VISVESVARAYA TECHNOLOGICAL UNIVERSITY
Jnanasangama, Belagavi, Karnataka

VI Semester
GRAPHICS PACKAGE

3D LANDSCAPE

Submitted By

1BI13CS063 Hitesh RS

for the academic year 2015-2016

BANGALORE INSTITUTE OF TECHNOLOGY
Department Computer Science & Engineering
K.R. Road, V.V.Puram, Bengaluru-560 004

VISVESVARAYA TECHNOLOGICAL UNIVERSITY
Jnanasangama, Belagavi, Karnataka

BANGALORE INSTITUTE OF TECHNOLOGY
K.R. Road, V.V.Puram, Bengaluru-560 004

Department of Computer Science & Engineering

<u>CERTIFICATE</u>

This is to certify that the implementation of **GRAPHICS PACKAGE** entitled **"3D LANDSCAPE"** has been successfully completed by **HITESH RS (1BI13CS063)** of VI semester B.E. for the partial fulfillment of the requirements for the Bachelor's degree in Computer Science & Engineering of the VISVESVARAYA TECHNOLOGICAL UNIVERSITY during the academic year 2015-2016.

Lab Incharges:

		Dr. S. NANDAGOPALAN
1. **M.Kempanna**	2. **Suma.L**	Professor and Head
Assistant Professor	Assistant Professor	Department of CS&E
Dept. of CS&E	Dept. of CS&E	Bangalore Institute of Technology
B.I.T	B.I.T	Bangalore
Bangalore	Bangalore	

External Examiner: **1.** **2.**

ACKNOWLEDGEMENT

I would like to express my deepest appreciation to all those who provided me the possibility to complete this report. A special thanks of gratitude to **Dr.S.Nandagopalan**, Head of the Department, Computer Science & Engineering for giving us the consent to carry out this project. I would also like to thank Professor **Girija.J** (Associate Professor) whose contribution in stimulating suggestions and encouragement helped me to coordinate my project especially in writing this report. I would like to express my thanks of gratitude to my lab incharges **M.Kempanna** (Assistant Professor) and **Suma.L** (Assistant Professor) whose suggestions and support helped me a lot in completing the project.

Furthermore I would also like to acknowledge with much appreciation the crucial role of the staff of Department of Computer Science & Engineering, who gave the permission to use all required equipment and the necessary materials to complete my Graphics Package. A special thanks goes to my friends who helped me by giving suggestion about my Graphics Package. I have to appreciate the guidance given by other Professors as well as that has improved my presentation skills thanks to their comment and advices.

HITESH RS

1BI13CS063

CONTENTS

Chapter-1
Introduction

Introduction

What Is OpenGL?

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

OpenGL is designed as a streamlined, hardware-independent interface to be implemented on many different hardware platforms. To achieve these qualities, no commands for performing windowing tasks or obtaining user input are included in OpenGL; instead, you must work through whatever windowing system controls the particular hardware you're using. Similarly, OpenGL doesn't provide high-level commands for describing models of three-dimensional objects. Such commands might allow you to specify relatively complicated shapes such as automobiles, parts of the body, airplanes, or molecules. With OpenGL, you must build up your desired model from a small set of *geometric primitives* - points, lines, and polygons.

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

OpenGL-Related Libraries

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

- The OpenGL Utility Library (GLU) contains several routines that use lower-level OpenGL commands to perform such tasks as setting up matrices for specific

- viewing orientations and projections, performing polygon tessellation, and rendering surfaces. This library is provided as part of every OpenGL implementation. GLU routines use the prefix **glu**.
- For every window system, there is a library that extends the functionality of that window system to support OpenGL rendering. For machines that use the X Window System, the OpenGL Extension to the X Window System (GLX) is provided as an adjunct to OpenGL. GLX routines use the prefix **glX**. For Microsoft Windows, the WGL routines provide the Windows to OpenGL interface. All WGL routines use the prefix **wgl**. For IBM OS/2, the PGL is the Presentation Manager to OpenGL interface, and its routines use the prefix **pgl**.
- The OpenGL Utility Toolkit (GLUT) is a window system-independent toolkit, written by Mark Kilgard, to hide the complexities of differing window system APIs. GLUT routines use the prefix **glut.**
- Open Inventor is an object-oriented toolkit based on OpenGL which provides objects and methods for creating interactive three-dimensional graphics applications. Open Inventor, which is written in C++, provides prebuilt objects and a built-in event model for user interaction, high-level application components for creating and editing three-dimensional scenes, and the ability to print objects and exchange data in other graphics formats. Open Inventor is separate from OpenGL.

Include Files

For all OpenGL applications, you want to include the gl.h header file in every file. Almost all OpenGL applications use GLU, the aforementioned OpenGL Utility Library, which 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 you are directly accessing a window interface library to support OpenGL, such as GLX, AGL, PGL, or WGL, you must include additional header files. For example, if you are calling GLX, you may need to add these lines to your code

```
#include <X11/Xlib.h>
#include <GL/glx.h>
```

If you are using GLUT for managing your window manager tasks, you should include

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

Note that glut.h includes gl.h, glu.h, and glx.h automatically, so including all three files is redundant. GLUT for Microsoft Windows includes the appropriate header file to access WGL.

GLUT, the OpenGL Utility Toolkit

As you know, OpenGL contains rendering commands but is designed to be independent of any window system or operating system. Consequently, it contains no commands for opening windows or reading events from the keyboard or mouse. Unfortunately, it's impossible to write a complete graphics program without at least opening a window, and most interesting programs require a bit of user input or other services from the operating system or window system.

In addition, since OpenGL drawing commands are limited to those that generate simple geometric primitives (points, lines, and polygons), GLUT includes several routines that create more complicated three-dimensional objects such as a sphere, a torus, and a teapot. This way, snapshots of program output can be interesting to look at. (Note that the OpenGL Utility Library, GLU, also has quadrics routines that create some of the same three-dimensional objects as GLUT, such as a sphere, cylinder, or cone.)

Important features of OpenGL Utility Toolkit (GLUT)

- Provides functionality common to all window systems.
- Open a window.
- Get input from mouse and keyboard.
- Menus.
- Event-driven.
- Code is portable but GLUT lacks the functionality of a good toolkit for a specific platform.
- No slide bars.
- OpenGL is not object oriented so that there are multiple functions for a given logical function :
- glVertex3f
    - glVertex2i
    - glVertex3dv
- Underlying storage mode is the same easy to create overloaded functions in C++ but issue is efficiency.
- **OpenGL Interface**
    - GL (OpenGL in Windows)
    - GLU (graphics utility library)
      uses only GL functions, creates common objects (such as spheres)
    - GLUT (GL Utility Toolkit)
      interfaces with the window system
    - GLX: glue between OpenGL and Xwindow, used by GLUT

Window Management

Five routines perform tasks necessary to initialize a window.

- **glutInit**(int **argc*, char ***argv*) initializes GLUT and processes any command line arguments (for X, this would be options like -display and -geometry).

glutInit() should be called before any other GLUT routine.

- **glutInitDisplayMode**(unsigned int *mode*) specifies whether to use an *RGBA* or color-index color model. You can also specify whether you want a single- or double-buffered window. (If you're working in color-index mode, you'll want to load certain colors into the color map; use **glutSetColor()** to do this.) Finally, you can use this routine to indicate that you want the window to have an associated depth, stencil, and/or accumulation buffer. For example, if you want a window with double buffering, the RGBA color model, and a depth buffer, you might call **glutInitDisplayMode**(*GLUT_DOUBLE* | *GLUT_RGB* | *GLUT_DEPTH*).
- **glutInitWindowPosition**(int *x*, int *y*) specifies the screen location for the upper-left corner of your window.
- **glutInitWindowSize**(int *width*, int *size*) specifies the size, in pixels, of your window.
- int **glutCreateWindow**(char **string*) creates a window with an OpenGL context. It returns a unique identifier for the new window. Be warned: Until **glutMainLoop()** is called (see next section), the window is not yet displayed.

The Display Callback

glutDisplayFunc(void (**func*)(void)) is the first and most important event callback function you will see. Whenever GLUT determines the contents of the window need to be redisplayed, the callback function registered by **glutDisplayFunc()** is executed. Therefore, you should put all the routines you need to redraw the scene in the display callback function.

If your program changes the contents of the window, sometimes you will have to call **glutPostRedisplay**(void), which gives **glutMainLoop()** a nudge to call the registered display callback at its next opportunity.

Running the Program

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

Graphics Programming Using OpenGL in Visual C++

- Opengl32.dll and glu32.dll should be in the system folder.
- Opengl32.lib and glu32.lib should be in the lib folder for VC++.
- gl.h and glu.h should be in a folder called GL under the include folder for VC++.
- Get glut32.lib, glut32.dll and glut.h from the course homepage and put them in the same places as the other files.
- Fire up visual studio.
- Create a new project as the following :

  File ◊ New ◊ Project (input your project name, then a directory (workspace) with the same name will be built)

  ◊ Win32 Console Application

  ◊ An Empty Application
- In the workspace click on "File View" to access (expand) the source code tree.
- In "Source Files", add in the source code (*.cpp files).
- Select Project ◊ Settings ◊ Link and in "Object/library modules" add "Opengl32.lib glu32.lib glut32.lib".
- Press "F7" to build "your_project.exe".

Graphics Functions

- Primitive functions:

  points, line segments, polygons, pixels, text, curves, surfaces
- Attributes functions:

color, pattern, typeface

Some Primitive Attributes

glClearColor (red, green, blue, alpha); - Default = (0.0, 0.0, 0.0, 0.0)
glColor3f (red, green, blue); - Default = (1.0, 1.0, 1.0)
glLineWidth (width); - Default = (1.0)
glLineStipple (factor, pattern) - Default = (1, 0xffff)
glEnable (GL_LINE_STIPPLE);
glPolygonMode (face, mode) - Default = (GL_FRONT_AND_BACK, GL_FILL)
glPointSize (size); - Default = (1.0)

- Viewing functions:
  position, orientation, clipping
- Transformation functions:
  rotation, translation, scaling
- *Input functions:*
  keyboards, mice, data tablets
- Control functions:
  communicate with windows, initialization, error handling
- Inquiry functions: number of colors, camera parameters/values

Matrix Mode

There are two matrices in OpenGL:

- Model-view: defines COP and orientation
- Projection: defines the projection matrix

```
glMatrixMode(GL_PROJECTION);
glLoadIdentity();
gluOrtho2D(0.0, 500.0, 0.0, 500.0);
glMatrixMode(GL_MODELVIEW);
```

Control Functions

- OpenGL assumes origin is bottom left
- glutInit(int *argcp, char **argv);
- glutCreateWindow(char *title);
- glutInitDisplayMode(GLUT_RGB | GLUT_DEPTH | GLUT_DOUBLE);
- glutInitWindowSize(480,640);
- glutInitWindowPosition(0,0);
- OpenGL default: RGB color, no hidden-surface removal, single buffering

Obtaining Values of OpenGL State Variables

```
glGetBooleanv (paramname, *paramlist);
glGetDoublev (paramname, *paramlist);
glGetFloatv (paramname, *paramlist);
glGetIntegerv (paramname, *paramlist);
```

Saving and Restoring Attributes

```
glPushAttrib (group);
glPopAttrib ( );
```

where group = GL_CURRENT_BIT, GL_ENABLE_BIT, GL_LINE_BIT, GL_POLYGON_BIT, etc.

