VISVESVARAYA TECHNOLOGICAL UNIVERSITY

JNANASANGAMA, BELAGAVI – 590018

Mini Project Report
On

WIND ENERGY

Submitted in partial fulfillment for the award of degree of

Bachelor of Engineering
In
Computer Science and Engineering

Submitted by
R Narendranath Reddy
1BG17CS075

Vidyayāmruthamashnuthe

B.N.M. Institute of Technology

Approved by AICTE, Affiliated to VTU, Accredited as grade A Institution by NAAC.
All UG branches – CSE, ECE, EEE, ISE & Mech.E accredited by NBA for academic years 2018-19 to 2020-21 & valid upto 30.06.2021
Post box no. 7087, 27th cross, 12th Main, Banashankari 2nd Stage, Bengaluru- 560070, INDIA
Ph: 91-80- 26711780/81/82 Email: principal@bnmit.in, www. bnmit.org

Department of Computer Science and Engineering

2019-20

B.N.M. Institute of Technology

Approved by AICTE, Affiliated to VTU, Accredited as grade A Institution by NAAC.
All UG branches – CSE, ECE, EEE, ISE & Mech.E accredited by NBA for academic years 2018-19 to 2020-21 & valid upto 30.06.2021
Post box no. 7087, 27th cross, 12th Main, Banashankari 2nd Stage, Bengaluru- 560070, INDIA
Ph: 91-80- 26711780/81/82 Email: principal@bnmit.in, www. bnmit.org

Department of Computer Science and Engineering

Vidyayāmruthamashnuthe

CERTIFICATE

Certified that the Mini Project entitled **WIND ENERGY** carried out by Mr. **R Narendranath Reddy** USN **1BG17CS075** a bonafide student of VI Semester B.E., **B.N.M Institute of Technology** in partial fulfillment for the Bachelor of Engineering in COMPUTER SCIENCE AND ENGINEERING of the **Visvesvaraya Technological University**, Belagavi during the year 2019-20. It is certified that all corrections/suggestions indicated for Internal Assessment have been incorporated in the report. The Project report has been approved as it satisfies the academic requirements in respect of Computer Graphics Laboratory with Mini Project prescribed for the said Degree.

Prof. Akshitha Katkeri
Assistant Professor & Lab-Incharge
Department of CSE
BNMIT, Bengaluru

Dr. Sahana D. Gowda
Professor and HOD
Department of CSE
BNMIT, Bengaluru

Name & Signature

Examiner 1:

Examiner 2:

Table of Contents

CONTENTS **Page No.**

List of Figures

ABSTRACT

The project Wind energy is to show about how efficiently a renewable energy can be utilised to generate electricity with the help of OpenGL functions. Simple OpenGL functions are used to draw rotating fans, powerhouse, street lights and houses in the proposed application. The project includes features such as movement of clouds indicating the presence of winds, Clockwise and anticlockwise rotation of windmill, fast and slow movements of wind and generation of electricity. The user is allowed to interact with the application with the help of mouse function and thereby choose any of the above mentioned functions. When the chosen action happens to involve presence of wind , the mechanical power of rotating blades is converted into electrical energy and thus the generated energy is used to light up the houses and street lights.

Computer graphics is an art of drawing pictures, lines, charts, etc. with the help of computer programming. The application Wind energy is designed and implemented using a graphics software system called OpenGl. The OpenGL transformations and the input interactions add more to the graphics. The project Visual Transformation Techniques using OpenGl is based on Rotation and Translation processes.

.

ACKNOWLEDGEMENT

The success and final outcome of this project required a lot of guidance and assistance from many people and I am extremely privileged to have got this all along the completion of my project.

I would like to thank **Shri. Narayan Rao R Maanay**, Secretary, BNMIT, Bengaluru for providing the excellent environment and infrastructure in the college.

I would like to sincerely thank **Prof. T J Rama Murthy**, Director, BNMIT, Bengaluru for having extended his constant support and encouragement during the course of this project.

I would like to express my gratitude to **Prof. Eishwar N Maanay**, Dean, BNMIT, Bengaluru for his relentless support and encouragement.

I would like to thank **Dr. Krishnamurthy G N**, Principal, BNMIT, Bengaluru for his constant encouragement.

I would like to thank, **Dr. Sahana D. Gowda**, Professor & Head of the Department of Computer Science and Engineering for the encouragement and motivation she provides.

I would also like to thank **Mrs. Akshitha Katkeri,** Assistant Professor, Department of Computer Science and Engineering for providing me with his valuable insight and guidance wherever required throughout the course of the project and its successful completion.

R NARENDRANATH REDDY

1BG17CS075

Chapter 1

INTRODUCTION

1.1 Overview

Graphics provides one of the most natural means of communicating with a computer, since highly developed 2D and 3D pattern recognition abilities allow to perceive and process pictorial data rapidly and efficiently.

Interactive computer graphics are the important means of producing pictures since the invention of photography and television. Some topics in computer graphics include user interface design, sprite graphics, vector graphics, 3D modelling ,shaders, GPU design, implicit surface visualization with ray tracing, and computer vision, among others. The overall methodology depends heavily on the underlying sciences of geometry, optics, and physics. It has the added advantage that, with the computer, can make pictures not only of concrete real world objects but also of abstract, synthetic objects, such as mathematical surfaces and of data that have no inherent geometry, such as survey results. To the user it appears that the picture is changing instantaneously in response to his commands. He can give a series of commands, each one generating a graphical response from the computer. In this way he maintains a conversation, or dialogue, with the computer.

1.2 Problem Statement

The aim of this project is to show the generation of electricity from wind energy using OPENGL and its functions. This main objective is to demonstrate the working of windmill to produce electric power. The application will also include user interaction through keyboard and mouse events, to view fast and slow movements of wind both clockwise and anticlockwise. The application will be implemented using the C programming language and the OpenGL API.

1.3 Motivation

Wind energy is the use of wind to generate electricity. The kinetic energy in the wind is converted to mechanical power, which in turn is used by rotor connected to main shaft to generate electricity. The ability to develop its visualization using C programming and the OpenGL API serves as a motivation to develop this application. The OpenGL transformations and the input interactions add more to the graphics and has become a widely accepted standard to design any graphic application.

1.4 Computer Graphics

Computer graphics and multimedia technologies are becoming widely used in educational applications because they facilitate non-linear, self-learning environments that particularly suited to abstract concepts and technical information.

Computer graphics are pictures and films created using computers. Usually, the term refers to computer-generated image data created with help from specialized graphical hardware and software. It is a vast and recent area in computer science. The phrase was coined in 1960, by computer graphics researchers Verne Hudson and William Fetter of Boeing. It is often abbreviated as CG, though sometimes erroneously referred to as CGI. Important topics in computer graphics include user interface design, sprite graphics, vector graphics, 3D modelling, shaders, GPU design, implicit surface visualization with ray tracing, and computer vision, among others.

The overall methodology depends heavily on the underlying sciences of geometry, optics, and physics. Computer graphics is responsible for displaying art and image data effectively and meaningfully to the user. It is also used for processing image data received from the physical world. Computer graphic development has had a significant impact on many types of media and has revolutionized animation, movies, advertising, video games, and graphic design generally.

