**VISVESVARAYA TECHNOLOGICAL UNIVERSITY**

Jnana Sangama, Belagavi-590018



**A**

**PROJECT REPORT**

**On**

***“WIND MILL STIMULATION “***

***Submitted in the partial fulfilment of Sixth Semester Mini Project Work***

**BACHELOR OF ENGINEERING**

**IN**

**COMPUTER SCIENCE AND ENGINEERING**

**Name : Pushpa B M USN: 1ME18CS048**

**Name : Anitha B USN: 1ME18CS010**

**Under the guidance of**

**Mrs. Dhanashree Kuthe**

**Assistant Professor, CSE Dept.**

**MSEC, Bangalore**



**Department of COMPUTER SCIENCE & Engineering**

**M S Engineering College**

NAAC Accredited, Affiliated to VTU, Belagavi, Approved By AICTE New Delhi,

Navarathna Agrahara, off Intl. Airport Road, Bangalore– 562110

2020-2021

**Department of Computer SCIENCE and Engineering**



**Certificate**

This is to certify that the project work entitled **“Wind Mill Stimulation”** carried out by **Pushpa B M (1ME18CS048)** and **Anitha B (1ME18CS010),** is a bonafide student of **M S ENGINEERING COLLEGE** submitted in partial fulfillment for the award of **Bachelor of Engineering** in **Computer Science and Engineering** of **Vishvesvaraya Technological University,** Belagavi, during the year **2020-2021**. It is certified that all corrections/suggestions indicated for internal assessment have been incorporated in the report, deposited in the department library. This project work report has been approved as it satisfies the academic requirements in respect of mini project work prescribed for **Bachelor of Engineering** Degree.

-------------------------------- -----------------------------------

**Signature of Guide Signature of HOD**

|  |  |  |
| --- | --- | --- |
| **Internal Examiner** | **Name** --------------------------- | **Signature** ------------------ |
| **External Examiner** | **Name** --------------------------- | **Signature** ------------------ |
|  |  |  |
| **Exam Date: -----------** |  |  |

**ABSTRACT**

In this project we designed the simulation of windmill using OpenGL. We used transformation functions like translate and rotate functions to design blades of the windmill.

We used many OpenGL inbuilt function to design the structure of windmill.

This projectconsist of many user defined function such as increasing windmill fan speed, decreasing windmill fan speed, side views, front and back views, custom angle of rotation of entire windmill structure.

It provides several options which can be interacted through menus. The user can also interact with program through mouse, keyboard functions.

We can rotate the entire windmill with respect to its axis using the arrow keys of keyboard. It can be rotate through 3600.

**ACKNOWLEDGEMENT**

I express my gratitude to our institution and management for providing us with good infrastructure, laboratory, facilities and inspiring staff, and whose gratitude was of immense help in completion of this seminar successfully.

I express my sincere gratitude to our principal, **Dr. K S Badarinarayan** for providing me required environment and for his valuable suggestion.

My sincere thanks to Dr.Malathesh**,** Head of the Dept. Computer Science and Engineering, M S Engineering College for his valuable support and for rendering us resources for this seminar work.

I express my gratitude to **Prof.Dhanashree Kuthe,** Assistant Professor, Dept. Computer Science and EngineeringM S Engineering College who guided me with valuable suggestions in completing this seminar at every stage.

My gratitude thanks should be rendered to many people who helped me in all possible ways.

**Name: Pushpa B M USN: 1ME18CS048**

**Name: Anitha B USN:1ME18CS010**

**TABLE OF CONTENTS**

|  |  |  |
| --- | --- | --- |
| **SL.No** | **TOPICS** | **Page Number** |
| **Chapter 1** | **INTRODUCTION** | 6-6 |
|  | 1.1. Computer Graphics | 7-8 |
|  | 1.2. History of Computer Graphics |  |
|  | 1.3. Image Types |  |
|  | 1.3. Application of Computer Graphics |  |
|  | 1.4. Introduction to OpenGL |  |
|  | 1.5. Introduction to GLUT | **8-10** |
| **Chapter 2** | **System Requirement** |  |
|  | 2.1. Software requirements |  |
|  | 2.2. Hardware requirements |  |
| **Chapter 3** | **About the Project** | **11-14** |
|  | 3.1. Introduction |  |
|  | 3.2. User defined functions | **15-16** |
|  | 3.3. Creation of “Unidentified Flying Objects” |  |
| **Chapter 4** | **Design** | **17-20** |
| **Chapter 5** | **Implementation** |  |
|  | 5.1. Built in functions |  |
|  | 5.2. Source Code | **20-21** |
| **Chapter 6** | **Testing** |  |
|  | 6.1. Unit Testing |  |
|  | 6.2. System Testing |  |
| **Chapter 7** | **Snapshots** |  |
|  |  |  |
| **Chapter 8** | **Conclusion** |  |
| **Chapter 9** | **References** | **47** |
|  |  |  |

**Chapter 1**

**INTRODUCTION**

**1.1 Introduction to computer graphics and opengl**

**COMPUTER GRAPHICS** is concerned with all aspects of producing pictures or images using a computer. The field began humbly almost 50 years ago, with the display of a few lines on a cathode-ray tube (CRT); now, we can create images by computer that are indistinguishable from photographs of real objects. We routinely train pilots with simulated airplanes, generating graphical displays of a virtual environment in real time. Feature-length movies made entirely by computer have been successful, both critically and financially. Massive multiplayer games can involve tens of thousands of concurrent participants [1][2].