Projection Transformations

```
glMatrixMode (GL_PROJECTION);
glLoadIdentity ( );
```

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

gluPerspective (fov, aspect, near, far);

glOrtho (left, right, bottom, top, near, far);

- Default = (-1.0, 1.0, -1.0, 1.0, -1.0, 1.0)

gluOrtho2D (left, right, bottom, top);

Modelview Transformations

glMatrixMode (GL_MODELVIEW);

glLoadIdentity ( );

gluLookAt (eye_x, eye_y, eye_z, at_x, at_y, at_z, up_x, up_y, up_z);

glTranslatef (dx, dy, dz);

glScalef (sx, sy, sz);

glRotatef (angle, axisx, axisy, axisz);

Writing Bitmapped Text

glPixelStorei (GL_UNPACK_ALIGNMENT, 1);

glColor3f (red, green, blue);

glRasterPos2f (x, y);

glutBitmapCharacter (font, character);

where font = GLUT_BITMAP_8_BY_13, GLUT_BITMAP_HELVETICA_10, etc.

Managing the Frame Buffer

glutInit (&argc, argv);

glutInitDisplayMode (GLUT_RGB | mode);

glutInitWindowSize (width, height);

glutInitWindowPosition (x, y);

glutCreateWindow (label);

glClear (GL_COLOR_BUFFER_BIT);

glutSwapBuffers ( );

where mode = GLUT_SINGLE or GLUT_DOUBLE.

Registering Callbacks

glutDisplayFunc (callback);

glutReshapeFunc (callback);

glutDisplayFunc (callback);

glutMotionFunc (callback);

glutPassiveMotionFunc (callback);

glutMouseFunc (callback);

glutKeyboardFunc (callback);

id = glutCreateMenu (callback);

glutMainLoop ( );

Display Lists

glNewList (number, GL_COMPILE);

glEndList ( );

glCallList (number);

glDeleteLists (number, 1);

Managing Menus

id = glutCreateMenu (callback);

glutDestroyMenu (id);

glutAddMenuEntry (label, number);

glutAttachMenu (button);

glutDetachMenu (button);

where button = GLUT_RIGHT_BUTTON or GLUT_LEFT_BUTTON.

Chapter-2
Design

Design

1. Aircraft Principle Axes

- **Normal axis**, or yaw axis — an axis drawn from top to bottom, and perpendicular to the other two axes.
- **Lateral axis**, transverse axis, or pitch axis — an axis running from the pilot's left to right in piloted aircraft, and parallel to the wings of a winged aircraft.
- **Longitudinal axis**, or roll axis — an axis drawn through the body of the vehicle from tail to nose in the normal direction of flight, or the direction the pilot faces.

Normally, these axes are represented by the letters X, Y and Z in order to compare them with some reference frame, usually named x, y, z. Normally, this is made in such a way that the X is used for the longitudinal axis, but there are other possibilities to do it.

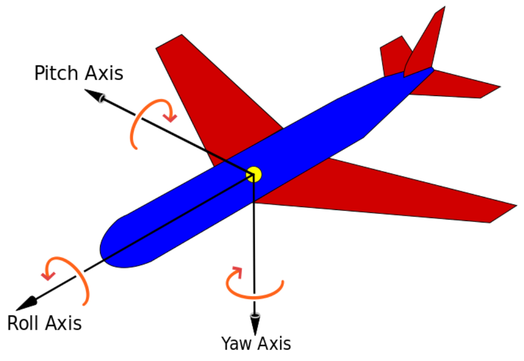


Fig. 2.1

1.1 Vertical axis (yaw)

The vertical **yaw axis** is defined to be perpendicular to the wings and to the normal line of flight with its origin at the center of gravity and directed towards the bottom of the aircraft. Yaw moves the nose of the aircraft from side to side. A positive yaw, or heading angle, moves the nose to the right. The rudder is the primary control of y

1.2 Lateral axis (pitch)

The **pitch axis** (also called **lateral** or **transverse axis**) passes through the plane from wingtip to wingtip. Pitch moves the aircraft's nose up and down. A positive pitch angle raises the nose and lowers the tail. The elevators are the primary control of pitch.

1.3 Longitudinal (roll)

The **roll axis** (or **longitudinal axis**) passes through the plane from nose to tail. The angular displacement about this axis is called **bank**. The pilot changes bank angle by increasing the lift on one wing and decreasing it on the other. A positive roll angle lifts the left wing and lowers the right wing. The ailerons are the primary control of bank. The rudder also has a secondary effect on bank.

2. Perspective projection

Perspective projection is a technique that is used to "paint" a scene in your 3D world onto a 2D surface such as a computer screen. It is responsible for making objects closer to you appear bigger than objects far away in the distance.

With OpenGL we are in the lucky position that a group of very smart computer scientists already took the time to work through all the complex mathematics used for perspective projection. All we need to do, is to set up the projection settings and OpenGL will automatically do the hard work for us.

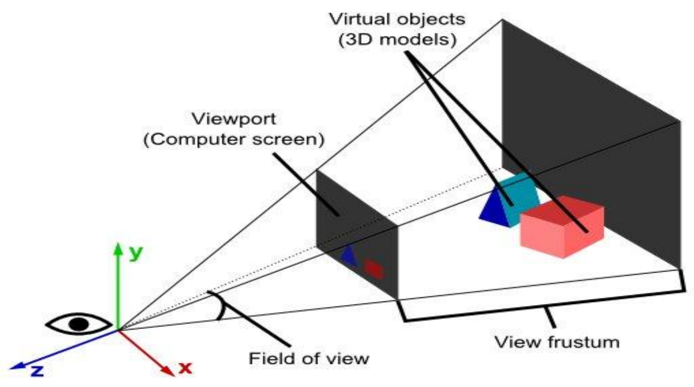


Fig. 2.2

The diagram above illustrates the idea behind perspective projection. As you can see, the virtual 3D models are mapped to a 2D surface such as your computer screen. Humans have an almost 180-degree forward-facing horizontal field of view. Some birds have a near-complete 360-degree field of view.

3. First Person Shooter (FPS)

A first person shooter (FPS) is a genre of action video game that is played from the point of view of the protagonist. FPS games typically map the gamer's movements and provide a view of what an actual person would see and do in the game. A FPS usually shows the protagonist's arms at the bottom of the screen, carrying whatever weapon is equipped. The gamer is expected to propel his avatar through the game by moving it forward, backward, sideways and so on using the game controller. Forward movements of the controller result in the avatar moving forward through the scenery, usually with a slight left-right rocking motion to properly simulate the human gait.

4. Explanation about the Package

This is a Graphics Package on 3D LANDSCAPE based on OPENGL API for the Computer Graphics and Visualization Laboratory (10CSL67). The project does a Walking Simulation, Drone Simulation and Aircraft Simulation in the Landscape. The project contains various objects such as Houses, Windmills, Trees, Clouds, Sun, Moon and an Aircraft with its Runway. The camera uses Perspective Viewing and the camera movement has been implemented in FPS (First Person Shooter) Style. The menus provided are to switch between Drone Mode, Walking Mode, and Aircraft Simulation. The Houses and Windmills can be added/removed with the help of menus. There is an option to switch between Day Mode and Night Mode and to start/stop the rotation of the Windmill. The lighting is simulated as the Sun is the source of light.

The starting screen of the package is the information window which displays the information about the title and author of the project. Pressing i will hide/show the information window. Once the simulation starts, pressing space bar will enable/disable the FPS movement.

The controls for the movement are:
w- Move Forward
a- Move Left
s- Move Backwards
d- Move Right
Mouse- Camera Rotation around a single point

The various objects are implemented as follows:

House

The basic structure of the house is implemented using the glut library function glutSolidCube().

The roof, windows and the door is implemented using GL_POLYGONS. The house is scaled, rotated and translated for placing it in different locations.

Windmill

The windmill consists of a pole which is implemented using glutSolidCone() and consists of a turbine which is implemented using glutSolidCube(). The blades are implemented using GL_POLYGONS. The pole, turbine and the blades are scaled and translated to form a windmill. The blades are rotated using translate and rotate functions.

Tree

The tree consists of a bark which is implemented using glutSolidCone(). The leaves of the tree are implemented using glutSolidCone() as well and translated in vertical direction to merge the cones one on top of another.

Clouds

The clouds are implemented using glutSolidSphere() and translated to get an effect of clouds. The clouds are repeated over the area using a for loop.

Sun & Moon

The sun and the moon are implemented using glutSolidSphere() and translated to its position.

Aircraft

The main body of the aircraft is implemented using gluSolidCylinder(). The tail is implemented using a glutSolidCone(). The front half sphere is done using glutSolidSphere() and merged into the cylinder body. The Main Wings, Tail Wing, and the fan are implemented using the GL_POLYGONS.

Runway

The Runway is implemented using glutSolidCube() with scaling and translation to for place and form the runway.

The concepts implemented in the Package are:

Translation

Rotation

Scaling

Perspective Viewing

Lighting

Keyboard Functions

Mouse Function

Chapter-3
Implementation

Source Code

```
/***********************************************************
************************************************************
****                Bangalore Institute of Technology                ****
****         Department of Computer Science & Engineering          ****
****                        VI Semester                             ****
****                      Graphics Package                          ****
****                 ************************                       ****
****                 **** 3D LANDSCAPE ****                         ****
****                 ************************                       ****
****                        Submitted By                            ****
****              1BI13CS063             HITESH RS                  ****
****                for the academic year 2015-16                   ****
************************************************************
************************************************************
*/
#include<stdlib.h>
#include<GL/glut.h>
#include<stdio.h>
#include<math.h>
static bool info_display = true;
static bool plane_move = true;
static bool drone_mode = true;
static bool walk_Mode = false;
static bool fly_Mode = false;
static bool windmill_rotate = true;
static bool dayLight = true;
static bool start_simulation = true;
static bool house1 = true, house2 = true, house3 = false, house4 = false, house5 = false;
static bool windmill1 = true, windmill2 = true, windmill3 = false, windmill4 = false, windmill5 = false;
static GLfloat angle = 0;
//FPS MODE Variables
bool g_key[256];
bool g_shift_down = false;
bool g_fps_mode = false;
int g_viewport_width = 0;
int g_viewport_height = 0;
bool g_mouse_left_down = false;
bool g_mouse_right_down = false;
// Movement settings
const float g_translation_speed = 200;
const float g_rotation_speed = (3.1416) / 180 * 0.2;
float m_x = 5000, m_y = 80, m_z = -4000;            // Position
float m_lx, m_ly, m_lz;                             // Direction vector of where we are looking at
float m_yaw = 20.0, m_pitch = 0.0;                  // Various rotation angles
float m_strafe_lx, m_strafe_lz;                     // Always 90 degree to direction vector
void Refresh(){
        // Camera parameter according to Riegl's co-ordinate system
        // x/y for flat, z for height
        m_lx = cos(m_yaw) * cos(m_pitch);
        m_ly = sin(m_pitch);
        m_lz = sin(m_yaw) * cos(m_pitch);
        m_strafe_lx = cos(m_yaw - (3.1416/2));
        m_strafe_lz = sin(m_yaw - (3.1416 / 2));
        glMatrixMode(GL_MODELVIEW);
        glLoadIdentity();
        gluLookAt(m_x, m_y, m_z, m_x + m_lx, m_y + m_ly, m_z + m_lz, 0.0, 1.0, 0.0);
}
void Move(float incr){
		float lx = cos(m_yaw)*cos(m_pitch);
		float ly = sin(m_pitch);
		float lz = sin(m_yaw)*cos(m_pitch);
		m_x = m_x + incr*lx;
		m_y = m_y + incr*ly;
		m_z = m_z + incr*lz;
		if (m_y <= 100){
			m_y = 100;
		}
		if (walk_Mode){
			if (m_y >= 150){
				m_y = 150;
			}
		}
		Refresh();
}
void Strafe(float incr){
		m_x = m_x + incr*m_strafe_lx;
		m_z = m_z + incr*m_strafe_lz;
		Refresh();
}
void Fly(float incr){
		m_y = m_y + incr;
		Refresh();
}
void RotateYaw(float angle){
		m_yaw = (m_yaw + angle);
		if (m_yaw > 90.0){
			m_yaw = (m_yaw + 90.0);
		}
		Refresh();
}
void RotatePitch(float angle){
		const float limit = 89.0 * (3.1416) / 180.0;
		m_pitch += angle;
		if (m_pitch < -limit)
			m_pitch = -limit;
		if (m_pitch > limit)
			m_pitch = limit;
		Refresh();
}
void drawHouse(GLfloat x, GLfloat y, GLfloat z, GLfloat R, GLfloat G, GLfloat B,GLfloat angle){
		GLfloat size = 50;
		GLfloat xscale = 3, yscale = 3, zscale = 5;
		glPushMatrix();
		glTranslatef(x, (yscale*size/2), z);
		glRotatef(angle , 0, 1, 0);
		glScalef(xscale, yscale, zscale);
		glColor3f(R/255.0, G/255.0, B/255.0);	//House Cube
		glutSolidCube(size);
		glColor3f(0.92, 0.082, 0.094);
		glBegin(GL_POLYGON);			//House Top Front Triangle
		glVertex3f(-(size / 2), (size / 2), (size / 2));
		glVertex3f((size / 2), (size / 2), (size / 2));
		glVertex3f(0, size, (size / 2));
		glEnd();
							//House Top Back Triangle
		glColor3f(0.92, 0.082, 0.094);
		glBegin(GL_POLYGON);			//House Top left side panel
		glVertex3f(-(size / 2), (size / 2), -(size / 2));
		glVertex3f(0, size, -(size / 2));
```

```
            glVertex3f(0, size, (size / 2));
            glVertex3f(-(size / 2), (size / 2), (size / 2));
            glEnd();
                                                                        //House Top right side panel
            glColor3f(0.47, 0.43, 0.06);
            glBegin(GL_POLYGON);                                        //Door
            glVertex3f((size / 4), -(size / 2), (size / 2)+0.2);
            glVertex3f((size / 4), 0, (size / 2) + 0.2);
            glVertex3f(-(size / 4), 0, (size / 2) + 0.2);
            glVertex3f(-(size / 4), -(size / 2), (size / 2) + 0.2);
            glEnd();
            glColor3f(1, 1, 0);
            glBegin(GL_POLYGON);                                        //Left Front Windows
            glVertex3f(-(size / 2) - 0.2, -(size / 4), (size / 8));
            glVertex3f(-(size / 2) - 0.2, -(size / 4), (size / 4));
            glVertex3f(-(size / 2) - 0.2, (size / 4),  (size / 4));
            glVertex3f(-(size / 2) - 0.2, (size / 4),  (size / 8));
            glEnd();
                                                                        //Left Back Windows
                                                                        //Right Front Windows
                                                                        //Right Back Windows
            glPopMatrix();
}
GLUquadricObj *Cylinder = gluNewQuadric();
void drawPlane(){
            GLfloat size = 20;
            GLfloat xscale = 3, yscale = 3, zscale = 3;
            GLfloat x, y, z;
            x = 0; y=0; z=8000;
            glPushMatrix();
            glRotatef(-90, 0, 1, 0);
            glTranslatef(x, y + (yscale*size) + (yscale * 62), z);
            glRotatef(9, 1, 0, 0);
            glScalef(xscale, yscale, zscale);
            glColor3f(83.0 / 255.0, 135.0 / 255.0, 5.0 / 255.0);        //Yellow
            gluCylinder(Cylinder, size, size, size * 10, 10, 10);       //Cylinder
            glutSolidSphere(size, 10, 10);                              //Sphere
            glTranslatef(0, 0, size*10);
            glutSolidCone(size, size * 7, 10, 10);                      //Cone
            glTranslatef(0, 0, -size * 10);
            glTranslatef(0, 0, size * 3);
            glBegin(GL_POLYGON);                                        //Front Wings Layer 1
            glVertex3f(0, size / 2, 0);
            glVertex3f(size * 10, size / 2, size*1.5);
            glVertex3f(size * 10, size / 2, size*3);
            glVertex3f(0, size / 2, size * 3);
            glVertex3f(-size * 10, size / 2, size*3);
            glVertex3f(-size * 10, size / 2, size*1.5);
            glVertex3f(0, size / 2, 0);
            glEnd();
                                                                        //Front Wings Layer 2
            glColor3f(0.0 / 255.0, 19.0 / 255.0, 161.0 / 255.0);        //Blue
            glBegin(GL_POLYGON);                                        //Front Wings thickness Fill
            glVertex3f(size * 10, size / 2 + 5, size*1.5);
            glVertex3f(-size * 10, size / 2 + 5, size*1.5);
            glVertex3f(-size * 10, size / 2 , size*1.5);
            glVertex3f(size * 10, size / 2 , size*1.5);
            glEnd();
            //Contd..

            glColor3f(83.0 / 255.0, 135.0 / 255.0, 5.0 / 255.0);        //Yellow
```

```
glTranslatef(0, 0, size * 11);
glScalef(0.5, 0.5, 0.5);
glBegin(GL_POLYGON);                                    //Back Wings Layer 1
glVertex3f(0, size / 2, 0);
glVertex3f(size * 10, size / 2, size*1.5);
glVertex3f(size * 10, size / 2, size * 3);
glVertex3f(0, size / 2, size * 3);
glVertex3f(-size * 10, size / 2, size * 3);
glVertex3f(-size * 10, size / 2, size*1.5);
glVertex3f(0, size / 2, 0);
glEnd();
                                                        //Back Wings Layer 2
glColor3f(0.0 / 255.0, 19.0 / 255.0, 161.0 / 255.0);    //Blue
glBegin(GL_POLYGON);                                    //Back Wings thickness Fill
glVertex3f(size * 10, size / 2 + 5, size*1.5);
glVertex3f(-size * 10, size / 2 + 5, size*1.5);
glVertex3f(-size * 10, size / 2, size*1.5);
glVertex3f(size * 10, size / 2, size*1.5);
glEnd();
//Contd..

glColor3f(83.0 / 255.0, 135.0 / 255.0, 5.0 / 255.0);    //Yellow
glTranslatef(0, 0, size*3);
glBegin(GL_POLYGON);                                    //Back Top Wing layer 1
glVertex3f(size / 4, size / 2, 0);
glVertex3f(size / 4, size / 2, -size*3);
glVertex3f(size / 4, size *5, -size * 2);
glVertex3f(size / 4, size *5, 0);
glEnd();
                                                        //Back Top Wing layer 2
glColor3f(0.0 / 255.0, 19.0 / 255.0, 161.0 / 255.0);    //Blue
glBegin(GL_POLYGON);                                    //Back Top Wing Thickness Fill
glVertex3f(size / 4, size / 2, 0);
glVertex3f(size / 4, size *5, 0);
glVertex3f(-size / 4, size *5, 0);
glVertex3f(-size / 4, size / 2, 0);
glEnd();
//Contd..
glColor3f(31.0 / 255.0, 26.0 / 255.0, 26.0 / 255.0);    //Brown
glTranslatef(0, 0, -size * 33);
glRotatef(80, 0, 0, 1);
glutSolidSphere(size /2, 10, 10);                       //Fan Centre
glColor3f(199.0 / 255.0, 10.0 / 255.0, 10.0/255.0);     //Pink
glBegin(GL_POLYGON);                                    //Fan Wings Layer 1
glVertex3f(0, size / 2, 0);
glVertex3f(size / 2, size * 4, 0);
glVertex3f(0, size * 6, 0);
glVertex3f(-size / 2, size * 4, 0);
glVertex3f(size / 2, -size * 4, 0);
glVertex3f(0, -size * 6, 0);
glVertex3f(-size / 2, -size * 4, 0);
glVertex3f(0, size / 2, 0);
glEnd();
                                                        //Fan Wings Layer 2
glColor3f(31.0 / 255.0, 26.0 / 255.0, 26.0 / 255.0);    //Brown
glTranslatef(0, 0, size * 20);
glRotatef(90, 1, 0, 0);
gluCylinder(Cylinder, size / 10, size / 10, size * 5, 10, 10);  //Back Leg
glRotatef(-90, 1, 0, 0);
glTranslatef(0, -size*5, 0);
glutSolidSphere(size / 2, 10, 10);                      //Back Wheel
```

```
		glTranslatef(0, size * 5, 0);

		glColor3f(31.0 / 255.0, 26.0 / 255.0, 26.0 / 255.0);				//Brown
		glTranslatef(0, 0, -size * 15);
		glRotatef(90, 0, 1, 0);
		glRotatef(60, 1, 0, 0);
		gluCylinder(Cylinder, size / 10, size / 10, size * 8, 10, 10);/ /Right Leg
		glRotatef(-90, 1, 0, 0);
		glTranslatef(0, -size * 8, 0);
		glutSolidSphere(size / 2, 10, 10);						//Right Wheel
		glTranslatef(0, size * 8, 0);
		glRotatef(90, 1, 0, 0);
		glColor3f(31.0 / 255.0, 26.0 / 255.0, 26.0 / 255.0);				//Brown
		glRotatef(-120, 1, 0, 0);
		glRotatef(180, 0, 1, 0);
		gluCylinder(Cylinder, size / 10, size / 10, size * 8, 10, 10);	//Left Leg
		glRotatef(-90, 1, 0, 0);
		glTranslatef(0, -size * 8, 0);
		glutSolidSphere(size / 2, 10, 10);						//Left Wheel
		glTranslatef(0, size * 8, 0);
		glRotatef(90, 1, 0, 0);
		glPopMatrix();
}

void drawSun(){
		GLfloat size = 200;
		GLfloat xscale = 3, yscale = 3, zscale = 3;
		GLfloat x, y, z;
		x = 5000; y = 8000; z = 5000;
		glPushMatrix();
		glTranslatef(x, y + (yscale*size), z);	//Sun
		glScalef(xscale, yscale, zscale);
		glColor3f(1,1,0);
		glutSolidSphere(size, 20, 20);
		glPopMatrix();
}
void drawMoon() {…}
void drawTree(GLfloat x, GLfloat y, GLfloat z){
		GLfloat size = 7;
		GLfloat xscale = 3, yscale = 10, zscale = 3;
		glPushMatrix();
		glTranslatef(x, (yscale*size / 8), z);
		glScalef(xscale, yscale, zscale);
		glRotatef(90 * 3, 1, 0, 0);
		glColor3f(181.0/255.0, 75.0/255.0, 0);
		glutSolidCone(size, size * 5, 1000, 1000);			//Tree Pole
		glPopMatrix();
		glPushMatrix();
		glTranslatef(x, (yscale*size), z);
		glScalef(xscale, yscale, zscale);
		glRotatef(90 * 3, 1, 0, 0);
		glColor3f(26.0 / 255.0, 161.0 / 255.0, 13.0/255.0);
		glutSolidCone(size*6, size * 3, 8, 8);				//Tree Leaves Base 1
		glPopMatrix();
										//Tree Leaves Base 2
										//Tree Leaves Base 3
										//Tree Leaves Base 4
										//Tree Leaves Base 5
}
void drawSingleBlade(GLfloat size){
		glBegin(GL_POLYGON);
```

```
	glVertex3f(-(size / 2), 0, (size / 8));
	glVertex3f((size / 2), 0, (size / 8));
	glVertex3f((size / 4), (size*10), (size / 8));
	glVertex3f(-(size / 4), (size * 10), (size / 8));
	glEnd();
	glBegin(GL_POLYGON);
	glVertex3f(-(size / 2), 0, (size / 4));
	glVertex3f((size / 2), 0, (size / 4));
	glVertex3f((size / 4), (size * 10), (size / 4));
	glVertex3f(-(size / 4), (size * 10), (size / 4));
	glEnd();
	glBegin(GL_POLYGON);
	glVertex3f(-(size / 2), 0, (size / 8));
	glVertex3f(-(size / 2), 0, (size / 4));
	glVertex3f((size / 2), 0, (size / 8));
	glVertex3f((size / 2), 0, (size / 4));
	glVertex3f((size / 4), (size * 10), (size / 8));
	glVertex3f((size / 4), (size * 10), (size / 4));
	glVertex3f(-(size / 4), (size * 10), (size / 8));
	glVertex3f(-(size / 4), (size * 10), (size / 4));
	glEnd();
}
