Contents

[Chapter 1 1](#_Toc483843552)

[INTRODUCTION 1](#_Toc483843553)

[1.1 Computer Graphics 1](#_Toc483843554)

[1.2 OpenGL 2](#_Toc483843555)

[1.2.1 Purpose 3](#_Toc483843556)

[1.2.2 Functionalities 3](#_Toc483843557)

[1.2.3 OpenGL contributions 3](#_Toc483843558)

[1.2.4 Weakness 4](#_Toc483843559)

[1.3 Procedural versus Descriptive 4](#_Toc483843560)

[1.4 Limitations 5](#_Toc483843561)

[1.5 Literature Survey 5](#_Toc483843562)

[Chapter 2 7](#_Toc483843563)

[RESOURCE REQUIREMENTS 7](#_Toc483843564)

[2.1 Hardware Requirements 7](#_Toc483843565)

[2.2 Software Requirements 7](#_Toc483843566)

[Chapter 3 8](#_Toc483843567)

[DESIGN 8](#_Toc483843568)

[3.1 Description 8](#_Toc483843569)

[Chapter 4 10](#_Toc483843570)

[IMPLEMENTATION 10](#_Toc483843571)

[4.1 Header Files Used 10](#_Toc483843572)

[4.2 Description of the functions 10](#_Toc483843573)

[Chapter 5 14](#_Toc483843574)

[SOURCE CODE 14](#_Toc483843575)

[Chapter 6 21](#_Toc483843576)

[TESTING 21](#_Toc483843577)

[6.1 Unit Testing 21](#_Toc483843578)

[6.2 Integration Testing 21](#_Toc483843579)

[6.3 System Testing 21](#_Toc483843580)

[6.4 User Acceptance Testing 22](#_Toc483843581)

[Chapter 7 23](#_Toc483843582)

[RESULTS 23](#_Toc483843583)

[Chapter 8 26](#_Toc483843584)

[CONCLUSION AND FUTURE ENHANCEMENT 26](#_Toc483843585)

[BIBLIOGRAPHY 27](#_Toc483843586)

**LIST OF FIGURES**

|  |  |  |
| --- | --- | --- |
| **Figure** | **Description** | **Page no.** |
| Figure 1.1 | Graphics system | 2 |
| Figure 1.2 | Graphics pipeline architecture | 2 |
| Figure 3.1 | Flow of the program | 9 |
| Figure 7.1 | Initial position of helicopter | 23 |
| Figure 7.2 | Position of helicopter on pressing key-w | 24 |
| Figure 7.3 | Position of helicopter on pressing key-a | 24 |
| Figure 7.4 | Position of helicopter on pressing key-d | 25 |
| Figure 7.5 | Position of helicopter on pressing key-s | 25 |

# Chapter 1

## INTRODUCTION

Computer graphics is concerned with all aspects of producing pictures or images using a computer. This field began almost 50 years ago, with the display of a few lines on a cathode ray tube. Now we can create images by computers that are indistinguishable from photographs of real objects.