VISUALIZATION is any technique for creating images, diagrams or animations to communicate a message.

**Application of Computer Graphics**

* Display of information
* Design
* Simulation and animation
* User interfaces

**Display of Information**

* Classical graphics techniques arose as a medium to convey information among people.
* We have computer plotting packages that provide a variety of plotting techniques and color tools that can handle multiple large data sets.
* The field of information visualization is becoming increasingly more important as we have to deal with understanding complex phenomena from problems in bioinformatics to detecting security threats.

**Design**

* Professional such as engineering and architecture are concerned with design.
* The use of interactive graphical tools in computer-aided design (CAD) pervades fields including as architecture, mechanical engineering, the design of very-large-scale integrated (VLSI) circuits, and the creation of characters for animations.

**Simulation and Animation**

* Once graphics systems evolved to be capable of generating sophisticated images in real time, engineers and researchers began to use them as simulators.
* One of the most important uses has been in the training of pilots. Graphical flight simulators have proved to increase safety and to reduce training expenses. The use of special VLSI chips has led to a generation of arcade games s as sophisticated as flight simulators.

**User Interfaces**

* Interaction with computers has become dominated by a visual paradigm that includes windows, icons, menus, and plotting device, such as mouse.
* User interfaces demonstrate the variety of the tools available in high level modeling packages and the interactive devices the user can employ in modeling geometric object.

**1.1.1 Image Types**

* **2D computer graphics:**

2D computer graphics are the computer-based generation of digital images mostly from two-dimensional models, such as 2D geometric models, text, and digital images, and by techniques specific to them.2D computer graphics are mainly used in applications that were originally developed upon traditional printing and drawing technologies, such as typography, cartography, technical drawing, advertising. Two-dimensional models are preferred, because they give more direct control of the image than 3D computer graphics, whose approach is more akin to photography than to typography.

There are two approaches to 2D graphics: vector and raster graphics.

* Pixel art: Pixel art is a form of digital art, created through the use of raster graphics software, where images are edited on the pixel level.
* Vector graphics: Vector graphics formats are complementary to raster graphics, which is the representation of images as an array of pixels, as it is typically used for the representation of photographic images.
* **3D computer graphics:**

With the birth of the workstation computers(like LISP machines,paintbox computers and Silicon Graphics workstations)came the 3D computer graphics.3D computer graphics in contrast to 2D computer graphics are graphics that use a three-dimensional representation of geometric data that is stored in the computer for the purposes of performing calculations and rendering 2D images.

**Some major advances in 3D computer graphics since then have been:**

* Flat shading: A technique that shades each polygon of an object based on the polygon's "normal" and the position and intensity of a light source.
* Gouraud shading: Invented by Henri Gouraud in 1971, a fast and resource-conscious technique used to simulate smoothly shaded surfaces by interpolating vertex colors across a polygon's surface.
* Texture mapping: A technique for simulating surface detail by mapping images(textures) onto polygons.
* Phong shading: Invented by Bui Toc Phong, a smooth shading technique that approximates curved-surface lighting by interpolating the vertex normals of a polygon across the surface; the lighting model includes glossy reflection with a controlable level of gloss.
* Bump mapping: Invented by Jim Blinn, a normal-perturbation technique used to simulate bumpy or wrinkled surfaces.
* Raytracing: A method based on the physical principles of geometric optics that can simulate multiple reflections and transparency.
* Radiosity: a technique for global illumination that uses radiative transfer theory to simulate indirect (reflected) illumination.

**1.1.2Applications of computer graphics**

Some of the applications of computer graphics are listed below:

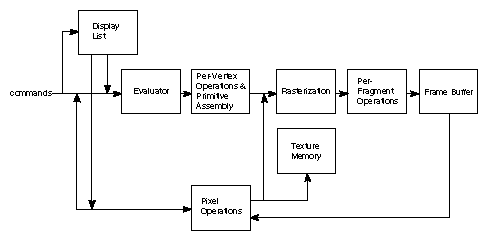
* Computational biology
* Computational physics
* Computer-aided design
* Computer simulation
* Digital art
* Education
* Graphic design
* Video Games
* Virtual reality
* Web design

**1.1.3 OpenGL**

As a software interface for graphics hardware, OpenGL's main purpose is to render two- and three-dimensional objects into a frame buffer. These objects are described as sequences of vertices (which define geometric objects) or pixels (which define images). OpenGL performs several processing steps on this data to convert it to pixels to form the final desired image in the frame buffer.