void drawThreeBlades(GLfloat x, GLfloat y, GLfloat z, GLfloat size, GLfloat xscale, GLfloat yscale, GLfloat zscale){
	glPushMatrix();
	glScalef(xscale, yscale / (xscale - 1), zscale * 2);
	drawSingleBlade(size);
	glRotatef(120, 0, 0, 1);
	drawSingleBlade(size);
	glRotatef(120, 0, 0, 1);
	drawSingleBlade(size);
	glRotatef(120, 0, 0, 1);
	glPopMatrix();
}
void drawWindMill(GLfloat x, GLfloat y, GLfloat z){
	GLfloat size = 5;
	GLfloat xscale = 3, yscale = 10, zscale = 3;
	glPushMatrix();
	glTranslatef(x, (yscale*size / 8), z);
	glScalef(xscale, yscale, zscale);
	glRotatef(90*3, 1, 0, 0);
	glColor3f(0.823, 0.819, 0.89);
	glutSolidCone(size, size*15, 10, 10);			//WindMill Pole
	glPopMatrix();
	glPushMatrix();
	glTranslatef(x, (yscale*size/2)+(size*130), z);
	glScalef(xscale, yscale/(xscale-1), zscale*2);
	glutSolidCube(size * 2);				//Windmill Machine
	glPopMatrix();
	glPushMatrix();					//Windmill Blades
	glTranslatef(x, (yscale*size / 2) + (size * 130), z+size);
	if (windmill_rotate){
		glRotatef(angle, 0, 0, 1);
	}
	drawThreeBlades(x, y, z, size, xscale, yscale, zscale);
	glPopMatrix();
}
void drawClouds(){
	GLfloat size = 10;
	GLfloat xscale = 5, yscale = 5, zscale = 10;
	GLfloat x, y, z;
	x = 0; y=0; z=0;
```

```
		for (z = 1; z <= 1000; z += 500){
			glPushMatrix();
			glEnable(GL_BLEND);
			glBlendFunc(GL_SRC_ALPHA, GL_ONE_MINUS_SRC_ALPHA);
			glAlphaFunc(GL_GREATER, 0.0 / 255.0);
			glEnable(GL_CULL_FACE);
			glCullFace(GL_FRONT);
			glColor4ub(255, 255, 255, 200);
			glTranslatef(x * 10, y + (yscale*size) + 3000, z * 10);			//Cloud 1
			glScalef(xscale, yscale, zscale);
			glutSolidSphere(size, 10, 10);
			glTranslatef(5, -4, -3);
			glutSolidSphere(size*0.8, 10, 10);
			glTranslatef(8, -4, 6);
			glutSolidSphere(size * 1, 10, 10);
			glTranslatef(10, 4, 2);
			glutSolidSphere(size*1.5, 10, 10);
			glTranslatef(5, -7, 5);
			glutSolidSphere(size*0.8, 10, 10);
			glTranslatef(5, -7, 5);
			glutSolidSphere(size*1.3, 10, 10);
			glPopMatrix();
												//Cloud 2
												//Cloud 3
												//Cloud 4
												//Cloud 5
												//Cloud 6
												//Cloud 7
												//Cloud 8
												//Cloud 9
		}
}
void drawRunway(){
		GLfloat size = 10;
		GLfloat xscale = 500, yscale = 1, zscale = 150;
		GLfloat x, y, z;
		x = -8000; y=1; z=0;
		float i;
		glPushMatrix();
		glTranslatef(x, 2, z);
		glScalef(xscale, yscale, zscale);
		glColor3f(8.0 / 255.0, 8.0 / 255.0, 10.0 / 255.0);
		glutSolidCube(size);
		glColor3f(1, 1, 1);
		glTranslatef(-5.5, 10, 0);
		for (i = 0.0; i < 10.0; i += 1.0){
			glTranslatef(1, 0, 0);
			glutSolidCube(size / 20);
		}
		glPushMatrix();
		glTranslatef(-4.5, 0, 4.5);
		glScalef(20, yscale, 1);
		glutSolidCube(size / 20);
		glTranslatef(0, 0, -9);
		glutSolidCube(size / 20);
		glPopMatrix();
		glPopMatrix();
}
void drawPlaneWindow(){
		glPushMatrix();
		glLoadIdentity();
```

```
        glMatrixMode(GL_PROJECTION);
        glPushMatrix();
        glLoadIdentity();
        gluOrtho2D(0, 1000, 0, 600);
        glDepthFunc(GL_ALWAYS);
        glColor3f(40.0 / 255.0, 40.0 / 255.0, 48.0 / 255.0);
        glBegin(GL_POLYGON);            //Window Divider
        glVertex2f(450, 600);
        glVertex2f(480, 0);
        glVertex2f(520, 0);
        glVertex2f(550, 600);
        glEnd();
                                        //Top Window Panel
                                        //Bottom Window Panel
                                        //Left Window Panel
                                        //Right Window Panel
        glPushMatrix();
        glEnable(GL_BLEND);
        glBlendFunc(GL_SRC_ALPHA, GL_ONE_MINUS_SRC_ALPHA);
        glAlphaFunc(GL_GREATER, 0.0 / 255.0);
        glEnable(GL_CULL_FACE);
        glCullFace(GL_FRONT);
        glColor4ub(149, 151, 207, 50);
        glBegin(GL_POLYGON);            //Translucent Glass shade
        glVertex2f(0, 0);
        glVertex2f(0, 600);
        glVertex2f(1000, 600);
        glVertex2f(1000, 0);
        glEnd();
        glDisable(GL_BLEND);
        glDisable(GL_CULL_FACE);
        glPopMatrix();
        glDepthFunc(GL_LESS);
        glPopMatrix();
        glMatrixMode(GL_MODELVIEW);
        glPopMatrix();
        glutPostRedisplay();
}
void drawText(GLfloat x, GLfloat y, GLfloat z, char *str){
        int i;
        glColor3f(1, 0, 0);
        glPushMatrix();
        glLoadIdentity();
        glMatrixMode(GL_PROJECTION);
        glPushMatrix();
        glLoadIdentity();
        gluOrtho2D(0, 1000, 0, 600);
        if (info_display){
                glColor3f(1, 1, 0);
                glBegin(GL_POLYGON);
                glVertex2f(0, 0);
                glVertex2f(1000, 0);
                glVertex2f(1000, 600);
                glVertex2f(0, 600);
                glEnd();
        }
        glDepthFunc(GL_ALWAYS);
        glColor3f(1, 0, 0);
        glRasterPos2f(x, y);
        for (i = 0; str[i] != '\0'; ++i)
                glutBitmapCharacter(GLUT_BITMAP_HELVETICA_18, str[i]);
        glDepthFunc(GL_LESS);
        glPopMatrix();
        glMatrixMode(GL_MODELVIEW);
        glPopMatrix();

}
void drawStrokeText(GLfloat x, GLfloat y, GLfloat sx, GLfloat sy, char str[], GLfloat width, GLubyte R, GLubyte G,
GLubyte B){
        int i;
        glPushMatrix();
        glLoadIdentity();
        glMatrixMode(GL_PROJECTION);
        glPushMatrix();
        glLoadIdentity();
        gluOrtho2D(0, 1000, 0, 600);
        if (info_display){
                glColor3f(0, 0, 0);
                glBegin(GL_POLYGON);
                glVertex2f(0, 0);
                glVertex2f(1000, 0);
                glVertex2f(1000, 600);
                glVertex2f(0, 600);
                glEnd();
        }
        glDepthFunc(GL_ALWAYS);
        glColor3ub(R, G, B);
        glLineWidth(width);
        glPushMatrix();
        glTranslatef(x, y, 0);
        glScalef(sx, sy, 0);
        for (i = 0; str[i] != '\0'; ++i)
                glutStrokeCharacter(GLUT_STROKE_ROMAN, str[i]);
        glPopMatrix();
        glDepthFunc(GL_LESS);
        glPopMatrix();
        glMatrixMode(GL_MODELVIEW);
        glPopMatrix();

}
void spin(){
        angle = (int)(angle + 20) % 360;
        glutPostRedisplay();
}
void keyboard(unsigned char key, int x, int y){
        if (key == 27)exit(0);
        if (key == 'i'){
                info_display = !info_display;
                g_fps_mode = false;
        }
        if (key == ' ') {
                plane_move = !plane_move;
                g_fps_mode = !g_fps_mode;
                if (info_display){
                        g_fps_mode = false;
                }
                if (g_fps_mode) {
                        glutSetCursor(GLUT_CURSOR_NONE);
                        glutWarpPointer(g_viewport_width / 2, g_viewport_height / 2);
                }
                else {
                        glutSetCursor(GLUT_CURSOR_LEFT_ARROW);
                }
        }
        if (glutGetModifiers() == GLUT_ACTIVE_SHIFT) {
                g_shift_down = true;
        }
        else {
                g_shift_down = false;
        }
        g_key[key] = true;
        glutPostRedisplay();
}
void KeyboardUp(unsigned char key, int x, int y){
        g_key[key] = false;
}
void Timer(int value){
        if (g_fps_mode) {
                if (g_key['w'] || g_key['W']) {
                        Move(g_translation_speed);
                }
                else if (g_key['s'] || g_key['S']) {
                        Move(-g_translation_speed);
                }
                else if (g_key['a'] || g_key['A']) {
                        Strafe(g_translation_speed);
                }
                else if (g_key['d'] || g_key['D']) {
                        Strafe(-g_translation_speed);
                }
                else if (g_mouse_left_down) {
                        Fly(-g_translation_speed);
                }
                else if (g_mouse_right_down) {
                        Fly(g_translation_speed);
                }
        }
        glutTimerFunc(1, Timer, 0);
}
void mouse(int btn, int state, int x, int y){
        if (state == GLUT_DOWN) {
                if (btn == GLUT_LEFT_BUTTON){
                        g_mouse_left_down = true;
                }
                else if (btn == GLUT_RIGHT_BUTTON){
                        g_mouse_right_down = true;
                }
        }
        else if (state == GLUT_UP) {
                if (btn == GLUT_LEFT_BUTTON){
                        g_mouse_left_down = false;
                }
                else if (btn == GLUT_RIGHT_BUTTON){
                        g_mouse_right_down = false;
                }
        }
        glutPostRedisplay();
}
void MouseMotion(int x, int y){
// This variable is hack to stop glutWarpPointer from triggering an event callback to Mouse(...)
// This avoids it being called recursively and hanging up the event loop
        static bool just_warped = false;
        if (just_warped) {
                just_warped = false;
                return;
        }
        if (g_fps_mode) {
                int dx = x - g_viewport_width / 2;
                int dy = y - g_viewport_height / 2;
                if (dx) {
                        RotateYaw(g_rotation_speed*dx);
                }
                if (dy) {
                        RotatePitch(g_rotation_speed*dy);
                }
                glutWarpPointer(g_viewport_width / 2, g_viewport_height / 2);
                just_warped = true;
        }
}
void reshape(int w, int h){
        g_viewport_width = w;
        g_viewport_height = h;
        glViewport(0, 0, w, h);
        glMatrixMode(GL_PROJECTION);
        glLoadIdentity();
        gluPerspective(70, (GLfloat)w/(GLfloat)h, 50, 50000);
        glMatrixMode(GL_MODELVIEW);
}
void lighting(){
        GLfloat mat_ambient[] = { 2, 2, 0.8, 1 };
        GLfloat mat_diffuse[] = { 0.8, 0.8, 0.8, 1.0 };
        GLfloat mat_specular[] = { 1.0, 1.0, 1.0, 1.0 };
        GLfloat mat_shininess[] = { 50.0 };
        GLfloat lightintensity[] = { 1, 1, 0.7, 1.0 };
        GLfloat lightPosition[] = { 3900, 8900, 3900, 0 };
        glColorMaterial(GL_FRONT_AND_BACK, GL_AMBIENT_AND_DIFFUSE);
        glLightfv(GL_LIGHT0, GL_AMBIENT, mat_ambient);
        glLightfv(GL_LIGHT0, GL_POSITION, lightintensity);
        glMaterialfv(GL_FRONT, GL_AMBIENT, mat_ambient);
        glMaterialfv(GL_FRONT, GL_DIFFUSE, mat_diffuse);
        glMaterialfv(GL_FRONT, GL_SPECULAR, mat_specular);
        glMaterialfv(GL_FRONT, GL_SHININESS, mat_shininess);
        glLightfv(GL_LIGHT0, GL_POSITION, lightPosition);
        glLightfv(GL_LIGHT0, GL_DIFFUSE, lightintensity);
}
void DroneMode(){
        drone_mode = true;
        walk_Mode = false;
        fly_Mode = false;
        m_x = 0, m_y = 80, m_z = 5000;
        m_yaw = 0.0, m_pitch = 0.0;
}
void WalkMode(){
        walk_Mode = true;
        fly_Mode = false;
        drone_mode = false;
        m_x = 0, m_y = 80, m_z = 5000;
        m_yaw = 0.0, m_pitch = 0.0;
}
void FlyMode(){
        fly_Mode = true;
        walk_Mode = false;
        drone_mode = false;
        plane_move = true;
```

```
	m_x = -10000, m_y = 80, m_z = 0;
	m_yaw = 0.0, m_pitch = 0.0;
}
void House(){
	//drawHouse(x, y, z, R, G, B, angle)
	if (house1) drawHouse(300, 0, -250, 9, 109, 143, 30);		//Sky Blue
	if (house2) drawHouse(8000, 0, 5500, 41, 21, 214, 60);		//Blue
	if (house3) drawHouse(1000, 0, 2000, 45, 134, 181, -45);	//Blue
	if (house4) drawHouse(-300, 0, 2500, 196, 123, 27, 120);	//Yellow
	if (house5) drawHouse(6000, 0, 4500, 60, 110, 0, 150);		//Green
}
void WindMill(){
	//drawWindMill(x, y, z)
	if (windmill1) drawWindMill(2000, 0, -3000);
	if (windmill2) drawWindMill(5000, 0, -5000);
	if (windmill3) drawWindMill(3000, 0, -4000);
	if (windmill4) drawWindMill(4000, 0, -1000);
	if (windmill5) drawWindMill(3000, 0, -2000);
}
void SimulationMenu(int item){
	switch (item){
	case 7: DroneMode(); break;
	case 1: WalkMode(); break;
	case 2: FlyMode(); break;
	case 3: dayLight = true; break;
	case 4:	dayLight = false; break;
	case 5: windmill_rotate = !windmill_rotate; break;
	case 6: exit(0);
	default:break;
	}
}
void AddMenu(int item){
	switch (item){
	case 1: house1 = !house1; break;
	case 2: house2 = !house2; break;
	case 3: house3 = !house3; break;
	case 4: house4 = !house4; break;
	case 5: house5 = !house5; break;
	case 6: windmill1 = !windmill1; break;
	case 7: windmill2 = !windmill2; break;
	case 8: windmill3 = !windmill3; break;
	case 9: windmill4 = !windmill4; break;
	case 10: windmill5 = !windmill5; break;
	default:break;

	}
}
void Menu(){
	int add_menu;
	add_menu = glutCreateMenu(AddMenu);
	glutAddMenuEntry("Add/Remove House 1", 1);
	glutAddMenuEntry("Add/Remove House 2", 2);
	glutAddMenuEntry("Add/Remove House 3", 3);
	glutAddMenuEntry("Add/Remove House 4", 4);
	glutAddMenuEntry("Add/Remove House 5", 5);
	glutAddMenuEntry("Add/Remove WindMill 1", 6);
	glutAddMenuEntry("Add/Remove WindMill 2", 7);
	glutAddMenuEntry("Add/Remove WindMill 3", 8);
	glutAddMenuEntry("Add/Remove WindMill 4", 9);
	glutAddMenuEntry("Add/Remove WindMill 5", 10);
	glutCreateMenu(SimulationMenu);
```

