**Visvesvaraya Technological University**

### Belgaum-590014



**A Project Report**

**On**

***“Namma Metro”***

*A project report submitted in partial fulfillment of the requirements for the award of the degree of* ***Bachelor of Engineering in Computer Science and Engineering***

*of Visvesvaraya Technological University, Belgaum*

Submitted by:

**SHEETHAL B.U. GUPTA 1RN11CS096**

**VIJETHA P.V 1RN11CS120**

Under the Guidance of:

**Prof. Sunil Kumar Prof. K.S. Sampada**

**Asst Professor Asst Professor**

**Dept. of CSE Dept. of CSE**



**Department of Computer Science and Engineering**

**RNS Institute of Technology**

**Channasandra, Uttarahalli-Kengeri Main Road, Bangalore-560 098**

**2013-2014**

**RNS Institute of Technology**

Channasandra, Uttarahalli-Kengeri Main Road,

Bangalore-560 061

**Department of Computer Science & Engineering**



**CERTIFICATE**

Certified that the project work entitled **“Namma Metro”** has been successfully carried out by **Sheethal B.U. Gupta** bearing USN **1RN11CS096** and **Vijetha P.V** bearing USN **1RN11CS120**, bonafide students of **RNS** **Institute of Technology,** in partial fulfillment of the requirements for the award of degree in **Bachelor of Engineering** **in Computer Science and Engineering** of **Visvesvaraya Technological University, Belgaum** during academic year 2013-2014. It is certified that all corrections/suggestions indicated for Internal Assessment have been incorporated in the report deposited in the departmental library. The project report has been approved as it satisfies the academic requirements in respect of project work for the said degree.

**Prof. Sunil Kumar Prof. K.S. Sampada Dr. G T Raju**

**Asst Prof. Asst Prof. Prof. and Head**

**Dept of CSE Dept of CSE Dept of CSE**

**External Viva:**

**Name of the Examiners Signature with Date**

**1.**

**2.**

**ACKNOWLEDGEMENT**

Any achievement, be it scholastic or otherwise does not depend solely on the individual efforts but on the guidance, encouragement and cooperation of intellectuals, elders and friends. A number of personalities, in their own capacities have helped me in carrying out this project work. I would like to take this opportunity to thank them all.

I would like to thank **Dr. H N Shivashankar**, Director, RNSIT, Bangalore, for his moral support towards completing my project.

I am grateful to **Dr. M K Venkatesha,** Principal, RNSIT, Bangalore, for his support towards completing my project.

I would like to thank **Dr. G T Raju**, Prof & Head of the Department, Computer Science & Engineering, RNSIT, Bangalore, for his valuable suggestions and expert advice.

I express my thanks to **Prof.Sunil Kumar**, Asst Professor, Department, Computer Science & Engineering, RNSIT, Bangalore, for helping me in completing the project and guiding me to organize the report in a systematic manner.

I deeply express my sincere gratitude to my guide **Prof. K.S. Sampada,** Asst Prof, Department of CSE, RNSIT, Bangalore,for her able guidance, regular source of encouragement and assistance throughout this project.

I thank all the teaching and non-teaching staff of Department of Computer Science & Engineering RNSIT, Bangalore for their constant support and encouragement.

Date :  **Students’ names**

Place : Bangalore Sheethal B.U. Gupta Vijetha P.V.

**ABSTRACT**

The project titled ‘NAMMA METRO’ is a Computer Graphics Application. The application is developed using OpenGL (Open Graphics Library) on Windows , which is a standard specification defining a cross-language, cross-platform API for writing applications that produce 2D and 3D computer graphics. The four major areas of computer graphics i.e. Display of information, Design, Simulation & Animation and User interfaces are attempted in the application.

The Project titled “Namma Metro”, attempts to show the movement of a metro train, from different persepectives. The project has three view modes: The Front View, Far View and a Side View. The user is initially instructed to provide with a keyboard key for each of the views. On input, the metro train is shown moving until a station arrives.

The Front view depicts how the metro train looks like from the front. A loco pilot, headlights, the tracks, etc can be seen. After a while, the train reaches the station, and stops.

The Far view depicts how the metro train looks like from the far. A tunnel, a flyover, and the train itself can be seen. After a while, the train reaches the tunnel, and stops.

The Side view depicts how the metro train looks from the side. Initially, one can see the train itself, moving and buildings, clouds in the background. After a while, the train reaches the station. While it reaches the station, the arrival-departure board at the station shows a countdown of when the train will arrive. Once it arrives at station, doors of the metro open, and a person is shown moving in.

The metro train movement is controlled by the 1,2,3 keys for the Front view ,Far view

and Side view ,respectively. The color of the train is Grey with blue window.

**CONTENTS**

|  |  |
| --- | --- |
|  |  |
| |  |  | | --- | --- | | **Chapter** | **Page No** | | **1. Introduction to computer graphics** | 1 | | 1.1 Overview of computer graphics | 1 | | 1.2 History of computer graphics | 2 | | 1.3 Applications of computer graphics | 3 | | **2. OpenGL** | 5 | | 2.1 Introduction to OpenGL | 5 | | 2.2 OpenGL libraries | 5 | | 2.3 Graphics pipeline architecture | 6 | | 2.4 OpenGL contributions | 8 | | 2.5 Limitations | 8 | | **3. Resource Requirements** | 10 | | 3.1 Hardware Requirements | 10 | | 3.2 Software Requirements | 10 | | **4. Function descriptions** | 11 | | **5. Implementation** | 14 | | **6. Snapshots** | 44 | | **7. Future enhancement** | 47 | | **8. Conclusion** | 48 | | **9. Bibliography** | 49 | |  |  | |  |
|  |  |
|  |  |

**CHAPTER 1**

**INTRODUCTION TO COMPUTER GRAPHICS**

* 1. **OVERVIEW OF COMPUTER GRAPHICS**

The term computer graphics has been used in a broad sense to describe almost everything on computers that is not text or sound. Typically, the term *computer graphics* refers to several different things:

* The [representation](http://en.wikipedia.org/wiki/Representation) and [manipulation](http://en.wikipedia.org/wiki/Photo_manipulation) of image data by a [computer](http://en.wikipedia.org/wiki/Computer)
* The various [technologies](http://en.wikipedia.org/wiki/Technologies) used to create and manipulate images
* The [images](http://en.wikipedia.org/wiki/Image) so produced, and
* The sub-field of [computer science](http://en.wikipedia.org/wiki/Computer_science) which studies methods for digitally synthesizing and manipulating visual content.

Today, computers and computer-generated images touch many aspects of daily life. Computer images is found on television, in newspapers, for example in weather reports, in all kinds of medical investigation and surgical procedures. A well-constructed [graph](http://en.wikipedia.org/wiki/Graph) can present complex statistics in a form that is easier to understand and interpret. In the media such graphs are used to illustrate papers, reports, thesis, and other presentation material.

Many powerful tools have been developed to visualize data. Computer generated imagery can be categorized into several different types: 2D, 3D, 4D, 7D, and animated graphics. As technology has improved, 3D computer graphics have become more common. Computer graphics has emerged as a sub-field of [computer science](http://en.wikipedia.org/wiki/Computer_science) which studies methods for digitally synthesizing and manipulating visual content. Over the past decade, other specialized fields have been developed like [information visualization](http://en.wikipedia.org/wiki/Information_visualization), and [scientific visualization](http://en.wikipedia.org/wiki/Scientific_visualization) more concerned with the visualization of [three dimensional](http://en.wikipedia.org/wiki/Three-dimensional_space) phenomena (architectural, meteorological, medical, [biological](http://en.wikipedia.org/wiki/Biological_Data_Visualization), etc.), where the emphasis is on realistic renderings of volumes, surfaces, illumination sources, and so forth, perhaps with a dynamic component.

* 1. **HISTORY OF COMPUTER GRAPHICS**

In 1959, the [TX-2](http://en.wikipedia.org/wiki/TX-2) computer was developed at [MIT's Lincoln Laboratory](http://en.wikipedia.org/wiki/Lincoln_Laboratory). The TX-2 integrated a number of new man-machine interfaces. A light pen could be used to draw sketches on the computer using [Ivan Sutherland](http://en.wikipedia.org/wiki/Ivan_Sutherland)'s revolutionary [Sketchpad software](http://en.wikipedia.org/wiki/Sketchpad).[[4]](file:///G:\Computer_graphics.htm#cite_note-WC03-3) Using a light pen, Sketchpad allowed one to draw simple shapes on the computer screen, save them and even recall them later. The light pen itself had a small photoelectric cell in its tip. This cell emitted an electronic pulse whenever it was placed in front of a computer screen and the screen's electron gun fired directly at it. By simply timing the electronic pulse with the current location of the electron gun, it was easy to pinpoint exactly where the pen was on the screen at any given moment. Once that was determined, the computer could then draw a cursor at that location. Also in 1961 another student at MIT, [Steve Russell](http://en.wikipedia.org/wiki/Steve_Russell), created the first video game, E. E. Zajac, a scientist at [Bell Telephone Laboratory](http://en.wikipedia.org/wiki/Bell_Labs) (BTL), created a film called "Simulation of a two-giro gravity attitude control system" in 1963.

