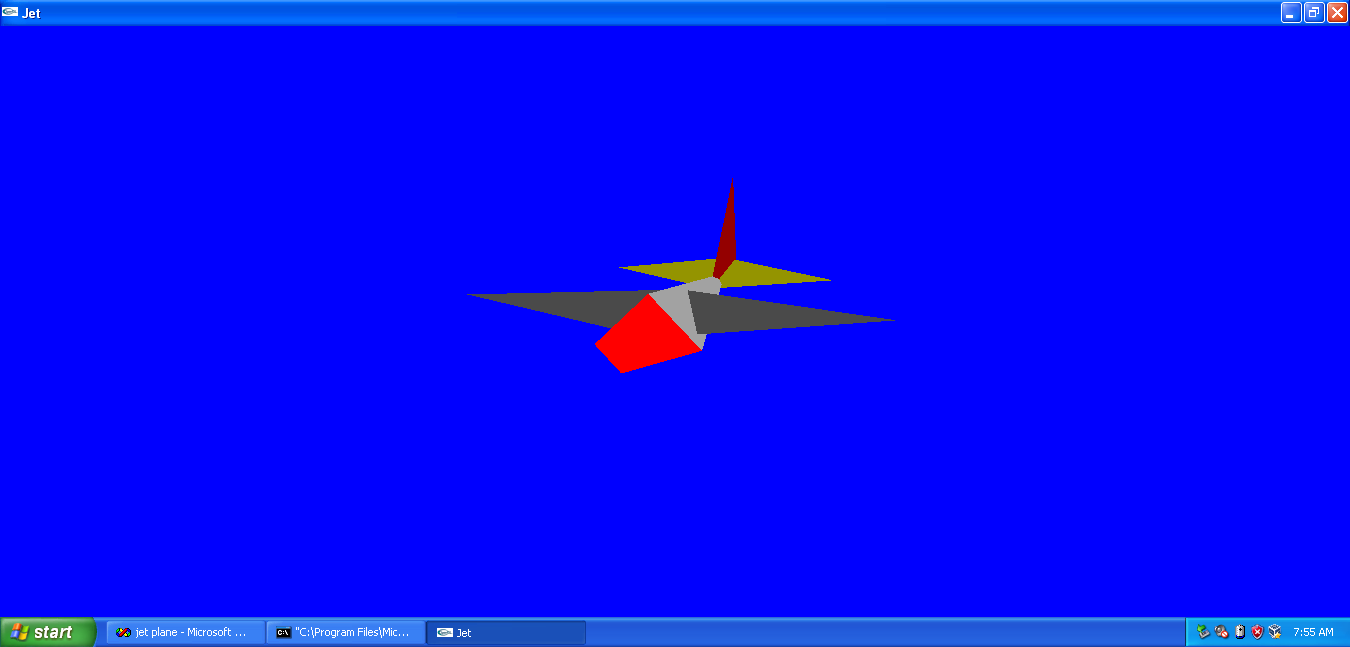
**DIFFERENT VIEWS OF THE JET PLANE MOVEMENTS USING CONTROLS**

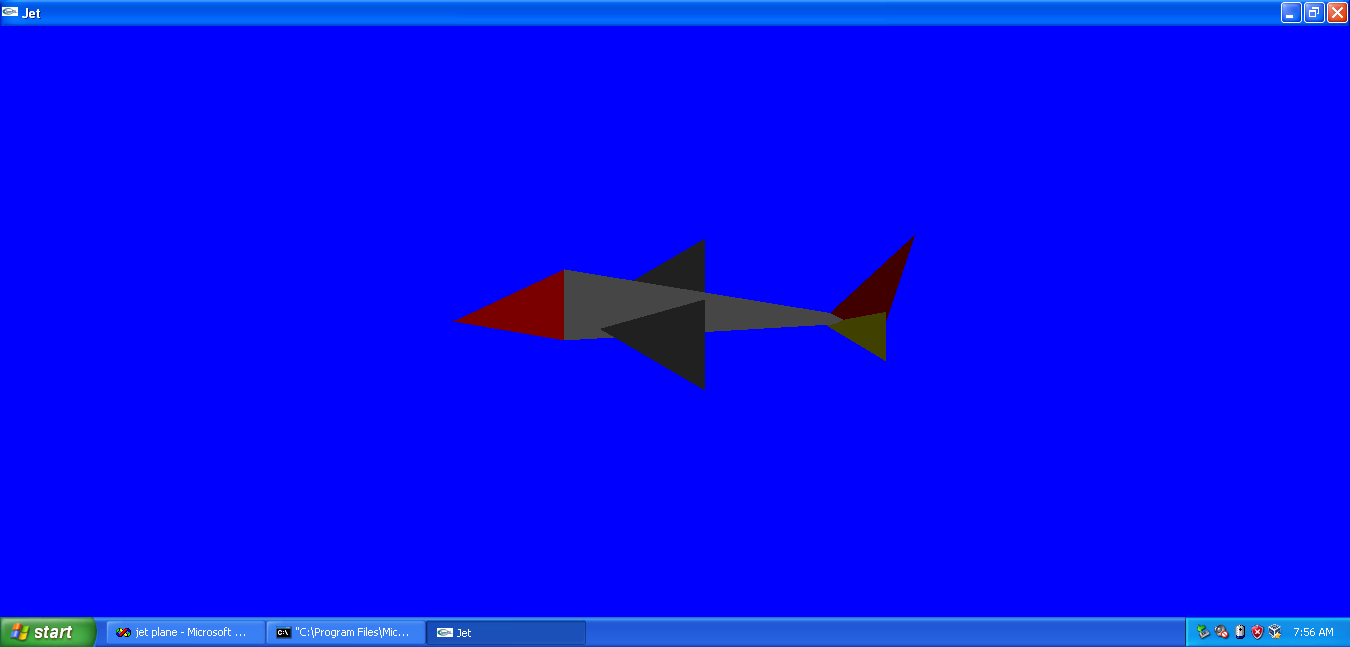
****

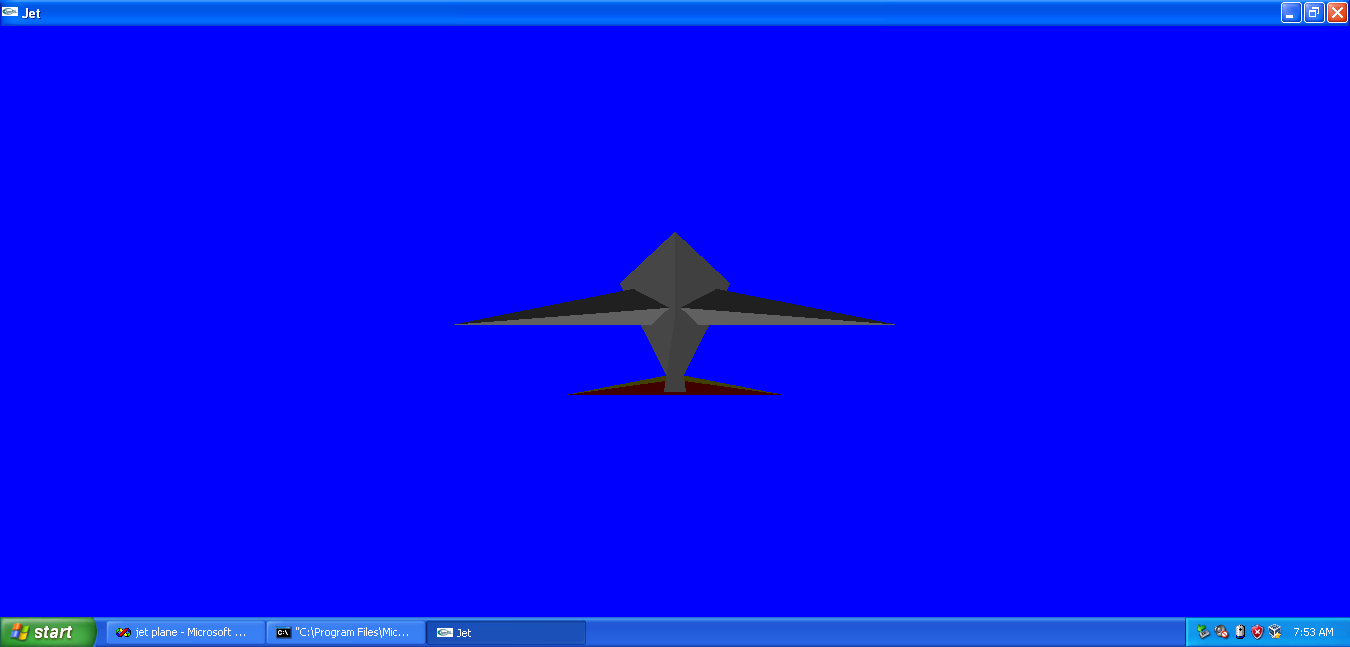


