Visvesvaraya Technological University
Belagavi-590 018, Karnataka

A Mini Project Report on

"RAILWAY STATION"

Mini Project Report submitted in partial fulfillment of the requirement for the Computer Graphics Laboratory with Mini Project [18CSL67]

Bachelor of Engineering
in Computer Science and Engineering

Submitted by,
Laharishree S[1J19CS119]
Sthuthi S [1JT19CS120]

Department of Computer Science and Engineering
Accredit by NBA, New Delhi
Jyothy Institute of Technology Tataguni, Bengaluru-560082

Jyothy Institute of Technology

Tataguni, Bengaluru-560082

Department of Computer Science and Engineering

CERTIFICATE

Certified that the mini project work entitled **“RAILWAY STATION”** carried out by **Laharishree S[1JT19CS119] and Sthuthi S[1JT18CS029]** bonafide students of Jyothy Institute of Technology, in partial fulfilment for the award of **Bachelor of Engineering** in **Computer Science and Engineering** department of the **Visvesvaraya Technological University, Belagavi** during academic the year **2020-2021**. 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 prescribed for the said Degree.

Mr. Vallabh Mahale	**Dr. Prabhanjan S**
Guide, Asst. Professor	Professor & HOD
Dept. of CSE	Dept. of CSE

External Viva Examiner

1. 2.

Signature with Date:

ACKNOWLEDGEMENT

Firstly, we are very grateful to this esteemed institution **“Jyothy Institute of Technology**” for providing us an opportunity to complete our project.

We express our sincere thanks to our **Principal Dr. Gopalakrishna K** for providing us with adequate facilities to undertake this project.

We would like to thank **Dr. Prabhanjan S, Professor and Head of Computer Science** and Engineering Department for providing his valuable support.

We would like to thank our guide **Mr. Vallabh Mahale, Assistant Professor** for his keen interest and guidance in preparing this work.

Finally, we would thank all our friends who have helped us directly or indirectly in this project.

Laharishree S[1JT19CS119]
Sthuthi S[1JT19CS120]

ABSTRACT

In this project, we have dealt with the Railway station, where it speaks about the movement of Train and its background where an aeroplane and comet movement occurs through a menu driven process.

This project displays the traffic signal used in the railways through which we can control the movement of train.

Firstly, an introduction to Computer graphics, OpenGL software is described, followed by a detailed explanation of the OpenGL architecture.

A sample code, followed by the main code, describes the implementation of Railway station in 2D graphics, where movement of objects can be seen just through a menu driven process and keyboard control. Followed by this, we have snapshots which witness the movement of the above mentioned objects.

We finally conclude our report with the conclusion and references, which gives an insight as to what all we gained from this project work.

Table of Contents

CHAPTER 1
INTRODUCTION

1.INTRODUCTION

1.1 Introduction to CG

Graphics is defined as any sketch or a drawing or a special network that pictorially represents some meaningful information. Computer Graphics is used where a set of image needs to be manipulated or the creation of the image in the form of pixels and is drawn on the computer. Computer Graphics can be used in digital photography, film, entertainment, electronic gadgets and all other core technologies which are required. It is a vast subject and area in the field of computer science. Computer Graphics can be used in UI design, rendering, geometric object, animation and many more. In most area, computer graphics is an abbreviation of CG. There are several tools used for implementation of Computer Graphics.

Computer Graphics refers to several things:

- The manipulation and the representation of the image or the data in a graphical manner.
- Various technologies required for the creation and manipulation.
- Digital synthesis and its manipulation.

Applications

- **Computer Graphics are used for aided design for engineering and architectural system-** These are used in electrical automobile, electro-mechanical, mechanical, electronic devices. For example: gears and bolts.
- **Computer Art –** MS Paint, Adobe Photoshop.
- **Presentation Graphics –** It is used to summarize financial statistical scientific or economic data. For example- Bar chart, Line chart.
- **Entertainment-** It is used in motion picture, music video, television gaming.
- **Education and training-** It is used to understand operations of complex system. It is also used for specialized system such for framing for captains, pilots and so on.
- **Visualization-** To study trends and patterns. For example- Analyzing satellite photo of earth.

