VISVESVARAYA TECHNOLOGICAL UNIVERSITY

JNANASANGAMA, BELAGAVI – 590018

Mini Project Report

On

A SCENIC VIEW OF A WINDMILL

Submitted in partial fulfilment for the award of degree of

Bachelor of Engineering

In

Computer Science and Engineering

Submitted by

Monalika P

1BG17CS119

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 A Scenic View of a Windmill carried out by Ms. Monalika P, USN 1BG17CS 119, a bonafide student of VI Semester B.E.,B.N.M Institute of Technology in partial fulfilment 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 Project prescribed for the said Degree.

Dr. Shreevidya R C
Associate 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

List of Figures

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.

.

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 **Dr. Sreevidya R C,** 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.

Monalika P

1BG17CS119

Chapter 1

INTRODUCTION

1.1 Overview

Interactive computer graphics provides us with the most natural means of communicating information through a computer. Over the years advancements in computer graphics have enabled us to not only make pictures of real-world objects but also visualize abstract, synthetic objects such as mathematical surfaces and of data that have no inherent geometry, such as survey results. 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. The computer on receiving signals from the input device can modify the displayed picture appropriately. 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 application is to demonstrate the scenic view of a windmill. 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 also be experienced.

1.3 Motivation

The scenic view of a windmill shows how the windmill is rotating and how its speed can be controlled at three different levels along with a car moving from left to right and a day-night view experience.

The ability to visualize how the scenic view of a windmill in the form of graphics will give us a valuable insight on its working. The ability to develop this visualization using C++ programming and the OpenGL API serves as a motivation to develop this 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 modeling, 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 to being language-independent,

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 scaled, biased and processed by the pixel map. The results are clamped

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.

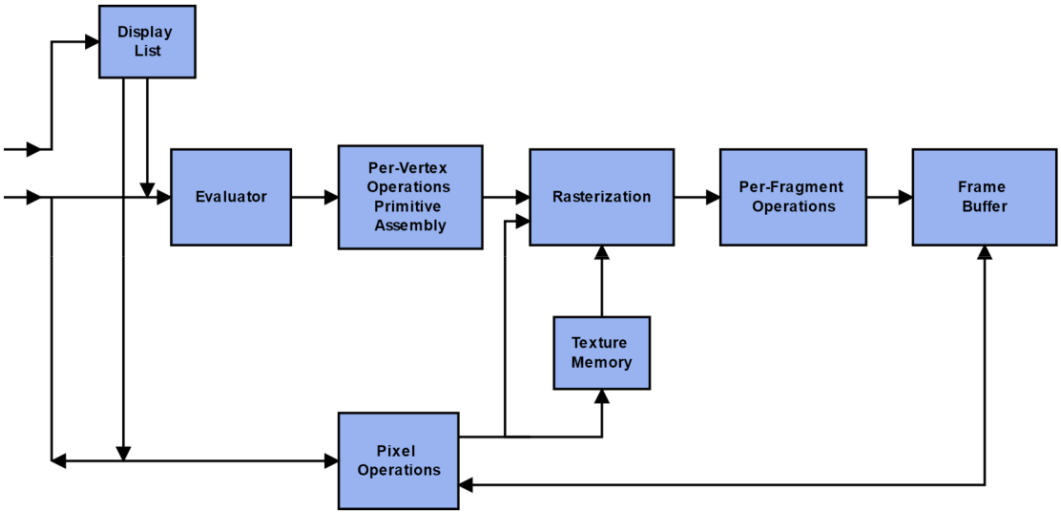

Fig 1.1 An illustration of the graphics pipeline process in OpenGL Architecture

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.

- **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. 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 the artificial world, move around in it, and interact with virtual features or items. The effect is commonly created by 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 consists of a scenic view of a windmill. It contains three windmills where the speed of rotation of the blades of a windmill can be controlled through the keyboard interactions. It 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.

4.2 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.