During 1970s, the first major advance in 3D computer graphics was created at UU by these early pioneers, the hidden-surface algorithm. In order to draw a representation of a 3D object on the screen, the computer must determine which surfaces are "behind" the object from the viewer's perspective, and thus should be "hidden" when the computer creates (or renders) the image.

In the 1980s, artists and graphic designers began to see the personal computer, particularly the [Commodore Amiga](http://en.wikipedia.org/wiki/Commodore_Amiga) and [Macintosh](http://en.wikipedia.org/wiki/Apple_Macintosh), as a serious design tool, one that could save time and draw more accurately than other methods. In the late 1980s, [SGI](http://en.wikipedia.org/wiki/Silicon_Graphics) computers were used to create some of the first fully computer-generated [short films](http://en.wikipedia.org/wiki/Short_film) at [Pixar](http://en.wikipedia.org/wiki/Pixar). The Macintosh remains a highly popular tool for computer graphics among graphic design studios and businesses. Modern computers, dating from the 1980s often use [graphical user interfaces](http://en.wikipedia.org/wiki/Graphical_user_interfaces) (GUI) to present data and information with symbols, icons and pictures, rather than text. Graphics are one of the five key elements of [multimedia](http://en.wikipedia.org/wiki/Multimedia) technology.

[3D graphics](http://en.wikipedia.org/wiki/3D_graphics) became more popular in the 1990s in [gaming](http://en.wikipedia.org/wiki/Video_game), [multimedia](http://en.wikipedia.org/wiki/Multimedia) and [animation](http://en.wikipedia.org/wiki/Animation). In 1996, [Quake](http://en.wikipedia.org/wiki/Quake), one of the first fully 3D [games](http://en.wikipedia.org/wiki/Game), was released. In 1995, [Toy Story](http://en.wikipedia.org/wiki/Toy_Story), the first full-length computer-generated animation film, was released in cinemas worldwide. Since then, computer graphics have only become more detailed and realistic, due to more powerful graphics hardware and 3D modelling software.

* 1. **APPLICATIONS OF COMPUTER GRAPHICS**

The applications of computer graphics can be divided into four major areas:

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

**Display of information**

Computer graphics has enabled architects, researchers and designers to pictorially interpret the vast quantity of data. Cartographers have developed maps to display the celestial and geographical information. Medical imaging technologies like Computerized Tomography (CT), Magnetic Resonance Imaging (MRI), Ultrasound, Positron Emission Tomography (PET) and many others make use of computer graphics.

**Design**

Professions such as engineering and architecture are concerned with design. They start with a set of specification; seek cost-effective solutions that satisfy the specification. Designing is an iterative process. Designer generates a possible design, tests it and then uses the results as the basis for exploring other solutions. The use of interactive graphical tools in Computer Aided Design (CAD) pervades the fields including architecture, mechanical engineering, and the design of very-large-scale integrated (VLSI) circuits and creation of characters for animation.

**Simulation and animation**

Once the graphics system evolved to be capable of generating sophisticated images in real time, engineers and researchers began to use them as simulators. Graphical flight simulators have proved to increase the safety and to reduce the training expenses. The field of virtual reality (VR) has opened many new horizons. A human viewer can be equipped with a display headset that allow him/her to see the images with left eye and right eye which gives the effect of stereoscopic vision. This has further led to motion pictures and interactive video games.

**User interfaces**

Computer graphics has led to the creation of graphical user interfaces (GUI) using which even naive users are able to interact with a computer. Interaction with the computer has been dominated by a visual paradigm that includes windows, icons, menus and a pointing device such as mouse. Millions of people are internet users; they access the internet through the graphical network browsers such as Microsoft internet explorer and Mozilla Firefox.

**CHAPTER 2**

**OpenGL**

**2.1 INTRODUCTION TO OpenGL**

OpenGL (Open Graphics Library) is a standard specification defining a cross-language, cross-platform API for writing applications that produce 2D and 3D computer graphics. The interface consists of over 250 different function calls which can be used to draw complex three-dimensional scenes from simple primitives. OpenGL was developed by Silicon Graphics Inc. (SGI) in 1992 and is widely used in CAD, virtual reality, scientific visualization, information visualization, and flight simulation.

OpenGL provides a set of commands to render a three dimensional scene. That means you provide the data in an OpenGL-useable form and OpenGL will show this data on the screen (render it). It is developed by many companies and it is free to use. You can develop OpenGL-applications without licensing. OpenGL is a hardware- and system-independent interface. An OpenGL-application will work on every platform, as long as there is an installed implementation. Because it is system independent, there are no functions to create windows etc., but there are helper functions for each platform. A very useful thing is GLUT.

**2.2 OpenGL LIBRARIES**

Computer Graphics are created using OpenGL, which became a widely accepted standard software system for developing graphics applications. 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.

OpenGL stands for ‘open graphics library’ graphics library is a collection of API’s (Applications Programming Interface). Graphics library functions are:

1. **GL library** (OpenGL in windows) – Main functions for windows.
2. **GLU** (OpenGL utility library) - Creating and viewing objects.
3. **GLUT** (OpenGL utility toolkit)- Functions that help in creating interface of windows

OpenGL draws *primitives*—points, line segments, or polygons—subject to several selectable modes. You can control modes independently of each other; that is, setting one mode doesn't affect whether other modes are set (although many modes may interact to determine what eventually ends up in the frame buffer). Primitives are specified, modes are set, and other OpenGL operations are described by issuing commands in the form of function calls. These libraries are included in the application program using preprocessor directives

#include<GL/glut.h>

**OpenGL User Interface Library** (**GLUI**) is a C++ user interface library based on the OpenGL Utility Toolkit (GLUT) which provides controls such as buttons, checkboxes, radio buttons, and spinners to OpenGL applications. It is window and operating system independent, relying on GLUT to handle all system-dependent issues, such as window and mouse management.

The **OpenGL Utility Library** (**GLU**) is a computer graphics library. It consists of a number of functions that use the base OpenGL library to provide higher-level drawing routines from the more primitive routines that OpenGL provides. It is usually distributed with the base OpenGL package.

**2.3 GRAPHICS PIPELINE ARCHITECTURE**

The figure shown 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.

The concept of the graphics pipeline is what really sets it apart from general CPUs, although the idea of a pipeline is the same as that used by general purpose CPUs.  
The graphics pipeline is built in stages. Every stage is specialized in precisely one element of the rendering process. Once we are familiar with these tasks, we will be able to recognize them in the designs of the GPU. Let’s now take a look at the separate stages

## pipeline_OpenGL

Figure1.Pipeline Architecture of Computer Graphics

As shown in the Figure 1, 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, per-fragment operations, which perform 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.

**2.4 OpenGL CONTRIBUTIONS**

It is very popular in the video games development industry where it competes with Direct3D (on Microsoft Windows).OpenGL is also used in CAD, virtual reality, and scientific visualization programs. OpenGL is very portable. It will run for nearly every platform in existence, and it will run well. It even runs on Windows NT 4.0 etc. The reason OpenGL runs for so many platforms is because of its Open Standard.

OpenGL has a wide range of features, both in its core and through extensions. Its extension feature allows it to stay immediately current with new hardware features, despite the mess it can cause.

**2.5 LIMITATIONS**

* OpenGL is case sensitive
* Line Color, Filled Faces and Fill Color not supported.
* Bump mapping is not supported.
* Shadow plane is not supported.
* Navigation Renderer is not supported.
* 3D measurement is not supported
* Streaming of individual 3D objectives is not supported

**CHAPTER 3**

**RESOURCE REQUIREMENTS**

**3.1 HARDWARE REQUIREMENTS**

The Hardware requirements are very minimal and the program can be run on most of the machines.

|  |  |  |
| --- | --- | --- |
| Processor | : | Pentium4 processor |
| Processor Speed | : | 2.4 GHz |
| RAM | : | 1 GB |
| Storage Space | : | 40 GB |
| Monitor Resolution | : | 1024\*768 or 1336\*768 or 1280\*1024 |

**3.2 SOFTWARE REQUIREMENTS**

Operating System : Windows XP

IDE : Microsoft Visual Studio with C++ (version 6)

OpenGL libraries, Header Files which includes GL/glut.h, Object File Libraries, glu32.lib,opengl32.lib,glut32.lib,DLLfiles,glu32.dll,glut32.dll,opengl32.dll.

**CHAPTER 4**

**FUNCTION DESCRIPTIONS**

The description of all the functions used in the program is given below:

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

This function requests a display with the properties in mode. The value of mode is determined by the logical OR of options including the colour model (GLUT\_RGB, GLUT\_INDEX) and buffering (GLUT\_SINGLE, GLUT\_DOUBLE).

* **void glutInitWindowPosition (int x, int y)**

This specifies the initial position of top-left corner of the windows in pixels.

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

This function specifies the initial height and width of the window in pixels.

* **void glutCreateWindow (char \*title)**

This function creates a window on the display the string title can be used to label the window. The return value provides a reference to the window that can be used when there are multiple windows.

