VISVESVARAYA TECHNOLOGICAL UNIVERSITY

Jnana Sangama, Belagavi – 590018, Karnataka, India

A MINI PROJECT REPORT ON

"HELICOPTER SIMULATION"

A Mini Project Work Report Submitted in partial fulfillment of the requirement for the degree of

BACHELOR OF ENGINEERING
In
COMPUTER SCIENCE & ENGINEERING

Submitted by

PRACHTI JB	**1RG17CS009**
PRAJWAL B MANI	**1RG17CS037**

Under the Guidance of

Mrs. GEETHA PAWAR

Asst. Prof. Dept. of CSE
RGIT, Bengaluru-32

Department of Computer Science & Engineering

RAJIV GANDHI INSTITUTE OF TECHNOLOGY

Cholanagar, R.T. Nagar Post, Bengaluru-560036

2019-2020

RAJIV GANDHI INSTITUTE OF TECHNOLOGY
(Affiliated to Visveshwaraya Technological University)
Cholanagar, R.T.Nagar Post, Bangalore-560032

Department of Computer Science Engineering

CERTIFICATE

This is to certify the Mini Project entitled "HELICOPTER SIMULATION" is a bonafide work carried out by **Ms. Prachiti JB (1RG17CS009)** and **Mr. Prajwal B Mani (1RG17CS037)** in partial fulfillment for the award of **Bachelor of Engineering in Computer Science Engineering** under **Visveshwaraya Technological University, Belagavi**, during the year **2019-2020**. It is certified that all corrections/suggestions given for the internal assessment have been incorporated in the report. This Mini Project has been approved as it satisfies the academic requirements in respect of seminar work.

Signature of Guide
Mrs. Geetha Pawar
Asst. Professor,
Dept of CSE
RGIT, Bengaluru

Signature of HOD
Mrs. Arudra A
Associate Professor & HOD
Dept of CSE
RGIT,Bengaluru

EXTERNAL VIVA

Name of the Examiners	**Signature with Date**
1.	
2.	

VISVESVARAYA TECHNOLOGICAL UNIVERSITY

JNANA SANGAMA, BELAGAVI-590018

RAJIV GANDHI INSTITUTE OF TECHNOLOGY

DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING

DECLARATION

We hereby declare that the mini project work entitled "HELICOPTER SIMULATION" submitted to the **Visvesvaraya Technological University, Belagavi** during the academic year **2019-2020**, is a record of an original work done by us under the guidance of **Mrs. Geetha Pawar, Assistant Professor, Department of Computer Science and Engineering, Rajiv Gandhi Institute of Technology, Bengaluru** and this project work is submitted in the partial fulfillment of requirements for the award of the degree of **Bachelor of Engineering in Computer Science & Engineering**. The results embodied in this thesis have not been submitted to any other University or Institute for award of any degree or diploma.

PRACHITI JB (1RG17CS009)

PRAJWAL B MANI (1RG17CS037)

ACKNOWLEDGEMENT

We take this opportunity to express our sincere gratitude and respect to the **Rajiv Gandhi Institute of Technology, Bengaluru** for providing us an opportunity to carry out our project work.

We express our sincere regards and thanks to **Dr. NAGARAJ A. M, Principal, RGIT, Bengaluru** and **Mrs. Arudra A , Associate Professor and Head, Department of Computer Science & Engineering, RGIT, Bengaluru,** for their encouragement and support throughout the Project.

With profound sense of gratitude, we acknowledge the guidance and support extended by **Mrs. Geetha Pawar, Asst. Prof and Miss.Rajani Kodagali, Asst.Prof, Department of Computer Science & Engineering, RGIT, Bengaluru.** Her incessant encouragement and valuable technical support have been of immense help in realizing this project. Her guidance gave us the environment to enhance our knowledge, skills and to reach the pinnacle with sheer determination, dedication and hard work.

We also extend our thanks to the entire faculty of the Department of CSE, RGIT, Bengaluru, who have encouraged us throughout the course of Bachelor Degree.