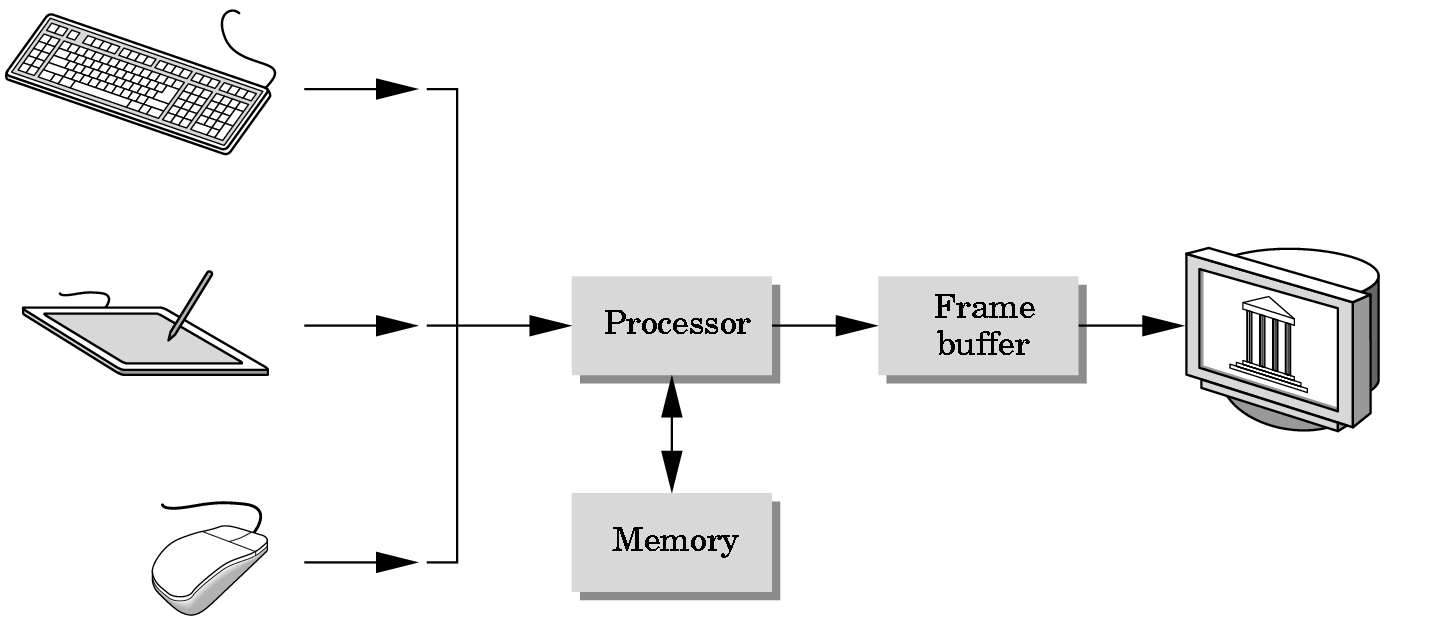
### 1.1 Computer Graphics

Computer graphics is one of the most effective and most commonly used means of communication with the user. It displays the information in the form of graphics objects such as pictures, charts, graphs and diagrams instead of simple text. The pictures or graphics objects may vary from engineering drawings, business graphs and architectural structures to animated movies. All the functionalities required for the development and presentation of such an environment or interface to the user is provided by the graphics package.

There is numerous number of ways in which computer graphics has made user interaction fast, effective and fun. Graphics has enabled the designers to introduce the concept of windows that act as virtual graphics terminals, each of which is capable of running an independent application. The introduction of the mouse has made the selection of objects on the interface easy by the “Point and Click” facility and a lot more.

With the speedily increasing enhancements in the field of computer graphics one can simulate real world objects, create motion by using the different strategies introduced in 2D, 3D and 4D dynamics, one can produce independent frames of objects and convey real time motion to the user through the fast and smooth movement of frames, produce packages for scientific and engineering visualizations, in the field of medicine for the study of human behavior and a lot more. So, we can see that computer graphics has become an integral part of life today and will continue to enhance and ease the usage of computers further more in the near future.

The below figure 1.1 shows the graphics system.

****

INPUT DEVICE OUTPUT DEVICE

Figure.1.1 graphics system

### 1.2 OpenGL

OpenGL was developed by Silicon Graphics Inc.(SGI) in 1992.OpenGL (Open Graphics Library) is a software interface to graphics hardware that produces 2D and 3D computer graphics. This interface consists of about 150 distinct commands that we use to specify the objects and operations needed to produce interactive 3D applications.

OpenGL is designed as a streamlined, hardware-independent interface to be implemented on many different hardware platforms. With OpenGL, we can build up our desired model from a small set of geometric primitives-points, lines and polygon. The below figure 1.2 shows the graphics pipeline architecture.

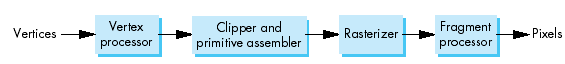


Figure.1.2 Graphics pipeline architecture

#### 1.2.1 Purpose

OpenGL serves two main purposes:

* To hide the complexities of interfacing with different 3D accelerators, by presenting the programmer with a single, uniform API.
* To hide the differing capabilities of hardware platforms, by requiring that all implementations support the full OpenGL feature set.

#### 1.2.2 Functionalities

Basic Functionalities in OpenGL include:

* Points, lines and polygons as basic primitives.
* A transform and lighting pipeline.
* Z-buffering.
* Texture mapping.
* Alpha blending.

#### 1.2.3 OpenGL contributions

* It is very popular in the video games development industry where it competes with Direct3D (on Microsoft Windows).
* OpenGL is also used in CAD, virtual reality, and scientific visualization programs.
* OpenGL is very portable. It will run for nearly every platform in existence, and it will run well. It even runs on Windows NT 4.0 etc. The reason OpenGL runs for so many platforms is because of its Open Standard.
* OpenGL has a wide range of features, both in its core and through extensions. Its extension feature allows it to stay immediately current with new hardware features, despite the mess it can cause.
* Because the ARB is made up of a diverse group of companies, the features available in OpenGL represent a wide range of interests, and thus make it useful in many different applications.