## Basic OpenGL Operation

The figure shown below gives an abstract, high-level block diagram of how OpenGL processes data. In the diagram, commands enter from the left and proceed through what can be thought of as a processing pipeline. Some commands specify geometric objects to be drawn, and others control how the objects are handled during the various processing stages.



**Figure 1.1 OpenGL Block Diagram**

As shown by the first block in the diagram, rather than having all commands proceeds immediately through the pipeline, you can choose to accumulate some of them in a *display list* for processing at a later time. The *evaluator* stage of processing provides an efficient means for approximating curve and surface geometry by evaluating polynomial commands of input values. During the next stage, *per-vertex operations and primitive assembly*, OpenGL processes geometric primitives—points, line segments, and polygons, all of which are described by vertices. Vertices are transformed and lit, and primitives are clipped to the viewport in preparation for the next stage.*Rasterization* produces a series of frame buffer addresses and associated values using a two-dimensional description of a point, line segment, or polygon.

Each *fragment* so produced is fed into the last stage, *a per-fragment operation, which performs* the final operations on the data before it's stored as pixels in the *frame buffer*. These operations include conditional updates to the frame buffer based on incoming and previously stored z-values (for z-buffering) and blending of incoming pixel colors with stored colors, as well as masking and other logical operations on pixel values.

Input data can be in the form of pixels rather than vertices. Such data, which might describe an image for use in texture mapping, skips the first stage of processing described above and instead is processed as pixels, in the *pixel operations* stage. The result of this stage is either stored as *texture memory*, for use in the rasterization stage, or rasterized and the resulting fragments merged into the frame buffer just as if they were generated from geometric data.All elements of OpenGL state, including the contents of the texture memory and even of the frame buffer, can be obtained by an OpenGL application.

**1.1.4GLUT**

GLUT is the OpenGL Utility Toolkit, a window system independent toolkit for writing OpenGL programs. It implements a simple windowing application programming interface (API) for OpenGL. GLUT makes it considerably easier to learn about and explore OpenGL programming. GLUT provides a portable API so you can write a single OpenGL program that works on both Win32 PCs and X11 workstations.

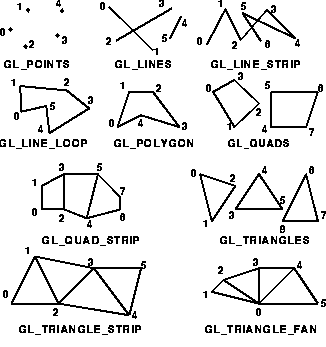
GLUT is designed for constructing small to medium sized OpenGL programs. While GLUT is well-suited to learning OpenGL and developing simple OpenGL applications, GLUT is not a full-featured toolkit so large applications requiring sophisticated user interfaces are better off using native window system toolkits like Motif. GLUT is simple, easy, and small. My intent is to keep GLUT that way.

**The GLUT library supports the following functionality:**

* Multiple windows for OpenGL rendering.
* Call back driven event processing.
* An `idle' routine and timers.
* Utility routines to generate various solid and wire frame objects.
* Support for bitmap and stroke fonts.
* Miscellaneous window management functions.

# OpenGL Primitives

# The programmer is provided the following primitives for use in constructing geometric objects.



**1.2 Overview of the project**

In this project we designed the simulation of windmill using OpenGL. We used transformation functions like translate and rotate functions to design blades of the windmill. We used many OpenGL inbuilt function to design the structure of windmill.

This projectconsist of many user defined function such as increasing windmill fan speed, decreasing windmill fan speed, side views, front and back views, custom angle of rotation of entire windmill structure.

It provides several options which can be interacted through menus. The user can also interact with program through mouse, keyboard functions. The options provided by the menu are views like side view, back view, front view, custom view. Using mouse, if we click left side it rotates to left and on successive clicking speed increases, if we click right button speed decreases and on successive clicking, it turns rotating towards right and vice versa.

We can rotate the entire windmill with respect to its axis using the arrow keys of keyboard. It can be rotate through 3600

**1.3 Aim of the project**

The aim of the project is to develop a suitable graphics package to simulate the Windmill and to implement the skills learned in Interactive Computer Graphics and Visualization theory, using OpenGL.

**REQUIREMENT SPECIFICATION**

**2.1 Functional Requirements**

These are statements of services the system should provide and do how the system reacts to particular inputs and how the system should behave in particular situations i.e. it describes what system should do.

* Simulation of Windmill should be implemented by using keyboard and mouse.
* Sequence of menus should be displayed on pressing middle mouse button as shown
* Side View 1
* Side View 2
* Back View
* Front View
* Custom View
* On pressing left mouse button, the windmill wheel starts rotating and on further clicks the speed of rotation increases.
* On pressing right mouse button, the rotating speed of the windmill wheel slows down and finally comes to halt.
* Simulation of Windmill shall also be done using the keys-

Right Arrow and Left Arrow keys can be used to rotate the whole Windmill Structure

**2.2Non Functional Requirements**

These are constraints on the services or functions offered by the system. They include timing constraints, constraints on the development process and standards. These requirements often apply to the system as a whole. They don’t usually just apply to individual system features or services. Therefore, they may specific system performance, security, availability, and other emergent properties.