PRACHITI JB (1RG17CS009)

PRAJWAL B MANI (1RG17CS037)

ABSTRACT

This project gives us an insight into the use of OpenGL API for Helicopter Simulation with Python as a supporting language. The main objective is to allow the user to interact with the rich graphics features. Graphical perspective is rendered from the viewpoint of the user and also it allows the user to strongly control through multiple camera positions using concepts of FOV (Field of View) and clipping window. This implementation provides the application developers with a flexible way to produce quality graphics

Table of Contents

CHAPTER 1

INTRODUCTION

1.1 Computer Graphics

Computer graphics is concerned with all aspects of producing pictures or images using a computer. The field began humbly 50 years ago, with the display of a few lines on a cathode-ray tube (CRT); now, we can generate images with the 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 computers have been successful, both critically and financially.

1.2 Applications of Computer Graphics

The development of Computer Graphics had been driven both by needs of the user community and by advances in hardware and software. The application of computer graphics are many and varied; we can, however, divide them into four major areas:

1. Display of information
2. Design
3. Simulation and animation
4. User interfaces

Although many applications span two or more of these areas, the development of the field was based on separate work in each.

1.2.1 Display of information

Medical Imaging possesses interesting and important data analysis problem. Modern imaging technologies such as Computed Tomography (CT), Magnetic Resonance Imaging (MRI), Ultrasound and Position Emission Tomography (PET), generate 3D data that must be subjected to algorithmic manipulation t provide useful information. The field of scientific visualization provides graphical tools that help the researchers interpret the fast quantity of data that generate.

1.2.2 Design

Professions such as engineering and architecture are concerned with design. Starting with a set of specifications, engineers and architects seek a cast effective and aesthetic solution that satisfies the specifications.
Design is an interactive process. Design problems are either over determined such that they possess no solution that satisfies all criteria, much less an optimal solution, or undetermined, such that they have multiple solutions that satisfies the design criteria.

1.2.3 Simulation and animation

Graphics system evolved to be capable of generating sophisticated images in real time, engineers and researchers began to use them as simulators. One of the most important use had been training of pilots. The use of special PLSI chips has led to a generation of arcade games as sophisticated as flight simulators.
The simulators can be used for designing the robot, planning its path, and simulating its behavior in complex environment. The success of flight simulators led to the use of Computer Graphics for animation in TV, motion pictures and advertising industries. Entire animated movies can now be made by computer at a cost less than that of movies with traditional ways.

1.2.4 User Interface

Our interaction with computers has become dominated by visual paradigm that includes icons, menus and pointing devices such as mouse. From users perspectives, winding system such as X and window system, Microsoft windows.

1.3 Introduction of OpenGL

Most of the applications will be designed to access OpenGL directly through functions in three libraries. Function 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. The second is the OpenGL Utility Library (GLU). This library uses only GL functions but contains code for creating common objects and simplifying programmers prefer not to write the code repeatedly. The GLU library is available in all OpenGL implementations; functions in the GLU library begins with letters glu.

To interface with the window system and to get input from external devices into programs, need atleast one more system-specific library that provides the "glue" between the window system and OpenGL. For the X window system, this library is functionality that should be expected in any modern windowing system. For this window system, GLUT will use GLX and the X libraries. The application program, however, can use only GLUT functions and thus be recompiled with the GLUT library for other window systems.

GLUT:

It is complete API written by Mark Kilogram, which lets us create windows and handle the messages. It exists for several platforms that a program, which uses GLUT, can be compiled on many platforms without any changes in the code.

ADDITIONAL LIBRARIES:

There are two library functions that we have learnt in OpenGL

1) **glu**

   It is a set of utility function, they are the easy way of doing things that is tedious with raw OpenGL.

2) **glx:**

   it allows you to open up X window and link it up with OpenGL so that it will draw that window.

1.4 Block Diagram of OpenGL

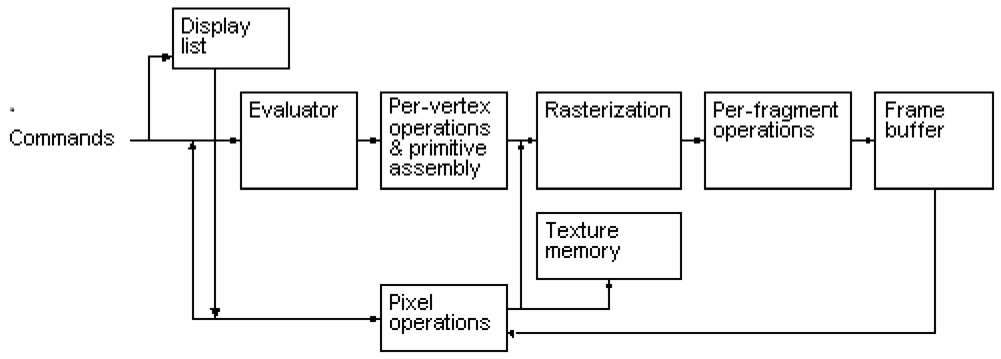

Fig 1: Block Diagram of OpenGL.