#### 1.2.4 Weakness

Though they are powerful, they do make code messy, very much so at times.

* They also make it confusing with any compiler that doesn't offer reference tracking (browse file). The worst part is, many newer extensions are completely card or vendor specific.
* Although useful for testing a graphics card's abilities, vendor-specific extensions are not frequently used by commercial applications.
* The function naming conventions can seem like overkill at times, since many IDEs have context sensitive help which can show you the parameters that are required, and in any case, if you know how to use a function, you should already know what parameters it takes. In addition, having 12 different names for a function may seem strange to a C++ programmer accustomed to function overloading (of course, since C doesn't support function overloading, there really isn't any way to get around it).

### 1.3 Procedural versus Descriptive

OpenGL provides you with fairly direct control over the fundamental operations of two-dimensional graphics. This includes specification of such parameters as transformation matrices, lighting equation coefficients, anti-aliasing methods, and pixel update operators. However, it doesn’t provide you with a means for describing or modeling complex geometric objects.

Thus, the OpenGL commands you issue specify how a certain result should be produced (what procedure should be followed) rather than what exactly that result should look like. That is, OpenGL is fundamentally procedural rather than descriptive. Because of this procedural nature, it helps to know how OpenGL works-the order in which it carries out its operations.

**Execution Model**

The model for interpretation of openGL commands is client-server. An application (the client) issues commands, which are interpreted and processed by OpenGL (the server).The server may or may not operate on the same computer as the client.

In this sense, OpenGL is network-transparent. A server can maintain several GL contexts, each of which is an encapsulated GL state. The required network protocol can be implemented by augmenting an already existing protocol (such as that of the X Window System) or by using an independent protocol. No OpenGL Commands are provided for obtaining user input.

### 1.4 Limitations

* + OpenGL is case sensitive
  + Line Color, Filled Faces and Fill Color not supported.
  + Bump mapping is not supported.
  + Shadow plane is not supported.
  + Navigation Renderer is not supported.
  + 3D measurement is not supported
  + Streaming of individual 3D objectives is not supported

### 1.5 Literature Survey

* **Translation:**

Translation is an operation that displaces points by a fixed distance in a given direction.To specify a translation, we need only to specify a displacement vector d, because the transformed points are given by

P’= p +d

For all points up on the object. Translation has three of degrees of freedom because we can specify the three components of the displacement vector arbitrarily.

* **Rotation:**

Rotation is more difficult to specify than translation because we must specify more parameters. For example if we want to rotate a two dimensional point(x, y) in a frame with respect to origin by an angle theta to the position (x’, y’), then we use a generalized formula:

X=

Y=

X’=

Y’=

Rotation also has three degree of freedom, the two angles necessary to specify the orientation of the vector and the angle that specifies the amount of the rotation about the vector rotation

* **Scaling:**

Scaling is an affine non rigid body transformation by which we can make an object bigger or smaller. To specify a scaling we can specify the fixed point, a direction in which we wish to scale and scale factor. Scaling has six degrees of freedom because we can specify an arbitrary fixed point and three independent scaling factors.

# 

# Chapter 2

## RESOURCE REQUIREMENTS

### 2.1 Hardware Requirements

The Hardware requirements are very minimal and the program can be run on most of the machines.

|  |  |  |
| --- | --- | --- |
| Processor | : | Pentium4 processor |
| Processor Speed | : | 2.4 GHz |
| RAM | : | 1 GB |
| Storage Space | : | 40 GB |
| Monitor Resolution | : | 1024\*768 or 1336\*768 or 1280\*1024 |

### 2.2 Software Requirements

Operating System : Windows family

IDE : Microsoft Visual C++ (version 8 )

OpenGL libraries

* **Header File**

1. GL/glut.h

* **Object File Libraries**

1. glu32.lib
2. opengl32.lib
3. glut32.lib

* **DLL files (Dynamic Link Libraries)**

1. glu32.dll
2. glut32.dll
3. opengl32.dll

* **The programming is done in C**

# Chapter 3

## DESIGN

Design is the second stage in the software life cycle of any engineered product or system. Design is a creative process. A good design is the key to effective system. It may be defined as “the process of applying various techniques and principles for the purpose of defining a process or a system in sufficient detail to permit its physical realization.”

### 3.1 Description

This application uses OpenGL functions to demonstrate the implementation of an interactive helicopter. The end user can interact with the onscreen helicopter using keyboard keys. The helicopter can move in forward, backward, up and down motions.