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

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

* **void glClearColor(GLclampfr,GLclampf g, GLclampfb,GLclampf a)**

This sets the present RGBA clear colour used when clearing the colour buffer. Variables of type GLclampf are floating point numbers between 0.0 and 1.0.

* **void glClear(GLbitfield mask)**

It clear buffers to present values. The value of mask is determined by the bitwise OR of options GL\_COLOR\_BUFFER\_BIT, GL\_DEPTH\_BUFFER\_BIT

* **void glutPostRedisplay ()**

This function requests that the display callback be executed after the current callback returns.

* **void glutReshapeFunc (void \*f (int width, int height)**

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

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

This function specifies a width\*height viewport in pixels whose lower left corner is at

(x, y) measured from the origin of the window.

* **void glMatrixMode (GLenum mode)**

This function specifies which matrix will be affected by subsequent transformations. Mode can be GL\_MODEL\_VIEW, GL\_PROJECTION, GL\_TEXTURE.

* **void glLoadIdentity ()**

This function sets the current transformation matrix to an identity matrix.

* **void gluOrtho2D(GLdoubleleft, GLdouble right, GLdouble bottom, GLdouble top)**

This function defines a two-dimensional viewing rectangle in the plane z=0.

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

This function registers the mouse callback function f. The callback function returns the button (GLUT\_LEFT\_BUTTON,GLUT\_MIDDLE\_BUTTON,GLUT\_RIGHT\_BUTTON), the state of the button after the event (GLUT\_UP, GLUT\_DOWN), and the position of the mouse relative to the top-left corner of the window.

* **void glVertex3f(TYPE xcoordinate, TYPE ycoordinate, TYPE zcoordinate)**

**void glVertex3fv(TYPE \*coordinates)**

This specifies the position of a vertex in 3 dimensions. If v is present, the argument is a pointer to an array containing the coordinates.

* **void glBegin(glEnum mode)**

This function initiates a new primitive of type mode and starts the collection of vertices. Values of mode include GL\_POINTS, GL\_LINES and GL\_POLYGON.

* **void glEnd()**

This function terminates a list of vertices.

* **void glutMainLoop()**

This function causes the program to enter an event processing loop. It should be the last statement in main.

* **void glPushMatrix(void);**

Pushes all matrices in the current stack down one level. The current stack is determined by glMatrixMode(). The topmost matrix is copied, so its contents are duplicated in both the top and second-from-the-top matrix. If too many matrices are pushed, an error is generated.

Void glRotate{fd} (TYPE angle, TYPE x, TYPE y, TYPE z);

void glTranslate{fd} (TYPE x, TYPE y, TYPE z);

Void glScale {fd} (TYPE x, TYPE y, TYPE z);

**21**. **Void glPopMatrix (void);**

Pops the top matrix off the stack, destroying the contents of the popped matrix. What was the second-from-the-top matrix becomes the top matrix. The current stack is determined by glMatrixMode(). If the stack contains a single matrix, calling glPopMatrix() generates an error.

**CHAPTER 5**

**IMPLEMENTATION**

**SOURCE CODE**

#include<stdio.h>

#include<stdlib.h>

#include<math.h>

#include<GL/glut.h>

#define CLOUD 1

#define WINDOW 2

void front\_view();

void far\_view();

void side\_view();

void station();

void train();

void run();

int ch=0,tran=0,movemany=0,farx=750,fary=750;

float s=0,open\_door1=0,open\_door2=0;//tran s and farx fary for tranlating train in side,front & far view

bool stop\_front=false,stop\_far=false,stop\_side=false;

int arrival=0,arrival2=0,arrival3=0;// to check when to stop

//train coordinates for side view

float v[7][2]={{-230,-130},{-230,100},{-200,130},{620,130},{620,-100},{590,-130},{590,100}};

float door1[6][2]={{0,-130},{0,70},{30,70},{30,-130},{60,-130},{60,70}};

float door2[6][2]={{370,-130},{370,70},{400,70},{400,-130},{430,-130},{430,70}};

float windows[4][2]={{-200,70},{550,70},{550,0},{-200,0}};

int x=0,y=0,z=0;//keeping track of number of turns train took in side view,front and far respectively

void print(char \*s)

{

glColor3f(1,1,1);

while(\*s)

glutBitmapCharacter(GLUT\_BITMAP\_TIMES\_ROMAN\_24, \*s++);

}

void mykey(unsigned char c,int x,int y)

{

switch(c)

{

case '0':ch=0;glutPostRedisplay();break;

case '1':ch=1;y=0;arrival2=0;s=0;glutPostRedisplay();break;

case '2':ch=2;z=0;arrival3=0;farx=0;fary=0;glutPostRedisplay();break;

case '3':ch=3;x=0;arrival=0;tran=0;glutPostRedisplay();break;

}

glutPostRedisplay();

}

void front\_train()

{

//outer polygon front

glLineWidth(1);

glColor3f( 0.4, 0.4, 0.4);

glBegin(GL\_POLYGON);

glVertex2f(-175,100);

glVertex2f(180,100);

glVertex2f(150,-400);

glVertex2f(-150,-400);

glEnd();

//train bottom buldge

glBegin(GL\_POLYGON);

glVertex2f(150,-400);

glVertex2f(-150,-400);

glVertex2f(-135,-475);

glVertex2f(155,-475);

glEnd();

//border for outer polygon front

glColor3f( 0,0,0);

glBegin(GL\_LINE\_LOOP);

glVertex2f(-175,100);

glVertex2f(180,100);

glVertex2f(150,-400);

glVertex2f(-150,-400);

glEnd();

//train bottom buldge border

glBegin(GL\_LINE\_LOOP);

glVertex2f(150,-400);

glVertex2f(-150,-400);

glVertex2f(-135,-475);

glVertex2f(155,-475);

glEnd();

//inner polygon

glColor3f(0.3, 0.3 ,0.858824);

glBegin(GL\_POLYGON);

glVertex2f(-120,50);

glVertex2f(120,50);

glVertex2f(100,-250);

glVertex2f(-100,-250);

glEnd();

//border for inner polygon

glColor3f( 0,0,0);

glBegin(GL\_LINE\_LOOP);

glVertex2f(-120,50);

glVertex2f(120,50);

glVertex2f(100,-250);

glVertex2f(-100,-250);

glEnd();

//mans face

glPushMatrix();

glColor3f(0,0,0);

glTranslatef(55,-125,0);

glutSolidSphere(25,10,10);

glPopMatrix();

//mans eye

glPushMatrix();

glColor3f(1,1,1);

glTranslatef(50,-120,0);

glutSolidSphere(5,10,10);

glPopMatrix();

glPushMatrix();

glColor3f(1,1,1);

glTranslatef(65,-120,0);

glutSolidSphere(5,10,10);

glPopMatrix();

//mans body

glColor3f(0,0,0);

glBegin(GL\_POLYGON);

glVertex2f(40,-150);

glVertex2f(70,-150);

glVertex2f(80,-250);

glVertex2f(30,-250);

glEnd();

//head light 1

glColor3f( 0.75, 0.2941, 0.1);

glBegin(GL\_POLYGON);

glVertex2f(-100,-300);

glVertex2f(-50,-300);

glVertex2f(-50,-350);

glVertex2f(-100,-350);

glEnd();

//border for headlight1

glColor3f( 0,0,0);

glBegin(GL\_LINE\_LOOP);

glVertex2f(-100,-300);

glVertex2f(-50,-300);

glVertex2f(-50,-350);

glVertex2f(-100,-350);

glEnd();

//head light 2

glColor3f( 0.75, 0.2941, 0.1);

glBegin(GL\_POLYGON);

glVertex2f(50,-300);

glVertex2f(100,-300);

glVertex2f(100,-350);

glVertex2f(50,-350);

glEnd();

//border for headlight2

glColor3f( 0,0,0);

glBegin(GL\_LINE\_LOOP);

glVertex2f(50,-300);

glVertex2f(100,-300);

glVertex2f(100,-350);

glVertex2f(50,-350);

glEnd();

}

void front\_station()

{

//platform left

glColor3f(0.81,0.71,0.23);//front face

glBegin(GL\_POLYGON);

glVertex2f(-500,-350);

glVertex2f(-350,-350);

glVertex2f(-350,-500);

glVertex2f(-500,-500);

glEnd();

glColor3f(0.7,0.7,0.7);//floor

glBegin(GL\_POLYGON);

glVertex2f(-500,-250);

glVertex2f(-500,-350);

glVertex2f(-350,-350);

glVertex2f(0,0);

glVertex2f(-150,-30);

glEnd();

glColor3f( 0.858824, 0.858824,0.439216 );//sidewall

glBegin(GL\_POLYGON);

glVertex2f(-350,-350);

glVertex2f(0,0);

glVertex2f(-350,-500);

glEnd();

glColor3f(1,1,0);//donot cross line

glBegin(GL\_POLYGON);

glVertex2f(-400,-350);

glVertex2f(0,0);

glVertex2f(-375,-350);

glEnd();