1.5 Helicopter Simulation

A helicopter (also often used: chopper or heli) is a kind of flying machine or aircraft. A helicopter lifts up off of the ground and moves because of its rotors. A rotor is several small wings, called rotor blades, that spin together around a shaft. For that reason, helicopters are sometimes called rotary-wing aircraft.
Since around 400 BC, the Chinese had a flying top that was used as a children's toy. The flying top was made from bamboo and used the same method of spinning wings to fly up in the air. Later flying tops were made of feathers tied to a stick. Leonardo da Vinci first thought of a helicopter flown by a man in 1490, and drew pictures of his ideas.
This project produces the simulation of the helicopter with the using computer graphics concepts

1.5.1 Advantages of Helicopter Simulation

- Efficient
- Dynamic
- Easy debugging
- Image manipulation

1.5.2 Disadvantages of helicopter simulation:

- Image rendering
- Unstable clipping window
- Static color and textures

CHAPTER 2

SYSTEM ANALYSIS

2.1 Existing System

The graphics project is designed using the graphics library utility toolkit, which contains graphics manipulation and creation APIs, which are implemented as part of the project. The object created during the course of development of this software consists of many geometric figures like lines, rectangle, squares, polygons, sphere, cube, etc. All the figures taken to be an array of X and Y coordinates, similarly rectangles are stored as a set of 4 points etc. System analysis and design is a very important process in any software development process. This process is called as requirement analysis. Once all the requirements are collected, we can then proceed with the actual designing of the system. During design process, we are involved in actual implementation of the system.

2.2 Proposed System

The proposed system is "Helicopter Simulation" developed using Python. It is implemented on windows. The 3D graphics package designed here provides an interface for the users handling the display. The proposed system provides a part of the visual simulation of the helicopter game, which shows the movement of the helicopter in the air that is controlled by the user. The scenario can be changed dynamically and viewed from a different camera's views.

2.3 Scope of the project

The proposed system provides a basic view of the helicopter. The project simulates the motion of the helicopter blade and moment of the helicopter . The keys, mouse or the touchpad controls the paddle.The simulation also provides the user most of the control. It can be imported into other PCs , gaming console and mobile devices.

2.4 Aim

The main aim of the project is to fly the helicopter and changing the perspective of the camera's position. This is achieved by the easy controls on the helicopter with respect to the main and the tail rotor blade movements. Also the helicopter can be flown in two different atmospheres as per the user’s desire.

CHAPTER 3

REQUIREMENT SPECIFICATIONS

3.1 User Requirements

- Code should be easy to understand and implementable by the user.
- The built-in functions should be utilized to maximum extent.
- Basic knowledge about the OpenGL library facilities.

3.2 Software Requirements

- Operating system - Windows 7 onwards
- Programming language version – Python 3.x
- Modules – PyOpenGL(V3.1.5), PyOpenGL-accelerate(V3.1.5), Pillow Library(V7.1.2)
- Any text editor can be used like Pycharm, VS Code etc

3.3 Hardware Requirements

- Processor – Intel Pentium D CPU with 2.66 GHz
- Memory type – RAM Memory of 768MB
- Mouse
- Keyboard 108 Standard
- Monitor Resolution – 800 x 600

CHAPTER 4

IMPLEMENTATION

4.1 Graphic Functions (Built-in Functions)

4.1.1 void glBegin(glEnum mode);

Initiates a new primitive of type mode and starts the collection of vertices. Values of mode include GL_POINTS, GL_LINES AND GL_POLYGON

4.1.2 void glEnd();

Terminates a list of vertices.

4.1.3 void glColor3f[i f d] (TYPE r, TYPE g, TYPE b);

Sets the present RGB colors. Valid types are int (i),float (f) and double (d). The maximum and minimum values of the floating-point types are 1.0 and 0.0 respectively

4.1.4 void glClearColor(GLclampf r, GLclampf g, GLclampf b, GLclampf a);

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

4.1.5 int glutCreateWindow(char *title);

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

4.1.6 void glutInitWindowSize(int width, int height);

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

4.1.7 void glutInitWindowPosition(int x, int y);

Specifies the initial position of the top-left corner of the window in pixels.