1.2 Introduction to OpenGL

Interface

OpenGL is an application program interface (API) offering various functions to implement primitives, models and images. This offers functions to create and manipulate render lighting, colouring, viewing the models. OpenGL offers different coordinate system and frames. OpenGL offers translation, rotation and scaling of objects. Functions in the main GL library have names that begin with gl and are stored in a library usually referred to as GL. The second is the OpenGL Utility Library (GLU). The library uses only GL functions but contains code for creating common objects and simplifying viewing. All functions in GLU can be created from the core GL library but application programmers prefer not to write the code repeatedly. The GLU library is available in all OpenGL implementations; functions in the GLU library begin with the letter glu. Rather than using a different library for each system we use a readily available library called OpenGL Utility Toolkit (GLUT), which provides the minimum functionality that should be expected in any modern windowing system.

Overview

- OpenGL (Open Graphics Library) is the interface between a graphics program and graphics hardware. It is streamlined. In other words, it provides low-level functionality. For example, all objects are built from points, lines and convex polygons. Higher level objects like cubes are implemented as six four-sided polygons.
- OpenGL supports features like 3-dimensions, lighting, anti-aliasing, shadows, textures, depth effects, etc.
- It is system-independent. It does not assume anything about hardware or operating• system and is only concerned with efficiently rendering mathematically described scenes. As a result, it does not provide any windowing capabilities.
- It is a state machine. At any moment during the execution of a program there is a current model transformation.
- It is a rendering pipeline. The rendering pipeline consists of the following steps:

  * Defines objects mathematically.
  * Arranges objects in space relative to a viewpoint.
  * Calculates the colour of the objects.

1.3 Introduction to Project-title

"", also known as the Lucas' Tower, or simply a pyramid puzzle, is a mathematical game, which consists of three rods and number of disks in different diameters, which can slide into any rod. This puzzle starts with the disks stacked on one rod in order of decreasing size, i.e., the smallest being at the top, thus approximating a conical shape.

This particular problem is used to understand the opengl functions and its primitives. It becomes interesting as the program has menu driven and keyboard control.
Hence, an algorithm is set for the problem, such that movement of objects is given by the user.

Sample C code to describe the connection of train:-

```
// train connector
glColor3f(0.309804,0.184314,0.184314);
glBegin(GL_POLYGON);
glVertex2f(350+i-xm,75);
glVertex2f(350+i-xm,95);
glVertex2f(300+i-xm,95);
glVertex2f(300+i-xm,75);
glEnd();
//train wheels
for(l=0;l<50;l++)
  {glColor3f(0.35,0.16,0.14);
draw_circle(425+i-xm,50,l);
draw_circle(700+i-xm,50,l);
  }
}
```