1.5 OpenGL API

Open Graphics Library (OpenGL) is a cross-language, cross-platform application programming interface (API) for rendering 2D and 3D vector graphics. The API is typically used to interact with a graphics processing unit (GPU), to achieve hardware-accelerated rendering. Silicon Graphics Inc., (SGI) began developing OpenGL in 1991 and released it on June 30, 1992; applications use it extensively in the fields of computer-aided design (CAD), virtual reality, scientific visualization, information visualization, flight simulation, and video games. Since 2006 OpenGL has been managed by the non-profit technology consortium Khronos Group.

The OpenGL specification describes an abstract API for drawing 2D and 3D graphics. Although it is possible for the API to be implemented entirely in software, it is designed to be implemented mostly or entirely in hardware. The API is defined as a set of functions which may be called by the client program, alongside a set of named integer constants. In addition, OpenGL is also cross-platform

Given that creating an OpenGL context is quite a complex process, and given that it varies between operating systems, automatic OpenGL context creation has become a common feature of several game-development and user-interface libraries, including SDL, Allegro, SFML, FLTK, and Qt. A few libraries have been designed solely to produce an OpenGL-capable window. The first such library was OpenGL Utility Toolkit (GLUT), later superseded by freeglut. GLFW is a newer alternative.

1.5.1 OpenGL API Architecture

Display Lists:

All data, whether it describes geometry or pixels, can be saved in a display list for current or later use. When a display list is executed, the retained data is sent from the display list just as if it were sent by the application in immediate mode.

Evaluators:

All geometric primitives are eventually described by vertices. Parametric curves and surfaces may be initially described by control points and polynomial functions called basis functions.

Per Vertex Operations:

For vertex data, next is the "per-vertex operations" stage, which converts the vertices into primitives. Some vertex data are transformed by 4 x 4 floating-point matrices. Spatial coordinates are projected from a position in the 3D world to a position on your screen.

Primitive Assembly:

Clipping, a major part of primitive assembly, is the elimination of portions of geometry which fall outside a half space, defined by a plane.

Pixel Operation:

While geometric data takes one path through the OpenGL rendering pipeline, pixel data takes a different route. Pixels from an array in system memory are first unpacked from one of a variety of formats into the proper number of components. Next the data is scaled, biased, and processed by a pixel map. The results are clamped and then either written into texture memory or sent to the rasterization step.

Rasterization:

Rasterization is the conversion of both geometric and pixel data into fragments.

Each fragment square corresponds to a pixel in the frame buffer. Colour and depth values are assigned for each fragment square.

Fragment Operations:

Before values are actually stored into the framebuffer, a series of operations are performed that may alter or even throw out fragments. All these operations can be enabled or disabled.

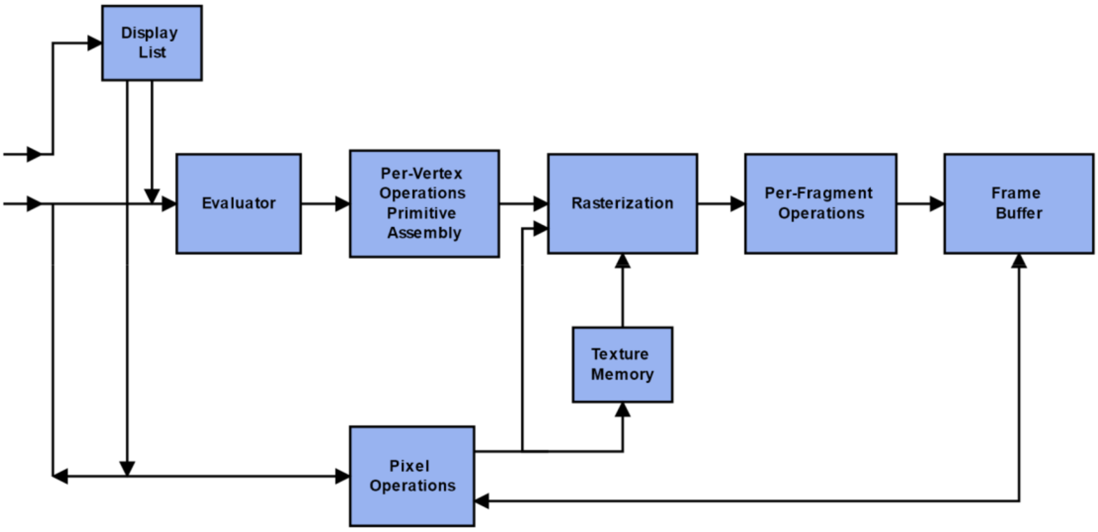

Fig 1.1 An illustration of the graphics pipeline process in OpenGL Architecture

1.6 Applications of Computer Graphics

Although many applications span two, three, or even all of these areas, the development of the field was based, for the most part, on separate work in each domain.
We can classify applications of computer graphics into four main areas:

1.6.1 Display of Information

Graphics has always been associated with the display of information. Examples of the use of orthographic projections to display floorplans of buildings can be found on 4000-year-old Babylonian stone tablets. Mechanical methods for creating perspective drawings were developed during the Renaissance. Countless engineering students have become familiar with interpreting data plotted on log paper. More recently, software packages that allow interactive design of charts incorporating colour, multiple data sets, and alternate plotting methods have become the norm. In fields such as architecture and mechanical design, hand drafting is being replaced by computer-based drafting systems using plotters and workstations. Medical imaging uses computer graphics in a number of exciting ways.

1.6.2 Design

Professions such as engineering and architecture are concerned with design. Although their applications vary, most designers face similar difficulties and use similar methodologies. One of the principal characteristics of most design problems is the lack of a unique solution. Hence, the designer will examine a potential design and then will modify it, possibly many times, in an attempt to achieve a better solution. Computer graphics has become an indispensable element in this iterative process.

1.6.3 Simulation

Some of the most impressive and familiar uses of computer graphics can be classified as simulations. Video games demonstrate both the visual appeal of computer graphics and our ability to generate complex imagery in real time. Computer-generated images are also the heart of flight simulators, which have become the standard method for training pilots.

1.6.4 User Interfaces

The interface between the human and the computer has been radically altered by the use of computer graphics. Consider the electronic office. The figures in this book were produced through just such an interface. A secretary sits at a workstation, rather than at a desk equipped with a typewriter. This user has a pointing device, such as a mouse, that allows him to communicate with the workstation.

Chapter 2

LITERATURE SURVEY

2.1 History of Computer Graphics

The term “computer graphics” was coined in 1960 by William Fetter, a designer at Boeing, to describe his own job, the field can be said to have first arrived with the publication in 1963 of Ivan Sutherland’s Sketchpad program, as part of his Ph.D. thesis at MIT. Sketchpad, as its name suggests, was a drawing program. Beyond the interactive drawing of primitives such as lines and circles and their manipulation – in particular, copying, moving and constraining – with use of the then recently invented light pen, Sketchpad had the first fully-functional graphical user interface (GUI) and the first algorithms for geometric operations such as clip and zoom. Interesting, as well, is that Sketchpad’s innovation of an object-instance model to store data for geometric primitives foretold object-oriented programming. Coincidentally, on the hardware side, the year 1963 saw the invention by Douglas Engelbart at the Stanford Research Institute of the mouse, the humble device even today carrying so much of GUI on its thin shoulders.