4.1.8 void glutInitDisplayMode(unsigned int mode);

Request a display with the properties in mode. The value of mode is determined by the logical OR of operation including the color model (GLUT_RGB,GLUT_INDEX) and buffering (GLUT_SINGLE,GLUT_DOUBLE);

4.1.9 void glFlush();

Forces any buffered, any OpenGL commands to execute

4.1.10 void glutInit (int argc, char **argv);

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

4.1.11 void glutMainLoop();

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

4.1.12 void glutDisplayFunc(void (*func) (void));

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

4.1.13 gluOrtho2D(GLdouble right, GLdouble bottom, GLdouble top);

Defines a two-dimensional viewing rectangle in the plane Z=0;

4.1.14 void glutBitmapCharacter(void *font, int char);

Renders the characters with ASCII code char at the current raster position using the raster font given by font. Fonts include GLUT_BITMAP_TIMES_ROMAN_10andGLUT_BITMAP_TIMES_ROMAN_8_Y_13. The raster position is incremented by the width of the character.

4.1.15 void glClear(GL_COLOR_BUFFER_BIT);

To make the screen solid and white.

4.1.16 void KeyboardFunc(key);

It is used for the implementation of keyboard interface. Passing control to void key(unsigned char key, int x, int y);

4.1.17 void translate[fd](TYPE x, TYPE y, TYPE z);

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

4.1.18 void glLoadMatric[fd] (TYPE *m);

Loads the 16-element array of TYPE GLfloat or GLdouble as a current matrix.

4.2 Simulation

```
import sys, os
from OpenGL.arrays import vbo
from OpenGL.GLUT import *
from OpenGL.GLU import *
from OpenGL.GL import *
from PIL import Image
from numpy import *
from math import *
import itertools
from scene import Scene, Heli
from controller import Controller
from camera import CameraType
import trafo

WIDTH, HEIGHT = 1000, 1000

# constants
FOV = 45
TIMER_MS = 10
# initial values
aspect = WIDTH/float(HEIGHT)

models = []
skybox_01_info = ('data/skybox/01/', '.png')
skybox_02_info = ('data/skybox/02/', '.png')
helicopter_info = ('data/helicopter/','HELICOPTER500D.obj')

def display():
   glClearColor(1.0, 1.0, 1.0, 1.0)
   glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT)
   drawScene()
   glutSwapBuffers()
```

```
def drawScene():
    global scene
    scene.draw()

def reshape(width, height):
    global WIDTH, HEIGHT

    WIDTH, HEIGHT = width, height
    aspect = width/float(height) if height > 0 else 1.0
    glViewport(0, 0, width, height)
    scene.updateProjection(FOV, aspect)

def initGL(width, height):
    glEnable(GL_TEXTURE_2D)
    glEnable(GL_DEPTH_TEST)
    glEnable(GL_LIGHTING)

def initScene():
    global scene
    scene = Scene(FOV, aspect)

    scene.addCamera(CameraType.FIX)
    scene.addCamera(CameraType.FOLLOW)
    scene.addCamera(CameraType.THIRD_PERSON)

    scene.addSkybox(skybox_01_info)
    scene.addSkybox(skybox_02_info)

    scene.addHelicopter(helicopter_info)

def animation(value):
    scene.helicopter.doRotor()
    scene.helicopter.fly()
    glutTimerFunc(TIMER_MS, animation, value)
    glutPostRedisplay()

def main():
    global helicopter_info
    args = sys.argv
    objpath = None
    if len(args) == 2:
        objpath = args[1]
        path = os.path.split(objpath)
        helicopter_info = (path[0] + "/", path[1])

    glutInit(sys.argv)
    glutInitDisplayMode(GLUT_DOUBLE | GLUT_RGBA | GLUT_DEPTH)
    glutInitWindowSize(WIDTH, HEIGHT)
    glutCreateWindow("Helicopter")

    initScene()
    controller = Controller(scene)

    glutDisplayFunc(display)     #register display function
    glutReshapeFunc(reshape)     #register reshape function
    glutKeyboardFunc(controller.handleKeyDown) #register keyboard function
    glutKeyboardUpFunc(controller.handleKeyUp) #register keyboard function
    glutTimerFunc(TIMER_MS, animation, 0)
    #glutIdleFunc(idlefunc)

    initGL(WIDTH,HEIGHT) #initialize OpenGL state
    glutMainLoop() #start even processing

if __name__ == "__main__":
   main()
```

