**Table of Contents**

[ACKNOWLEDGEMENT V](#_TOC_250011)

[ABSTRACT VI](#_TOC_250010)

TABLE OF CONTENTS VII

LIST OF FIGURES III

LIST OF TABLES IV

CHAPTER 1

INTRODUCTION VII

* 1. [Aim VII](#_TOC_250009)
  2. [Introduction to Open GL **VII**](#_TOC_250008)
  3. [Project related concepts **VIII**](#_TOC_250007)
  4. Interface **IX**

[CHAPTER 2](#_TOC_250006)

REQUIREMENT SPECIFICATION X

* 1. [Software requirements **X**](#_TOC_250005)
  2. [Hardware requirements **X**](#_TOC_250004)

[CHAPTER 3](#_TOC_250003)

DESIGN XII

* 1. Window design **XI**
  2. Explanation **XI**
  3. Flow Diagram **XII**

[CHAPTER 4](#_TOC_250002)

[IMPLEMENTATION XII](#_TOC_250001)

* 1. [Functions Used **XII**](#_TOC_250000)
  2. Functions explanation **XIV**

#### CHAPTER 5 SOURCE

#### CODE XV

#### CHAPTER 6 TEST

**CASES XLIV**

**CHAPTER 7**

**SNAPSHOTS XLV**

**CHAPTER 8**

**CONCLUSION & FUTURE WORK LI**

**REFERENCES LII**

**List of Figures**

|  |  |  |
| --- | --- | --- |
| **Figure No.** | **Figure Name** | **Page No.** |
| Figure 1.1 | Flow of control | 10 |
| Figure 7.1 | Day View | 25 |

|  |  |  |
| --- | --- | --- |
| Figure 7.2 | Night View | 25 |

**List of Tables**

|  |  |  |
| --- | --- | --- |
| **Table No.** | **Table Name** | **Page No.** |
| Table 6.1 | Test cases for Mouse interface | 15 |
| Table 6.2 | Test cases for Keyboard interface | 16 |

# ACKNOWLEDGEMENT

The completion of Mini project brings with and sense of satisfaction, but it is never completed without thanking the persons who are all responsible for its successful completion. First and foremost I wish to express our deep sincere feelings of gratitude to my Institution, **Sai Vidya Institute of Technology**, for providing mean opportunity to do our education.

I would like to thank the **Management** and **Prof. M R Holla,** Director, Sai Vidya Institute of Technology for providing the facilities.

I extend my deep sense of sincere gratitude to **Dr. H S Ramesh Babu**, Principal, Sai Vidya Institute of Technology, Bengaluru, for having permitted to carry out the Mini project work on **“A SCENIC VIEW OF A WINDMILL”** successfully**.**

I am thankful to **Prof. A M Padma Reddy**, Additional Director and Professor, Department of Computer Science and Engineering, Sai Vidya Institute of Technology, for his constant support and motivation.

I express my heartfelt sincere gratitude to **Dr. K. Ananthapadmanabha,** Professor and Head, Department of Computer Science and Engineering, Sai Vidya Institute of Technology, Bengaluru, for his valuable suggestions and support.

I express my sincere gratitude to **Dr. Sangeetha V,** Associate Professor, Department of ISE**,** and **Prof. Sunil G L,** Assistant Professor, Department of CSE, Sai Vidya Institute of Technology, Bengaluru, for their constant support in completing the Mini project.

Finally, I would like to thank all the Teaching, Technical faculty and supporting staff members of Department of Information Science and Engineering, Sai Vidya Institute of Technology, Bengaluru, for their support.

Satwik Ram K 1VA17CS047

# ABSTRACT

Computer Graphics is concerned with all aspects of producing pictures or images using a computer. Here, a particular graphics system, OpenGL which has become a widely accepted standard for developing graphics applications. A 2D OpenGL animation that demonstrates the use of glPushMatrix and glPopMatrix to implement hierarchical modelling.

It is being coded in such a way that Windmill keeps rotating. The rotation speed can be varied .A car keeps moving across the screen from left to right and one car can be stopped. The day and night view can be experienced.

# CHAPTER 1 INTRODUCTION

## Aim

The aim of this project is to develop a 2-D graphics package which supports basic operations which include representation of client-server communication system and the processes involved in it. The package must also have a user-friendly interface that may be menu-oriented, iconic or a combination of both.

All the basic graphical features are made available with minimum complexity. This model is designed for upload/download/connection of communication operations.