* The implementation of windmill simulation shall render in real-time.
* This project shall maintain a simple user interface.

**2.3Software requirement and hardware requirements**

**2.3.1 Software Requirements**

* Operating System : Windows 98/XP or Higher
* Programming Language : C,C++
* Microsoft Visual Studio 2005 or higher: This Software package containing visual basics in C++ language is required.
* Toolkit : GLUT Toolkit, VC++

**2.3.2 Hardware Requirements**

This package has been developed on:

* Processor : Pentium Processor
* Processor Speed : 333 MHz
* RAM : 32 MB or Higher
* Graphics Card : 512MB
* Monitor : Color
* Keyboard : Low Profile, Dispatch able Type
* I/O Parts : Mouse, Monitor

**Chapter 3**

**DESIGN**

**3.1 Initialization**

Initialize the interaction with the windows. Initialize the display mode- double buffer and depth buffer. Initialize the various callback functions for drawing and redrawing the polygon, for mouse and keyboard interface, for movement of the image in different directions. Initialize the window position and size and create the window to display the output.

**3.2 Flow of control**

The flow of control in the above flow chart is respected to the Texture Package. For any of the program flow chart is compulsory to understand the program. We consider the flow chartfor the texture project in which the flow starts from start and proceeds to the main function after which it comes to the initialization of call back functions and further it proceeds to mouse and keyboard functions after all the function,the flow comes to quit which is the end of the flow chart.

**FLOWCHART**

MAIN

INITIALIZE CALLBACK FUNCTIONS

MAIN SCREEN DISPLAYED

MOUSE BUTTON

RIGHT BUTTON

MIDDLE BUTTON

LEFT BUTTON

SIDE VIEW 2

BACK VIEW

FRONT VIEW

CUSTOM VIEW

SIDE VIEW 1

EVENTS

**Fig 3.1: Flowchart for Simulation of Windmill**

**Chapter 4**

**IMPLEMENTATION**

**4.1 Source Code**

#include <GL/glut.h>

#include<stdio.h>

#include<conio.h>

#include<process.h>

#define M\_SIDE1 20

#define M\_SIDE2 21

#define M\_BACK 22

#define M\_FRONT 23

#define M\_CUSTOM 24

#define SIZE 500

float x\_angle = 0.0;

float y\_angle = 0.0;

float z\_angle = 0.0;

float camera\_angle=0.0;

float c=1.0;

GLfloat pos[] = { 0.0, 0.0, -10.0, 1.0 };

GLfloat white[] = { 2.5, 2.5, 6.0, 6.0 };

GLfloat red[] = { 0.7, .4, 0.0, 1.0 };

GLfloat deep\_blue[] = { 0.3, 0.3, 0.9, 1.0 };

GLfloat shiny[] = { 50.0 };

GLfloat dull[] = { 0.0 };

GLUquadricObj \*Cylinder;

enum { X, Y, Z } axis = X;

void change\_view (int sel)

{

switch (sel)

{

case M\_CUSTOM:{printf("ENTER VIEWANGLE:");

scanf("%f",&camera\_angle);};

break;

case M\_SIDE1: {camera\_angle=90;}

break;

case M\_SIDE2: camera\_angle=-90;

break;

case M\_BACK: camera\_angle=180;

break;

case M\_FRONT:camera\_angle=0;

default: break;

}

}

void initialize\_menu (void)

{

glutCreateMenu(change\_view );

glutAddMenuEntry("SIDE VIEW 1", 20);

glutAddMenuEntry("SIDE VIEW 2", 21);

glutAddMenuEntry("BACK VIEW", 22);

glutAddMenuEntry("FRONT VIEW", 23);

glutAddMenuEntry("CUSTOM VIEW", 24);

glutAttachMenu(GLUT\_MIDDLE\_BUTTON);

}

void mouse\_button (int button, int state, int x, int y)

{

if (button == GLUT\_LEFT\_BUTTON )

{

axis=Z; c=c+0.4;

printf("WIND SPEED INCREASE\t SPEED=%fKm/Hr\n\n",c\*1.5);

glutPostRedisplay();

}

else if (button == GLUT\_RIGHT\_BUTTON )

{

axis = Z; c=c-0.4;

printf("WIND SPEED DECREASE\t SPEED=%fKm/Hr\n\n",c\*1.5);

glutPostRedisplay();

}

}

void spin(void)

{

switch (axis)

{

case X: x\_angle += 1.0;

break;

case Y: y\_angle += 1.0;

break;

case Z: z\_angle += c ;

break;

default: break;

}

glutPostRedisplay();

}

void display (void)