4.3 Camera

```
from numpy import *
from math import *
import trafo

def enum(**enums):
    return type('Enum', (), enums)

CameraType = enum(FIX = 1, FOLLOW = 2, THIRD_PERSON = 3)

class Camera(object):
    INIT_EYE = [0, 0, 2]
    INIT_CENTER = [0, 0, -1]
    INIT_UP = [0, 1, 0]

    def __init__(self, *information):
        param_count = len(information)
        self.e = information[0] if param_count else Camera.INIT_EYE
        self.c = information[1] if param_count else Camera.INIT_CENTER
        self.up = information[2] if param_count else Camera.INIT_UP

    def getLookAt(self):
        return trafo.lookAtMatrix(self.e[0],self.e[1],self.e[2], self.c[0], self.c[1], 
self.c[2], self.up[0], self.up[1], self.up[2])

    def update(self, helipos, orientation):
        return

class ThirdPersonCamera(Camera):
    INIT_DIR = [0, 0, -1, 0]
    INIT_DIST = 3.0

    def __init__(self, *information):
        param_count = len(information)
        if param_count == 0:
            self.third_person_dir = array(ThirdPersonCamera.INIT_DIR)
            self.third_person_dist = float(ThirdPersonCamera.INIT_DIST)
            Camera.__init__(self)
        if param_count == 2:
            self.third_person_dir = array(information[0])
            self.third_person_dist = float(information[1])
            Camera.__init__(self)
        if param_count == 3:
            self.third_person_dir = array(ThirdPersonCamera.INIT_DIR)
            self.third_person_dist = float(ThirdPersonCamera.INIT_DIST)
            Camera.__init__(self, *information)
        if param_count == 5:
            self.third_person_dir = array(ThirdPersonCamera.INIT_DIR)
            self.third_person_dist = float(ThirdPersonCamera.INIT_DIST)
            Camera.__init__(self, *information[:3])

    def update(self, helipos, orientation):
        self.c = helipos
        direction = dot(orientation, self.third_person_dir)
        direction[1] = 0
        self.e = (array(helipos) + (self.third_person_dist * direction)[:3]).tolist()
        self.up = [0,1,0]


class FixCamera(Camera):
    def __init__(self, *information):
        Camera.__init__(self, *information)


class FollowCamera(Camera):
    def __init__(self, *information):
        Camera.__init__(self, *information)

    def update(self, helipos, orientation):
        self.c = helipos
```

4.4 Controller

```
import sys, time
from OpenGL.GLUT import *

ttime = 0
class Controller(object):
```

```
def __init__(self, scene):
    self.scene = scene
    self.pressedKeys = set()

def handleKeyDown(self, key, x, y):
    key=key.decode('utf-8')
    if key == chr(27):
        sys.exit()
    scene = self.scene
    helicopter = scene.helicopter

    if key == 'c':
        scene.switchCam()

    if key == '1':
        scene.switchSky()
    if key in 'adjlikws':
        self.pressedKeys.add(key)

    if key in 'ad':
        global ttime
        ttime = time.time()

    for k in self.pressedKeys:
        if k == 'a':
            helicopter.gier(False)
        if k == 'd':
            helicopter.gier(True)

        if k == 'j':
            helicopter.roll(True)
        if k == 'l':
            helicopter.roll(False)

        if k == 'i':
            helicopter.nick(True)
        if k == 'k':
            helicopter.nick(False)

        if k == 'w':
            helicopter.pitch(True)
        if k == 's':
            helicopter.pitch(False)

def handleKeyUp(self, key, x, y):
    scene = self.scene
    helicopter = scene.helicopter
    key=key.decode('utf-8')
    if key in 'ad':
        x = time.time()-ttime
        if x > 0.05:
            helicopter.gierSwingout()

    if key in 'adjlikws':
        self.pressedKeys.remove(key)
```