## Introduction to OpenGL

OpenGL is an open specification for an applications program interface for defining 2D and 3D objects. The specification is cross-language, cross-platform API for writing applications that produce 2D and 3D computer graphics. It renders 3D objects to the screen, providing the same set of instructions on different computers and graphics adapters. Thus it allows us to write an application that can create the same effects in any operating system using any OpenGL-adhering graphics adapter. [2]

Computer graphics, a 3-dimensional primitive can be anything from a single point to an n-sided polygon. From the software standpoint, primitives utilize the basic 3-dimensional rasterization algorithms such as Bresenham's line drawing algorithm, polygon scan line fill, texture mapping and so forth. OpenGL's basic operation is to accept primitives such as points, lines and polygons, and convert them into pixels. This is done by a graphics pipeline known as the OpenGL state machine. Most OpenGL commands either issue primitives to the graphics pipeline, or configure how the pipeline processes these primitives.

OpenGL is a low-level, procedural API, requiring the programmer to dictate the exact steps required to render a scene. OpenGL's low-level design requires programmers to have a good knowledge of the graphics pipeline, but also gives a certain amount of freedom to implement novel rendering algorithms.

#### GLUT

GLUT, short for OpenGL Utility Toolkit, is a set of support libraries available on every major platform. OpenGL does not directly support any form of windowing, menus, or input. That's where GLUT comes in. It provides basic functionality in all of those areas, while remaining platform independent, so that you can easily move GLUT-based applications from, for example, Windows to UNIX with few, if any, changes. [1]

## Project Related Concepts

The objective is to build a graphics editor having all required operations that a normal graphics editor should offer. The coding is implemented for a single canvas graphics editor. In this editor importance is given to a simple user interface and implementation of all basic graphic routines and basic graphic editing options.

The basic requirements of the graphics editor were analysed to be:

1. User Interface- The presentation of the model is crucial to its acceptance by its users. The package should provide easy selection and implementation of the features through menu interfaces.
2. Process Selection - To select a suitable option to display the process. The model implements support for graphical representation of the process. The implementation is done using the graphics scan conversion algorithm.
3. Keyboard interaction – The model provides option to perform various actions by the use of keys. The performed action will be displayed as soon as the model recognizes the key pressed.
   * c : The actual screen of the scenic view of the windmill will be obtained.
   * d : Day view of the model can be experienced
   * n : Night view of the model can be experienced
   * r : The motion of one of the cars in the model can be stopped
   * s : The speed of rotation of the windmills in the model can be altered.

## INTERFACES

## Keyboard Interface

The model provides option to perform various actions by the use of keys. The performed action will be displayed as soon as the model recognizes the key pressed.

* + c : The actual screen of the scenic view of the windmill will be obtained.
  + d : Day view of the model can be experienced
  + n : Night view of the model can be experienced
  + r : The motion of one of the cars in the model can be stopped
  + s : The speed of rotation of the windmills in the model can be altered.

# CHAPTER 2

## REQUIREMENTS SPECIFICATION

Visual Studio 2005 delivers on Microsoft’s vision of smart client applications by letting developers quickly create connected applications that deliver the highest quality rich user experiences. This new version lets any size organization create more secure, more manageable, and more reliable applications that take advantage of Windows Vista, windows7, 2007 Office System and the Web. By building these new types of applications, organizations will find it easier than ever to capture and analyze information so that they can make effective business decisions.

## Software Requirements

* + - An MS-DOS based operating system like Windows 98, Windows 2000 or Windows XP, vista, windows 7 is the platform required to develop the 2D and 3D graphics applications.
    - A Visual C/C++ compiler is required for compiling the source code to make the executable file which can then be directly executed.
    - A built in graphics library like glut and glut32, and header file like GL\glut.h and also dynamic link libraries like glut and glut32 are required for creating the 3D layout.