The user can use the following keys to interact with the helicopter.

* a – forward
* d – backward
* s – down
* w – up
* r – reset

The figure 3.1 below shows the flow of the program. The main() function calls display() and keyboard() functions. The display() function calls sky() and drawheli() functions which help in specific displaying. The keys used to interact with helicopter are a, d, s, w and r which comes under keyboard() function.

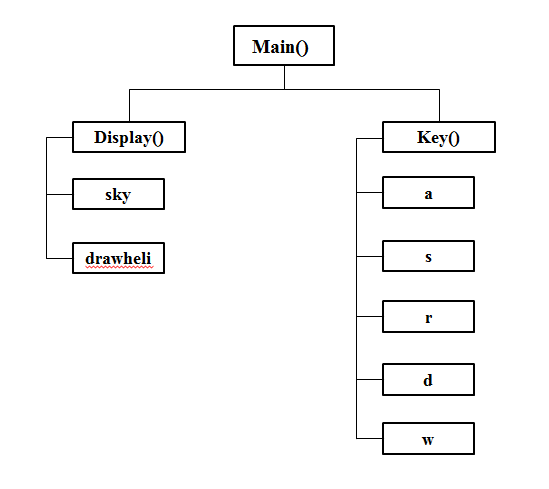
****

Figure:3.1 flow of the program

# Chapter 4

## IMPLEMENTATION

### 4.1 Header Files Used

* **#include<stdio.h>**

The standard input/output library requires the use of a header file called stdio.h. The include command is a directive that tells the compiler to use the information in the header file called stdio.h. The initials stdio stands for standard input output, and stdio.h file contains all the instructions the compiler needs to work with disk files and send information to the output device.

* **#include<GL/glut.h>**

GL is the fundamental openGL library. It provides functions that are permanent part of OpenGL. The functions start with characters ‘gl’.

GLUT, the GL utility tool kit supports developing and managing menus, and managing events. The functions start with characters ‘glut’.

GLU, the GL Utility Library provides high-level routines to handle certain matrix operations, drawing of quadric surfaces such as sphere and cylinders. The functions start with characters ‘glu’.

* **#include<math.h>**

math.h header file contains mathematical operations defined in it.

### 4.2 Description of the functions

* **main()**

The execution of the program starts from main().

* **glutInit() :**

Initializes GLUT. The arguments from main are passed in and can be used by the application.

* **glutInitDisplayMode() :**

Requests a display with the properties in mode. The value of mode is determined by the logical OR of options including the color model (GLUT\_RGB,GLUT\_INDEX) and buffering (GLUT\_SINGLE,GLUT\_DOUBLE).

* **glutInitWindowSize() :**

Specifies the initial height and width of the window in pixels.

* **glutCreateWindow() :**

Creates a window on the display. The string can be used to label the window. The return value provides a reference to the window that can be used when there are multiple windows.

* **glutDisplayFunc() :**

Registers the display function that is executed when the window needs to be redrawn.

* **glutKeyboardFunc():**

Registers the keyboard callback function. The callback function returns the key and the position of the mouse relative to the top- left corner of window.

* **glutIdleFunc() :**

Registers the display callback function that is executed whenever there are no other events to be handled.

* **glutMainLoop() :**

Cause the program to enter an event-processing loop. It should be the last statement in main.

* **myinit()**

The function is defined to initialize the window parameters.

* **glClearColor()**

Sets the present RGBA clear color used when clearing the color buffer. Variables of type GLclampf are floating-point numbers between 0.0 and 1.0.

* **glMatrixMode() :**

Specifies which matrix will be affected by subsequent transformations. mode can be GL\_MODELVIEW,GL\_PROJECTION or GL\_TEXTURE.

* **glLoadIdentity() :**

Sets the current transformation matrix to an identity matrix.

* **glOrtho() :**

Defines an orthographic viewing volume with all parameters measured from the center of the projection plane.

* **display()**

The entire working of the program is graphically displayed on the screen by the contents defined in this function. This thus forms the heart of the program.

Here, all the conditions are checked the program instance must satisfy. Then suitably the instance and its current state are displayed.

* **glColor() :**

Sets the present RGB 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.

* **glutSwapBuffers() :** Swaps the front and back buffers.
* **glPushMatrix():**

Pushes the current matrix stack down by one, duplicating the current matrix i.e., after a glPushMatrix call, the matrix on top of the stack is identical to one below it.