4.5 Trafo

```
from numpy import *
from math import *

import sys, os

def rotationMatrix(angle, axis):
    c, mc = cos(angle), 1-cos(angle)
    s = sin(angle)
    l = sqrt(dot(array(axis), array(axis)))
    x, y, z = array(axis)/l
    r = array([
        [x*x*mc+c, x*y*mc-z*s, x*z*mc+y*s, 0],
        [x*y*mc+z*s, y*y*mc+c, y*z*mc-x*s, 0],
        [x*z*mc-y*s, y*z*mc+x*s, z*z*mc+c, 0],
        [0, 0, 0, 1]])
    return r.T

def scaleMatrix(sx, sy, sz):
    s = array([[sx, 0, 0, 0],
               [0, sy, 0, 0],
               [0, 0, sz, 0],
               [0, 0, 0, 1]])
    return s

def translationMatrix(tx, ty, tz):
    t = array([[1, 0, 0, tx],
               [0, 1, 0, ty],
               [0, 0, 1, tz],
               [0, 0, 0, 1]])
    return t

def lookAtMatrix(ex, ey, ez, cx, cy, cz, ux, uy, uz):
    e = array([ex, ey, ez])
    c = array([cx, cy, cz])
    up = array([ux, uy, uz])

    #normalize
    uplength = sqrt(dot(up, up))
    up = up / uplength
    # viewdirection
    f = c-e
    flength = sqrt(dot(f,f))
    f = f / flength

    s = cross(f, up)
    slength = sqrt(dot(s, s))
    s = s /slength

    u = cross(s, f)
    la = array([
        [s[0], s[1], s[2], -dot(s,e)],
        [u[0], u[1], u[2], -dot(u,e)],
        [-f[0], -f[1], -f[2], dot(f,e)],
        [0, 0, 0, 1]])

    return la

def perspectiveMatrix(fov, aspect, zNear, zFar):
    f = 1.0/tan(fov/2.0)
    aspect = float(aspect)
    zNear = float(zNear)
    zFar = float(zFar)
    p = array([
        [f/aspect, 0, 0, 0],
        [0, f, 0, 0],
        [0, 0, (zFar+zNear)/(zNear-zFar), (2*zFar*zNear)/(zNear- zFar)],
        [0, 0, -1, 0]])
    return p
```

4.6 Sender

```
from OpenGL.GL import *
from OpenGL.GL.shaders import *

from numpy import *
from math import *

class Sender(object):
    def __init__(self, shaderprogram, varnames):
        self.locations = {}
        for varname in varnames:
            self.locations[varname] = glGetUniformLocation(shaderprogram, varname)

    def _getloc(self, varname):
        return self.locations[varname]

    def sendMat4(self, varname, matrix):
        glUniformMatrix4fv(self._getloc(varname), 1, GL_TRUE, matrix.tolist())

    def sendMat3(self, varname, matrix):
        glUniformMatrix3fv(self._getloc(varname), 1, GL_TRUE, matrix.tolist())

    def sendVec4(self, varname, value):
        glUniform4f(self._getloc(varname), *value)

    def sendVec3(self, varname, value):
        glUniform3f(self._getloc(varname), *value)

    def sendValue(self, varname, value):
        glUniform1f(self._getloc(varname), value)
```