## Hardware Requirements

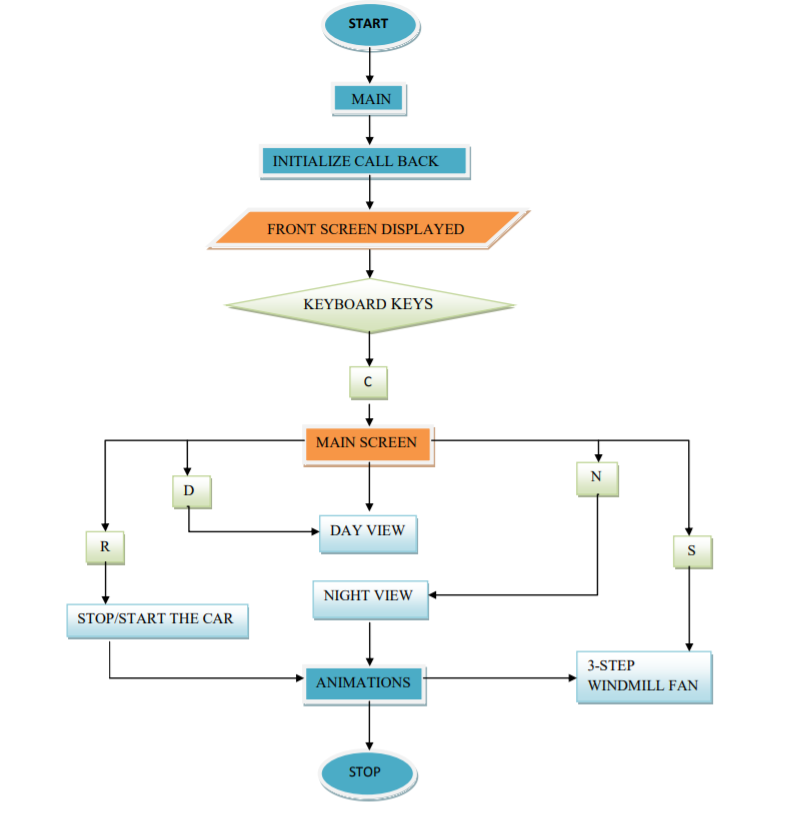
The hardware requirements are very minimal and the software can run on most of the machines.

* + - Processor - Intel 486/Pentium processor or above.
    - Processor Speed - 500 MHz or above
    - RAM - 64MB or above Storage Space - 2 MB or above, hard disk -10MB.
    - Monitor resolution - A color monitor with a minimum resolution of 1000\*700
    - Supports both single &double buffering.

# CHAPTER 3

DESIGN

The entire project is itself designed in C++. It contains three windmills where the speed of rotation of the blades of a windmill can be controlled through the keyboard interactions.The model also consists of two cars that move from left to right across the screen and one of the cars can be stopped by using keyboard interactions. The day and night view can also be experienced here.



## Fig.1.1. Flow of control

# CHAPTER 4

### IMPLEMENTATION

## 4.1 Functions used

**glutInit :** Initialises GLUT. The arguments passed from main can be used by the applications

**glutInitDisplayMode(unsigned int mode) :** Requests a display with properties in mode. The value of the mode is determined by logical OR operation. Mode values used are GLUT\_DOUBLE, GLUT \_RGB, GLUT\_DEPTH.

**glutInitWindowSize :** Specifies the initial width and height of the window in pixels.

**glutCreateWindow :** Gives the name of the window being created.

**glutMainLoop:** Enters the GLUT event processing loop.

**glRotatef:** Performs a rotation of angle degrees around vectors x,y,z.

**glLoadIndentity:** Replaces the current matrix with the identity matrix.

**glutDisplayFunc:** The glutDisplayFunc sets the display callback for the current window specified by ‘func’.

**glFlush:** Empties all of these buffers, causing all issued commands to be executed as quickly they are accepted by the actual rendering engine.

# CHAPTER-5

## Source Code

#include<stdio.h>

#include <GL/glut.h>

#include <math.h>

#include<string.h>

//int label;

const double PI = 3.141592654;

int night=0,stop=0,day=0;

int speed=0;

int frameNumber = 0;

int frameNumberOfCart=0;

int win1,win2;

int storeframe=0;

void circle1(GLfloat x,GLfloat y,GLfloat radius);

void display();

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

{

if(key=='n') {

night=1;}

if(key=='d')

night=0;

if(key=='s')

speed=(1+speed)%3;

if(key=='r'){

stop=1-stop;}

}

void Write(double x,double y,double z,double scale,char \*s)

{

int i,l=strlen(s);

glPushMatrix();

glTranslatef(x,y,z);

glScalef(scale,scale,scale);

for(i=0;i<l;i++)

glutStrokeCharacter(GLUT\_STROKE\_ROMAN,s[i]);

glPopMatrix();

}

void doFrame(int v) {

frameNumber++;

frameNumberOfCart++;

glutPostRedisplay();

glutTimerFunc(30,doFrame,0);

}

/\*

\* This method is called from main() to initialize OpenGL.