CHAPTER 2
OpenGL Architecture

2. OpenGL Architecture

1) Pipeline Architectures

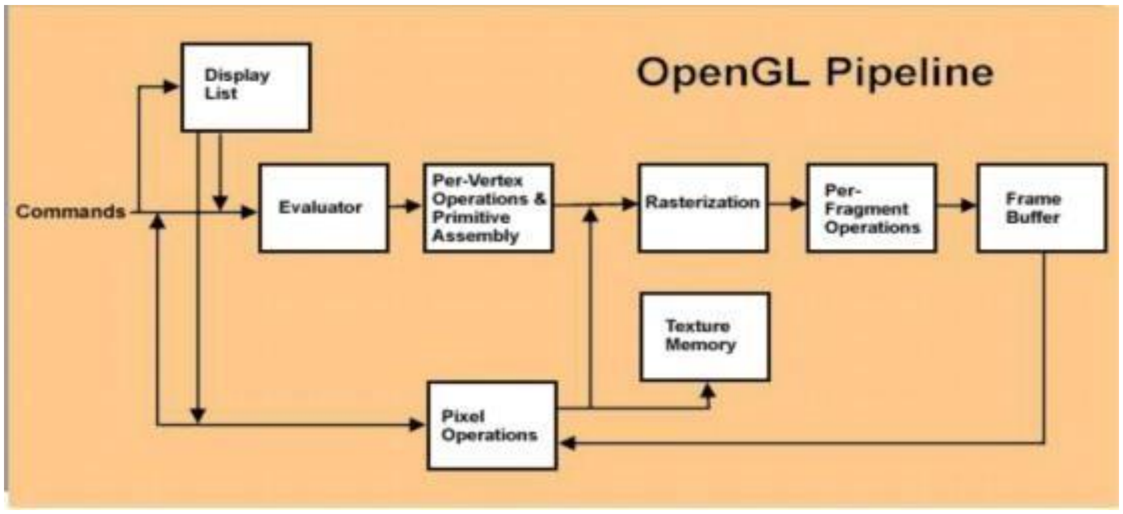

FIG:2.1 OPENGL PIPELINE ARCHITECTURE

- **Display Lists:** All data, whether it describes geometry or pixels, can be saved in a display list for current or later use. (1 alternative to retaining data in a display list is processing the data immediately - also known as immediate mode.) 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. Evaluators provide a method to derive the vertices used to represent the surface from the control points. The method is a polynomial mapping, which can produce surface normal, texture coordinates, colours, and spatial coordinate values from the control points.
- **Per-Vertex Operations:** For vertex data, next is the "per-vertex operations" stage, which converts. The vertices into primitives. Some vertex data (for example, spatial coordinates) 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. If advanced features are enabled, this stage is even busier. If texturing is used, texture coordinates may be generated and transformed here. If lighting is enabled, the lighting calculations are performed using the transformed vertex, surface normal, light source position, material properties, and other lighting information to produce a colour value.
- **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. Point clipping simply passes or rejects vertices; line or polygon clipping can add additional vertices depending upon how the line or polygon is clipped. In some cases,

this is followed by perspective division, which makes distant geometric objects appear smaller than closer objects. Then view port and depth (z coordinate) operations are applied. If culling is enabled and the primitive is a polygon, it then may be rejected by a culling test. Depending upon the polygon mode, a polygon may be drawn as points or lines. The results or this stage are complete geometric primitives, which are the transformed and clipped vertices with related colour, depth, and sometimes texturecoordinate values and guidelines for the rasterization step.

- **Pixel Operations:** While geometric data takes one path through the OpenGL rendering pipeline, pixel data takes a different route. Pixels from an array in system memory are first unpacked from one of a variety of formats into the proper number of components. Next the data is scaled, biased, and processed by a pixel map. The results are clamped and then either written into texture memory or sent to the rasterization step. If pixel data is read from the frame buffer, pixel-transfer operations (scale, bias, mapping, and clamping) are performed. Then these results are packed into an appropriate format and returned to an array in system memory. There are special pixel copy operations to copy data in the frame buffer to other parts of the frame buffer or to the texture memory. A single pass is made through the pixel transfer operations before the data is written to the texture memory or back to the frame buffer.
- **Texture Assembly:** An OpenGL application may wish to apply texture images onto geometric objects to make them look more realistic. If several texture images are used, it's wise to put them into texture objects so that you can easily switch among them. Some OpenGL implementations may have special resources to accelerate texture performance. There may be specialized, high-performance texture memory. If this memory is available, the texture objects may be prioritized to control the use of this limited and valuable resource.
- **Rasterization:** Rasterization is the conversion of both geometric and pixel data into fragments. Each fragment square corresponds to a pixel in the frame buffer. Line and polygon stipples, line width, point size, shading model, and coverage calculations to support ant aliasing are taken into consideration as vertices are connected into lines or the interior pixels are calculated for a filled polygon. Color and depth values are assigned for each fragment square.