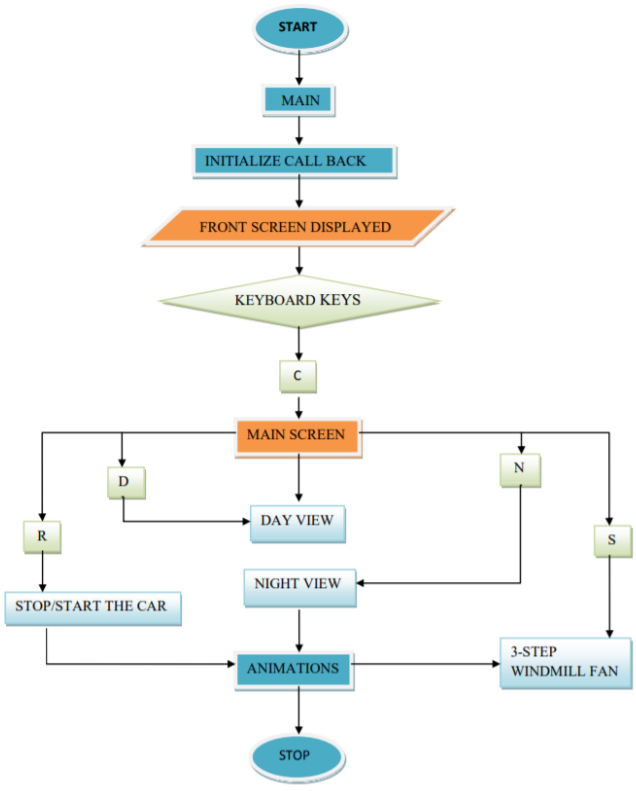

Figure 4.1 Flowchart of the Proposed System

Chapter 5

IMPLEMENTATION

5.1 Module Description

- **void myInit(void):**

  Responsible for the initial initialisation of point attributes, clearing the screen

- **void Display(void):**

  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 main(int argc, char**argv):**

  Main function from which execution begins and which calls all other functions.

- **void keys(unsigned char key, int x, int y):**

  Keyboard function where few keys are given

- **void circle1(GLfloat x, GLfloat y, GLfloat radius):**

  Using Circle1 function draw the circle with some specified radius.

- **void drawWheel();**

  Draw a wheel, centred at (0,0) and with radius 1. The wheel has 15 spokes that rotate in a clockwise direction as the animation proceeds.

- **void drawCart();**

  Draw a cart of required by using GL_POLYGONS. And insert its wheels using push and pop matrix functions.

- **void drawSun();**

  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 drawWindmill();**

  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.

5.2 High Level Code

5.2.1 Built-In Functions

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

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

- **void 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.

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

  Specifies the initial width and height of window in pixels.

- **int glutCreateWindow(const char *title);**

  Gives a name to the window which is created.

- **void glutMainLoop(void);**

  The glutMainLoop enters the GLUT event processing loop

- **void glutDisplayFunc(void (*func)(void)):**

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

- **void glFlush(void);**

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

- **void glLoadIdentity(void);**

  Replaces the current matrix with the identity matrix.

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

  Produces a rotation of angle degrees around vector x,y,z

5.2.2 User Implementation

- **Keys Function:**

```
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;}
```

- **Keyboard1 Function:**

```
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);

}
}
```

- **Circle1 Function:**

```
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();
  }
```

- **DrawDisk Function**

```
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();
}
```

- **DrawWheel Function**

```
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();
}
```

- **DrawCart Function**

```
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();
}
```

- **DrawCart1 Function**

```
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();
}
```

- **DrawSun1 Function**

```
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);
}
```

- **DrawSun2 Function**

```
void drawSun2() {
if(night==1){
glColor3f(0.0,0.0,0.0);
 drawDisk(0.35);
}
```

- **DrawWindmill Function**

```
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();
}
}
```

- **Display Function**

```
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.

glPushMatrix();
if(night==0)
glTranslated(5.8,3,0);
else
glTranslated(100.8,300,0);

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);
/* Draw three windmills */
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();
}// end of display
```

Chapter 6

RESULTS

- **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.

Figure 6.1 Day view

- **Night View**

  In this page, 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.

Figure 6.2 Night view

Chapter 7

CONCLUSION AND FUTURE ENHANCEMENTS

An attempt has been made to develop an OpenGL graphics package, which meets the necessary requirements of the user successfully. It enables us to learn about the basic concepts in OpenGL graphics and know standard library graphics functions and also to explore some other functions.

OpenGL graphics is a huge library which consist of numerous functions. These function have been used creatively to build the programs which may be to draw figures of various shapes at lower level or to stimulate any real thing, animation etc at higher level. This project has given us an insight into the use of computer graphics. As we have had to use many built-in and user defined functions, we have managed to get a certain degree of familiarity with these functions and have generally understood the power of these functions and were able to comprehend the true nature of the most powerful tool graphics in OpenGL and also have understood to a reasonable extent the reason why Graphics is so powerful for game programmers. We can now converse with a certain degree of confidence about Graphics in OpenGL and finally we have successfully completed the implementation of this project using OpenGL.

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.

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