\*/

void init() {

glClearColor(0.5f, 0.5f, 1, 1);

glMatrixMode(GL\_PROJECTION);

glLoadIdentity();

glOrtho(0, 7, -1, 4, -1, 1);

glMatrixMode(GL\_MODELVIEW);

}

void keyboard1(unsigned char key ,int x ,int y)

{

if(key=='c' || key=='C')

{

glutDestroyWindow(win1);

glutInitDisplayMode(GLUT\_DOUBLE);

win2=glutCreateWindow("Windmill");

init();

glutDisplayFunc(display);

glutTimerFunc(200,doFrame,0);

glutKeyboardFunc(keys);

}

}

void circle1(GLfloat x, GLfloat y, GLfloat radius)

{

float angle;

glBegin(GL\_POLYGON);

for(int i=0;i<100;i++)

{

angle = i\*2\*(PI/100);

glVertex2f(x+(cos(angle)\*radius),y+(sin(angle)\*radius));

}

glEnd();

}

void drawDisk(double radius) {

int d;

glBegin(GL\_POLYGON);

for (d = 0; d < 32; d++) {

double angle = 2\*PI/32 \* d;

glVertex2d( radius\*cos(angle), radius\*sin(angle));

}

glEnd();

}

/\*

\* Draw a wheel, centered at (0,0) and with radius 1. The wheel has 15 spokes

\* that rotate in a clockwise direction as the animation proceeds.

\*/

void drawWheel() {

int i;

glColor3f(0,0,0);

drawDisk(1);

glColor3f(0.75f, 0.75f, 0.75f);

drawDisk(0.8);

glColor3f(0,0,0);

drawDisk(0.2);

glRotatef(frameNumber\*20,0,0,1);

glBegin(GL\_LINES);

for (i = 0; i < 15; i++) {

glVertex2f(0,0);

glVertex2d(cos(i\*2\*PI/15), sin(i\*2\*PI/15));

}

glEnd();

}

void drawCart() {

glPushMatrix();

glTranslatef(-1.5f, -0.1f, 0);

glScalef(0.8f,0.8f,1);

drawWheel();

glPopMatrix();

glPushMatrix();

glTranslatef(1.5f, -0.1f, 0);

glScalef(0.8f,0.8f,1);

drawWheel();

glPopMatrix();

glColor3f(.1,.25,0);

glBegin(GL\_POLYGON); // start drawing a polygon

//glColor3f(0.5f,0.2f,0.0f); // Set The Color To Red

glVertex3f(-3.0f, 3.0f, 0.0f); // Top left

glVertex3f(1.2f, 3.0f, 0.0f);

glColor3f(1.0f,1.5f,1.6f);

glVertex3f(3.0f, 1.2f, 0.0f);

// Set The Color To Green

glVertex3f( 3.0f,0.0f, 0.0f); // Bottom Right

// glColor3f(0.0f,0.0f,1.0f); // Set The Color To Blue

glVertex3f(-3.0f,0.0f, 0.0f);

glEnd();

}

void drawCart1() {

glPushMatrix();

glTranslatef(-1.5f, -0.1f, 0);

glScalef(0.8f,0.8f,1);

drawWheel();

glPopMatrix();

glPushMatrix();

glTranslatef(1.5f, -0.1f, 0);

glScalef(0.8f,0.8f,1);

drawWheel();

glPopMatrix();

glColor3f(0.3,0.2,0);

glBegin(GL\_POLYGON); // start drawing a polygon

glVertex2f(-2.5f,0);

glVertex2f(2.5f,0);

glColor3f(1,0.2,0);

glVertex2f(2.5f,2);

glVertex2f(-2.5f,2);

glEnd();

}

/\*

\* Draw a sun with radius 0.5 centered at (0,0). There are also 13 rays which

\* extend outside from the sun for another 0.25 units.

\*/

void drawSun() {

int i;

glColor3f(1,1,0);

for (i = 0; i < 13; i++) { // Draw 13 rays, with different rotations.

glRotatef( 360 / 13, 0, 0, 1 ); // Note that the rotations accumulate!

glBegin(GL\_LINES);

glVertex2f(0, 0);

glVertex2f(0.75f, 0);

glEnd();

}

drawDisk(0.5);

//else{

//glColor3f(1,1,1);

// }drawDisk(0.5);

// glColor3f(0,0,0);

}

void drawSun1() {

if(night==1){

glColor3f(1.0,1.0,1.0);

drawDisk(0.35);}

// glColor3f(1.0,1.0,1.0);

// circle(0.9);

// glFlush();

// glutSwapBuffers();

//drawDisk(0.5);

//drawDisk(0.5);

}

void drawSun2() {

if(night==1){

glColor3f(0.0,0.0,0.0);

drawDisk(0.35);}

// glColor3f(1.0,1.0,1.0);

// circle(0.9);

// glFlush();

// glutSwapBuffers();

//drawDisk(0.5);

//drawDisk(0.5);

}

/\*

\* Draw a windmill, consisting of a pole and three vanes. The pole extends from the

\* point (0,0) to (0,3). The vanes radiate out from (0,3). A rotation that depends

\* on the frame number is applied to the whole set of vanes, which causes the windmill

\* to rotate as the animation proceeds. Note that this method changes the current

\* transform in the GL context gl! The caller of this subroutine should take care

\* to save and restore the original transform, if necessary.