2) OpenGL Engine and Drivers

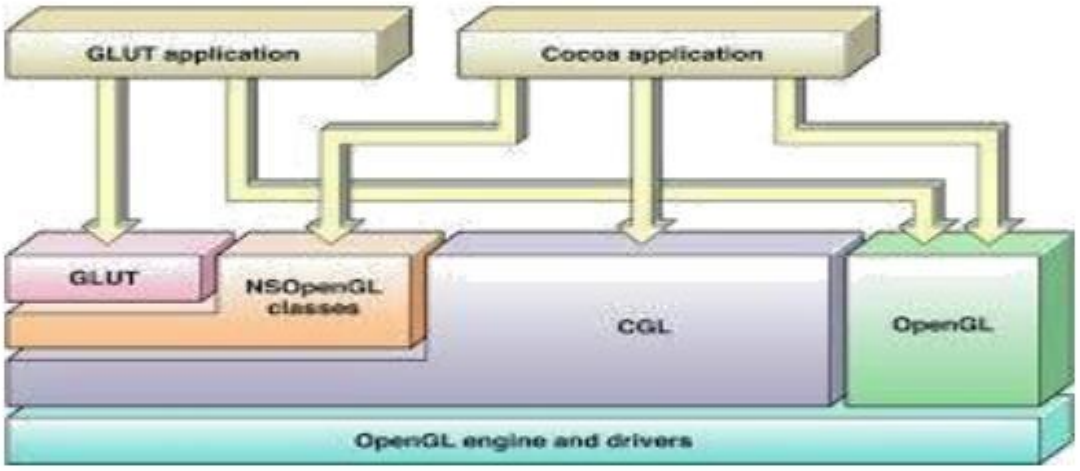

FIG:2.2 OPENGL ENGINE AND DRIVERS

CPU-GPU Cooperation

The architecture of OpenGL is based on a client-server model. An application program written to use the OpenGL API is the "client" and runs on the CPU. The implementation of the OpenGL graphics engine (including the GLSL shader programs we write) is the "server" and runs on the GPU. Geometry and many other types of attributes are stored in buffers called Vertex Buffer Objects (or VBOs). These buffers are allocated on the GPU and filled by your CPU program. We will get our first glimpse into this process (including how these buffers are allocated, used, and deleted) in the first sample program we will study.

Modeling, rendering, and interaction is very much a cooperative process between the CPU client program and the GPU server programs written in GLSL. An important part of the design process is to decide how best to divide the work and how best to package and communicate required information from the CPU to the GPU. There is no standard "best way" to do this that is applicable to all programs, but we will study a few very common approaches.

Window Manager Interfaces

OpenGL is a pure output-oriented modeling and rendering API. It has no facilities for creating and managing windows, obtaining runtime events, or any other such window system dependent operation. OpenGL implicitly assumes a window-system interface that fills these needs and invokes user-written event handlers as appropriate.

It is beyond the scope of these notes to list, let alone compare and contrast, all window manager interfaces that can be used with OpenGL. Two very common window system interfaces are:
GLFW: Runs on Linux, Macintosh, and Windows.
GLUT: (actually *freeglut*): A window interface that used to be the most common one used when teaching OpenGL. It, too, runs on Linux, Macintosh, and Windows.

3) Application Development-API's

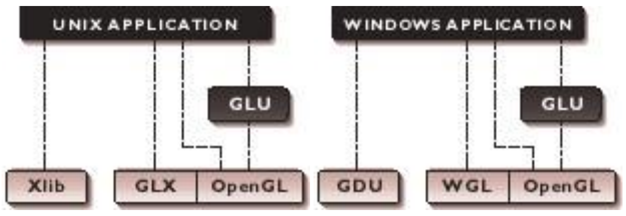

FIG:2.3 APPLICATIONS DEVELOPMENT(API'S)

This diagram demonstrates the relationship between OpenGL GLU and windowing APIs.

Leading software developers use OpenGL, with its robust rendering libraries, as the 2D/3D graphics foundation for higher-level APIs. Developers leverage the capabilities of OpenGL to deliver highly differentiated, yet widely supported vertical market solutions. For example, Open Inventor provides a cross-platform user interface and flexible scene graph that makes it easy to create OpenGL applications. IRIS Performer < leverages OpenGL functionality and delivers additional features tailored for the demanding high frame rate markets such as visual simulation and virtual sets OpenGL Optimizer is a toolkit for real-time interaction, modification, and rendering of complex surface-based models such as those found in CAD/CAM and special effects creation. OpenGL Volumizer is a high-level immediate mode volume rendering API for the energy, medical and sciences markets. OpenGL Shader provides a common interface to support realistic visual effects, bump mapping, multiple textures, environment maps, volume shading and an unlimited array of new effects using hardware acceleration on standard OpenGL graphics cards.