{

Cylinder = gluNewQuadric();

gluQuadricDrawStyle( Cylinder, GLU\_FILL);

gluQuadricNormals( Cylinder, GLU\_SMOOTH);

gluQuadricOrientation( Cylinder, GLU\_OUTSIDE );

glClear(GL\_COLOR\_BUFFER\_BIT | GL\_DEPTH\_BUFFER\_BIT);

glMatrixMode(GL\_MODELVIEW);

glLoadIdentity();

glEnable(GL\_TEXTURE\_2D);

//Bottom

glTexParameteri(GL\_TEXTURE\_2D,GL\_TEXTURE\_MIN\_FILTER,

GL\_NEAREST);

glTexParameteri(GL\_TEXTURE\_2D, GL\_TEXTURE\_MAG\_FILTER,

GL\_NEAREST);

glMaterialfv(GL\_FRONT, GL\_AMBIENT, white);

glBegin(GL\_QUADS);

glNormal3f(0.0, 1.0f, 0.0f);

glTexCoord2f(0.0f, 0.0f);

glVertex3f(-25.0,-25.0,-44);

glTexCoord2f(0.0f, 1.0f);

glVertex3f(-25.0,25.0,-44);

glTexCoord2f(1.0f, 1.0f);

glVertex3f(25.0,25.0,-44);

glTexCoord2f(1.0f, 0.0f);

glVertex3f(25.0,-25.0,-44);

glEnd();

glDisable(GL\_TEXTURE\_2D);

glRotatef(camera\_angle,0.0,1.0,0.0);

gluCylinder(Cylinder,.4,.4,4,16,20);

glMaterialfv(GL\_FRONT\_AND\_BACK, GL\_AMBIENT\_AND\_DIFFUSE, red);

glMaterialfv(GL\_FRONT\_AND\_BACK, GL\_SPECULAR, red);

glMaterialfv(GL\_FRONT\_AND\_BACK, GL\_SHININESS, shiny);

glPushMatrix();

glutSolidTorus (1.4, 1.4, 6, 6);

glutSolidCube(2.5);

glPushMatrix();

glTranslatef(0.0,-2.0,0.0);

glMaterialfv(GL\_FRONT\_AND\_BACK, GL\_DIFFUSE, red);

//material property for the base of the windmill

glMaterialfv(GL\_FRONT\_AND\_BACK, GL\_SPECULAR, red);

glPushMatrix();

glRotatef(90.0,1.0,0.0,0.0);

glTranslatef(0.0,0.0,-2.0);

gluCylinder(Cylinder,1.0,1.5,27,50,50);

glPopMatrix();

glPopMatrix();

glRotatef(z\_angle, 0.0, 0.0, 1.0);

glMaterialfv(GL\_FRONT\_AND\_BACK, GL\_AMBIENT\_AND\_DIFFUSE, red);

glMaterialfv(GL\_FRONT\_AND\_BACK, GL\_SPECULAR, white);

glPushMatrix();

glTranslatef(0.0,0.0,1.5);

glutSolidCone(1.5,2.5,50,50);

glPopMatrix();

glPushMatrix(); //blade 1

glTranslatef(0.0,0.0,2.2);

glRotatef(90.0,1.0,0.0,0.0);

glPushMatrix();

glRotatef(120,0.0,1.0,0.0);

glutSolidCone(0.9, 16.0, 15, 15);

glPopMatrix();

glPopMatrix();

glPushMatrix(); //blade 2

glTranslatef(0.0,0.0,2.2);

glRotatef(90.0,1.0,0.0,0.0);

glPushMatrix();

glRotatef(-120,0.0,1.0,0.0);

glutSolidCone(0.9, 16.0, 15, 15);

glPopMatrix();

glPopMatrix();

glPushMatrix(); //blade 3

glTranslatef(0.0,0.0,2.2);

glRotatef(90.0,1.0,0.0,0.0);

glutSolidCone(0.9, 16.0, 15, 15);

glPopMatrix();

glLightfv(GL\_LIGHT1, GL\_POSITION, pos);

glutSwapBuffers();

}

void init (void)

{

glMatrixMode(GL\_PROJECTION);

glLoadIdentity();

glOrtho(-25.0, 25.0, -25.0, 25.0, -250.0, 250.0);

glEnable(GL\_LIGHTING);

glEnable(GL\_LIGHT0);

glEnable(GL\_LIGHT1);

glEnable(GL\_NORMALIZE);

}

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

{

switch (key)

{

case GLUT\_KEY\_LEFT: {axis = X; camera\_angle--;

printf("\n\nVIEW ANGLE=%f", camera\_angle);};

glutPostRedisplay();

break;

case GLUT\_KEY\_RIGHT: {axis = Y; camera\_angle++;

printf("\n\nVIEW ANGLE=%f", camera\_angle);};

glutPostRedisplay();

break;

case GLUT\_KEY\_UP: c=c+0.4;

glutPostRedisplay();

break;

case GLUT\_KEY\_END: exit(0);

case GLUT\_KEY\_DOWN:{c=c-0.4;};

glutPostRedisplay();

break;

default: break;