//platform right

glColor3f(0.81,0.71,0.23);//front face

glBegin(GL\_POLYGON);

glVertex2f(500,-350);

glVertex2f(350,-350);

glVertex2f(350,-500);

glVertex2f(500,-500);

glEnd();

glColor3f(0.7,0.7,0.7);//floor

glBegin(GL\_POLYGON);

glVertex2f(350,-350);

glVertex2f(500,-350);

glVertex2f(500,-250);

glVertex2f(150,-30);

glVertex2f(0,0);

glEnd();

glColor3f( 0.858824, 0.858824,0.439216 );//sidewall

glBegin(GL\_POLYGON);

glVertex2f(350,-350);

glVertex2f(0,0);

glVertex2f(350,-500);

glEnd();

glColor3f(1,1,0);//donot cross line

glBegin(GL\_POLYGON);

glVertex2f(375,-350);

glVertex2f(400,-350);

glVertex2f(0,0);

glEnd();

glPushMatrix();

glColor3f(0,0,0);

glTranslatef(-375,-150,0);

glutSolidSphere(25,10,10);

glPopMatrix();

//body

glColor3f(1,0,0);

glBegin(GL\_POLYGON);

glVertex2f(-385,-170);

glVertex2f(-365,-170);

glVertex2f(-350,-220);

glVertex2f(-400,-220);

glEnd();

//hands

glColor3f(0,0,0);

glBegin(GL\_LINES);

glVertex2f(-365,-170);

glVertex2f(-335,-190);

glEnd();

glColor3f(0,0,0);

glBegin(GL\_LINES);

glVertex2f(-385,-170);

glVertex2f(-415,-190);

glEnd();

//legs

glColor3f(0,0,0);

glBegin(GL\_LINES);

glVertex2f(-365,-220);

glVertex2f(-365,-250);

glEnd();

glColor3f(0,0,0);

glBegin(GL\_LINES);

glVertex2f(-385,-220);

glVertex2f(-385,-250);

glEnd();

glColor3f(0,0,0);

glBegin(GL\_POLYGON);//Board

glVertex2f(340,250);

glVertex2f(500,250);

glVertex2f(500,100);

glVertex2f(340,100);

glEnd();

glLineWidth(10);

glBegin(GL\_LINES);

glVertex2f(425,100);

glVertex2f(425,-220);

glEnd();

glLineWidth(1);

glColor3f(1,1,1);

glRasterPos2f(350,175);

print("MG ROAD");

}

void front\_view()

{

glClearColor(0.496078,0.7,0.8,1);

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

glPushMatrix(); //clouds on left

glTranslatef(-300,450,0);

glCallList(1);

glPopMatrix();

glPushMatrix(); //clouds on right

glTranslatef(300,450,0);

glCallList(1);

glPopMatrix();

glColor3f(0,0,0);//track

glBegin(GL\_LINES);

glVertex2f(0,0);

glVertex2f(160,-500);

glEnd();

glBegin(GL\_LINES);

glVertex2f(0,0);

glVertex2f(-150,-500);

glEnd();

glPushMatrix();

glScalef(s,s,s);

front\_train();

glPopMatrix();

if(y==1)

{

front\_station();

glPushMatrix();

glScalef(s,s,s);

front\_train();

glPopMatrix();

arrival++;

if(arrival>=750)

{

glutIdleFunc(NULL);

stop\_front=true;

}

}

}

void far\_train()

{

glPushMatrix();

glLineWidth(1);

glTranslatef(30,0,0);

//train top

glColor3f(0.3,0.30,0.30);

glBegin(GL\_POLYGON);

glVertex2f(210,210);

glVertex2f(160,230);

glVertex2f(-300,-50);

glVertex2f(-250,-70);

glEnd();

//train front

glColor3f( 0.6,0.6,0.6);

glBegin(GL\_POLYGON);

glVertex2f(-300,-50);

glVertex2f(-250,-70);

glVertex2f(-275,-165);

glVertex2f(-330,-150);

glEnd();

//train front window

glColor3f( 0.196078, 0.196078,0.8);

glBegin(GL\_POLYGON);

glVertex2f(-295,-70);

glVertex2f(-260,-80);

glVertex2f(-280,-145);

glVertex2f(-320,-135);

glEnd();

//train front bottom

glColor3f( 0.6,0.6,0.6);

glBegin(GL\_POLYGON);

glVertex2f(-330,-150);

glVertex2f(-275,-165);

glVertex2f(-270,-180);

glVertex2f(-320,-185);

glEnd();

//train side

glColor3f( 0.5,0.5,0.5);

glBegin(GL\_POLYGON);

glVertex2f(-250,-70);

glVertex2f(-275,-165);

glVertex2f(-270,-180);

glVertex2f(210,150);

glVertex2f(210,210);

glEnd();

glPopMatrix();

//windows

glColor3f(0.3, 0.3 ,0.858824);

glBegin(GL\_POLYGON);

glVertex2f(-190,-70);

glVertex2f(230,195);

glVertex2f(230,160);

glVertex2f(-190,-120);

glEnd();

//door

glColor3f(1,1,1);

glBegin(GL\_POLYGON);

glVertex2f(-100,10);

glVertex2f(-60,30);

glVertex2f(-50,-50);

glVertex2f(-90,-75);

glEnd();

//door outline

glColor3f(0,0,0);

glBegin(GL\_LINE\_LOOP);

glVertex2f(-100,10);

glVertex2f(-80,20);

glVertex2f(-70,-63);

glVertex2f(-90,-75);

glEnd();

glBegin(GL\_LINE\_LOOP);

glVertex2f(-80,20);

glVertex2f(-70,-63);

glVertex2f(-50,-50);

glVertex2f(-60,30);

glEnd();

//door 2

glColor3f(1,1,1);

glBegin(GL\_POLYGON);

glVertex2f(150,155);

glVertex2f(180,175);

glVertex2f(180,105);

glVertex2f(150,85);

glEnd();

//door 2 outline

glColor3f(0,0,0);

glBegin(GL\_LINE\_LOOP);

glVertex2f(150,155);

glVertex2f(165,165);

glVertex2f(165,95);

glVertex2f(150,85);

glEnd();

glBegin(GL\_LINE\_LOOP);

glVertex2f(165,165);

glVertex2f(165,95);

glVertex2f(180,105);

glVertex2f(180,175);

glEnd();

}

void far\_view()

{

glClearColor( 0.74902, 0.847059, 1,1);

glClear(GL\_COLOR\_BUFFER\_BIT);

float cx=20,cy=20,r=160,dy=0.07,x1,y1;

int i,theta,n=1200;

glPushMatrix(); //clouds on left

glTranslatef(-450,350,0);

glCallList(1);

glPopMatrix();

glPushMatrix(); //clouds on middle

glTranslatef(-50,450,0);

glCallList(1);

glPopMatrix();

glColor3f(0.9,0.9,0.9);// the flyover

glBegin(GL\_POLYGON);

glVertex2f(-500,-300);

glVertex2f(-500,-200);

glVertex2f(350,500);

glVertex2f(500,500);

glVertex2f(500,400);

glVertex2f(300,200);

glVertex2f(250,150);

glVertex2f(150,50);

glVertex2f(50,-50);

glVertex2f(-50,-150);

glVertex2f(-150,-250);

glVertex2f(-250,-350);

glVertex2f(-400,-500);

glVertex2f(-500,-500);

glEnd();// end of flyover

glColor3f(0,0,0);

glBegin(GL\_POLYGON); // pillar for flyover

glVertex2f(0,-95);

glVertex2f(0,-500);

glVertex2f(-50,-500);

glVertex2f(-50,-145);

glEnd();

glBegin(GL\_POLYGON); // pillar for flyover

glVertex2f(300,200);

glVertex2f(300,-300);

glVertex2f(250,-300);

glVertex2f(250,150);

glEnd();

//call train

glPushMatrix();

glTranslatef(farx,fary,0);

far\_train();

glPopMatrix();

//tunnel

glPushMatrix();

//tunnel shade

glColor3f(0.196078, 0.8,0.196078);

glTranslatef(150,200,0);

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

{

glBegin(GL\_POINTS);

for(theta=0;theta<=180;theta++)

{

x1=cx+r\*cos(3.14\*theta/180);

y1=cy+r\*sin(3.14\*theta/180);

glVertex2f(x1,y1);

glFlush();

}

glEnd();

cy=cy+dy;

cx=cx+dy;

}

//outline for tunnel

glColor3f( 0.137255 , 0.556863, 0.137255);

glPointSize(5);

glBegin(GL\_POINTS);

for(theta=0;theta<=180;theta++)

{

x1=15+r\*cos(3.14\*theta/180);

y1=15+r\*sin(3.14\*theta/180);

glVertex2f(x1,y1);

glFlush();