CHAPTER 3
IMPLEMENTATION

CODE SECTION:

```
#include<stdio.h>
#include<GL/glut.h>
#include <GL/gl.h>
#include <stdlib.h>
#define SPEED 30.0

float i=0.0,m=0.0,n=0.0,o=0.0,c=0.0;

int light=1,day=1,plane=0,comet=0,xm=900,train=0;
char ch;

void declare(char *string)
{
    while(*string)
      glutBitmapCharacter(GLUT_BITMAP_TIMES_ROMAN_24, *string++);
}

void draw_pixel(GLint cx, GLint cy)
{

        glBegin(GL_POINTS);
                glVertex2i(cx,cy);
        glEnd();
}

void plotpixels(GLint h,GLint k, GLint x,GLint y)
{
        draw_pixel(x+h,y+k);
        draw_pixel(-x+h,y+k);
        draw_pixel(x+h,-y+k);
        draw_pixel(-x+h,-y+k);
        draw_pixel(y+h,x+k);
        draw_pixel(-y+h,x+k);
        draw_pixel(y+h,-x+k);
        draw_pixel(-y+h,-x+k);
}

void draw_circle(GLint h, GLint k, GLint r)
{
        GLint d=1-r, x=0, y=r;
        while(y>x)
        {
                plotpixels(h,k,x,y);
                if(d<0) d+=2*x+3;
                else
                {
                        d+=2*(x-y)+5;
```

```
                                        --y;
                        }
                        ++x;
                }
                plotpixels(h,k,x,y);
}


void draw_object()
{
int l;
if(day==1)
{
//sky
glColor3f(0.0,0.9,0.9);
glBegin(GL_POLYGON);
glVertex2f(0,380);
glVertex2f(0,700);
glVertex2f(1100,700);
glVertex2f(1100,380);
glEnd();

//sun



        for(l=0;l<=35;l++)
{
                glColor3f(1.0,0.9,0.0);
                draw_circle(100,625,l);
}



//plane
if(plane==1)
{
        glColor3f(1.0,1.0,1.0);
        glBegin(GL_POLYGON);
        glVertex2f(925+n,625+o);
glVertex2f(950+n,640+o);
        glVertex2f(1015+n,640+o);
        glVertex2f(1030+n,650+o);
        glVertex2f(1050+n,650+o);
        glVertex2f(1010+n,625+o);
glEnd();

        glColor3f(0.8,0.8,0.8);
        glBegin(GL_LINE_LOOP);
        glVertex2f(925+n,625+o);
```

```
glVertex2f(950+n,640+o);
          glVertex2f(1015+n,640+o);
          glVertex2f(1030+n,650+o);
          glVertex2f(1050+n,650+o);
          glVertex2f(1010+n,625+o);
glEnd();

}

//cloud1


          for(l=0;l<=20;l++)
          {
                    glColor3f(1.0,1.0,1.0);
                    draw_circle(160+m,625,l);

          }


          for(l=0;l<=35;l++)
          {
                    glColor3f(1.0,1.0,1.0);
                    draw_circle(200+m,625,l);
                    draw_circle(225+m,625,l);
          }

          for(l=0;l<=20;l++)
          {
                    glColor3f(1.0,1.0,1.0);
                    draw_circle(265+m,625,l);
          }

//cloud2


          for(l=0;l<=20;l++)
          {
                    glColor3f(1.0,1.0,1.0);
                    draw_circle(370+m,615,l);
}




          for(l=0;l<=35;l++)
          {
```

```
                    glColor3f(1.0,1.0,1.0);
                    draw_circle(410+m,615,l);
                    draw_circle(435+m,615,l);
                    draw_circle(470+m,615,l);
          }

for(l=0;l<=20;l++)
          {
                    glColor3f(1.0,1.0,1.0);
                    draw_circle(500+m,615,l);