* **glPopMatrix():**

pops the current matrix on top of stack ,replacing the current matrix with one below it on the stack.

* **glFlush() :**

Forces any buffered OpenGL commands to execute.

* **glutPostRedisplay() :**

Requests that the display callback be executed after the current callback returns to 1. By checking for the above mentioned conditions, we display the queen suitably.

* **glutWireSphere():**

This function draws a wire frame sphere centered at origin. The “equatorial” great circle lies in the xy-plane.

* **glTranslatef() :**

Alters the current matrix by a displacement of (x,y,z). Type is either GLfloat or GLdouble

# Chapter 5

## SOURCE CODE

#include<stdio.h>

#include<math.h>

#include<GL/glut.h>

static GLfloat spin = 0,i=0,j=0;

void sky()//sky

{

glBegin(GL\_POLYGON);

glColor3f(0,0.6,0.8);

glVertex2f(0,400);

glVertex2f(650,350);

glColor3f(1.0,1.0,1.0);

glVertex2f(600,300);

glVertex2f(0,100);

glEnd();

}

void drawheli()

{

glPushMatrix();

glTranslatef(i/5,j/5,0);

glPushMatrix();

glTranslated(150,80,0);

glColor3f(1.0,0.0,0.0);

glBegin(GL\_POLYGON);

glVertex3f(55,40,0);

glVertex3f(70,0+20,0);

glVertex3f(140,0+20,0);

glVertex3f(160,40,0);

glVertex3f(160,70+20,0);

glVertex3f(140,100+20,0);

glVertex3f(70,100+20,0);

glVertex3f(55,78,0);

glEnd();

glColor3f(1.0,0.0,0.0);

glBegin(GL\_POLYGON);

glVertex3f(82,20,0);

glVertex3f(82,12,0);

glVertex3f(87,12,0);

glVertex3f(87,20,0);

glEnd();

glColor3f(1.0,0.0,0.0);

glBegin(GL\_POLYGON);

glVertex3f(128,20,0);

glVertex3f(128,12,0);

glVertex3f(123,12,0);

glVertex3f(123,20,0);

glEnd();

glColor3f(0.7,0.0,0.0);

glBegin(GL\_POLYGON);

glVertex3f(70,12,0);

glVertex3f(140,12,0);

glVertex3f(140,9,0);

glVertex3f(70,9,0);

glEnd();

glColor3f(0.7,0.0,0.0);

glBegin(GL\_POLYGON);

glVertex3f(140,12,0);

glVertex3f(143,14,0);

glVertex3f(140,9,0);

glEnd();

glColor3f(0.7,0.0,0.0);

glBegin(GL\_POLYGON);

glVertex3f(70,12,0);

glVertex3f(67,14,0);

glVertex3f(70,9,0);

glEnd();

glColor3f(0.8,0.8,0.8);

glBegin(GL\_POLYGON);

glVertex3f(70,100+20,0);

glVertex3f(55,78,0);

glVertex3f(55,40,0);

glVertex3f(65,50,0);

glEnd();

glColor3f(0.8,0.8,0.8);

glBegin(GL\_POLYGON);

glVertex3f(74,112-5,0);

glVertex3f(70,60-5,0);

glVertex3f(97,60-5,0);

glVertex3f(97,112-5,0);

glEnd();

glColor3f(0.7,0.0,0.0);

glBegin(GL\_POLYGON);

glVertex3f(100+5,112,0);

glVertex3f(130+5,112,0);

glVertex3f(145+5,90,0);

glVertex3f(145+5,70,0);

glVertex3f(125+5,30-2,0);

glVertex3f(100+5,30-2,0);

glEnd();

glColor3f(1.0,0.0,0.0);

glBegin(GL\_POLYGON);

glVertex3f(160,40,0);

glVertex3f(260,80,0);

glVertex3f(260,120,0);

glVertex3f(255,120,0);

glVertex3f(250,90,0);

glVertex3f(160,90,0);

glEnd();

glColor3f(1.0,0.0,0.0);

glBegin(GL\_POLYGON);

glVertex3f(108,100+20,0);

glVertex3f(108,130,0);

glVertex3f(102,130,0);

glVertex3f(102,100+20,0);

glEnd();

glPopMatrix();

glPushMatrix();

glTranslatef(255,210,-100);

spin=spin+2;

glColor3f(0.8, 0.0, 0.0);

glRotatef(spin,0,1,0);

glScalef(1.1,0.07,0);

glutWireSphere(100, 100, 10);

glutPostRedisplay();

glPopMatrix();

glPushMatrix();

glTranslatef(398,181,-100);