}

glEnd();

glPopMatrix(); //end of tunnel

//Sun

glPushMatrix();

glColor3f(1,0.25,0);

glTranslatef(-300,400,0);

glutSolidSphere(25,10,10);

glPopMatrix();

glBegin(GL\_LINES);

glVertex2f(-340,420);//sun ray 1

glVertex2f(-265,375);

glVertex2f(-300,440);//sun ray 2 mid

glVertex2f(-300,360);

glVertex2f(-340,395);//sun ray 3

glVertex2f(-260,395);

glVertex2f(-260,420);//sun ray 4

glVertex2f(-340,380);

glEnd();// end of sun

if(z==1)

{

arrival3++;

if(arrival3==350)

{

glutIdleFunc(NULL);

stop\_far=true;

}

}

}

void door\_open()

{

glClearColor(0,1,1,1);

glClear(GL\_COLOR\_BUFFER\_BIT);

if(ch==4)

{

glPushMatrix();

station();

glTranslatef(tran,0,0);//Translate the train to the last position when it stopped

train();

glPopMatrix();

}

}

void train()

{

//Train

glColor3f( 0.752941, 0.752941, 0.752941);

glBegin(GL\_TRIANGLES);

glVertex2fv(v[1]);

glVertex2fv(v[0]);

glVertex2f(v[0][0]-80,v[0][1]+30);

glVertex2fv(v[0]);

glVertex2f(v[0][0]-80,v[0][1]+30);

glVertex2f(v[0][0]-80,v[0][1]);

glEnd();

glBegin(GL\_POLYGON);

glVertex2fv(v[0]);

glVertex2fv(v[1]);

glVertex2fv(v[2]);

glVertex2fv(v[3]);

glVertex2fv(v[4]);

glVertex2fv(v[5]);

glVertex2fv(v[6]);

glEnd();

//train back

glBegin(GL\_TRIANGLES);

glVertex2fv(v[6]);

glVertex2fv(v[5]);

glVertex2f(v[5][0]+30,v[5][1]);

glEnd();

glColor3f(0.3,0.3,0.3);

glBegin(GL\_POLYGON);

glVertex2fv(v[3]);

glVertex2fv(v[6]);

glVertex2f(v[5][0]+30,v[5][1]);

glVertex2f(v[4][0]+45,v[4][1]+10);

glEnd();

//train top

glColor3f(0.4,0.4,0.4);

glBegin(GL\_POLYGON);

glVertex2fv(v[1]);

glVertex2fv(v[2]);

glVertex2fv(v[3]);

glVertex2fv(v[6]);

glEnd();

//window 1

glColor3f(0.3, 0.3 ,0.858824);

glBegin(GL\_POLYGON);

glVertex2fv(windows[0]);

glVertex2fv(windows[1]);

glVertex2fv(windows[2]);

glVertex2fv(windows[3]);

glEnd();

//door 1 and door 2 background when door opens

glColor3f(1,1,1);

glBegin(GL\_POLYGON);

glVertex2fv(door1[0]);

glVertex2fv(door1[1]);

glVertex2fv(door1[5]);

glVertex2fv(door1[4]);

glEnd();

glBegin(GL\_POLYGON);

glVertex2fv(door2[0]);

glVertex2fv(door2[1]);

glVertex2fv(door2[5]);

glVertex2fv(door2[4]);

glEnd();

//door 1 left side

glPushMatrix();

glTranslatef(open\_door1,0,0);/\*open door to left, open\_door1 will be > 0,

only after it arrives to station, ch=4 \*/

glColor3f(1,1,1);

glBegin(GL\_POLYGON);

glVertex2fv(door1[0]);

glVertex2fv(door1[1]);

glVertex2fv(door1[2]);

glVertex2fv(door1[3]);

glEnd();

glColor3f(0,0,0);

glBegin(GL\_LINE\_LOOP);

glVertex2fv(door1[0]);

glVertex2fv(door1[1]);

glVertex2fv(door1[2]);

glVertex2fv(door1[3]);

glEnd();

glPopMatrix();

//door 1 right side

glPushMatrix();

glTranslatef(open\_door2,0,0);/\* open door to right, open\_door2 will be > 0,

only after it arrives to station, ch=4\*/

glColor3f(1,1,1);

glBegin(GL\_POLYGON);

glVertex2fv(door1[3]);

glVertex2fv(door1[2]);

glVertex2fv(door1[5]);

glVertex2fv(door1[4]);

glEnd();

glColor3f(0,0,0);

glBegin(GL\_LINE\_LOOP);

glVertex2fv(door1[3]);

glVertex2fv(door1[2]);

glVertex2fv(door1[5]);

glVertex2fv(door1[4]);

glEnd();

glPopMatrix();

//door 2 left side

glPushMatrix();

glTranslatef(open\_door1,0,0);//open door to left, open\_door1 will be > 0, only when ch=4

glColor3f(1,1,1);

glBegin(GL\_POLYGON);

glVertex2fv(door2[0]);

glVertex2fv(door2[1]);

glVertex2fv(door2[2]);

glVertex2fv(door2[3]);

glEnd();

glColor3f(0,0,0);

glBegin(GL\_LINE\_LOOP);

glVertex2fv(door2[0]);

glVertex2fv(door2[1]);

glVertex2fv(door2[2]);

glVertex2fv(door2[3]);

glEnd();

glPopMatrix();

//door 2 right side

glPushMatrix();

glTranslatef(open\_door2,0,0);// open door to right, open\_door2 will be > 0,

//only after it arrives to station, ch=4

glColor3f(1,1,1);

glBegin(GL\_POLYGON);

glVertex2fv(door2[3]);

glVertex2fv(door2[2]);

glVertex2fv(door2[5]);

glVertex2fv(door2[4]);

glEnd();

glColor3f(0,0,0);

glBegin(GL\_LINE\_LOOP);

glVertex2fv(door2[3]);

glVertex2fv(door2[2]);

glVertex2fv(door2[5]);

glVertex2fv(door2[4]);

glEnd();

glPopMatrix();

}

void station()

{

glClearColor(0,1,1,1);

int counter=1000;//timer

char s[10];// for displaying time as string

//Plat form front

glColor3f(0.25,0.25,0.25);

glBegin(GL\_LINES);

glVertex2f(-500,-90);

glVertex2f(500,-90);

glEnd();

glColor3f(0.9,0.9,0.9); //Floor

glBegin(GL\_POLYGON);

glVertex2f(-500,-130);

glVertex2f(500,-130);

glVertex2f(500,-500);

glVertex2f(-500,-500);

glEnd();

//Dont cross line

glColor3f(1,1,0);

glBegin(GL\_POLYGON);

glVertex2f(-500,-150);

glVertex2f(500,-150);

glVertex2f(500,-170);

glVertex2f(-500,-170);

glEnd(); //End of platform front

//plat form back

glColor3f( 0.858824, 0.858824,0.439216 );//side wall

glBegin(GL\_POLYGON);

glVertex2f(-450,130);

glVertex2f(500,130);

glVertex2f(500,200);

glVertex2f(-450,200);

glEnd();

//floor

glColor3f( 0.7,0.7,0.7);

glBegin(GL\_POLYGON);

glVertex2f(-450,200);

glVertex2f(500,200);

glVertex2f(500,250);

glVertex2f(-400,250);

glEnd();

//dont cross line

glColor3f( 1,1,0);

glBegin(GL\_POLYGON);

glVertex2f(-450,210);

glVertex2f(500,210);

glVertex2f(500,220);

glVertex2f(-420,220);

glEnd(); //end of platform back

//Board

glColor3f(0.137255,0.137255,0.556863);

glBegin(GL\_POLYGON);

glVertex2f(-350,320);

glVertex2f(150,320);

glVertex2f(150,400);

glVertex2f(-350,400);

glEnd();

//text on the Board

glColor3f(1,0.5,0);

glRasterPos2f(-330,365);

print("Arrival/destination : MG Road");

glColor3f(1,0.5,0);

glRasterPos2f(-330,335);

print("Train will come in :");

while(counter)

{

if(counter==1000&&arrival==10)

{

glRasterPos2f(-80+(counter/100),335);

\_itoa\_s(counter/100,s,10);

print(s);

}

if(counter==900&&arrival==100)

{

glRasterPos2f(-80+(counter/100),335);

\_itoa\_s(counter/100,s,10);

print(s);

}

if(counter==800&&arrival==200)

{

glRasterPos2f(-80+(counter/100),335);

\_itoa\_s(counter/100,s,10);

print(s);

}

if(counter==700&&arrival==300)

{

glRasterPos2f(-80+(counter/100),335);

\_itoa\_s(counter/100,s,10);

print(s);

}

if(counter==600&&arrival==400)

{

glRasterPos2f(-80+(counter/100),335);

\_itoa\_s(counter/100,s,10);

print(s);

}

if(counter==500&&arrival==500)

{

glRasterPos2f(-80+(counter/100),335);

\_itoa\_s(counter/100,s,10);

print(s);

}

if(counter==400&&arrival==600)

{

glRasterPos2f(-80+(counter/100),335);

\_itoa\_s(counter/100,s,10);

print(s);

}

if(counter==300&&arrival==700)

{

glRasterPos2f(-80+(counter/100),335);

\_itoa\_s(counter/100,s,10);

print(s);

}

if(counter==200&&arrival==800)