}

}

void reshape (int width, int height)

{

GLfloat w, h;

glViewport(0, 0, width, height);

glMatrixMode(GL\_PROJECTION);

glLoadIdentity();

if (width > height)

{

w = (25.0 \* width) / height;

h = 25.0;

}

else

{

w = 25.0;

h = (25.0 \* height) / width;

}

glOrtho(-w, w, -h, h, -250.0, 250.0);

glutPostRedisplay();

}

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

{

glutInit(&argc, argv);

glutInitDisplayMode(GLUT\_RGBA | GLUT\_DOUBLE);

glutInitWindowSize(SIZE, SIZE);

glutInitWindowPosition(100, 50);

glutCreateWindow("SIMULATION OF WINDMILL");

glutIdleFunc(spin);

glutDisplayFunc(display);

glutSpecialFunc(special);

glutMouseFunc(mouse\_button);

initialize\_menu();

glutReshapeFunc(reshape);

init();

glEnable(GL\_DEPTH\_TEST);

glutMainLoop();

}

**4.2 OpenGL Functions:**

This Project is implemented by using the below OpenGL Functions[4].

**Window Management functions**

Five routines perform tasks necessary to initialize a window.

* **glutInit**(int \**argc*, char \*\**argv*)

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

* **glutInitDisplayMode**(unsigned int *mode*)

Requests a display with the properties in mode. The value of mode is determined by the logical OR

* **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 on display. The string title can be used to label the window.

* **Void glutMainLoop()**

Cause the program to enter an event processing loop.

**Transformation Functions**

* **void glRotatef( GLfloat angle, GLfloat x, GLfloat y, GLfloat z);**

The glRotated and glRotatef functions multiply the current matrix by a rotation matrix.

* **void glTranslate( TYPE x, TYPE y, TYPEz);**

The glTranslated and glTranslatef functions multiply the current matrix by a translation matrix.

**Call Back Functions**

* **glutDisplayFunc(void (\* *func*)(void))**

Whenever GLUT determines the contents of the window need to be redisplayed, the callback function registered by **glutDisplayFunc()** is executed.

* **glutReshapeFunc**

glutReshapeFunc sets the reshape callback for the current window.

SYNTAX:

void glutReshapeFunc(void (\*func)(int width, int height));

* **void glutIdleFunc(void (\*func)(void));**

glutIdleFunc sets the global idle callback. This function is invoked when there are no other events. Its default is the NULL function pointer .Atypical use of idle function is to continue to generate graphical primitives through a display function while nothing is happening.

**Interactive Functions**

* **void glutKeyboardFunc(void (\*func)(unsigned char key, int x, int y));**

Registers the keyboard callback function func. The callback function returns the ASCII code of the key pressed and the position of the mouse.

* **void glutMouseFunc(void (\*func)(int button, int state, int x, int y));**

Registers the mouse callback function func. The callback function returns the button(GLUT\_LEFT\_BUTTON,GLUT\_RIGHT\_BUTTON,GLUT\_MIDDLE\_BUTTON), the state of the button after the event (GLUT\_UP,GLUT\_DOWN) and the position of the mouse with respect to the top left corner of the window.

* **int glutCreateMenu(void (\*func)(int value));**

Returns an identifier for a top-level menu and registers the callback function f that returns an integer value corresponding to the menu entry selected. glutCreateMenu creates a new pop-up menu and returns a unique small integer identifier.

* **void glutAddMenuEntry(char \*name, int value);**

This function adds an entry with the string name displayed to the current menu. Value is returned to the menu callback when the entry is selected

* **void glutAttachMenu(int button);**

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

* **Void glPushMatrix() & void glPopMatrix**

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

* **void glOrtho(GLdouble left, GLdouble right, GLdouble top, GLdouble bottom, GLdouble near, GLdouble far)**

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

**Chapter 5**

**TESTING**

Testing in general means validation and verification. It shows that the system conforms to its specifications and system meets all expectation of the user.

**Table 5.1: Test case for menu option**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Sl no. | Test case description | Expected Result | Actual Result | Remarks |
| 1. | Click the MIDDLE mouse button on the display screen | Menu with   * Side View 1 * Side View 2 * Back View * Front View * Custom View   Should be displayed | Menu with   * Side View 1 * Side View 2 * Back View * Front View * Custom View   Is displayed | Pass |
| 2. | Click on Side View 1 option | A Side View of the Windmill structure (Right Side)  Should be displayed | A Side View of the Windmill structure  Is displayed | Pass |
| 3. | Click on Side View 2 option | A Side View of the Windmill structure (Left Side)  Should be displayed | A Side View of the Windmill structure  Is displayed | Pass |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 4. | Click on Back View option | A Back Side View of the Windmill structure  Should be displayed | A Back Side View of the Windmill structure  Is displayed | Pass |
| 5. | Click on Front View option | A Front Side View of the Windmill structure  Should be displayed | A Front Side View of the Windmill structure  Is displayed | Pass |
| 6 | Click on Custom View option | Option to enter any custom angle.  Should be displayed  The Computer asks for any custom angle, on entering the values, it rotates the Windmill Structure to that angle. | Option to enter any custom angle.  Is displayed  The Computer asks for any custom angle, on entering the value, it rotates the Windmill Structure to that angle. | Pass |