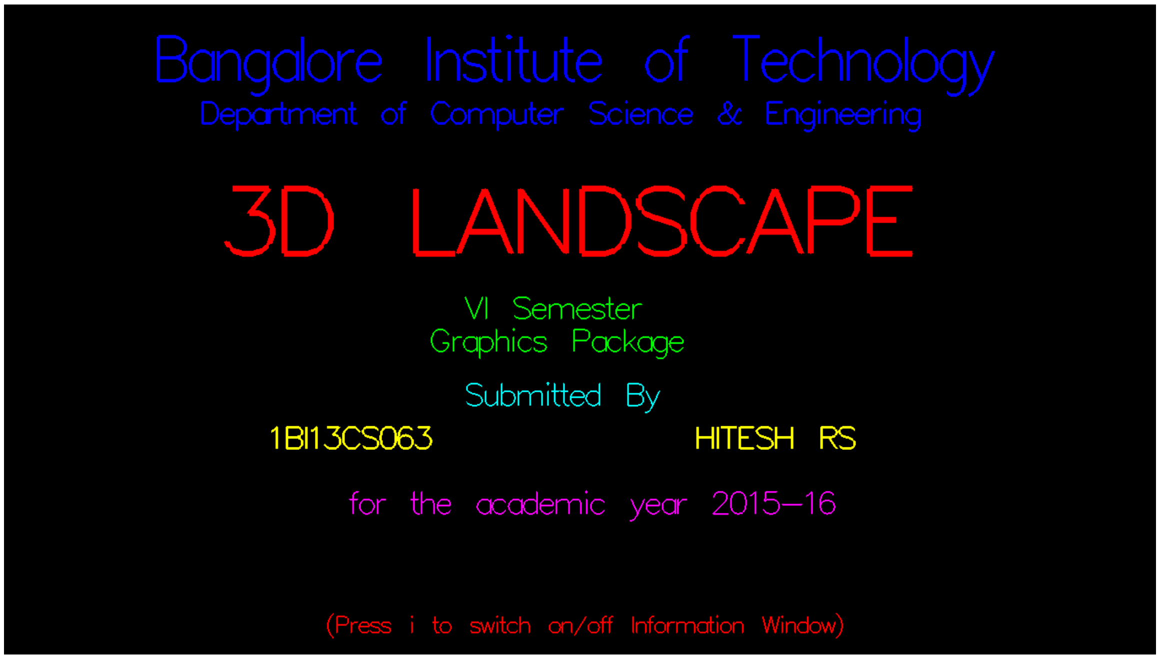
```
        glutAddMenuEntry("Drone Mode", 7);
        glutAddMenuEntry("Walking Simulation ", 1);
        glutAddMenuEntry("Aircraft Simulation", 2);
        glutAddMenuEntry("Day Mode", 3);
        glutAddMenuEntry("Night Mode", 4);
        glutAddMenuEntry("Start/Stop WindMill", 5);
        glutAddSubMenu("Add/Remove Objects", add_menu);
        glutAddMenuEntry("Exit        (ESC)", 6);
        glutAttachMenu(GLUT_RIGHT_BUTTON);

}
void displayInfo(){
        //drawStrokeText(x, y, sx, sy, str[], width, R, G, B);
        drawStrokeText(130, 530, 0.35, 0.4, "Bangalore Institute of Technology", 4, 0, 0, 255);
        drawStrokeText(170, 490, 0.2, 0.2, "Department of Computer Science & Engineering", 2.5, 0, 0, 255);
        drawStrokeText(190, 370, 0.6, 0.6, "3D LANDSCAPE", 20, 255, 0, 0);
        drawStrokeText(400, 310, 0.2, 0.2, "VI Semester", 2.2, 0, 255, 0);
        drawStrokeText(370, 280, 0.2, 0.2, "Graphics Package", 2.2, 0, 255, 0);
        drawStrokeText(400, 230, 0.2, 0.2, "Submitted By", 2.2, 0, 255, 255);
        drawStrokeText(230, 190, 0.2, 0.2, "1BI13CS063", 2.5, 255, 255, 0);
        drawStrokeText(600, 190, 0.2, 0.2, "HITESH RS", 2.5, 255, 255, 0);
        drawStrokeText(300, 130, 0.2, 0.2, "for the academic year 2015-16", 2.2, 255, 0, 255);
        drawStrokeText(280, 20, 0.15, 0.15, "(Press i to switch on/off Information Window)", 2, 255, 0, 0);
}
void display(){
        char str[50];
        glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
        if (dayLight){
                glClearColor(150.0 / 255.0, 201.0 / 255.0, 242.0 / 255.0, 1);
        }
        else{
                glClearColor(29.0 / 255.0, 63.0 / 255.0, 82.0 / 255.0, 1);
        }
        glLoadIdentity();
        lighting();
        Refresh();
        if (info_display){
                displayInfo();
        }
        glColor3f(2.0 / 255.0, 64.0 / 255.0, 0.0 / 255.0);                    //World Base
        glBegin(GL_POLYGON);
        glVertex3f(-10000, 0, -10000);
        glVertex3f(-10000, 0, 10000);
        glVertex3f(10000, 0, 10000);
        glVertex3f(10000, 0, -10000);
        glEnd();
        if (dayLight){
                drawSun();
        }
        else{
                drawMoon();
        }
        House();
        WindMill();
        //drawTree(x, y, z);
        drawTree(5000, 0, 5000);
        drawTree(4000, 0, 3000);
        drawTree(4000, 0, 2000);
        drawTree(3000, 0, 4000);
        drawClouds();
        drawRunway();
        if (!fly_Mode){
                drawPlane();
        }
        if (fly_Mode){
                drawPlaneWindow();
                g_fps_mode = true;
                glutSetCursor(GLUT_CURSOR_NONE);
                if (plane_move){
                        Move(g_translation_speed);
                }

        }
        if (!info_display){
                if (!g_fps_mode){
                        drawText(10, 580, 0, "Press SPACEBAR to start/stop movement");
                }
                else{
                        if (drone_mode) drawText(10, 580, 0, "DRONE MODE: ON");
                        if (walk_Mode) drawText(10, 580, 0, "WALK MODE: ON");
                        if (fly_Mode) drawText(10, 580, 0, "AIRPLANE MODE: ON");
                        drawText(10, 560, 0, "w: Forward");
                        drawText(10, 540, 0, "a: Left");
                        drawText(10, 520, 0, "d: Right");
                        drawText(10, 500, 0, "s:Backward");
                        drawText(10, 480, 0, "Mouse: Rotation");
                }
                if (fly_Mode){
                        sprintf(str, "Altitude: %f", m_y);
                        drawText(440, 540, 0, str);
                }
        }
        glutSwapBuffers();
}
int main(int argc, char **argv){
        glutInit(&argc, argv);
        glutInitDisplayMode(GLUT_DOUBLE | GLUT_RGB | GLUT_DEPTH);
        glutInitWindowSize(1000, 600);
        glutCreateWindow("3D Landscape");
        glutDisplayFunc(display);
        Menu();
        glutReshapeFunc(reshape);
        glutKeyboardFunc(keyboard);
        glutMotionFunc(MouseMotion);
        glutPassiveMotionFunc(MouseMotion);
        glutKeyboardUpFunc(KeyboardUp);;
        glutTimerFunc(1, Timer, 0);
        glutIdleFunc(spin);
        glutFullScreen();
        glEnable(GL_LIGHTING);
        glEnable(GL_LIGHT0);
        glShadeModel(GL_SMOOTH);
        glEnable(GL_NORMALIZE);
        glEnable(GL_COLOR_MATERIAL);
        glEnable(GL_DEPTH_TEST);
        glutMainLoop();
        return 0;
}
```