}






//grass
glColor3f(0.0,0.9,0.0);
glBegin(GL_POLYGON);
glVertex2f(0,160);
glVertex2f(0,380);
glVertex2f(1100,380);
glVertex2f(1100,160);
glEnd();

}


else
{

//sky
glColor3f(0.0,0.0,0.0);
glBegin(GL_POLYGON);
glVertex2f(0,380);
glVertex2f(0,700);
glVertex2f(1100,700);
glVertex2f(1100,380);
glEnd();

//moon
int l;

          for(l=0;l<=35;l++)
          {
                    glColor3f(1.0,1.0,1.0);
                    draw_circle(100,625,l);
```

```
            }

//star1

glColor3f(1.0,1.0,1.0);
glBegin(GL_TRIANGLES);
glVertex2f(575,653);
glVertex2f(570,645);
glVertex2f(580,645);
glVertex2f(575,642);
glVertex2f(570,650);
glVertex2f(580,650);
glEnd();

//star2
glColor3f(1.0,1.0,1.0);
glBegin(GL_TRIANGLES);
glVertex2f(975,643);
glVertex2f(970,635);
glVertex2f(980,635);
glVertex2f(975,632);
glVertex2f(970,640);
glVertex2f(980,640);
glEnd();

//star3
glColor3f(1.0,1.0,1.0);
glBegin(GL_TRIANGLES);
glVertex2f(875,543);
glVertex2f(870,535);
glVertex2f(880,535);
glVertex2f(875,532);
glVertex2f(870,540);
glVertex2f(880,540);
glEnd();

//star4
glColor3f(1.0,1.0,1.0);
glBegin(GL_TRIANGLES);
glVertex2f(375,598);
glVertex2f(370,590);
glVertex2f(380,590);
glVertex2f(375,587);
glVertex2f(370,595);
glVertex2f(380,595);
glEnd();

//star5
```

```
glColor3f(1.0,1.0,1.0);
glBegin(GL_TRIANGLES);
glVertex2f(750,628);
glVertex2f(745,620);
glVertex2f(755,620);
glVertex2f(750,618);
glVertex2f(745,625);
glVertex2f(755,625);
glEnd();

//star6
glColor3f(1.0,1.0,1.0);
glBegin(GL_TRIANGLES);
glVertex2f(200,628);
glVertex2f(195,620);
glVertex2f(205,620);
glVertex2f(200,618);
glVertex2f(195,625);
glVertex2f(205,625);
glEnd();

//star7
glColor3f(1.0,1.0,1.0);
glBegin(GL_TRIANGLES);
glVertex2f(100,528);
glVertex2f(95,520);
glVertex2f(105,520);
glVertex2f(100,518);
glVertex2f(95,525);
glVertex2f(105,525);
glEnd();

//star8
glColor3f(1.0,1.0,1.0);
glBegin(GL_TRIANGLES);
glVertex2f(300,468);
glVertex2f(295,460);
glVertex2f(305,460);
glVertex2f(300,458);
glVertex2f(295,465);
glVertex2f(305,465);
glEnd();

//star9
glColor3f(1.0,1.0,1.0);
glBegin(GL_TRIANGLES);
glVertex2f(500,543);
glVertex2f(495,535);
```

```
glVertex2f(505,535);
glVertex2f(500,532);
glVertex2f(495,540);
glVertex2f(505,540);
glEnd();


//comet
if(comet==1)
{
	for(l=0;l<=7;l++)
	{
		glColor3f(1.0,1.0,1.0);
		draw_circle(300+c,675,l);
	}

glColor3f(1.0,1.0,1.0);
	glBegin(GL_TRIANGLES);
	glVertex2f(200+c,675);
	glVertex2f(300+c,682);
	glVertex2f(300+c,668);
	glEnd();
}

//Plane
if(plane==1)
{

	for(l=0;l<=1;l++)
	{
		glColor3f(1.0,0.0,0.0);
		draw_circle(950+n,625+o,l);
		glColor3f(1.0,1.0,0.0);
		draw_circle(954+n,623+o,l);

	}



}

//grass
glColor3f(0.0,0.3,0.0);
glBegin(GL_POLYGON);
glVertex2f(0,160);
glVertex2f(0,380);
glVertex2f(1100,380);
glVertex2f(1100,160);
```

```
glEnd();

}


//track boundary
glColor3f(1.0,1.0,1.0);
glBegin(GL_POLYGON);
glVertex2f(0,150);
glVertex2f(0,160);
glVertex2f(1100,160);
glVertex2f(1100,150);
glEnd();

//platform

glColor3f(0.560784,0.560784,0.737255);
glBegin(GL_POLYGON);
glVertex2f(0,160);
glVertex2f(0,250);
glVertex2f(1100,250);