Subsequent advances through the sixties came thick and fast: raster algorithms, the implementation of parametric surfaces, hidden-surface algorithms and the representation of points by homogeneous coordinates, the latter crucially presaging the foundational role of projective geometry in 3D graphics, to name a few. Flight simulators were the killer app of the day and companies such as General Electric and Evans & Sutherland, 6 co-founded by Douglas Evans and Ivan Sutherland, wrote simulators with real-time graphics.

The seventies, brought the Z-buffer for hidden surface removal, texture mapping, Phong’s lighting model – all crucial components of the OpenGL API (Application Programming Interface) we’ll be using soon – as well as keyframe-based animation. Photorealistic rendering of animated movie keyframes almost invariably deploys ray tracers, which were born in the seventies too.

Through the nineties, as well, the use of 3D effects in movies became pervasive. The Terminator and Star Wars series, and Jurassic Park, were among the early movies to set the standard for CGI. Toy Story from Pixar, 8 released in 1995, has special importance in the history of 3D CGI as the first movie to be entirely computer generated no scene was ever pondered through a glass lens, nor any recorded on a photographic reel! It was cinema without film. Quake, released in 1996, the first of the hugely popular Quake series of games, was the first fully 3D game.

Another landmark from the nineties of particular relevance to us was the release in 1992 of OpenGL, the open-standard cross-platform and cross language 3D graphics API, by Silicon Graphics. OpenGL is actually a library of calls to perform 3D tasks, which can be accessed from programs written in various languages and running over various operating systems. That OpenGL was high-level (in that it frees the applications programmer from having to care about such low-level tasks as representing primitives like lines and triangles in the raster, or rendering them to the window) and easy to use (much more so than its predecessor 3D graphics API, PHIGS, standing for Programmer's Hierarchical Interactive Graphics System) first brought 3D graphics programming to the "masses". What till then had been the realm of a specialist was now open to a casual programmer following a fairly amicable learning curve.

Since its release OpenGL has been rapidly adopted throughout academia and industry. It's only among game developers that Microsoft's proprietary 3D API, Direct3D, which came soon after OpenGL bearing an odd similarity to it but optimized for Windows, is more popular.

The story of the past decade has been one of steady progress, rather than spectacular innovations in CG. Hardware continues to get faster, better, smaller and cheaper, continually pushing erstwhile high-end software down market, and raising the bar for new products. The almost complete displacement of CRT monitors by LCD and the emergence of high-definition television are familiar consequences of recent hardware evolution.

2.2 Related Work

- **Computer Aided Design (CAD):**

Most of engineering and Architecture students are concerned with Design. CAD is used to design various structures such as Computers, Aircrafts, Building, in almost all kinds of Industries. Its use in designing electronic systems is known as electronic design automation (EDA). In mechanical design it is known as mechanical design automation(MDA) or computer aided drafting (CAD), which includes the process of creating a technical drawing with the use of computer software.

- **Computer Simulation**

Computer simulation is the reproduction of the behaviour of a system using a computer to simulate the outcomes of a mathematical model associated with said system. Since they allow to check the reliability of chosen mathematical models, computer simulations have become a useful tool for the mathematical modelling of many natural systems in physics (computational physics), astrophysics, climatology, chemistry, biology and manufacturing, human systems in economics, psychology, social science, health care and engineering. Simulation of a system is represented as the running of the system's model. It can be used to explore and gain new insights into new technology and to estimate the performance of systems too complex for analytical solutions.

- **Digital Art**

Digital art is an artistic work or practice that uses digital technology as part of the creative or presentation process. Since the 1970s, various names have been used to describe the process, including computer art and multimedia art. Digital art is itself placed under the larger umbrella term new media art. With the rise of social media and the internet, digital art application of computer graphics. After some initial resistance, the impact of digital technology has transformed activities such as painting, drawing, sculpture and music/sound art, while new forms, such as net art, digital installation art, and virtual reality, have become recognized artistic practices. More generally the term digital artist is used to describe an artist who makes use of digital technologies in the production of art. In an expanded sense, "digital art" is contemporary art that uses the methods of mass production or digital media.

- **Virtual Reality**

Virtual reality (VR) is an experience taking place within a computer-generated reality of immersive environments can be similar to or completely different from the real world. Applications of virtual reality can include entertainment (i.e. gaming) and educational purposes (i.e. medical or military training). Other, distinct types of VR style technology include augmented reality and mixed reality. Currently standard virtual reality systems use either virtual reality headsets or multi-projected environments to generate realistic images, sounds and other sensations that simulate a user's physical presence in a virtual environment. A person using virtual reality equipment is able to look around VR headsets consisting of a head-mounted display with a small screen in front of the eyes ,but can also be created through specially designed rooms with multiple large screens. Virtual reality typically incorporates auditory and video feedback, but may also allow other types of sensory and force feedback through haptic technology.

- **Video Games**

A video game is an electronic game that involves interaction with a user interface to generate visual feedback on a two- or three-dimensional video display device such as a TV screen, virtual reality headset or computer monitor. Since the 1980s, video games have become an increasingly important part of the entertainment industry, and whether they are also a form of art is a matter of dispute. The electronic systems used to play video games are called platforms. Video games are developed and released for one or several platforms and may not be available on others. Specialized platforms such as arcade games, which present the game in a large, typically coin-operated chassis, were common in the 1980s in video arcades, but declined in popularity as other, more affordable platforms became available. These include dedicated devices such as video game consoles, as well as general-purpose computers like a laptop, desktop or handheld computing devices.

Chapter 3

SYSTEM REQUIREMENTS

3.1 Software Requirements

Software requirements deal with defining software resource requirements and prerequisites that need to be installed on a computer to provide optimal functioning of an application.

The following are the software requirements for the application :

- Operating System: Windows 10
- Compiler: GNU C/C++ Compiler
- Development Environment: Visual Studio 2019 Community Edition
- API: OpenGL API & Win32 API for User Interface and Interaction

3.2 Hardware Requirements

The most common set of requirements defined by any operating system or software application is the physical computer resources, also known as hardware.

- CPU: Intel or AMD processor
- Cores: Dual-Core (Quad-Core recommended)
- RAM: minimum 4GB (>4GB recommended)
- Graphics: Intel Integrated Graphics or AMD Equivalent
- Secondary Storage: 250GB
- Display Resolution: 1366x768 (1920x1080 recommended)

Chapter 4

SYSTEM DESIGN

4.1 Proposed System

The proposed system is designed to simulate Wind energy using OpenGL. Using OpenGL inbuilt functions the structure of the windmill has been designed. Transformation functions like translate and rotate have been used to design blades of the windmill. Various user defined functions have been used for increasing or decreasing fan speed along with clockwise and anticlockwise rotation of blades.

A wind farm with a power house and a few turbines is constructed. Clouds can also be seen in the sky. As the wind flows, the speed of the clouds increase. Wind can flow in either direction, and the turbine blades ensue the movement of winds. The user is allowed to interact with the application through mouse functions. The following menus are provided with the right click of the mouse function.

No wind : There is no wind flowing.

Wind CW : Wind flow in Clockwise direction.

Wind ACW : Wind flow in Anticlockwise direction.

Fast Wind CW : Wind flowing faster in Clockwise direction.

Fast Wind CW : Wind flowing faster in Anticlockwise direction.