glColor3f(1.0, 1.0, 1.0);

glutSolidSphere(17, 50, 20);

glutPostRedisplay();

glPopMatrix();

glPushMatrix();

glTranslatef(398,181,-100);

glColor3f(1.0, 0.0, 0.0);

glutWireSphere(18, 20, 20);

glutPostRedisplay();

glPopMatrix();

glPushMatrix();

glTranslatef(398,181,-100);

spin=spin+2;

glColor3f(0.8, 0.0, 0.0);

glRotatef(spin,0,0,1);

glScalef(1.1,0.2,0);

glutWireSphere(16, 50, 20);

glutPostRedisplay();

glPopMatrix();

glPopMatrix();

glFlush();

}

void display()

{

glClear(GL\_COLOR\_BUFFER\_BIT);

sky();

drawheli();

glFlush();

glutSwapBuffers();

}

void myinit()

{

glClearColor(1.0,1.0,1.0,1.0);

glColor3f(1.0,1.0,1.0);

glMatrixMode(GL\_PROJECTION);

glLoadIdentity();

glOrtho(0.0,600.0,0.0,350,0.0,250.0);

glMatrixMode(GL\_MODELVIEW);

}

void upper()

{

j++;

glutPostRedisplay();

}

void down()

{

j--;

glutPostRedisplay();

}

void lefter()

{

i--;

glutPostRedisplay();

}

void restrt()

{

i=j=0;

i--;

glutPostRedisplay();

}

void righter()

{

i++;

glutPostRedisplay();

}

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

{

if(key=='w')

{

glutIdleFunc(upper);

}

if(key=='s')

{

glutIdleFunc(down);

}

if(key=='a')

{

glutIdleFunc(lefter);

}

if(key=='d')

{

glutIdleFunc(righter);

}

if(key=='r')

{

glutIdleFunc(restrt);

}

}

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

{

glutInit(&argc,argv);

glutInitDisplayMode(GLUT\_DOUBLE | GLUT\_RGB);

glutInitWindowSize(1200,700);

glutInitWindowPosition(0,0);

glutCreateWindow("Helicopter | w-UP a-FRONT s-DOWN r-RESTART (d-BACK)");

glutDisplayFunc(display);

glutKeyboardFunc(keys);

myinit();

glutMainLoop();

}

# Chapter 6

## TESTING

### 6.1 Unit Testing

Unit testing implies the first stage of dynamic testing process. According to software development expert Barry Boehm, a fault discovered and corrected in the unit testing phase is more than a hundred times cheaper than if it is done after delivery to the customer.

It involves analysis of the written code with the intention of eliminating errors. It also verifies that the codes are efficient and adheres to the adopted coding standards. Testing is usually white box. It is done using the Unit test design prepared during the module design phase. This may be carried out by software developers.

### 6.2 Integration Testing

In integration testing the separate modules will be tested together to expose faults in the interfaces and in the interaction between integrated components. Testing is usually black box as the code is not directly checked for errors.

### 6.3 System Testing

System testing will compare the system specifications against the actual system. After the integration test is completed, the next test level is the system test. System testing checks if the integrated product meets the specified requirements. Why is this still necessary after the component and integration tests? The reasons for this are as follows:

Reasons for system test

a) In the lower test levels, the testing was done against technical specifications, i.e., from the technical perspective of the software producer. The system test, though, looks at the system from the perspective of the customer and the future user. The testers validate whether the requirements are completely and appropriately met.

Example - The customer (who has ordered and paid for the system) and the user (who uses the system) can be different groups of people or organizations with their own specific interests and requirements of the system.

b) Many functions and system characteristics result from the interaction of all system components, consequently, they are only visible on the level of the entire system and can only be observed and tested there.

### 6.4 User Acceptance Testing

Acceptance testing is the phase of testing used to determine whether a system satisfies the requirements specified in the requirements analysis phase. The acceptance test design is derived from the requirements document. The acceptance test phase is the phase used by the customer to determine whether to accept the system or not.

The following description is unacceptable in and overview article Acceptance testing:

* To determine whether a system satisfies its acceptance criteria or not.
* To enable the customer to determine whether to accept the system or not.
* To test the software in the "real world" by the intended audience.

Purpose of acceptance testing

* To verify the system or changes according to the original needs.

# Chapter 7

## RESULTS

****

Figure:7.1 initial position of the helicopter

The above figure 7.1 shows the initial position of helicopter before any key is pressed.