glVertex2f(1100,160);
glEnd();

//table 1

glColor3f(1.0,0.498039,0.0);
glBegin(GL_POLYGON);
glVertex2f(140,190);
glVertex2f(140,210);
glVertex2f(155,210);
glVertex2f(155,190);
glEnd();

glColor3f(0.2,0.2,0.2);
glBegin(GL_POLYGON);
glVertex2f(130,210);
glVertex2f(130,215);
glVertex2f(225,215);
glVertex2f(225,210);
glEnd();

glColor3f(1.0,0.498039,0.0);
glBegin(GL_POLYGON);
glVertex2f(200,190);
glVertex2f(200,210);
glVertex2f(215,210);
glVertex2f(215,190);
```

```
glEnd();

//table 2

glColor3f(1.0,0.498039,0.0);
glBegin(GL_POLYGON);
glVertex2f(390,190);
glVertex2f(390,210);
glVertex2f(405,210);
glVertex2f(405,190);
glEnd();

glColor3f(0.2,0.2,0.2);
glBegin(GL_POLYGON);
glVertex2f(380,210);
glVertex2f(380,215);
glVertex2f(475,215);
glVertex2f(475,210);
glEnd();

glColor3f(1.0,0.498039,0.0);
glBegin(GL_POLYGON);
glVertex2f(450,190);
glVertex2f(450,210);
glVertex2f(465,210);
glVertex2f(465,190);
glEnd();

//table 3

glColor3f(1.0,0.498039,0.0);
glBegin(GL_POLYGON);
glVertex2f(840,190);
glVertex2f(840,210);
glVertex2f(855,210);
glVertex2f(855,190);
glEnd();

glColor3f(0.2,0.2,0.2);
glBegin(GL_POLYGON);
glVertex2f(830,210);
glVertex2f(830,215);
glVertex2f(925,215);
glVertex2f(925,210);
glEnd();

glColor3f(1.0,0.498039,0.0);
glBegin(GL_POLYGON);
```

```
glVertex2f(900,190);
glVertex2f(900,210);
glVertex2f(915,210);
glVertex2f(915,190);
glEnd();

//below track
glColor3f(0.623529,0.623529,0.372549);
glBegin(GL_POLYGON);
glVertex2f(0,0);
glVertex2f(0,150);
glVertex2f(1100,150);
glVertex2f(1100,0);
glEnd();

//Railway track

glColor3f(0.0,0.0,0.0);
glBegin(GL_POLYGON);
glVertex2f(-100,0);
glVertex2f(-100,20);
glVertex2f(1100,20);
glVertex2f(1100,0);
glEnd();

glBegin(GL_POLYGON);
glVertex2f(-100,80);
glVertex2f(-100,100);
glVertex2f(1100,100);
glVertex2f(1100,80);
glEnd();

glBegin(GL_POLYGON);
glVertex2f(0,0);
glVertex2f(0,80);
glVertex2f(10,80);
glVertex2f(10,0);
glEnd();

glBegin(GL_POLYGON);
glVertex2f(80,0);
glVertex2f(80,80);
glVertex2f(90,80);
glVertex2f(90,0);
glEnd();

glBegin(GL_POLYGON);
glVertex2f(150,0);
```

```
glVertex2f(150,80);
glVertex2f(160,80);
glVertex2f(160,0);
glEnd();

glBegin(GL_POLYGON);
glVertex2f(220,0);
glVertex2f(220,80);
glVertex2f(230,80);
glVertex2f(230,0);
glEnd();

glBegin(GL_POLYGON);
glVertex2f(290,0);
glVertex2f(290,80);
glVertex2f(300,80);
glVertex2f(300,0);
glEnd();

glBegin(GL_POLYGON);
glVertex2f(360,0);
glVertex2f(360,80);
glVertex2f(370,80);
glVertex2f(370,0);
glEnd();

glBegin(GL_POLYGON);
glVertex2f(430,0);
glVertex2f(430,80);
glVertex2f(440,80);
glVertex2f(440,0);
glEnd();

glBegin(GL_POLYGON);
glVertex2f(500,0);
glVertex2f(500,80);
glVertex2f(510,80);
glVertex2f(510,0);
glEnd();

glBegin(GL_POLYGON);
glVertex2f(570,0);
glVertex2f(570,80);
glVertex2f(580,80);
glVertex2f(580,0);
glEnd();

glBegin(GL_POLYGON);
```

```
glVertex2f(640,0);
glVertex2f(640,80);
glVertex2f(650,80);
glVertex2f(650,0);
glEnd();

glBegin(GL_POLYGON);
glVertex2f(710,0);
glVertex2f(710,80);
glVertex2f(720,80);
glVertex2f(720,0);
glEnd();

glBegin(GL_POLYGON);
glVertex2f(770,0);
glVertex2f(770,80);