As and when the wind energy is converted to electric energy , the lights in the houses and street lights light up thereby indicating that the wind energy is being harnessed.

The advantages are:

- OpenGL is designed as a streamlined.
- It is a hardware independent interface; it can be implemented on many different hardware platforms.
- With OpenGL, we can draw a small set of geometric primitive such as points, lines and polygons etc.
- It provides double buffering which is vital in providing transformations.
- It is event driven software.

- It provides call back function.

The header files used are:

1. #include<stdlib.h>:This is C library function for standard input and output.
2. #include<GL/glut.h>:This header is included to read the glut.h, gl.h and glu.h.
3. #include<math.h>:This is a C library function for performing certain mathematical operations.

4.2 Data Flow Diagram

A data-flow diagram (DFD) is a way of representing a flow of a data of a process or a system (usually an information system). The DFD also provides information about the outputs and inputs of each entity and the process itself. A data-flow diagram has no control flow, there are no decision rules and no loops. For each data flow, at least one of the endpoints (source and / or destination) must exist in a process. The interaction with the windows is initialized using glutInit() openGL API. The display mode- double buffer and depth buffer is, various callback functions for drawing and redrawing ,for mouse and keyboard interfaces ,input and calculate functions for various mathematical calculations, the window position and size are also initialized and create the window to display the output .

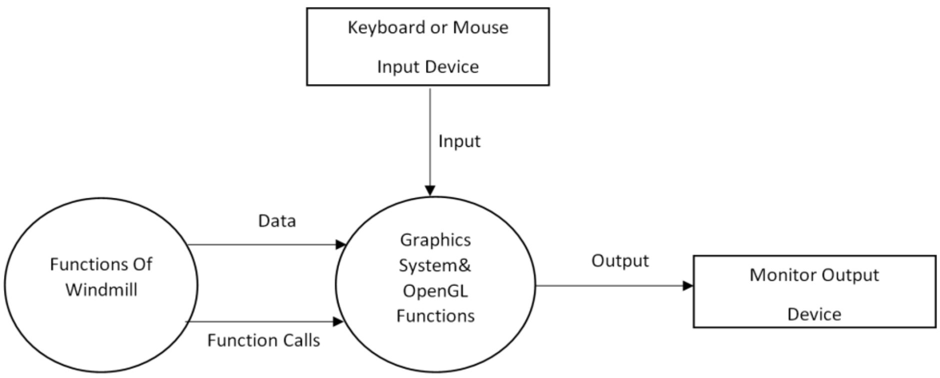

Figure 4.1 Graphics function flow

Figure 4.1 shows the Level 0 Dataflow diagram of the proposed application for a Wind energy simulator. The keyboard and mouse devices are used for input to the application. The graphics system processes these user keyboard/mouse interactions using built in OpenGL functions like glutKeyboardFunc(void *) and glutMouseFunc(*void). The refined representation of a process can be done in another data-flow diagram, which subdivides this process into sub-processes.

4.3 Flowchart

A flowchart is a visual representation of the sequence of steps and decisions needed to perform a process. Each step in the sequence is noted within a diagram shape. Steps are linked by connecting lines and directional arrows.

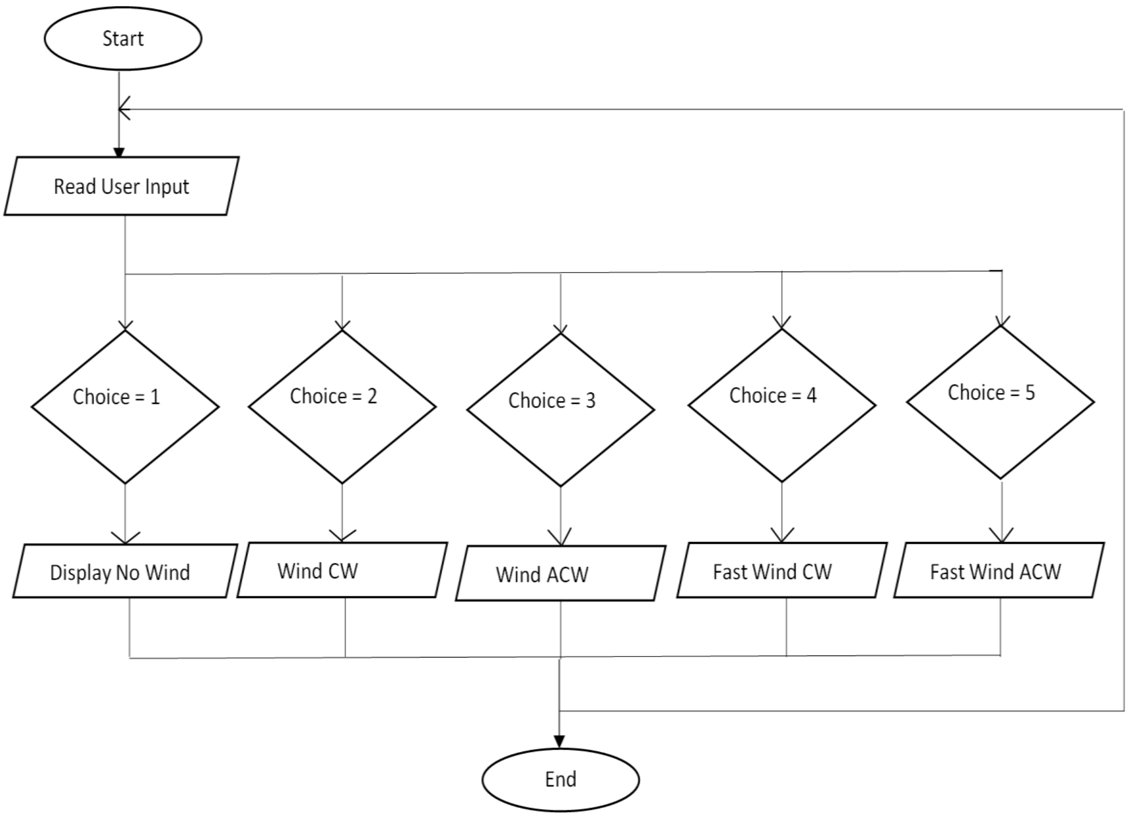

Figure 4.2 Flowchart of the visualization

Chapter 5

IMPLEMENTATION

5.1 Module Description

- **void text():**

  Function to display the text message on the application through raster function .

- **void hut():**

  Function to draw the huts in the application.

- **void streetlight():**

  Function to construct the streetlights which light up when electricity is generated through wind.

- **void display():**

  Clears the window (and the depth buffer) and draws all the compone, wiresand background.nts of the application, i.e, hut ,streetlight ,fan, fanpole, cloud, powers station, road.

- **void reshape (Glint w , Glint h):**

The reshape function defines what to do when the window is resized. It must have a void return type, and takes two int **parameters** (the new width and height of the window).

- **void menu():**

  Function that provides options to be chosen by the user.

- **void spinclockwise() / void anticlockwise():**

  Rotates the fans of the windmill either clockwise or anticlockwise.

- **void fan():**

  To draw the windmills in the farm.

- **void powerstation():**

  To build the power station where electricity is generated.

5.2 High Level Code

5.2.1 Built-In Functions

- **void glLoadIdentity();**

Sets the current transformation matrix to identity matrix.