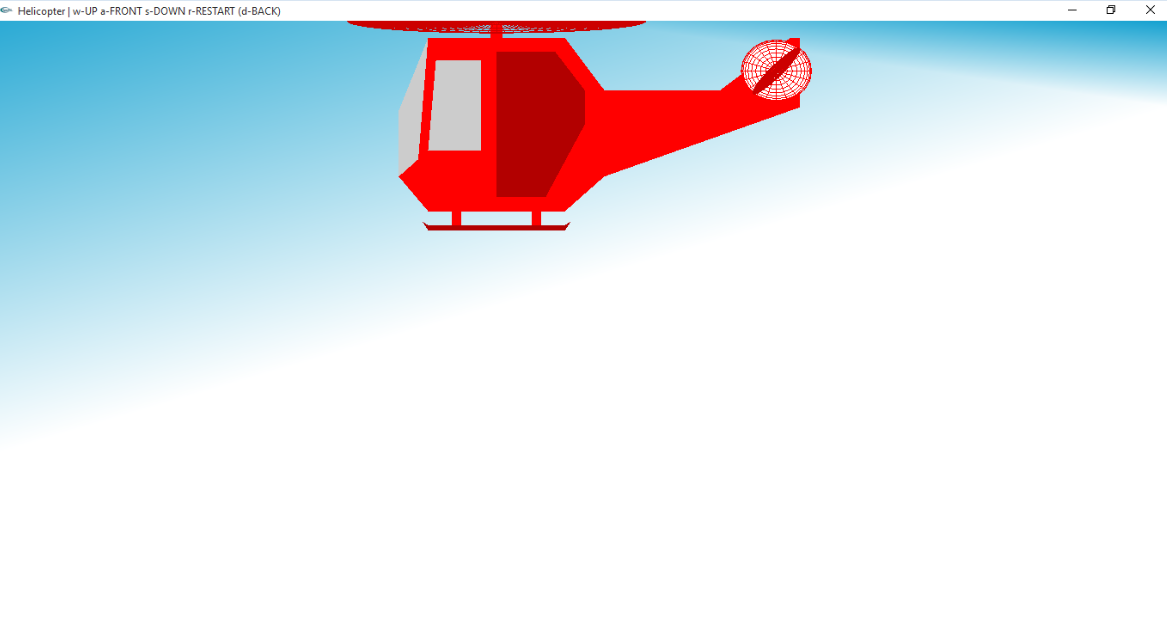


Figure :7.2 position of helicopter on pressing key-w

The above figure 7.2 shows the position of helicopter when ‘w’ key is pressed. When this key is pressed, the helicopter moves upwards.

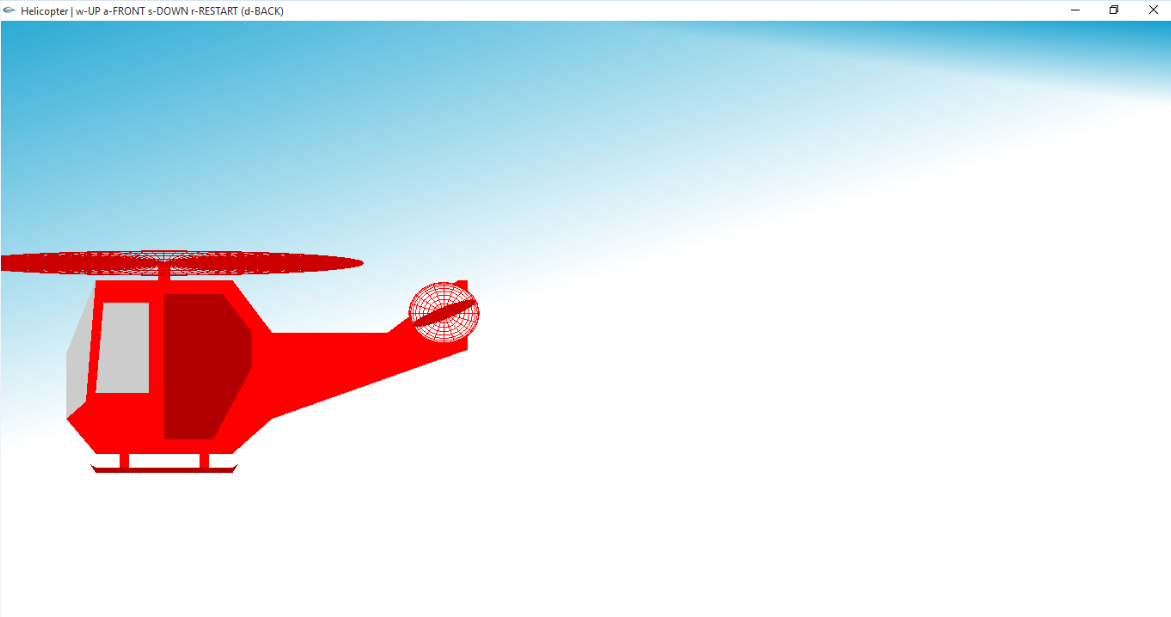


Figure 7.3:position of helicopter on pressing key-a

The above figure 7.3 shows the position of helicopter when ‘a’ key is pressed. When this key is pressed, the helicopter moves forward.