**Table 5.2: Test case for mouse**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Slno. | Test case Description | Expected result | Actual Result | Remarks |
| 1. | Click on left button on display screen. | The Speed of the Windmill wheel should increase. | Increases the speed of the Windmill wheel. | Pass |
| 2. | Click on right button on display screen. | The Speed of the Windmill wheel should decrease. | Decreases the speed of the Windmill wheel. | Pass |

**Table 5.3: Test case for keyboard**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Sl no. | Test case Description | Expected result | Actual result | Remarks |
| 1. | Press Left Arrow key on the display screen. | Free anti-clockwise movement the Windmill Structure | Able to move the windmill structure anti-clockwise freely | Pass |
| 2. | Press Right Arrow key on the display screen. | Free clockwise movement the Windmill Structure | Able to move the windmill structure clockwise freely | Pass |

**Chapter 6**

**RESULTS**

**6.1 Initial screen**

****

Fig 6.1: Initial screen

The above diagram shows the snapshot having the windmill. The objects are of windmill shape and are placed towards the right of the display window. And the display window is of size (600,600).

**6.2Screen displaying the menus**



Fig 6.2: Screen displaying the menus.

The above diagram shows the snapshothavingoptions such as Data\_Structuresand displaying the menu which consists of.:-

* Side View 1
* Side View 2
* Back View
* Front View
* Custom View

**6.3 Back View**

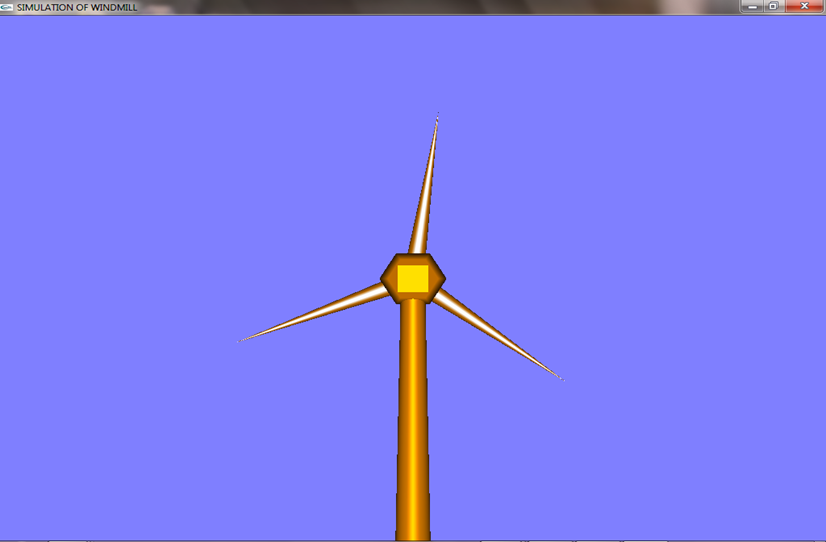


Fig 6.3: Back View

The above diagram shows the snapshot about the option of Back View , in which the Back View of the Windmill structure is displayed .