- **void glClear(glEnum mode);**

  Clears the buffers namely color buffer and depth buffer. mode refers to GL_COLOR_BUFFER_BIT or DEPTH_BUFFER_BIT.

- **void glTranslate[fd](TYPE x, TYPE y, TYPE z);**

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

- **void glutSwapBuffers();**

  Swaps the front and back buffers.

- **void glMatrixMode(GLenum mode);**

  Specifies which matrix will be affected by subsequent transformations, Mode can be GL_MODELVIEW or GL_PROJECTION.

- **void glLoadIdentity( );**

  Sets the current transformation matrix to identity matrix.

- **void glEnable(GLenum feature);**

  Enables an OpenGL feature. Feature can be GL_DEPTH_TEST (enables depth test for hidden surface removal), GL_LIGHTING (enables for lighting calculations), GL_LIGHTi (used to turn on the light for number i), etc.

- **void glutBitMapCharacter(void *font, int character);**

  Without using any display lists, glutBitmapCharacter renders the character in the named bitmap font.

- **void glRasterPos[234][ifd](GLfloat x, GLfloat y, GL float z);**

  This position, called the raster position, is used to position pixel and bitmap write operations.

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

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

- **void glutInitDisplayMode(unsigned int mode);**

  Requests a display with the properties in the 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).

- **void glutCreateWindow(char *title);**

  Creates a window on display; the string title 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.

- **void glutMainLoop();**

  Causes the program to enter an event-processing loop.

- **void glutDisplayFunc(void (*func)(void));**

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

- **void glutMouseFunc(void *f(int button, int state, int x, int y)**

  Registers the mouse callback function f. The callback function returns the button (GLUT_LEFT_BUTTON,etc., the state of the button after the event (GLUT_DOWN), and the position of the mouse relative to the top-left corner of the window

- **void glClearColor(GLclampf r,GLclampf g, GLclamp b, Glclamp a)**

  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.

- **void glViewport(int x ,int y, GLsizei width, GLsizei height)**

  Specifies the width*height viewport in pixels whose lower left corner is at (x,y) measured from the origin of the window.

- **void glColor3[b I f d ub us ui](TYPE r, TYPE g, TYPE b)**

  Sets the present RGB colors. Valid types are bytes(b), int(i), float(f), double(d), unsigned byte(ub), unsigned short(us), and unsigned int(ui). The maximum and minimum value for floating point types are 1.0 and 0.0 respectively, whereas the maximum and minimum values of the discrete types are those of the type, for eg, 255 and 0 for unsigned bytes.

- **void glutInitWindowSize(int width, int height);**

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

- **void glutReshapeFunc(void *f(int width, int height));**

  Registers the reshape callback function f. the callback function returns the height and width of the new window. The reshape callback invokes the display callback.

- **void createMenu(void);**

  This function is used to create menus which are used as options in program.

5.2.2 User Implementation Code

- **Main Function**

```
int main(int argc,char **argv)
{
glutInit(&argc,argv);
glutInitDisplayMode(GLUT_DOUBLE|GLUT_RGB|GLUT_DEPTH);
glutInitWindowSize(500,500);
glutInitWindowPosition(100,100);
glutCreateWindow("WIND ENERGY");
init();
glutDisplayFunc(display);
glutReshapeFunc(reshape);
glutKeyboardFunc(mykey);
glutCreateMenu(menu);
glutAddMenuEntry("No Wind",1);
glutAddMenuEntry("Wind CW",2);
glutAddMenuEntry("Wind ACW",3);
glutAddMenuEntry ("Fast Wind CW",4);
glutAddMenuEntry("Fast Wind ACW",5);
glutAddMenuEntry("Quit",6);
glutAttachMenu(GLUT_RIGHT_BUTTON);
glutMainLoop();
return 0;
}
```

- **Streetlight function**

```
void streetlight()
{
 glPushMatrix();/*1st street light*/
 glLoadIdentity();
 glColor3f(0.2,0.2,0.2);
 glBegin(GL_POLYGON);
 glVertex3f(28.0,-20.0,2.0);
 glVertex3f(29.0,-20.0,3.0);
 glVertex3f(29.0,10.0,4.0);
 glVertex3f(28.0,10.0,2.0);
 glEnd();
 glPopMatrix();
 glPushMatrix();
 glColor3f(0.3,0.15,0.1);
 glBegin(GL_POLYGON);
 glVertex3f(26.0,6.0,2.0);
 glVertex3f(31.0,7.0,3.0);
 glVertex3f(31.0,6.0,4.0);
 glVertex3f(26.0,7.0,2.0);
 glEnd();
 glPopMatrix();
 glPushMatrix();
 glColor3f(b,b,b);
 glTranslatef(24.5,4.0,1.0);
 glRotatef(260,0,0,1);
 glScalef(1,3.5,1);
 glutSolidCube(2);
 glPopMatrix();
 glPushMatrix();/*2nd street light*/
 glLoadIdentity();
 glColor3f(0.2,0.2,0.2);
 glBegin(GL_POLYGON);
 glVertex3f(16.1,-10.0,2.0);
 glVertex3f(16.9,-10.0,3.0);
 glVertex3f(16.9,14.0,4.0);
 glVertex3f(16.1,14.0,2.0);
 glEnd();
 glPopMatrix();
 glPushMatrix();
 glColor3f(0.3,0.15,0.1);
```

```
glBegin(GL_POLYGON);
glVertex3f(14.5,12.0,2.0);
glVertex3f(18.5,13.0,3.0);
glVertex3f(18.5,12.0,4.0);
glVertex3f(14.5,13.0,2.0);

glEnd();
glPopMatrix();
glPushMatrix();
glColor3f(b,b,b);
glTranslatef(13.5,10.5,1.0);
glRotatef(260,0,0,1);
glScalef(1,3.5,1);
glutSolidCube(1.5);
glPopMatrix();
}
```

- **Background function**

```
void background()
{
glColor3f(0.0,0.1,0.0);
glBegin(GL_POLYGON);//green ground
glVertex2i(-250.0,-250.0);
glVertex2i(250.0,-250.0);
glVertex2i(250.0,0.0);
glVertex2i(-250.0,0.0);
glEnd();
glColor3f(0.1 ,0.1,0.1);
glBegin(GL_POLYGON);//mid night blue sky
glVertex2i(-250.0,0.0);
glVertex2i(-250.0,250.0);
glVertex2i(250.0,250.0);
glVertex2i(250.0,0.0);
glEnd();
}
```

- **Text Function**

```
void text()
{
glColor3f(u,v,w);
glRasterPos2f(0,13);
output("POWER HOUSE");
```

```
glRasterPos2f(20,13);
output("STREET LIGHT");
}
```

- **Fan Function**

  // Similar code for 4 fans

```
void fan1()
{
glPushMatrix();
glLoadIdentity();
glColor3f(1,1,1);
glTranslatef (-8.0,20.0, 2.0);/*rotation about fixed point*/
glRotatef(spin,0.0,0.0,1.0);
glTranslatef (8.0,-20.0, -2.0);
glBegin(GL_TRIANGLES);/*1st fan*/
glVertex3f(-8.0,20.0,2.0);
glVertex3f(-12.0,16.0,3.0);
glVertex3f(-12.0,18.0,4.0);
glVertex3f(-8.0,20.0,2.0);
glVertex3f(-4.0,24.0,3.0);
glVertex3f(-4.0,22.0,4.0);
glEnd();
glPopMatrix();
}
```