Chapter-4
Results

Result



Bangalore Institute of Technology
Department of Computer Science & Engineering

3D LANDSCAPE

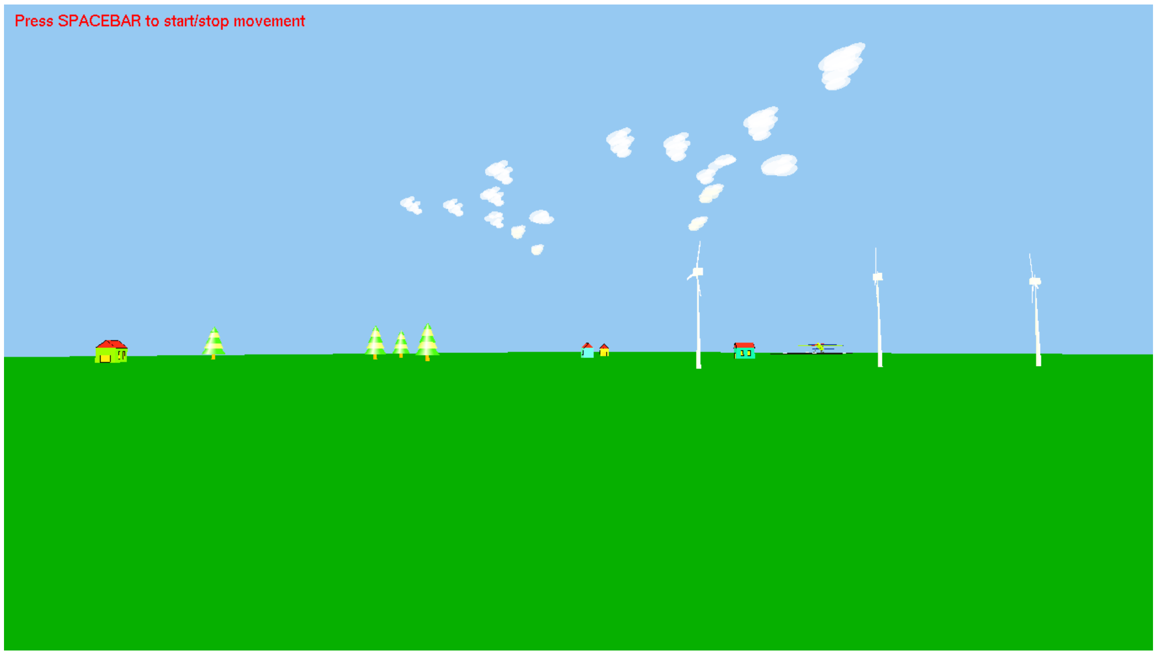
VI Semester
Graphics Package

Submitted By
1BI13CS063 HITESH RS

for the academic year 2015–16

(Press i to switch on/off Information Window)