CHAPTER 5

RESULT

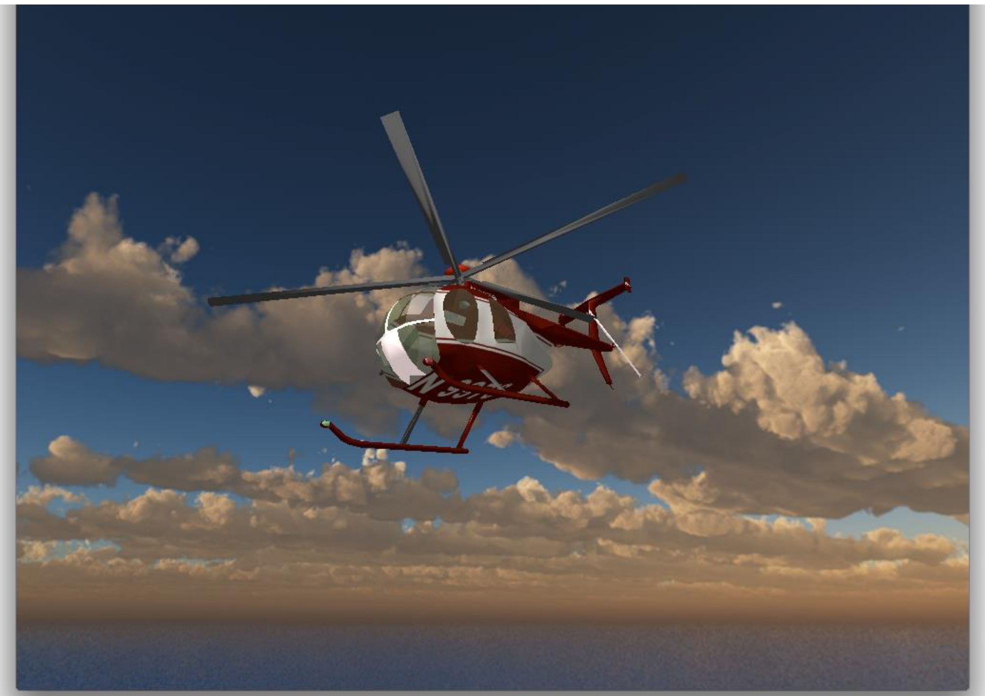

Figure 1: Long View

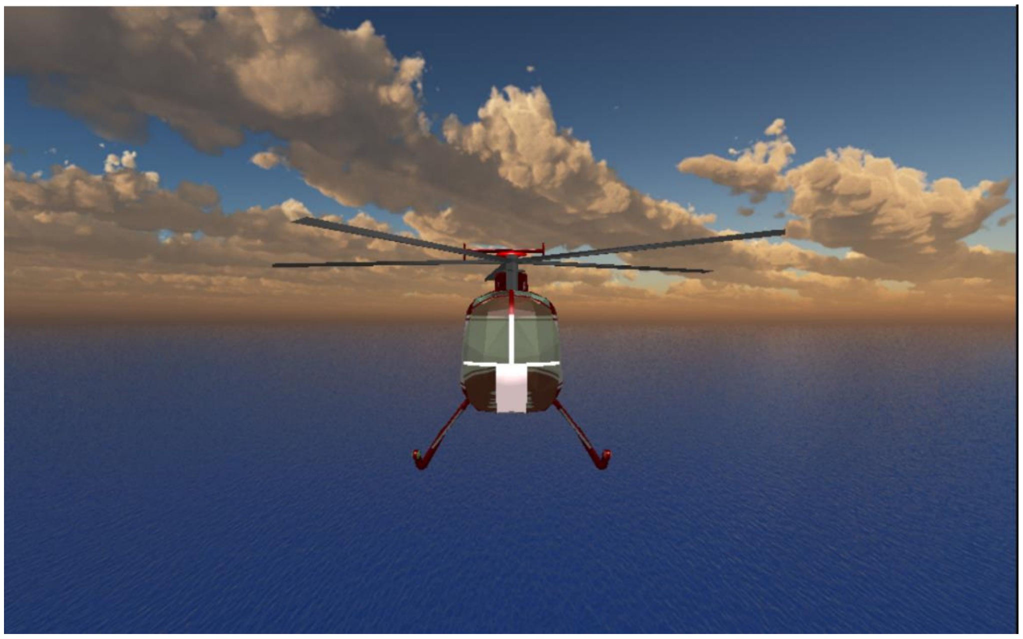

Figure 2: Front View

Figure 3: Back View

Figure 4: Top View

Figure 5: Land View

CHAPTER 6

CONCLUSION

This proposed project has successfully completed simulating the helicopter
From the graphics which have been rendered by changing the camera perspectives, the user can have effective and easy access to the controls of the helicopter. The graphics are constructed in such a way that it is visually appealing for the user. There is additional room for future up-gradation that would improve the quality of this project and make it, even more, user friendly. Some of the features that can be enhanced are:

- Adding dynamic color and textures
- Have a stable clipping window
- Make user controls smoother
- Better image rendering
- Reducing the complexity of the program

BIBLIOGRAPHY

Textbooks Referred:

→ Interactive Computer Graphics, A Top –Down Approach with OpenGL-Edward Angel, 5th Edition, Addison-Wesley-2008

→ The Official Guide to Learning OpenGL, by Jackie Nedier, Tom Davis, Mason Woo(THE RED BOOK)

→ OpenGL Super Bible by Richard S, Wright. Jr and Micheal Sweet

→ Donald Hearn &amp; Pauline Baker: Computer Graphics with OpenGL Version, 3rd /4thEdition, Pearson Education,2011

Website Referred:

- https://www.opengl.org/
- https://stackoverflow.com/
- http://pyopengl.sourceforge.net/documentation/index.html
- https://pillow.readthedocs.io/en/stable/
- https://docs.python.org/3/