- **Road Function**

```
void road()
{
glColor3f(0.0,0.0,0.0);
glBegin(GL_POLYGON);
glVertex2f(-1,0);
glVertex2f(4,0);
glVertex2f(43,-39);
glVertex2f(37,-40);
glEnd();
}
```

- **Display Function**

```
 void display(void)
{
int b=0;
glClear(GL_COLOR_BUFFER_BIT|GL_DEPTH_BUFFER_BIT);
if(z<5){
 for(z=0;z<=1500;z++)
 {
 front();
 glutPostRedisplay();
 glutSwapBuffers();
 glFlush();
 }
 }
 Else
 {
 background();
 fanhouse();
 text();
 road();
 hut(-29,-23,-24,-33);
 hut(0,-11,5,-21);
 hut(-21,-1,-14,-11);
 clouds();
 powerstation();
 wires();
 streetlight();
 woman();
 fanpole1();
 fanpole2();
 fanpole3();
 fan1();
 fan2();
 fan3();
 fan4();
 glutSwapBuffers();
 glFlush();
 }
 }
```

- **Power station Function**

```
void powerstation()
{
GLint ax=1.5,ay=8;
glColor3f(1.0,0.25,0.1);
glBegin(GL_POLYGON);//from tip(anti clkwise)
glVertex2i(ax,ay);//a
glVertex2i(ax-2,ay-2);//b
glVertex2i(ax-2,ay-8);//c
glVertex2i(ax+2,ay-8);//d
glVertex2i(ax+2,ay-2);//e
glEnd();
glColor3f(0.7,0.5,0.3);
glBegin(GL_POLYGON);//roof (from a)
glVertex2i(ax,ay+3);//a
glVertex2i(ax-3,ay-3);//b
glVertex2i(ax+3,ay-3);//e
glEnd();
glColor3f(v,v,w);
glBegin(GL_POLYGON);/* door */
glVertex2i(ax-1,ay-5.0);//top left
glVertex2i(ax-1.0,ay-8.0);//bottom left
glVertex2i(ax+1.0,ay-8.0);// bottom right
glVertex2i(ax+1.0,ay-5.0);//top right
glEnd();
}
```

- **Cloud Function**

```
void clouds()
{
glPushMatrix();
glColor3f (0.4, 0.7,0.9);
glLoadIdentity (); /* clear the matrix */
/* viewing transformation */
glTranslatef(a+1, 40.0, -9.0);
glScalef (2.0, 1.0, 1.0); /* modeling transformation */
glutSolidSphere (2.0,50,56);
glLoadIdentity();
glTranslatef(a-2.0,40.0, -9.0);
glScalef (2.0, 1.0, 1.0);
glutSolidSphere (2.0,50,56);
glLoadIdentity();
glTranslatef(a+7.0,40.0, -9.0);
glScalef (2, 1.0, 1.0);
glutSolidSphere (2.0,50,56);
glLoadIdentity();
glTranslatef(a-7.0,40.0, -9.0);
```

```
glScalef (2, 1.0, 1.0);
glutSolidSphere (2.0,50,56);
glLoadIdentity();
glTranslatef(a+18.0,40.0, -9.0);
glScalef (2, 1.0, 1.0);
glutSolidSphere (2.0,50,56);
glLoadIdentity();
glTranslatef(a+25.0,40.0, -9.0);
glScalef (2, 1.0, 1.0);
glutSolidSphere (2.0,50,56);
glLoadIdentity();
glTranslatef(a+36.0,40.0, -9.0);
glScalef (3.0, 1.0, 1.0);
glutSolidSphere (2.0,50,56);
glLoadIdentity();
glTranslatef(a+50.0,40.0, -9.0);
glScalef (2, 1.0, 1.0);
glutSolidSphere (2.0,50,56);
glLoadIdentity();
glTranslatef(a+56.0,40.0, -9.0);
glScalef (2, 1.0, 1.0);
glutSolidSphere (2.0,50,56);
glPopMatrix();
}
```

- **Roof Function**

```
void roof(GLint rux,GLint ruy,GLint rdx,GLint rdy)
{
glPushMatrix();
glColor3f(1,0.25,0.1);
glBegin(GL_LINE_STRIP);
glVertex2i(rux,ruy);//roof up
glVertex2i(rdx,rdy);//roof down
glEnd();
glPopMatrix();
}
```

- **Hut Function**

```
void hut(GLint rux,GLint ruy,GLint rdx,GLint rdy)
{
GLint blx=rdx,bly=rdy-9,brx=rdx+10,bry=rdy-9,kx=rdx-8,ky=rdy+1;
GLfloat i;
for(i=0;i<440;i=i+1)
roof(rux+i/40,ruy,rdx+i/40,rdy);/* draw straws */
glColor3f(0.3,0.25,0.1);
glBegin(GL_POLYGON);/* front wall */
glVertex2i(rdx,rdy);//roof left
glVertex2i(rdx+10,rdy);//roof right
```

```
glVertex2i(brx,bry);//base right
glVertex2i(blx,bly);//base left
glEnd();
glColor3f(0.3,0.15,0.1);
glBegin(GL_POLYGON);/* side wall */
glVertex2i(rux,ruy);//roof up
glVertex2i(kx,ky);//bacK
glVertex2i(kx,ky-6);//bacK base
glVertex2i(blx,bly);
glVertex2i(rdx,rdy);
glEnd();
glColor3f(v,v,w);
glBegin(GL_POLYGON);/* window */
glVertex2i(kx+2,ky-2.5);//top left
glVertex2i(kx+2,ky-5.5);//bottom left
glVertex2i(blx-3,bly+3.5);//bottom right
glVertex2i(blx-3,bly+6.9);//top right
glEnd();
glBegin(GL_POLYGON);/* door */
glVertex2i(blx+3.5,bly);//bottom left
glVertex2i(brx-3.5,bry);//bottom right
glVertex2i(brx-3.5,bry+5);//top right
glEnd();
}
```

- **Fanpole Function**

```
void fanpole1()                    // Similar to 3 fanpoles.
{
glColor3f(1.0,1.0,1.0);
glBegin(GL_TRIANGLE_STRIP);
glVertex2f(-8.1,20.0);
glVertex2f(-7.9,20.0);
glVertex2f(-8.5,0.0);
glVertex2f(-7.5,0.0);
glEnd();
}
```