{

glRasterPos2f(-80+(counter/100),335);

\_itoa\_s(counter/100,s,10);

print(s);

}

if(counter==100&&arrival==900)

{

glRasterPos2f(-80+(counter/100),335);

\_itoa\_s(counter/100,s,10);

print(s);

}

counter--;

}

//board holders

glColor3f(0.184314,0.184314,0.309804);

glBegin(GL\_POLYGON);

glVertex2f(-250,400);

glVertex2f(-230,400);

glVertex2f(-230,500);

glVertex2f(-250,500);

glEnd();

glColor3f(0.184314,0.184314,0.309804);

glBegin(GL\_POLYGON);

glVertex2f(30,400);

glVertex2f(50,400);

glVertex2f(50,500);

glVertex2f(30,500);

glEnd();

//seats 1

glColor3f(1,0.5,0);

glBegin(GL\_POLYGON);

glVertex2f(-440,-380);

glVertex2f(-360,-380);

glVertex2f(-380,-320);

glVertex2f(-420,-320);

glEnd();

glColor3f(0,1,0.4);

glBegin(GL\_POLYGON);

glVertex2f(-360,-380);

glVertex2f(-340,-320);

glVertex2f(-300,-320);

glVertex2f(-280,-380);

glEnd();

glColor3f(0.7,0.65,0.25);

glBegin(GL\_POLYGON);

glVertex2f(-280,-380);

glVertex2f(-260,-320);

glVertex2f(-220,-320);

glVertex2f(-200,-380);

glEnd();

glColor3f(0.7,0.65,0.25);

glBegin(GL\_POLYGON);

glVertex2f(-200,-380);

glVertex2f(-180,-355);

glVertex2f(-235,-355);

glEnd();

glColor3f(0,0,0);

glBegin(GL\_LINE\_LOOP);

glVertex2f(-440,-380);

glVertex2f(-440,-430);

glEnd();

glColor3f(0,0,0);

glBegin(GL\_LINE\_LOOP);

glVertex2f(-200,-380);

glVertex2f(-200,-430);

glEnd();

glColor3f(0,0,0);

glBegin(GL\_LINE\_LOOP);

glVertex2f(-180,-355);

glVertex2f(-180,-430);

glEnd();

//seats 2

glPushMatrix();

glTranslatef(0,-60,0);

glColor3f(1,0,0);

glBegin(GL\_POLYGON);

glVertex2f(300,350);

glVertex2f(310,410);

glVertex2f(350,410);

glVertex2f(360,350);

glEnd();

glColor3f(1,0,0);

glBegin(GL\_POLYGON);

glVertex2f(300,350);

glVertex2f(280,330);

glVertex2f(340,330);

glVertex2f(360,350);

glEnd();

glColor3f(1,1,0);

glBegin(GL\_POLYGON);

glVertex2f(360,350);

glVertex2f(370,410);

glVertex2f(410,410);

glVertex2f(420,350);

glEnd();

glColor3f(1,1,0);

glBegin(GL\_POLYGON);

glVertex2f(360,350);

glVertex2f(340,330);

glVertex2f(400,330);

glVertex2f(420,350);

glEnd();

glColor3f(1,0,1);

glBegin(GL\_POLYGON);

glVertex2f(420,350);

glVertex2f(430,410);

glVertex2f(470,410);

glVertex2f(480,350);

glEnd();

glColor3f(1,0,1);

glBegin(GL\_POLYGON);

glVertex2f(420,350);

glVertex2f(400,330);

glVertex2f(460,330);

glVertex2f(480,350);

glEnd();

//seat legs

glColor3f(0,0,0);

glBegin(GL\_LINES);

glVertex2f(480,350);

glVertex2f(480,300);

glEnd();

glColor3f(0,0,0);

glBegin(GL\_LINES);

glVertex2f(460,330);

glVertex2f(460,300);

glEnd();

glColor3f(0,0,0);

glBegin(GL\_LINES);

glVertex2f(280,330);

glVertex2f(280,280);

glEnd();

glColor3f(0,0,0);

glBegin(GL\_LINES);

glVertex2f(300,330);

glVertex2f(300,300);

glEnd();

glPopMatrix();

//person 1

glPushMatrix();

glColor3f(0,0,0);

glTranslatef(59,-350,0);

glutSolidSphere(25,10,10);

glPopMatrix();

//body

glColor3f(1,0,0);

glBegin(GL\_POLYGON);

glVertex2f(56,-370);

glVertex2f(65,-370);

glVertex2f(81,-450);

glVertex2f(40,-450);

glEnd();

//hands

glColor3f(0,0,0);

glBegin(GL\_LINES);

glVertex2f(65,-370);

glVertex2f(95,-410);

glEnd();

glColor3f(0,0,0);

glBegin(GL\_LINES);

glVertex2f(56,-370);

glVertex2f(26,-410);

glEnd();

//legs

glColor3f(0,0,0);

glBegin(GL\_LINES);

glVertex2f(48,-450);

glVertex2f(48,-490);

glEnd();

glColor3f(0,0,0);

glBegin(GL\_LINES);

glVertex2f(73,-450);

glVertex2f(73,-490);

glEnd(); // end of person 1

if(ch==5) // after the door opens, train disappears, to make it reappear

{

glPushMatrix();

glTranslatef(tran,0,0);

movemany=170;

train();

glPopMatrix();

}

// if train stopped, doors opened, move the man inside train

glPushMatrix();

glTranslatef(50,150+movemany,0);

//person 2

//head

glPushMatrix();

glColor3f(0,0,0);

glTranslatef(245,-325,0);

glutSolidSphere(25,10,10);

glPopMatrix();

//hand

glBegin(GL\_LINES);

glVertex2f(245,-400);

glVertex2f(265,-375);

glEnd();

glBegin(GL\_LINES);

glVertex2f(235,-390);

glVertex2f(225,-375);

glEnd();

//shirt

glColor3f( 0.858824,0.439216, 0.576471);

glBegin(GL\_POLYGON);

glVertex2f(235,-400);

glVertex2f(255,-400);

glVertex2f(255,-350);

glVertex2f(235,-350);

glEnd();

//leg

glColor3f(0,0,1);

glBegin(GL\_POLYGON);

glVertex2f(235,-400);

glVertex2f(245,-400);

glVertex2f(235,-450);

glVertex2f(225,-450);

glEnd();

//leg

glBegin(GL\_POLYGON);

glVertex2f(245,-400);

glVertex2f(255,-400);

glVertex2f(265,-450);

glVertex2f(255,-450);

glEnd(); // end of second person

glPopMatrix();

}

void draw\_window(float x,float y)

{

glPushMatrix();

glTranslatef(x,y,0);

glCallList(2);

glPopMatrix();

}

void side\_view()

{

glClearColor(0,1,1,1);

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

glLineWidth(5);

glPushMatrix(); //clouds on left

glTranslatef(-200,400,0);

glCallList(1);

glPopMatrix();

glPushMatrix();//clouds in middle

glTranslatef(0,200,0);

glCallList(1);

glPopMatrix();

glPushMatrix();//clouds in right

glTranslatef(300,400,0);

glCallList(1);

glPopMatrix();

glColor3f(0,0,0);

glBegin(GL\_LINES);//tracks

glVertex2f(-500,-90);

glVertex2f(500,-90);

glVertex2f(-500,-135);

glVertex2f(500,-135);

glEnd();

glColor3f(0.9,0.9,0.9);

glBegin(GL\_POLYGON);//tracks background

glVertex2f(-500,-90);

glVertex2f(500,-90);

glVertex2f(500,-130);

glVertex2f(-500,-130);

glEnd();

glColor3f(0,0,0);//building1

glColor3f(0,0,1);

glBegin(GL\_POLYGON);

glVertex2f(-500,-90);

glVertex2f(-400,-90);

glVertex2f(-400,170);

glVertex2f(-500,170);

glEnd();

draw\_window(-490,100);

draw\_window(-450,100);

draw\_window(-490,30);

draw\_window(-450,30);

draw\_window(-490,-50);

draw\_window(-450,-50);

glColor3f(0,0,0);

glColor3f(1,0,0);//building 2

glBegin(GL\_POLYGON);

glVertex2f(-400,-90);

glVertex2f(-400,400);

glVertex2f(-250,400);

glVertex2f(-250,-90);

glEnd();

draw\_window(-390,330);

draw\_window(-350,330);

draw\_window(-310,330);

draw\_window(-390,260);

draw\_window(-350,260);

draw\_window(-310,260);

draw\_window(-390,190);

draw\_window(-350,190);

draw\_window(-310,190);

draw\_window(-390,120);

draw\_window(-350,120);

draw\_window(-310,120);

glColor3f(1,1,0);//building 6

glBegin(GL\_POLYGON);

glVertex2f(500,-90);

glVertex2f(400,-90);

glVertex2f(400,300);

glVertex2f(500,300);

glEnd();

draw\_window(410,230);

draw\_window(410,160);

draw\_window(410,90);

draw\_window(410,20);

draw\_window(450,230);

draw\_window(450,160);

draw\_window(450,90);