Fig. 4.1 Information Window



Press SPACEBAR to start/stop movement

Fig. 4.2 Starting View

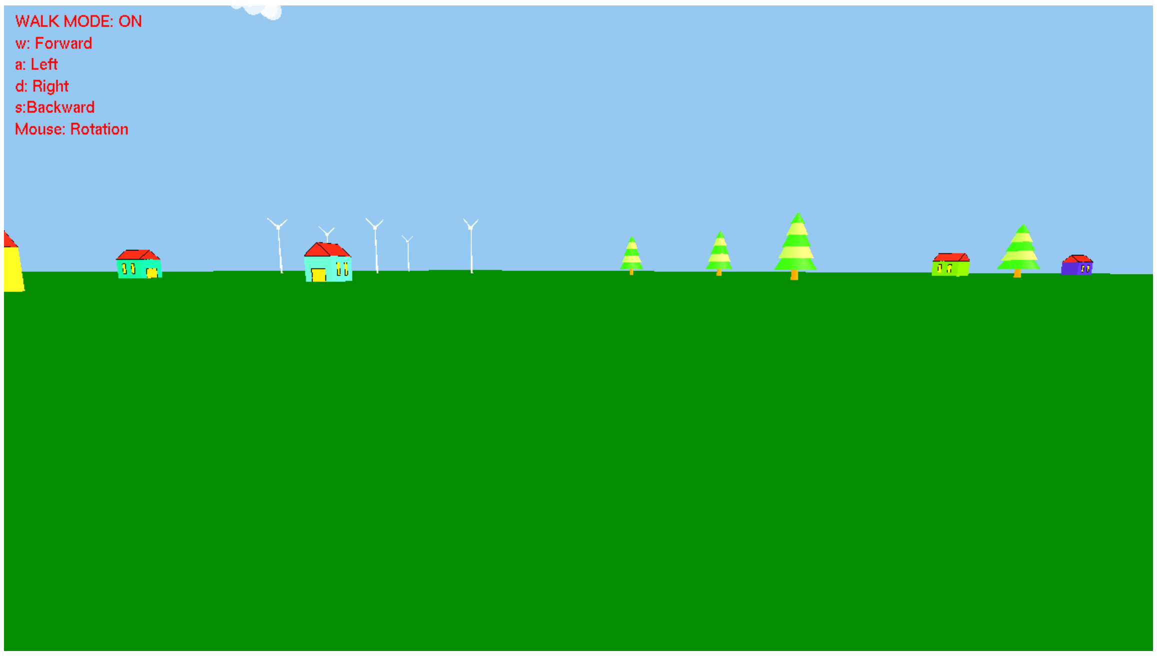


Fig. 4.3 Walk Mode

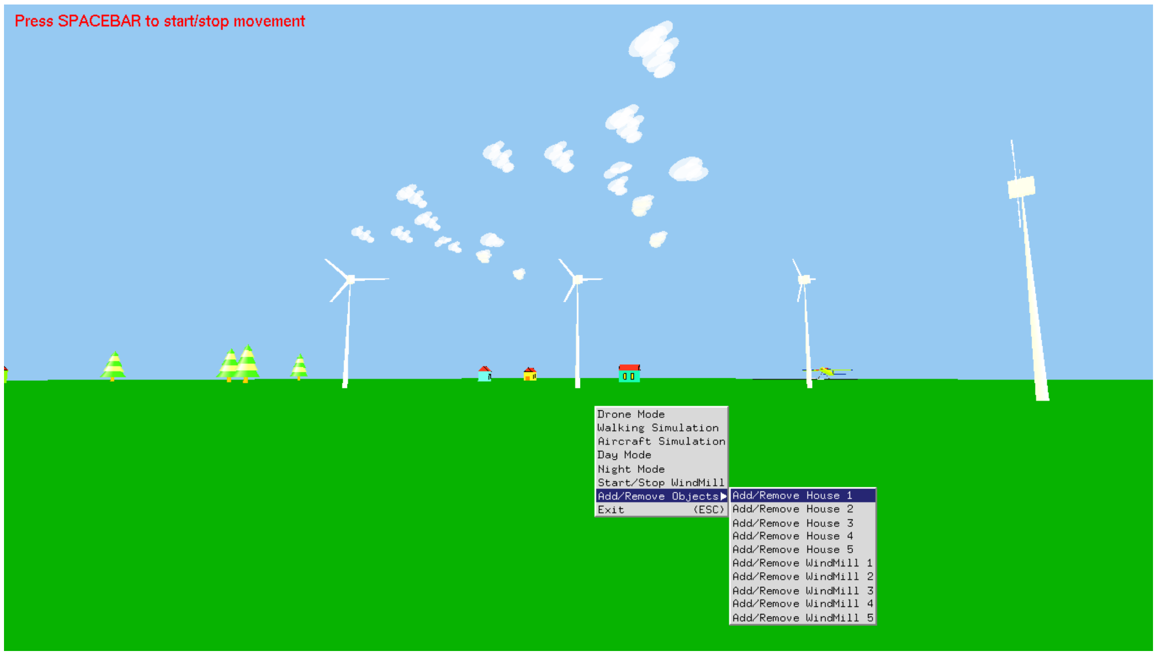
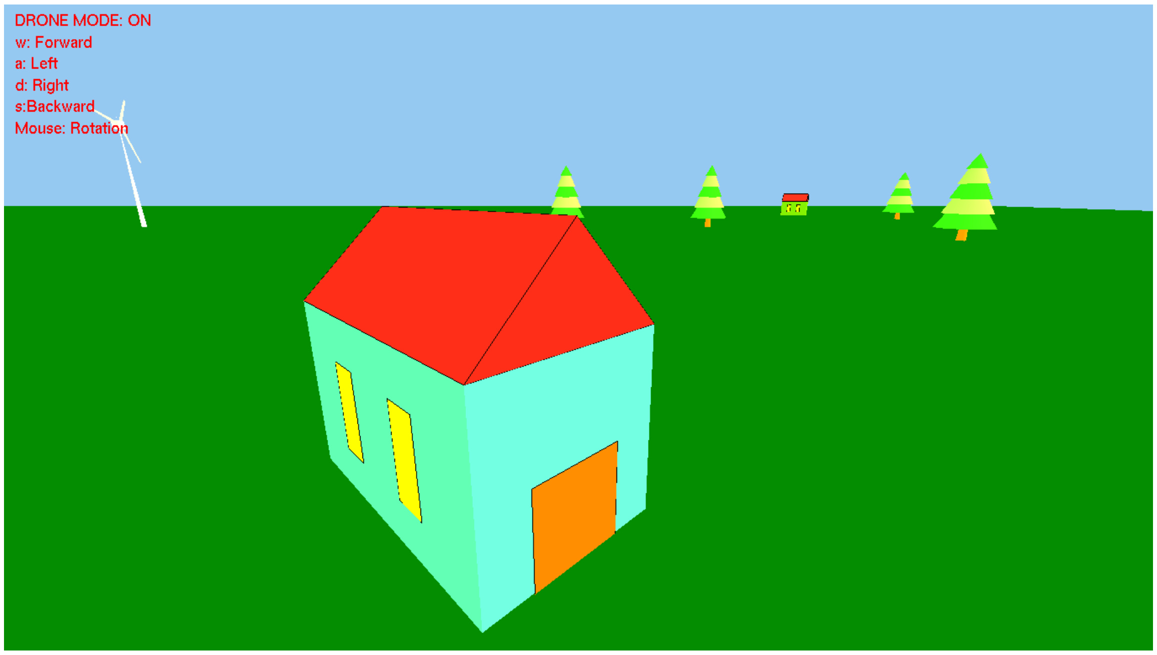
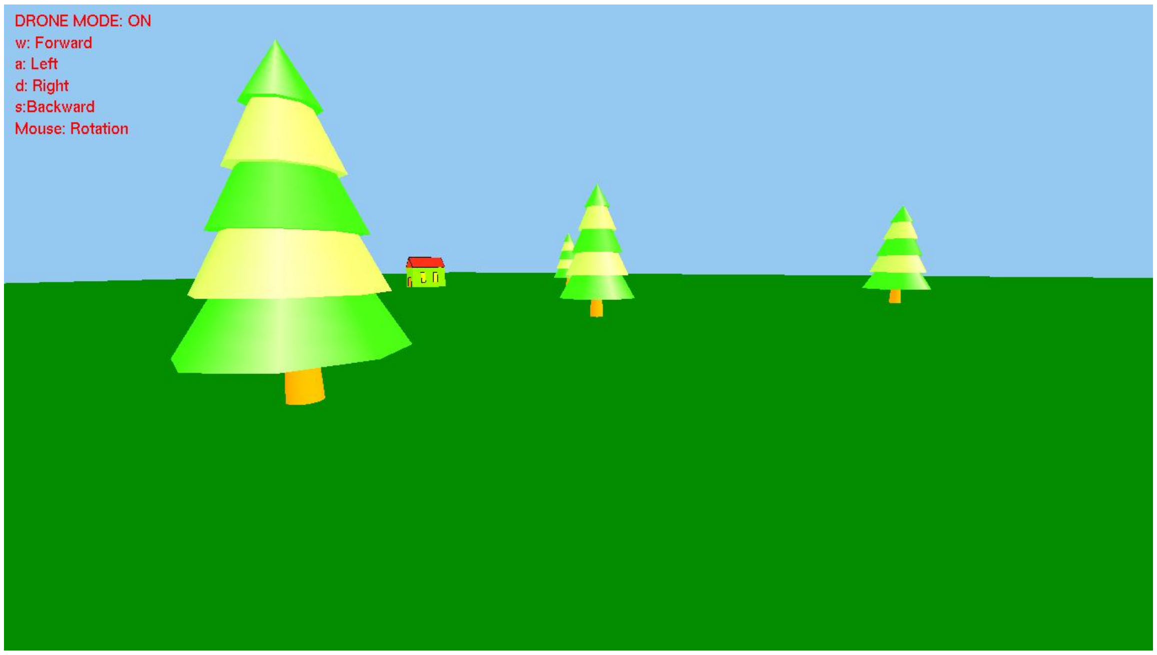