\*/

void drawWindmill() {

int i;

glColor3f(0.8f, 0.8f, 0.9f);

glBegin(GL\_POLYGON);

glVertex2f(-0.05f, 0); //thick ness of the windmill pole

glVertex2f(0.05f, 0);

glVertex2f(0.05f, 3);

glVertex2f(-0.05f, 3);

glEnd();

glTranslatef(0, 3, 0);

glRotated(frameNumber \* (180.0/(10\*(5\*speed+4))), 0, 0, 1);

glColor3f(0.4f, 0.4f, 0.8f);

for (i = 0; i < 3; i++) {

glRotated(120, 0, 0, 3); // Note: These rotations accumulate.

glBegin(GL\_POLYGON);

glVertex2f(0,0);

glVertex2f(0.5f, 0.1f);

glVertex2f(1.5f,0);

glVertex2f(0.5f, -0.1f);

glEnd();

}

}

/\*

\* This function is called when the image needs to be redrawn.

\* It is installed by main() as the GLUT display function.

\* It draws the current frame of the animation.

\*/

void display() {

glClear(GL\_COLOR\_BUFFER\_BIT); // Fills the scene with blue.

glLoadIdentity();

if(night==1){

glClearColor(0, 0, 0, 1);

glColor3f(1.0,1.0,1.0);

circle1(1.87,2.85,0.01);

circle1(1.9,2.65,0.01);

circle1(1.56,3.5,0.02);

circle1(0.85,3.5,0.01);

circle1(1.345,2.5,0.01);

circle1(2.02,2.69,0.01);

circle1(1.8,3.75,0.02);

circle1(0.99,3.45,0.01);

circle1(1.345,2.5,0.01);

circle1(7.40,2.8,0.01);

circle1(6.120,2.5,0.01);

circle1(4.90,7.5,0.01);

circle1(8.0,3.5,0.01);

circle1(6.40,3.5,0.01);

circle1(4.55,3.5,0.01);

circle1(7.65,3.5,0.01);

circle1(8.23,3.1,0.01);

circle1(6.2,2.5,0.01);

circle1(4.95,2.85,0.01);

circle1(6.15,2.75,0.01);

}

else

glClearColor(0.4,0.7,1.0,1);

if(night==1)

/\* Draw three green triangles to form a ridge of hills in the background \*/

glColor3f(0,0.2f,0.0f);

else

glColor3f(0, 0.6f, 0.2f);

glBegin(GL\_POLYGON);

glVertex2f(-3,-1);

glVertex2f(1.5f,1.65f);

glVertex2f(5,-1);

glEnd();

glBegin(GL\_POLYGON);

glVertex2f(-3,-1);

glVertex2f(3,2.1f);

glVertex2f(7,-1);

glEnd();

glBegin(GL\_POLYGON);

glVertex2f(0,-1);

glVertex2f(6,1.2f);

glVertex2f(20,-1);

glEnd();

/\* Draw a bluish-gray rectangle to represent the road. \*/

if(night==0)

glColor3f(0.4f, 0.4f, 0.5f);

else

glColor3f(0.3f,0.3f,0.4f);

glBegin(GL\_POLYGON);

glVertex2f(0,-0.4f);

glVertex2f(7,-0.4f);

glVertex2f(7,0.4f);

glVertex2f(0,0.4f);

glEnd();

/\* Draw a white line to represent the stripe down the middle

\* of the road. \*/

glLineWidth(4);

// Set the line width to be 6 pixels.

if(night==1) glColor3f(0.9,0.9,0.9);

else

glColor3f(1,1,1);

glBegin(GL\_LINES);

glVertex2f(0,0);

glVertex2f(0.5,0);

glEnd();

glBegin(GL\_LINES);

glVertex2f(0.75,0);

glVertex2f(1.25,0);

glEnd();

glBegin(GL\_LINES);

glVertex2f(1.5,0);

glVertex2f(2.0,0);

glEnd();

glBegin(GL\_LINES);

glVertex2f(2.25,0);

glVertex2f(2.75,0);

glEnd();

glBegin(GL\_LINES);

glVertex2f(3.0,0);

glVertex2f(3.5,0);

glEnd();

glBegin(GL\_LINES);

glVertex2f(3.75,0);

glVertex2f(4.25,0);

glEnd();

glBegin(GL\_LINES);

glVertex2f(4.5,0);