**6.4 Side View 1**

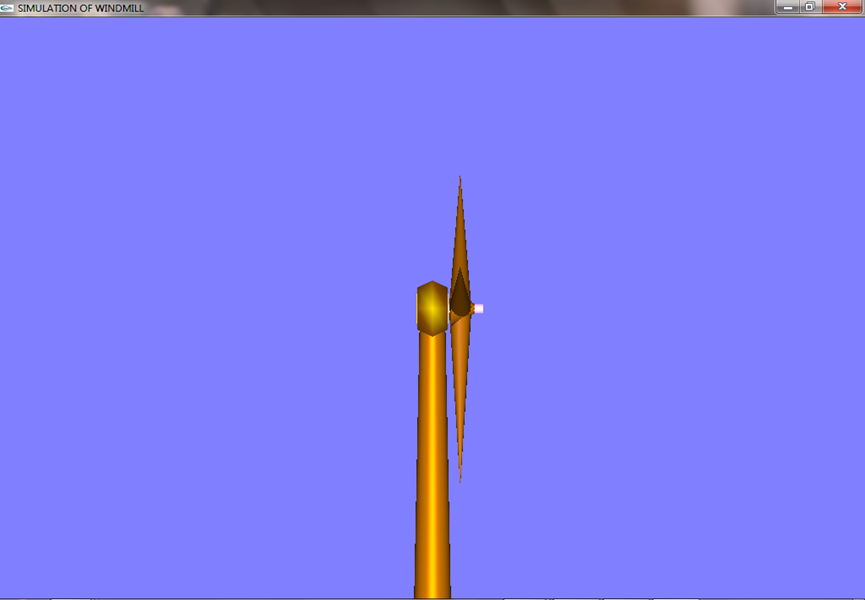
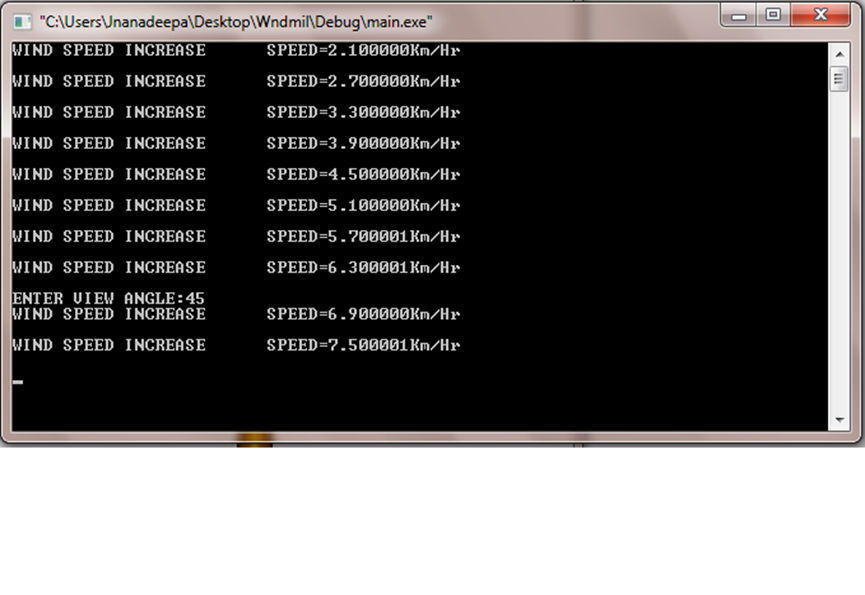


Fig 6.4: Side View 1

The above diagram shows the snapshot about the option of Back View , in which the Side View of the Windmill structure is displayed

**6.5 Speed Tracking on Mouse Clicks**

Fig 6.5: Speed tracking using Mouse Clicks

The above diagram shows the snapshot about the simulation of the windmill, When the wheel of the windmill is made to rotate using the Left and Right Button of the mouse.

When the Left Button of the mouse is clicked, it starts rotating and on further clicks the speed of the rotation increases.

When the Right Button of the mouse is clicked, the rotation speed decreases and finally comes to halt.

**6.6 Free Movement using Keys**



Fig 6.6: Free Movement using Keys

The above diagram shows the snapshot about free movement of the Windmill structure using the keys. The Right Arrow key is used to rotate the angle of the windmill structure clockwise, whereas the Left Arrow Key is used to rotate the angle of windmill structure anti-clockwise.

**CHAPTER 7**

**CONCLUSION**

**Simulation of Windmill** is a designed and implemented using a graphics software system called **OpenGL** which has became a widely accepted standard for developing graphic application. Using openGL functions user can create geometrical objects and can use **translation, rotation**, scaling with respect to the co-ordinate system.

The project Visual Transformation Techniques using openGL is based on Rotation and Translation processes using shading effects and is demonstrated using Visual C++.

The development of the **Simulation of Windmill** project has given us a good exposure to OpenGL by which we have learnt some of the technique which help in development of animated pictures, gaming. Hence it is helpful for us even to take up this field as our career too and develop some other features in OpenGL and provide as a token of contribution to the graphics world.

**Simulation of Windmill** consist of many user defined function such as increasing windmill fan speed, decreasing windmill fan speed, side views, front and back views, custom angle of rotation of entire windmill structure. All these function makes this project an example of animation in **OpenGL.**

**CHAPTER 8**

**FUTURE ENHANCEMENT**

This project has been designed using C++, which works on the windows platform. The project can be designed using other languages and better graphical interfaces. The following features could have been incorporated.

* Circular and priority queues can be implemented.
* Different shading effects can be added.
* We can give transparency and fogging to the objects.

**CHAPTER 9**

**BIBLIOGRAPHY**

* Edward Angel:
  + - Interactive Computer Graphics: A Top-Down Approach with OpenGL
* F.S. Hill, Jr.:
  + - Computer Graphics Using OpenGL
* Donald Hearn and Pauline Baker:
  + - Computer Graphics- OpenGL Version
* Richard S. Wright, Jr. and Michael Sweet:
  + - The OpenGL Super Bible
* Dave Shreiner, Mason Woo, Jackie Neider and Tom Davis:
  + - OpenGL Programming Guide
* Shalini Govil-Pai:
  + - Principles of Computer Graphics: Theory and Practice Using OpenGL
* Websites:
  + - <http://www.OpenGl.org>
    - [www.opengl.org](http://www.opengl.org)
    - [www.sourcecode.com](http://www.sourcecode.com)