Fig. 4.4 Menus



DRONE MODE: ON
w: Forward
a: Left
d: Right
s:Backward
Mouse: Rotation

Fig. 4.5 House



DRONE MODE: ON
w: Forward
a: Left
d: Right
s:Backward
Mouse: Rotation

Fig. 4.6 Tree

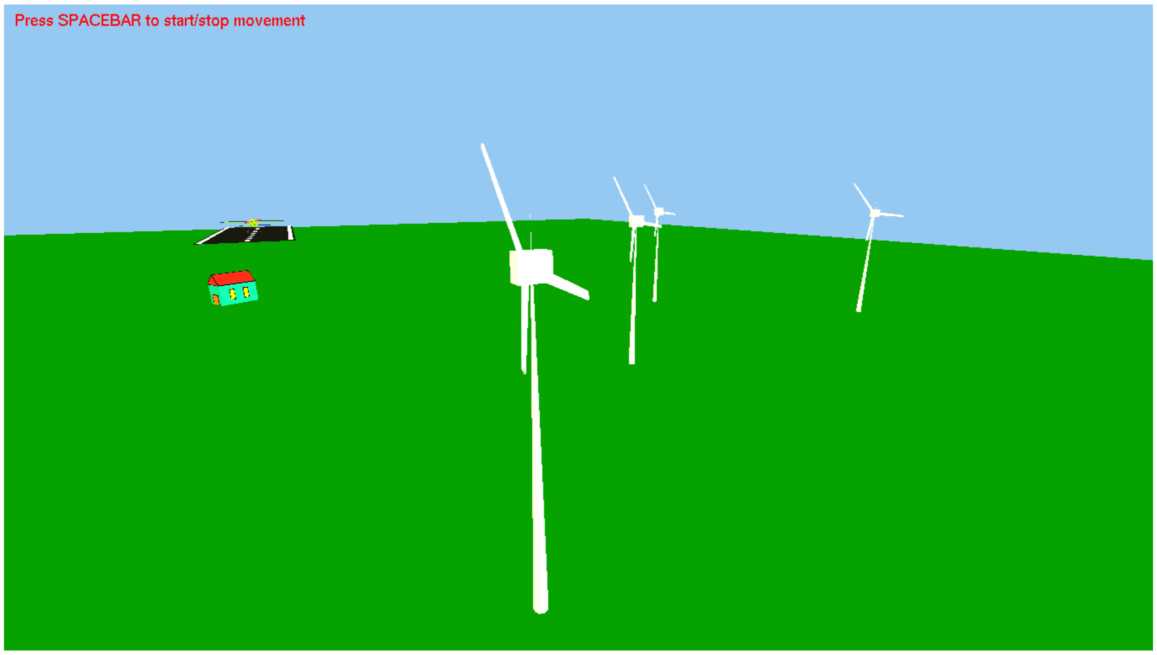


Fig. 4.7 Windmill

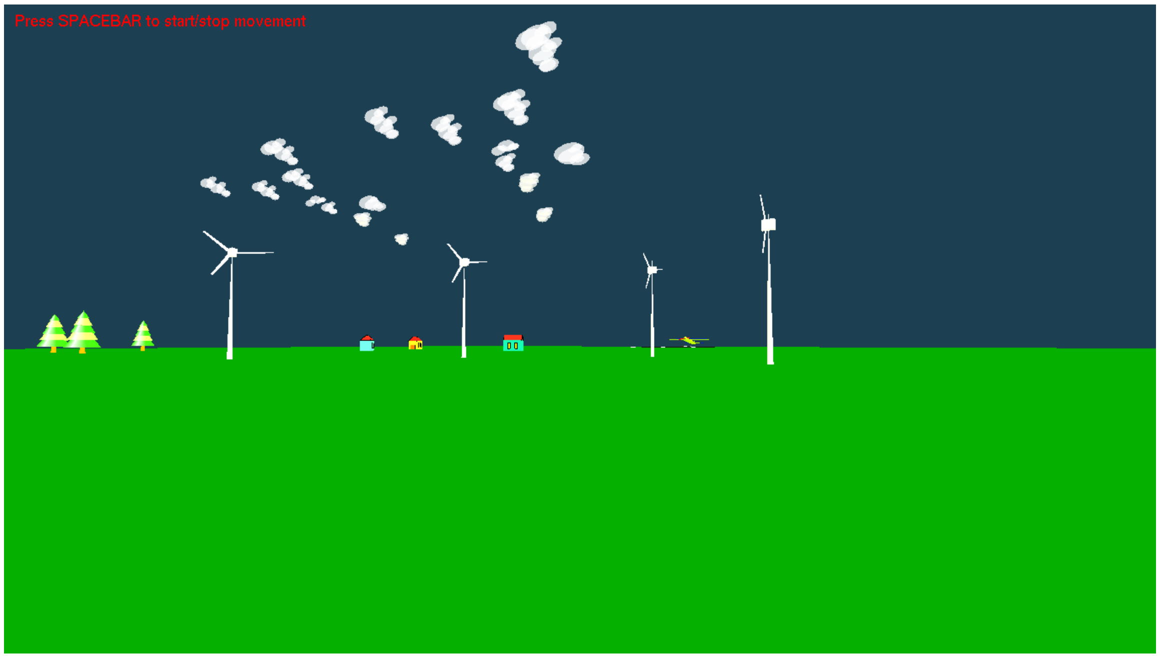


Fig. 4.8 Night Mode

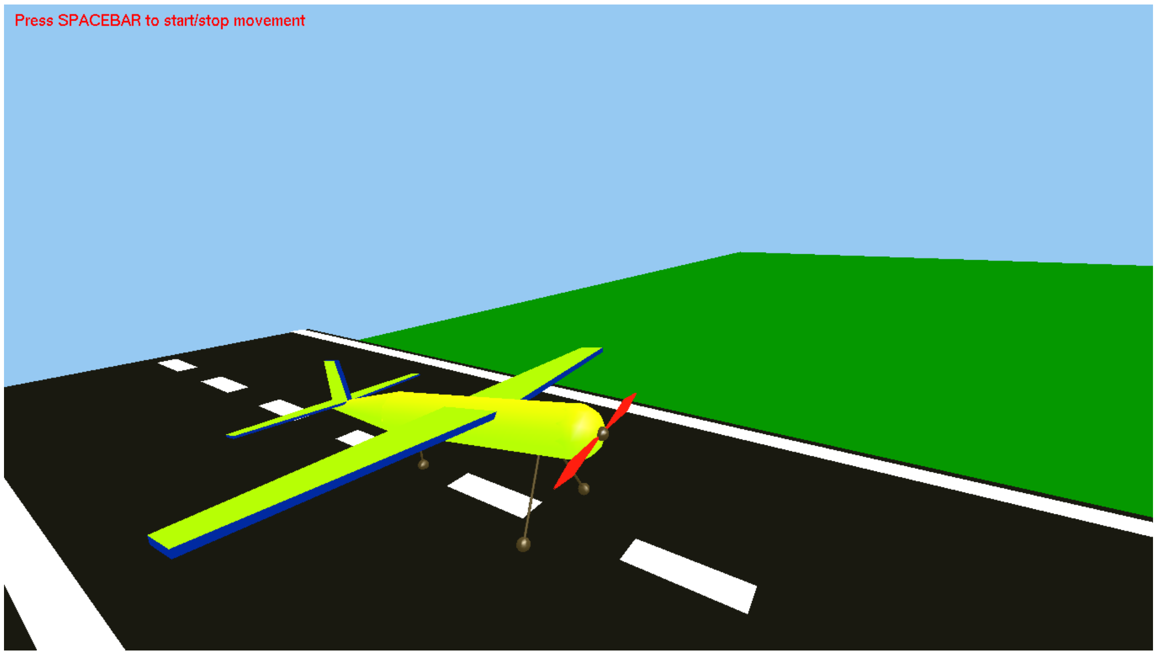

Press SPACEBAR to start/stop movement

Fig. 4.9 Aircraft

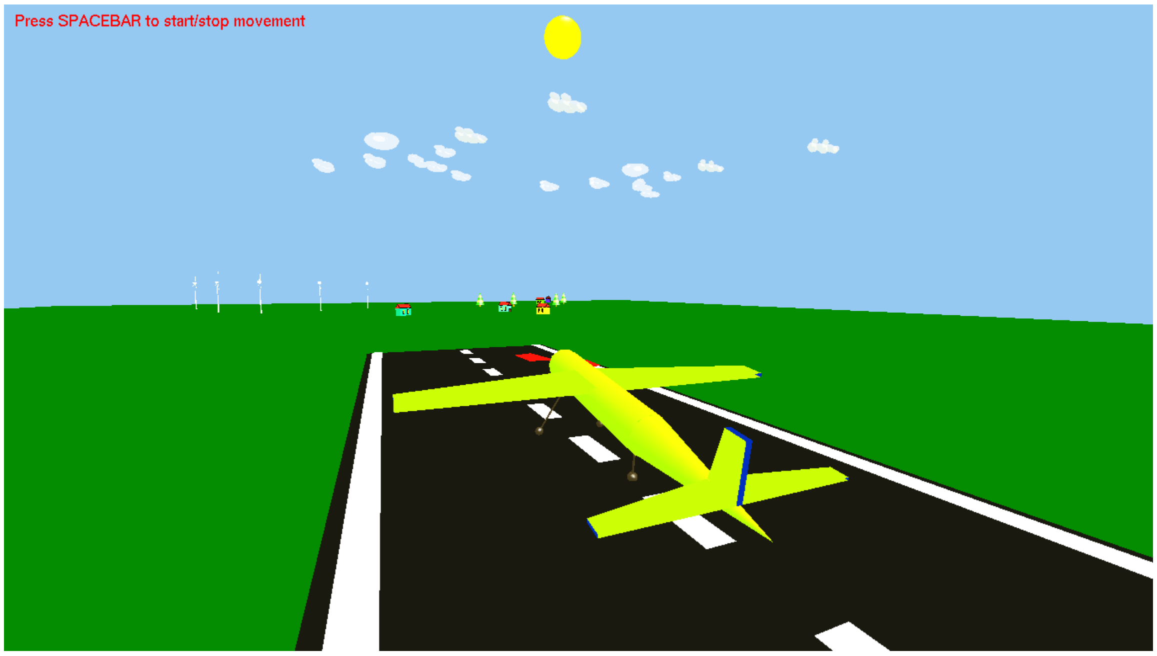

Press SPACEBAR to start/stop movement

Fig. 4.10 Aircraft View

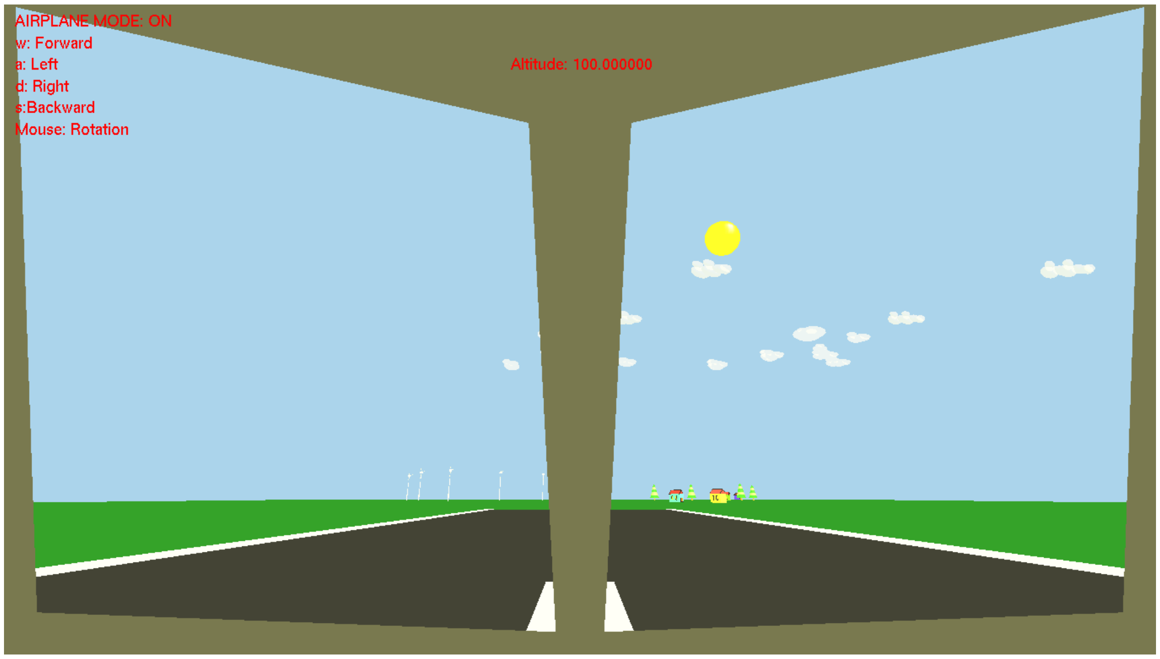


Fig. 4.11 Aircraft Takeoff Simulation

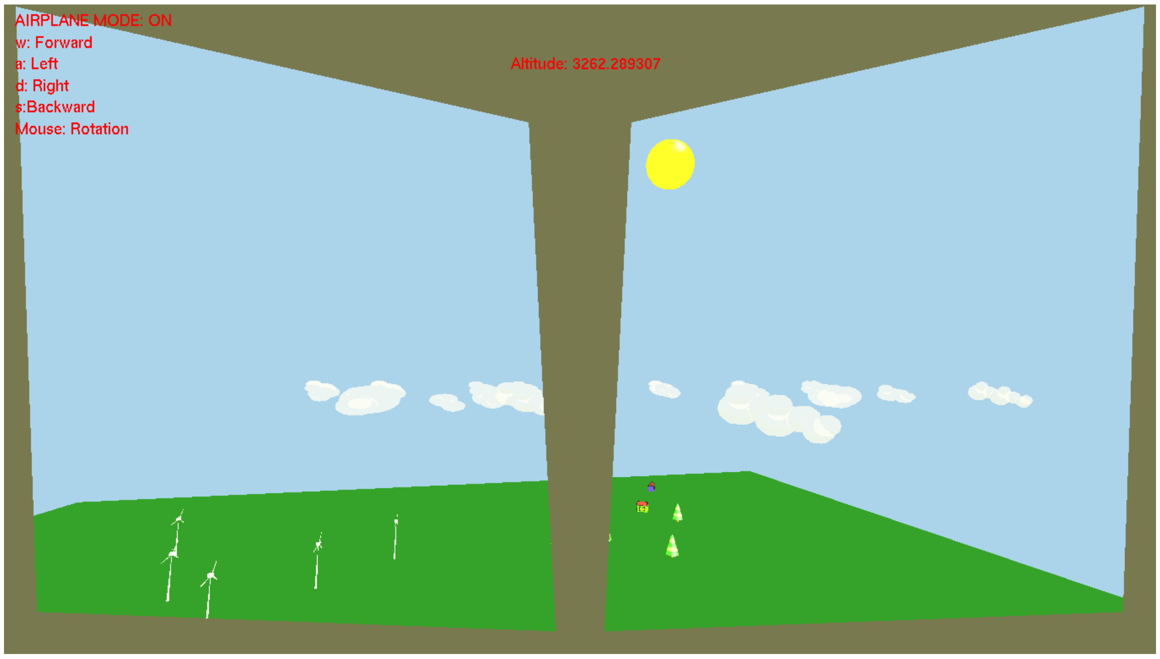


Fig. 4.12 Aircraft Flying Simulation

Chapter-5
Conclusion

Conclusion

The Project succeeded in creating a 3D Landscape with an imaginative graphics of all the objects present in it. It gives a virtual experience of a human present in the landscape enabling him to simulate walking and drift the skies with Aircraft Simulation. He can hover around the landscape with the Drone mode ON. To conclude, this project simulates a real-world with low graphics and can be used to develop any games.

Future Works

In the future, this project can be developed further by adding Textures to all the surfaces of the landscape which will simulate all the objects nearly to a real-world experience. The Aircraft Simulation can be improved by adding different views in the simulation like outer view of the aircraft while in motion. The landscape can be made as an infinite world where the landscape keeps generating as and when the viewer keeps navigating the landscape.

Chapter-6
Bibliography

Bibliography

- Interactive Computer Graphics – Edward Angel
- OPENGL API Documentation at https://www.opengl.org/documentation/
- OpenGL Programming Guide: The Official Guide to Learning OpenGL - Dave Shreiner
- http://www.glprogramming.com/red