****

Figure :7.4 position of helicopter on pressing key-d

The above figure 7.4 shows the position of helicopter when ‘d’ key is pressed. When this key is pressed, the helicopter moves backwards

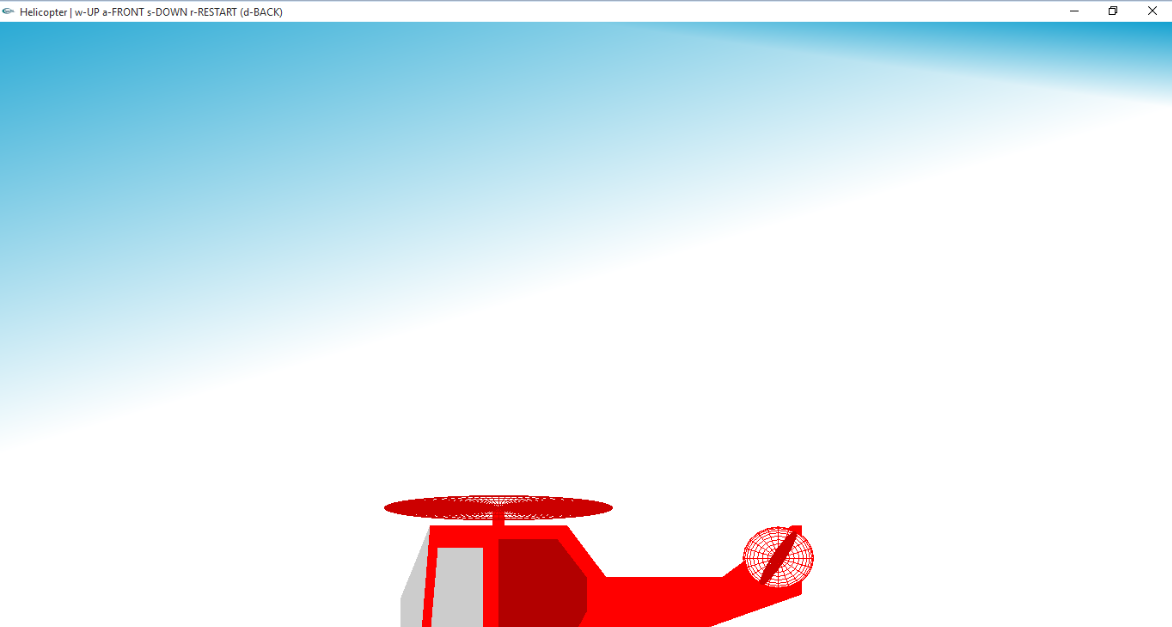
****

Figure:7.5 position of helicopter on pressing key-s

The above figure 7.5 shows the position of helicopter when ‘s’ key is pressed. When this key is pressed, the helicopter moves down.

# Chapter 8

## CONCLUSION AND FUTURE ENHANCEMENT

Moving is implemented using OpenGL and its library functions. The program graphically displays the movement of the helicopter using keyboard inputs. It takes the input from the user and moves the onscreen helicopter accordingly. The primary objective of this project

is to graphically show the movement of helicopter. The onscreen helicopter can be moved in up, down, forward and backward motions. The idea was to simplify the understanding of the concept.

The present project shows the movement of helicopter in 2D. The further enhancements that can be made to this project are:

* Make the project visually appealing by giving the display a 3D perspective.
* Lighting and shading can be implemented.
* Introduce user-interaction by availing options to zoom in and zoom out.

# BIBLIOGRAPHY

[1] Interactive Computer Graphics A Top-Down Approach with OpenGL, Edward Angel, 5th Edition, Addison- Wesley, 2008.

[2] Introduction to the Design and Analysis of Algorithms, Anany Levitin, 2nd Edition, Pearson Education, 2009.

[3] http://www.cprogramming.com/tutorial/opengl\_introduction.html

[4] www.opengl.org/resources/code/samples/redbook/