glVertex2f(5.0,0);

glEnd();

glBegin(GL\_LINES);

glVertex2f(5.25,0);

glVertex2f(5.75,0);

glEnd();

glBegin(GL\_LINES);

glVertex2f(6.0,0);

glVertex2f(6.5,0);

glEnd();

glBegin(GL\_LINES);

glVertex2f(6.75,0);

glVertex2f(7.25,0);

glEnd();

glBegin(GL\_LINES);

glVertex2f(7.5,0);

glVertex2f(8.0,0);

glEnd();

glColor3f(1,1,1);

glBegin(GL\_LINES);

glVertex2f(0,0.33);

glVertex2f(7,0.33);

glEnd();

glColor3f(1,1,1);

glBegin(GL\_LINES);

glVertex2f(0,-0.33);

glVertex2f(7,0-0.33);

glEnd();

glLineWidth(1); // Reset the line width to be 1 pixel.

/\* Draw the sun. The drawSun method draws the sun centered at (0,0). A 2D translation

\* is applied to move the center of the sun to (5,3.3). A rotation makes it rotate\*/

glPushMatrix();

if(night==0)

glTranslated(5.8,3,0);

else

glTranslated(100.8,300,0);

//glRotated(-frameNumber\*0.7,0,0,1);

drawSun();

glPopMatrix();

glPushMatrix();

glTranslated(2.8,3.5,0);

drawSun1();

glPopMatrix();

glPushMatrix();

glTranslated(2.75,3.68,0);

drawSun2();

glPopMatrix();

if(night==0)

glColor3f(0.5,0.5,0.5);

else

glColor3f(0.1,0.1,0.1);

circle1(1.6,3.13,.200);

circle1(1.75,3.23,.300);

circle1(1.1,3.28,.400);

circle1(1.5,3.33,.300);

glPushMatrix();

glTranslated(0.75,1,0);

glScaled(0.6,0.6,1);

drawWindmill();

glPopMatrix();

glPushMatrix();

glTranslated(2.2,1.6,0);

glScaled(0.4,0.4,1);

drawWindmill();

glPopMatrix();

glPushMatrix();

glTranslated(5.7,0.8,0);

glScaled(0.7,0.7,1);

drawWindmill();

glPopMatrix();

glPushMatrix();

if(stop==0){

glTranslated(-3 + 13\*(frameNumberOfCart % 500) / 300.0, 0, 0);

storeframe=frameNumberOfCart;

//printf("%d\n",frameNumberOfCart);

}

else{

glTranslated(-3 + 13\*(storeframe % 500) / 300.0, 0, 0);

frameNumberOfCart=storeframe;

}

glScaled(0.22,0.22,1);

drawCart();

glPopMatrix();

glPushMatrix();

glTranslated(-3 + 13\*(frameNumber % 300) / 200.0, 0, 0);

glScaled(0.3,0.3,1);

drawCart1();

glPopMatrix();

glFlush();

glutSwapBuffers();

} // end display

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

{

glutInit(&argc, argv);

glutInitDisplayMode(GLUT\_SINGLE|GLUT\_RGB);

glutInitWindowSize(700,500);

glutInitWindowPosition(100,100);

win1=glutCreateWindow("INTRODUCTION");

glutDisplayFunc(redisplay);

// glutTimerFunc(200,doFrame,0);

glutKeyboardFunc(keyboard1);

glutMainLoop();

//return 0;

}

# CHAPTER 6

## TESTING

* 1. **TEST CASES**

Table 6.1 : Test cases for Mouse interface

The actual screen of the scenic view of the windmill will be obtained.