- **Fanhouse Function**

```
void fanhouse()
{ GLint ax=28,ay=30;
glColor3f(0.7,0.5,0.3);
glBegin(GL_POLYGON);//from tip(anti clkwise)
glVertex2i(ax,ay);//a
glVertex2i(ax-3,ay-2);//b
glVertex2i(ax-3,ay-9);//c
glVertex2i(ax+3,ay-9);//d
glVertex2i(ax+3,ay-2);//e
glEnd();
```

```
glColor3f(0.0,0.3,0.3);
glBegin(GL_POLYGON);//roof (from a)
glVertex2i(ax,ay+3);//a
glVertex2i(ax-4,ay-3);//b
glVertex2i(ax+4,ay-3);//e
glEnd();
glColor3f(0.3,0.15,0.1);
glBegin(GL_TRIANGLE_STRIP);
glVertex2f(30.1,21.0);
glVertex2f(29.9,21.0);
glVertex2f(30.5,0.0);
glVertex2f(29.5,0.0);
glEnd();
glBegin(GL_TRIANGLE_STRIP);
glVertex2f(26.1,21.0);
glVertex2f(25.9,21.0);
glVertex2f(26.5,0.0);
glVertex2f(25.5,0.0);
glEnd();
glColor3f(0.0,0.3,0.3);
glBegin(GL_TRIANGLE_STRIP);
glVertex2f(30.0,22.0);
glVertex2f(29.5,22.0);
glVertex2f(29.0,19.0);
glVertex2f(28.5,19.0);
glEnd();
glBegin(GL_TRIANGLE_STRIP);
glVertex2f(26.5,22.0);
glVertex2f(26.0,22.0);
glVertex2f(25.5,19.0);
glVertex2f(25.0,19.0);
glEnd();
glColor3f(0,0,d);
glEnable(GL_LINE_STIPPLE);
glLineStipple(1,0x00FF);
glBegin(GL_LINES);
glVertex2f(25.5,19.0);
glVertex2f(25.5,-1.0);
glVertex2f(25.0,19.0);
glVertex2f(25.0,-1.0);
glVertex2f(25.25,19.0);
glVertex2f(25.25,-1.0);
glEnd();
glColor3f(0,0,e);
glBegin(GL_LINES);
glVertex2f(28.5,19.0);
glVertex2f(28.5,-1.0);
glVertex2f(29.0,19.0);
glVertex2f(29.0,-1.0);
glVertex2f(28.75,19.0);
```

```
glVertex2f(28.75,-1.0);
glEnd();
glDisable(GL_LINE_STIPPLE);
glColor3f(0,0,1);
glBegin(GL_POLYGON);
glVertex2f(25.0,-1.0);
glVertex2f(29.5,-1.0);
glVertex2f(29.5,-2.0);
glVertex2f(25.0,-2.0);
glEnd();
glColor3f(0.3,0.15,0.1);
glBegin(GL_POLYGON);
glVertex2f(24.5,-1.0);
glVertex2f(25.0,-1.0);
glVertex2f(25.0,-2.0);
glVertex2f(24.5,-2.0);
glEnd();
glBegin(GL_POLYGON);
glVertex2f(24.5,-2.0);
glVertex2f(30.0,-2.0);
glVertex2f(30.0,-3.0);
glVertex2f(24.5,-3.0);
glEnd();
glBegin(GL_POLYGON);
glVertex2f(29.5,-1.0);
glVertex2f(30.0,-1.0);
glVertex2f(30.0,-2.0);
glVertex2f(29.5,-2.0);
glEnd();
}
```

- **Reshape Function**

```
void reshape(int w, int h)
{
glViewport(0, 0, (GLsizei) w, (GLsizei) h);
glMatrixMode(GL_PROJECTION);
glLoadIdentity();
glOrtho(-35.0, 35.0, -45.0, 45.0, -20.0, 20.0);
glMatrixMode(GL_MODELVIEW);
glLoadIdentity();
}
```

- **Wires Function**

```
glColor3f(.7,.5,.7);
glEnable(GL_LINE_STIPPLE);
glLineStipple(1,0x00FF);
```

```
glBegin(GL_LINES);
glVertex2f(-8.0,7.0);
glVertex2f(-32.0,7.0);
glVertex2f(-8.0,10.0);
glVertex2f(1.5,10.0);
glVertex2f(26.5,7.0);
glVertex2f(14.5,12.0);
glVertex2f(31.0,7.0);
glVertex2f(18.0,12.0);
glEnd();
glDisable(GL_LINE_STIPPLE);
}
```

- **Woman Function**

```
glClearColor(0.48,0.5,0.5,0.0);
glColor3f(1.0,0.0,0.0);
glBegin(GL_POLYGON);//veil
glVertex2f(21.0+x,-17.0+y);
glVertex2f(22.0+x,-17.0+y);
glVertex2f(22.5+x,-20.0+y);
glVertex2f(20.5+x,-20.0+y);
glEnd();
glBegin(GL_POLYGON);//face
glVertex2f(21.0+x,-18.0+y);
glVertex2f(21.0+x,-17.0+y);
glVertex2f(22.0+x,-17.0+y);
glVertex2f(22.0+x,-18.0+y);
glEnd();
glBegin(GL_POLYGON);//neck
glVertex2f(21.5+x,-17.0+y);
glVertex2f(21.6+x,-17.0+y);
glVertex2f(21.6+x,-18.5+y);
glVertex2f(21.5+x,-18.5+y);
glEnd();
glColor3f(0.97,0.45,0.84);
glBegin(GL_POLYGON);//body
glVertex2f(21.0+x,-18.5+y);
glVertex2f(22.1+x,-18.5+y);
glVertex2f(22.1+x,-20.0+y);
glVertex2f(21.0+x,-20.0+y);
glColor3f(0.59,0.137,0.985);
glBegin(GL_POLYGON);//skirt
glVertex2f(21.0+x,-20.0+y);
glVertex2f(22.1+x,-20.0+y);
glVertex2f(22.7+x,-21.0+y);
glVertex2f(20.5+x,-21.0+y);
glEnd();
glColor3f(0.0,0.0,0.0);
glPointSize(1.4);
```

```
glBegin(GL_POINTS);//eyes
glVertex2f(21.3+x,-17.30+y);
glVertex2f(21.8+x,-17.30+y);
glEnd();
glBegin(GL_LINES);//nose
glVertex2f(21.6+x,-17.6+y);
glVertex2f(21.6+x,-17.3+y);
glEnd();
glBegin(GL_LINES);//smile
glVertex2f(21.5+x,-17.8+y);
glVertex2f(21.8+x,-17.8+y);
glEnd();
glColor3f(0.3,0.15,0.1);
glBegin(GL_POLYGON);//hand 1
glVertex2f(21.0+x,-18.5+y);
glVertex2f(20.5+x,-20.0+y);
glVertex2f(21.0+x,-19.0+y);
glBegin(GL_POLYGON);//hand 2
glVertex2f(22.1+x,-18.5+y);
glVertex2f(22.7+x,-19.0+y);
glVertex2f(22.1+x,-19.0+y);
glEnd();
glColor3f(1,1,0.4);
glBegin(GL_POLYGON);//torch
glVertex2f(22.7+x,-19.0+y);
glVertex2f(22.4+x,-19.0+y);
glVertex2f(22.5+x,-19.7+y);
glVertex2f(23.0+x,-19.7+y);
glEnd();
glColor3f(c,c,c);
glBegin(GL_POLYGON);//torch light
glVertex2f(22.7+x,-19.2+y);
glVertex2f(22.4+x,-19.2+y);
glVertex2f(22.5+x,-19.7+y);
glVertex2f(23.0+x,-19.7+y);
glEnd();
glColor3f(c,c,c);
glEnable(GL_LINE_STIPPLE);
glLineStipple(1,0x00FF);
glBegin(GL_LINES);
glVertex2f(22.5+x,-19.2+y);
glVertex2f(24.0+x,-25.0+y);
glVertex2f(22.6+x,-19.2+y);
glVertex2f(25.5+x,-25.0+y);
glVertex2f(22.7+x,-19.2+y);
glVertex2f(27.0+x,-25.0+y);
glEnd();
}
```