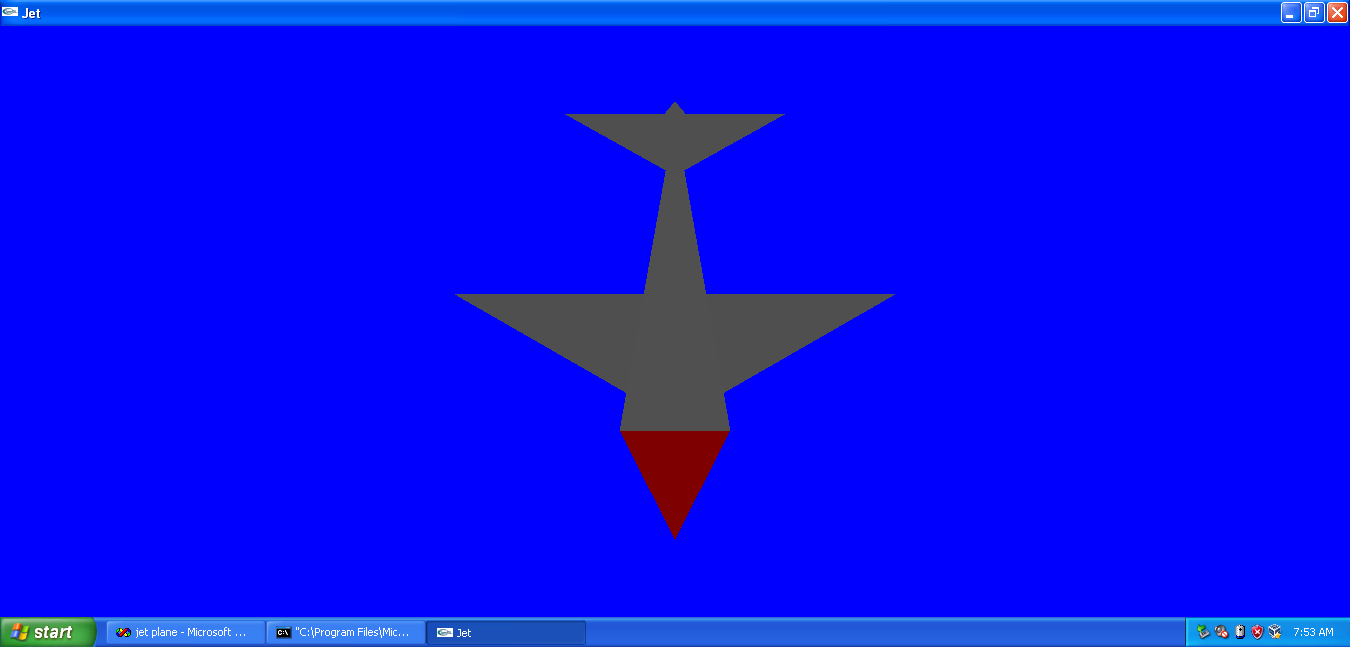


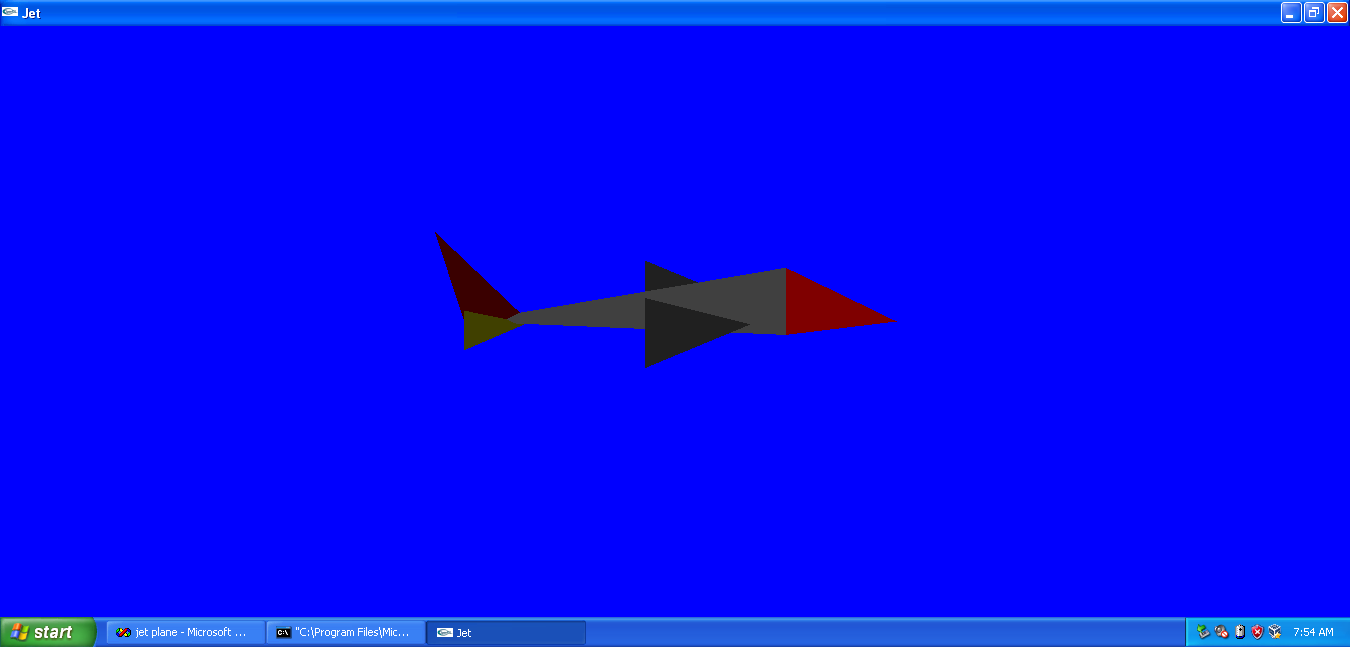


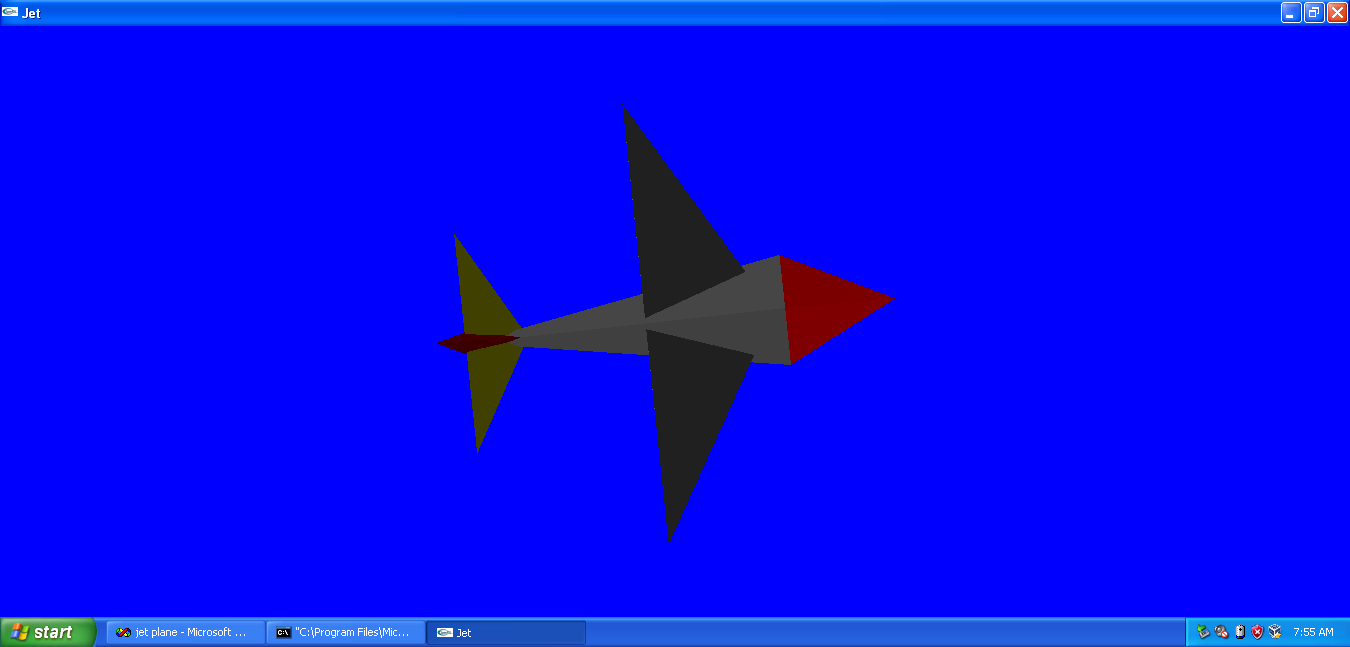


****

****



****

****

##### **BIBLIOGRAPHY**

**Reference Books and E-books:**

* **Interactive Computer Graphics A Top-Down Approach with OpenGL** -Edward Angel, 5th Edition, Addison-Wesley, 2008.
* **OpenGL Red Book.**
* **OpenGL Super Bible** - Richard S. Wright Jr. ,Benjamin Lipchack,Nicholas Haemel

**Websites for Reference:**

[www.opengl.org/resources/code/samples/redbook](http://www.opengl.org/resources/code/samples/redbook)