Day view of the model can be experienced

Night view of the model can be experienced

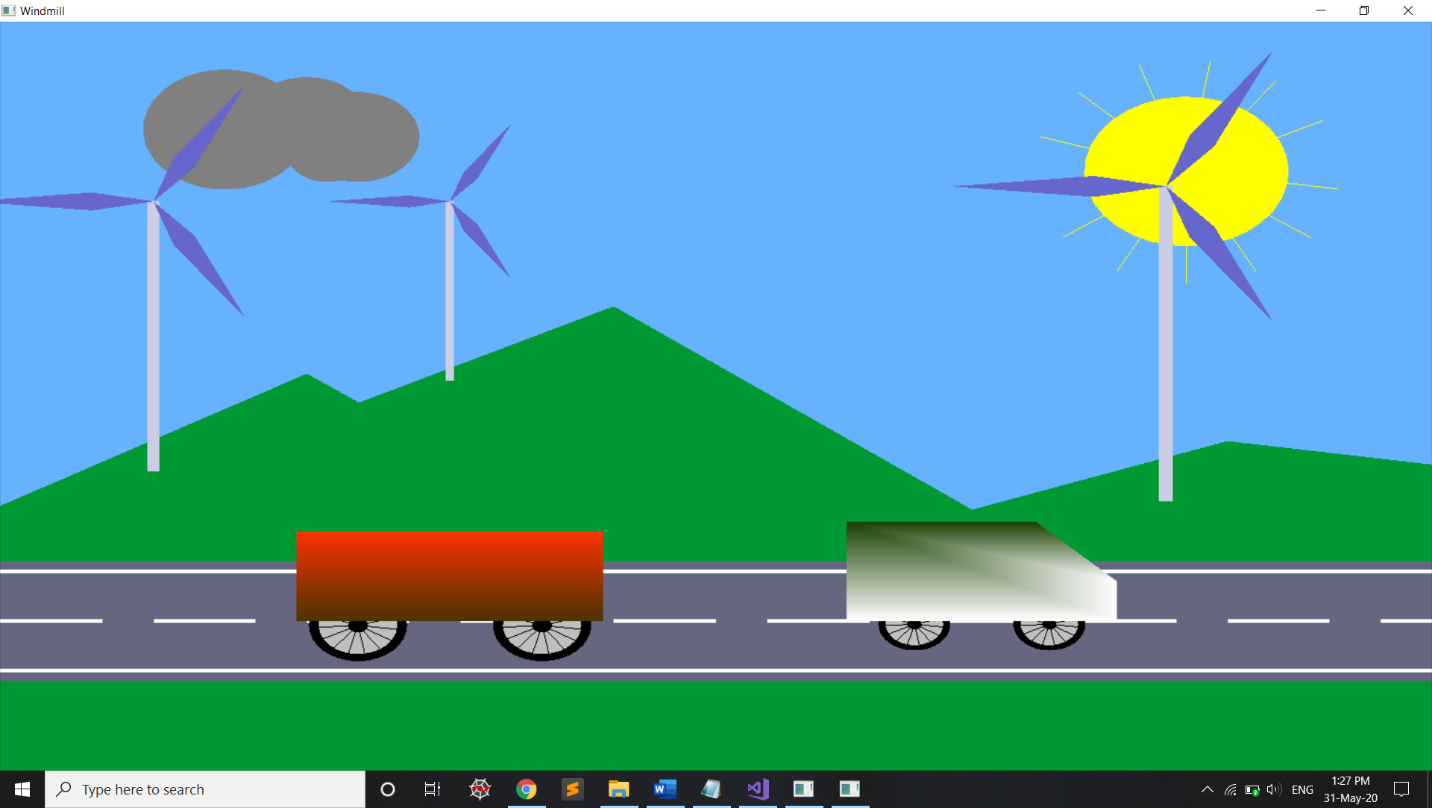
The motion of one of the cars in the model can be stopped

The speed of rotation of the windmills in the model can be altered.

|  |  |  |  |
| --- | --- | --- | --- |
| Sl.  No | **Functionality** | **Comments** | Remarks |
| 1. | Key functions   1. Key “c” 2. Key “d” 3. Key “n” 4. Key “r 5. Key “s” |  | Pass |
| 2. | Other Features | Menu option has multiple operation choices that  will be performed on selection. | Pass |