- **Clockwise Function**

```
void spinclockwise(void)
{
w=0.3;u=0;v=1;b=0.5;c=1;d=1;e=0;
a=a+0.1;
if(a>40)
a-=100.0;
spin=spin-1.0;
if(spin<0)
spin=spin+360.0;
glutPostRedisplay();
}
```

- **Anticlockwise Function**

```
void anticlockwise(void )
{
u=0;w=.3;v=1;b=0.5;c=1;d=1,e=0;
if(a==40)
a=40;
a=a-0.1;
if(a<-100)
a+=100.0;
if(spin==360.0)
spin=spin-360;
spin=spin+1.0;
if(spin>360.0)
spin=spin-360.0;
glutPostRedisplay();
}
```

- **Key Function**

```
void mykey(unsigned char key,int m,int n)
{
if(key=='w') y+=.1,x-=0.1;
if(key=='s') y-=.1,x+=0.1;
glutPostRedisplay();
}
```

- **Menu Function**

```
void menu(int id )
{
switch(id)
{
case 1: u=v=w=b=0.45;c=d=e=1;
glutIdleFunc(display);
break;
case 2: glutIdleFunc(spinclockwise);
break;
case 3: glutIdleFunc(anticlockwise);
break;
case 4: glutIdleFunc(spinclockwise1);
break;
case 5: glutIdleFunc(anticlockwise1);
break;
case 6:exit(0);
}
}
```

Chapter 6

RESULTS

- **No Wind :**

The below screenshot is the first page of the project which displays the application when there is no wind movement.

Figure 6.1 No wind No electricity

- **Wind Clockwise or Anticlockwise**

On Clicking on the mouse button the windmill fans rotate either in clockwise or anticlockwise direction and light up the houses and street lights.

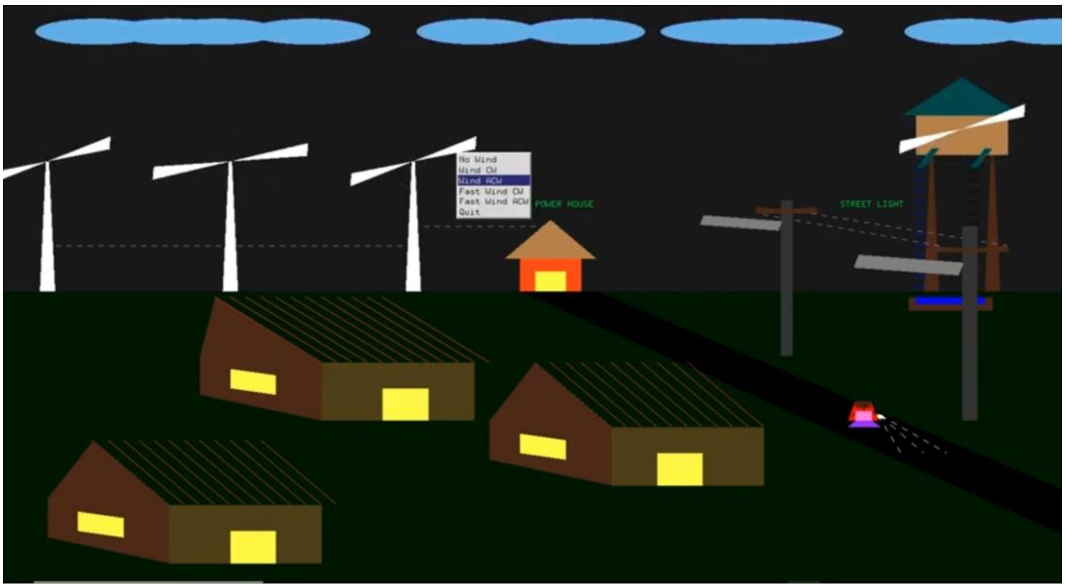

Figure 6.2 Wind Clockwise or Anticlockwise

- **Fast Wind Clockwise or Anticlockwise:**
  Faster movement of blades is observed either clockwise or anticlockwise.

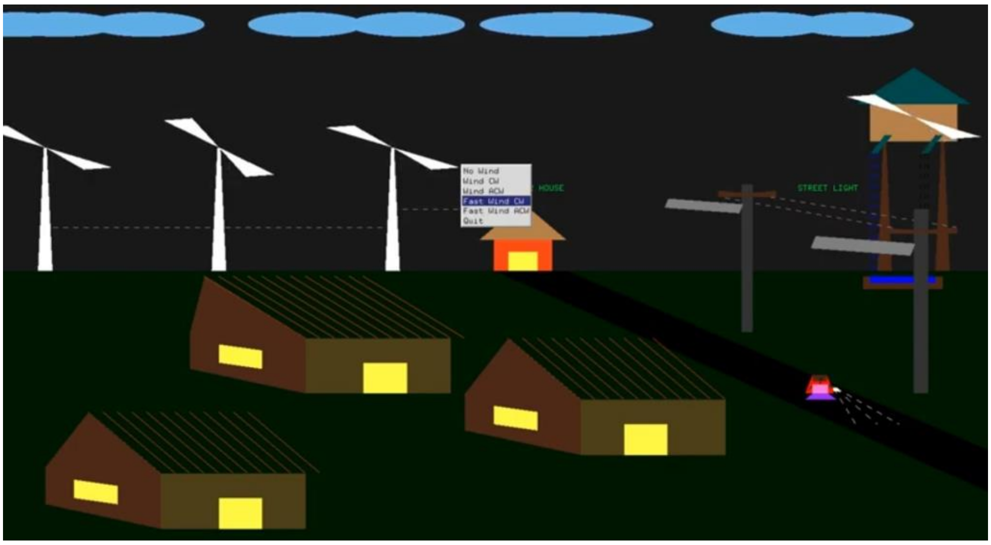

Figure 6.4 Fast Wind Clockwise or Anticlockwise

Chapter 7

CONCLUSION AND FUTURE ENHANCEMENTS

This application is implemented using openGL. Simple openGL functions have been used to draw windmills, powerhouse, turbines and show the generation of electricity when the fans are rotated. This project helps in analyzing how the different standard functions are used for inducing changes of rotation of fans in windmill(clockwise, anticlockwise). This package is very useful since it provides knowledge about various OpenGl functions. The code implemented for project works well to the best of knowledge. The functionality of all the modules and module level integration is found to be satisfactory. Best efforts are being made to make the model realistic and user friendly. The development of this application has given a good exposure to OpenGL and has helped in gaining knowledge about its techniques so as to apply it in developing animations and gaming.

In the future the application can be implemented as 3D model and be viewed on all sides of the screen. The graphics can be made in a way so as to portray that wind energy can be harnessed near oceans or tides. Windmill, turbines and power stations can be displayed along with their internal working.

BIBLIOGRAPHY

[1]Edward Angel: Interactive Computer Graphics: A Top Down Approach 5th Edition, Addison – Wesley, 2008

[2] Donald Hearn and Pauline Baker: OpenGL, 3rd Edition, Pearson Education, 2004

[3] http://www.opengl.org

[4] Wikipedia: Computer Graphics – https://en.wikipedia.org/wiki/ComputerGraphics