draw\_window(450,20);

glColor3f(1,0,0);// building 5

glBegin(GL\_POLYGON);

glVertex2f(400,-90);

glVertex2f(400,175);

glVertex2f(300,175);

glVertex2f(300,-90);

glEnd();

draw\_window(310,105);

draw\_window(350,105);

draw\_window(310,35);

draw\_window(350,35);

draw\_window(310,-35);

draw\_window(350,-35);

glColor3f(0,0,0.5);// building 4

glBegin(GL\_POLYGON);

glVertex2f(300,-90);

glVertex2f(300,290);

glVertex2f(200,290);

glVertex2f(200,-90);

glEnd();

draw\_window(210,220);

draw\_window(250,220);

draw\_window(210,150);

draw\_window(250,150);

draw\_window(210,80);

draw\_window(250,80);

draw\_window(210,10);

draw\_window(250,10);

glColor3f(0,0.5,0.5);//building 3

glBegin(GL\_POLYGON);

glVertex2f(75,190);

glVertex2f(200,190);

glVertex2f(200,-90);

glVertex2f(75,-90);

glEnd();

draw\_window(85,120);

draw\_window(125,120);

draw\_window(165,120);

draw\_window(85,50);

draw\_window(125,50);

draw\_window(165,50);

draw\_window(85,-20);

draw\_window(125,-20);

draw\_window(165,-20);

glLineWidth(2);

glPushMatrix();

glTranslatef(tran,0,0);//translating of train, tran increments in run()

train();

glPopMatrix();

if(x==1) //after it roams enough, goes to station.

{

glClearColor(0,1,1,1);

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

station();

glPushMatrix();

glTranslatef(tran,0,0);// train is still translating, tran increments in run()

train();

glPopMatrix();

arrival++;// to keep track of when to stop the train in station

if(arrival==1000)

{

glutIdleFunc(NULL);// stop the train

stop\_side=true;

ch=4;//door open choice

glutIdleFunc(run);// run the idle function for choice ch=4

}

}

}

void welcome()

{

glBegin(GL\_POLYGON);

glColor3f(0.5,0.5,1);

glVertex2f(-500,-500);

glVertex2f(-500,500);

glColor3f(0,0,1);

glVertex2f(500,500);

glVertex2f(500,-500);

glEnd();

glColor3f(1,1,1);

glRasterPos2f(-150,450);

print("R.N.S. Institute of Technology");

glRasterPos2f(-100,350);

print("Namma Metro");

glColor3f(0,0,0);

glRasterPos2f(-270,290);

print("Instructions : Click on one of the keyboard inputs");

glRasterPos2f(-270,250);

print("Train coming from Far(Front View): 1");

glRasterPos2f(-270,210);

print("Tunnel and the Train(Far View): 2");

glRasterPos2f(-270,170);

print("Train and the Station(Side View): 3");

glRasterPos2f(-270,130);

print("Wait till each of the animation is completed,before you");

glRasterPos2f(-270,90);

print("choose next");

glRasterPos2f(-450,-100);

print("Project members : ");

glRasterPos2f(-450,-200);

print("Sheethal B.U. Gupta. (1RN11CS096)");

glRasterPos2f(50,-200);

print("&");

glRasterPos2f(100,-200);

print("Vijetha P.V.(1RN11CS120)");

glFlush();

}

void display()

{

glClear(GL\_COLOR\_BUFFER\_BIT);

glutIdleFunc(run);

if(ch==0) welcome();

else if(ch==1) //front view

{

if(stop\_front)

glutIdleFunc(NULL);

front\_view();

}

else if(ch==2) //far view

{

if(stop\_far)

glutIdleFunc(NULL);

far\_view();

}

else if(ch==3) //side view

{

if(stop\_side)

glutIdleFunc(NULL);

side\_view();

}

else if(ch==4)//open door

{

door\_open();

}

else if(ch==5)//move person after door opens

{

station();

}

glNewList(CLOUD,GL\_COMPILE);// definition of cloud

glPushMatrix();

glColor3f(1,1,1);

glutSolidSphere(25,10,10);//left circle

glTranslatef(40,0,0);

glutSolidSphere(25,10,10);//right circle

glTranslatef(-25,25,0);

glutSolidSphere(25,10,10);//middle circle

glPopMatrix();

glEndList();

glNewList(WINDOW,GL\_COMPILE);// definition of window

glColor3f(1,1,1);

glBegin(GL\_POLYGON);

glVertex2f(10,10);

glVertex2f(30,10);

glVertex2f(30,50);

glVertex2f(10,50);

glEnd();

glEndList();

glutSwapBuffers();

glFlush();

}

void myinit()

{

//glMatrixMode(GL\_PROJECTION);

//glLoadIdentity();

glClearColor(1,1,1,1);

glColor3f(1.0,0.0,0.0);

glOrtho(-500,500,-500,500,-500,500);

//glMatrixMode(GL\_MODELVIEW);

}

void run()

{

int i,j=0;

if(ch==1) //front view

{

s=s+0.001;

if(s>=1)

{

s=0;

y++;

}

for(i=0;i<1000000;i++);

}

if(ch==2)//far view

{

farx--;

fary--;

if(farx==-750 && fary==-750)

{

farx=750;fary=750;

z++;

}

//for(i=0;i<6000;i++);

}

if(ch==3) //side view

{

tran-=1;

if(tran<-1200)

{

tran=900;

x++;

}

for(i=0;i<6000000;i++);

}

if(ch==4) //open door

{

open\_door1=open\_door1-0.01;

if(open\_door1<=-20)

{

glutIdleFunc(NULL);

ch=5;//set choice for man moving inside train

glutIdleFunc(run);

}

open\_door2=open\_door2+0.01;

if(open\_door2>=20)

{

glutIdleFunc(NULL);

}

}

glutPostRedisplay();

}

void main()

{

glutInitDisplayMode(GLUT\_SINGLE|GLUT\_RGB|GLUT\_DEPTH);

glutInitWindowSize(800,600);

glutInitWindowPosition(100,100);

glutCreateWindow("Namma Metro");

glutDisplayFunc(display);

glutKeyboardFunc(mykey);

glutIdleFunc(run);

myinit();

glutMainLoop();