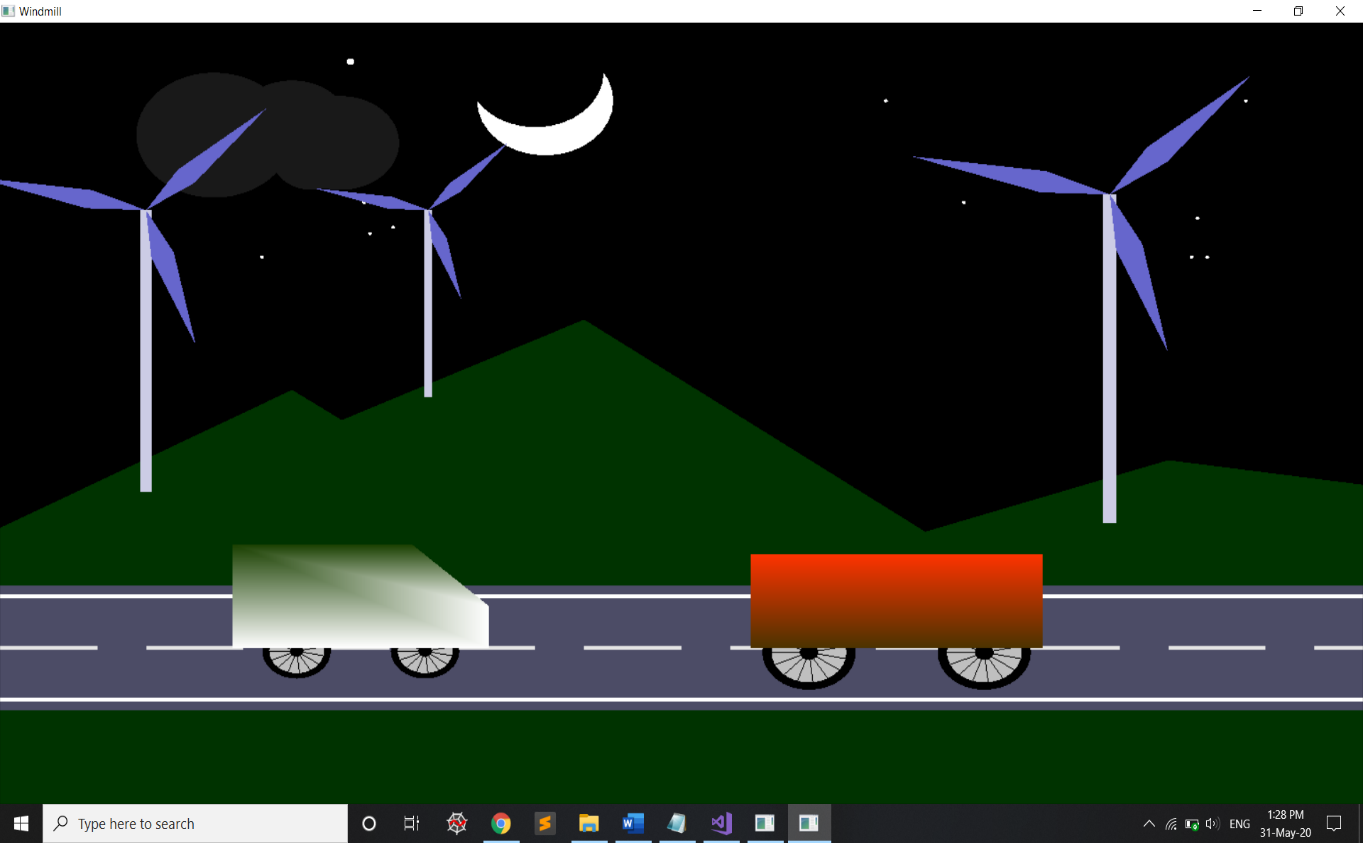
Table 6.2 : Test cases for Keyboard interface

# Chapter 7 Snapshots



### Fig.7.1.Day view

This is the starting page of the project. It consists of windmills that are rotating and the two carts will be moving from left to right on the screen. This is the day view of the scenery. The user can change the day view into night, change the speed of rotation of the blades of the windmill and stop one of the carts by using the keyboard interactions.



### Fig.7.2.Night View

Here, the user can change the night view into day, change the speed of rotation of the blades of the windmill and stop one of the carts by using the keyboard interactions.

CHAPTER 8

**CONCLUSION**

This graphics is very user friendly tool. The user can very easily use this tool to display the working of a client–server system. The interface is mouse driven and the user can select a function by clicking on an option representing that function. For the user’s help the function performed by each option is displayed on the screen as soon as the process starts. We have tried our best to make this editor very realistic, so that the user does not face any trouble when switching over from any real life graphics editor to this highly useful one.

## FUTURE ENHANCEMENTS

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.

• This 2D design can be converted in 3D for more better look.

• Various lighting elements can be implemented to make it more realistic.

• Design of car, mountain and other background elements can be more realistic by using shadow techniques.

# REFERENCES

1. Interactive Computer Graphics A Top-Down Approach with OpenGL -Edward Angel, 5th Edition, Addison-Wesley, 2008.
2. Computer Graphics Using OpenGL – F.S. Hill Jr. 2nd Edition, Pearson Education, 2001.
3. Computer Graphics – James D Foley, Andries Van Dam, Steven K Feiner, John F Hughes, Addison-Wesley 1997.
4. Computer Graphics - OpenGL Version – Donald Hearn and
5. Pauline Baker, 2nd Edition, Pearson Education, 2003.

#### [www.google.com](http://www.google.com/)

1. [**www.openglforum.org**](http://www.openglforum.org/)
2. [**http://en.wikipedia.org/wiki/Visual\_studio**](http://en.wikipedia.org/wiki/Visual_studio)
3. [**http://en.wikipedia.org/wiki/Open\_GL**](http://en.wikipedia.org/wiki/Open_GL)

# 