}

**CHAPTER 6**

**SNAPSHOTS**

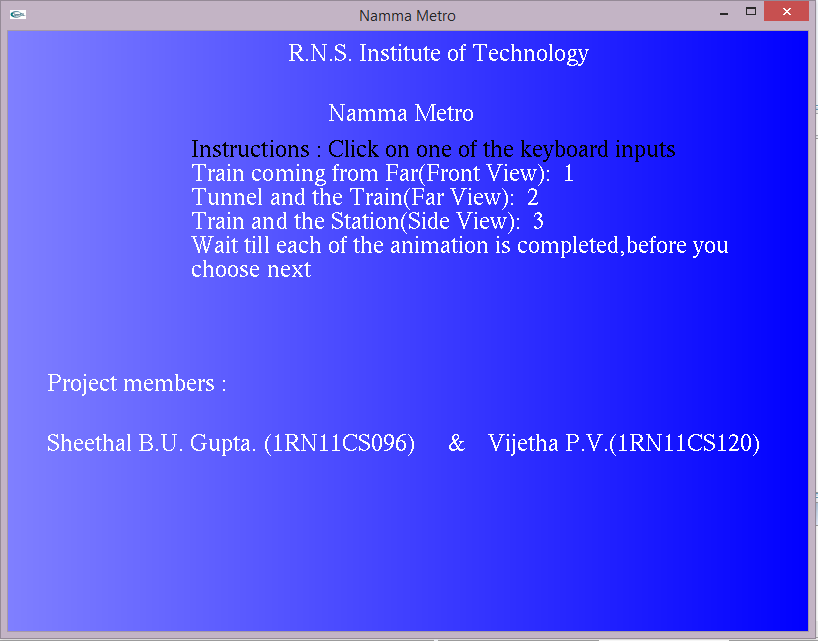
****

Figure 6.1. Start Screen

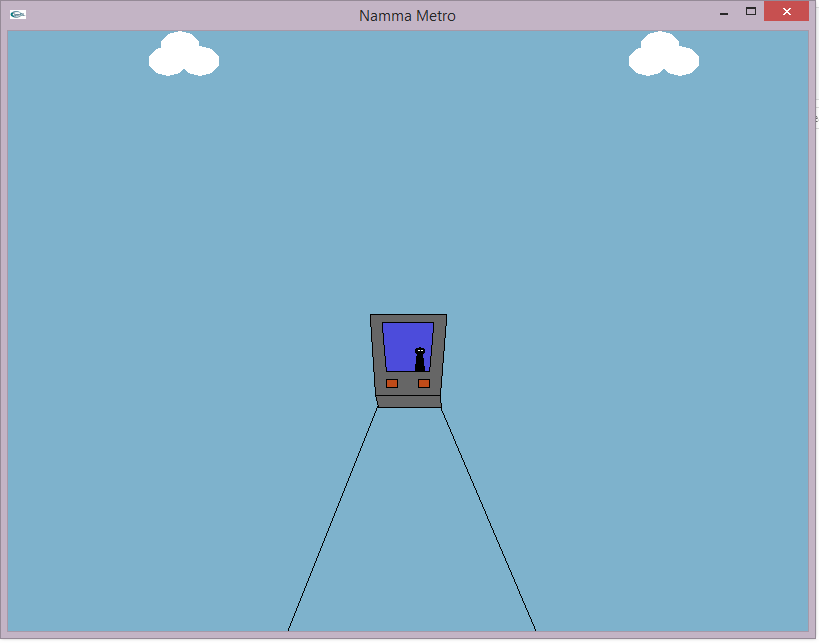
****

Figure 6.2. Front view

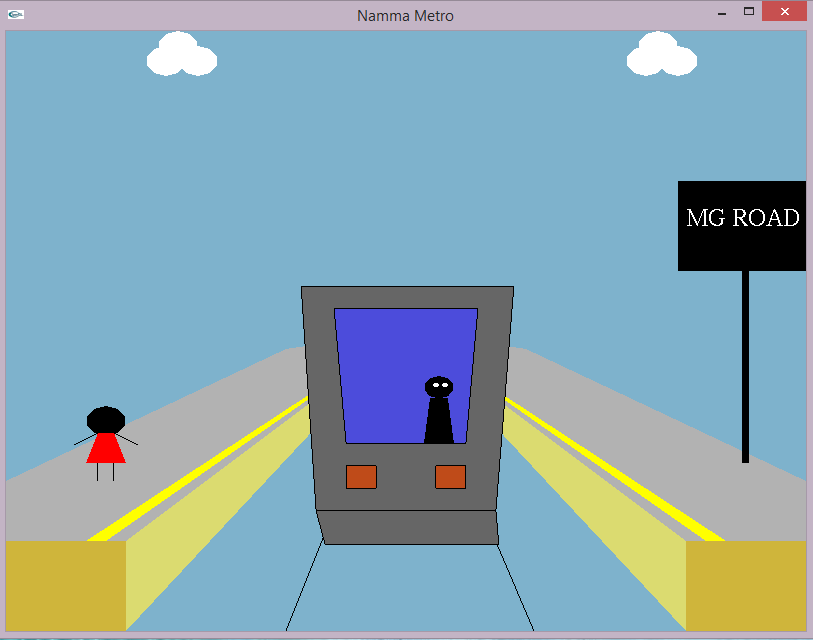
****

Figure 6.3. Station Front View

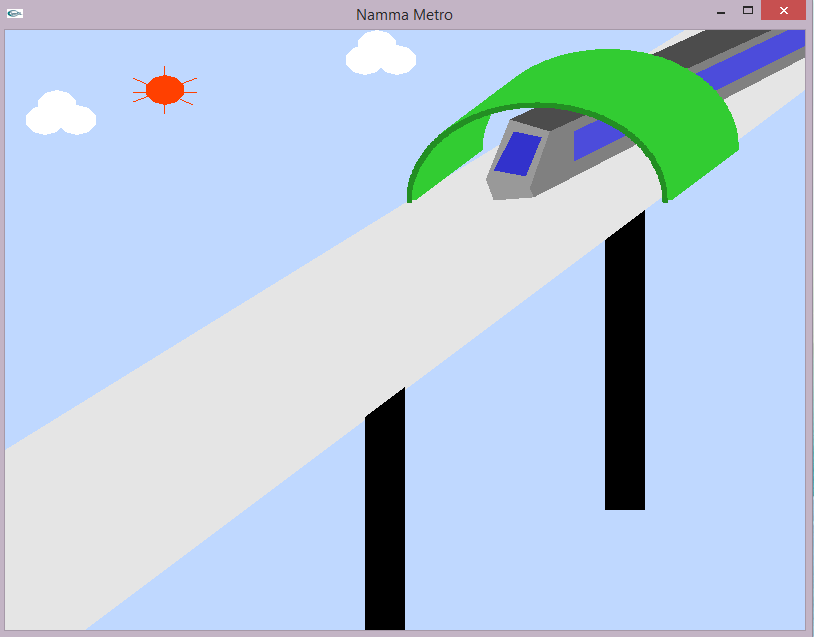
****

Figure 6.4. Far view

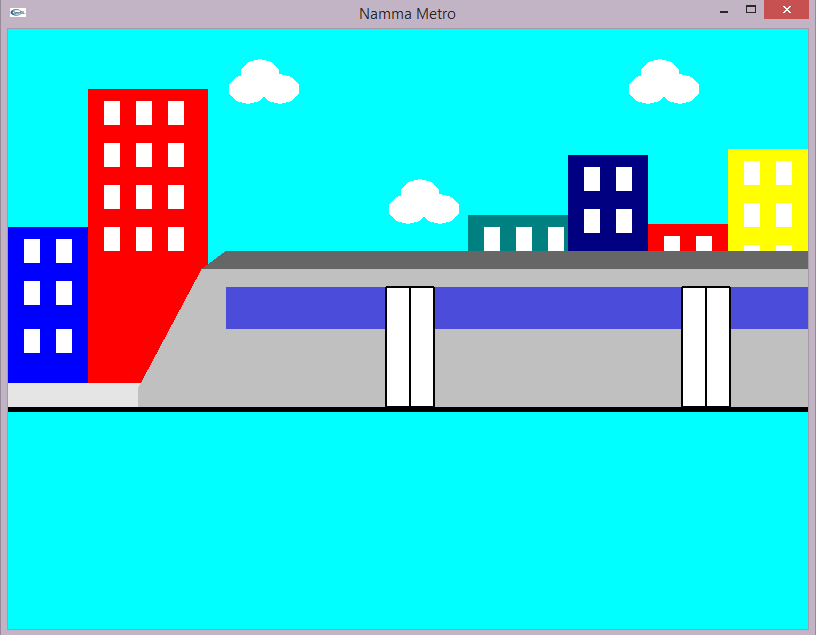
****

Figure. 6.5 Side View

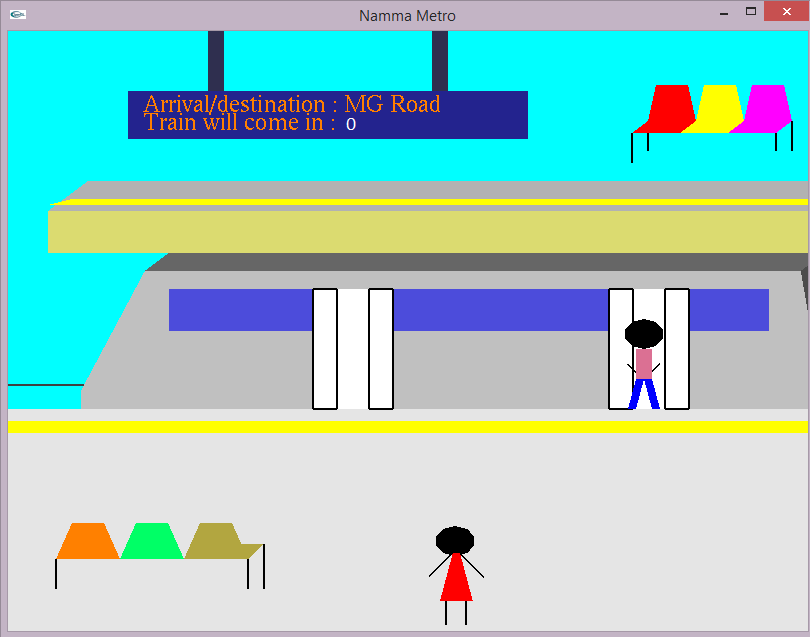
****

Figure 6.6. Station Side View

**CHAPTER 7**

**FUTURE ENHANCEMENT**

Since the project developed is mainly academics based, the project consists of only a few functions and implementation of few concepts. Developing of all possible functions present in OpenGL can further enhance this project. We can also include textures in order to make the background and also the train appear, real. Typically, textures are read from an image file, since specifying a texture programmatically could take hundreds of lines of code.

We can also add audio for the moving train and for announcements in the station, using OpenAL, Open Audio Library or other technologies. As the source code is available to the User, he may make changes to the existing shapes, colors, etc. This project may be implemented in 3D, also.

Further, it may be used as an education tool for educating the people in the city of, their conduct in the stations.

**CONCLUSION**

We started with modest aim with no prior experience in any programming projects as this, but ended up in learning many things, fine tuning the programming skills and getting into the real world of software development.

This is a simple interactive application. It is extremely user friendly and has the features which makes simple graphics project.

Checking and verification of all possible types of the functions are taken care. Care was taken to avoid bugs. Bugs may be reported to creator as the need may be .So, we conclude on note that we are looking forward to develop more such projects with an appetite to learn more in computer graphics.

**BIBLIOGRAPHY**

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 – OpenGL Version – Donald Hearn and Pauline Baker, 2nd Edition, Pearson Education, 2003.
4. <http://www.opengl.org/discussion_boards/showthread.php/132502-Color-tables>
5. <http://www.videotutorialsrock.com/>
6. http://www.stackoverflow.com/