glVertex2f(780,80);
glVertex2f(780,0);
glEnd();

glBegin(GL_POLYGON);
glVertex2f(840,0);
glVertex2f(840,80);
glVertex2f(850,80);
glVertex2f(850,0);
glEnd();

glBegin(GL_POLYGON);
glVertex2f(900,0);
glVertex2f(900,80);
glVertex2f(910,80);
glVertex2f(910,0);
glEnd();

glBegin(GL_POLYGON);
glVertex2f(970,0);
glVertex2f(970,80);
glVertex2f(980,80);
glVertex2f(980,0);
glEnd();

glBegin(GL_POLYGON);
glVertex2f(1040,0);
glVertex2f(1040,80);
glVertex2f(1050,80);
glVertex2f(1050,0);
glEnd();
```

```
//track bounbary
glColor3f(0.647059,0.164706,0.164706);
glBegin(GL_POLYGON);
glVertex2f(-100,100);
glVertex2f(-100,150);
glVertex2f(1100,150);
glVertex2f(1100,100);
glEnd();

//railway station boundary (fence)
glColor3f(0.647059,0.164706,0.164706);
glBegin(GL_POLYGON);
glVertex2f(0,250);
glVertex2f(0,310);
glVertex2f(5,320);
glVertex2f(10,310);
glVertex2f(10,250);
glEnd();

glBegin(GL_POLYGON);
glVertex2f(90,250);
glVertex2f(90,310);
glVertex2f(95,320);
glVertex2f(100,310);
glVertex2f(100,250);
glEnd();

glBegin(GL_POLYGON);
glVertex2f(140,250);
glVertex2f(140,310);
glVertex2f(145,320);
glVertex2f(150,310);
glVertex2f(150,250);
glEnd();

glBegin(GL_POLYGON);
glVertex2f(190,250);
glVertex2f(190,310);
glVertex2f(195,320);
glVertex2f(200,310);
glVertex2f(200,250);
glEnd();

glBegin(GL_POLYGON);
glVertex2f(240,250);
glVertex2f(240,310);
glVertex2f(245,320);
glVertex2f(250,310);
```

```
glVertex2f(250,250);
glEnd();

glBegin(GL_POLYGON);
glVertex2f(340,250);
glVertex2f(340,310);
glVertex2f(345,320);
glVertex2f(350,310);
glVertex2f(350,250);
glEnd();

glBegin(GL_POLYGON);
glVertex2f(390,250);
glVertex2f(390,310);
glVertex2f(395,320);
glVertex2f(400,310);
glVertex2f(400,250);
glEnd();

glBegin(GL_POLYGON);
glVertex2f(950,250);
glVertex2f(950,310);
glVertex2f(955,320);
glVertex2f(960,310);
glVertex2f(960,250);
glEnd();

glBegin(GL_POLYGON);
glVertex2f(1000,250);
glVertex2f(1000,310);
glVertex2f(1005,320);
glVertex2f(1010,310);
glVertex2f(1010,250);
glEnd();

glBegin(GL_POLYGON);
glVertex2f(1050,250);
glVertex2f(1050,310);
glVertex2f(1055,320);
glVertex2f(1060,310);
glVertex2f(1060,250);
glEnd();

glBegin(GL_POLYGON);
glVertex2f(950,295);
glVertex2f(950,305);
glVertex2f(1100,305);
glVertex2f(1100,295);
```

```
glEnd();

glBegin(GL_POLYGON);
glVertex2f(950,265);
glVertex2f(950,275);
glVertex2f(1100,275);
glVertex2f(1100,265);
glEnd();

glBegin(GL_POLYGON);
glVertex2f(0,295);
glVertex2f(0,305);
glVertex2f(400,305);
glVertex2f(400,295);
glEnd();

glBegin(GL_POLYGON);
glVertex2f(0,265);
glVertex2f(0,275);
glVertex2f(400,275);
glVertex2f(400,265);
glEnd();

//tree 1
glColor3f(0.9,0.2,0.0);
glBegin(GL_POLYGON);
glVertex2f(50,185);
glVertex2f(50,255);
glVertex2f(65,255);
glVertex2f(65,185);
glEnd();


        for(l=0;l<=30;l++)
        {
                glColor3f(0.0,0.5,0.0);
                draw_circle(40,250,l);
                draw_circle(80,250,l);
        }

        for(l=0;l<=25;l++)
        {
                glColor3f(0.0,0.5,0.0);
                draw_circle(50,290,l);
                draw_circle(70,290,l);
        }

        for(l=0;l<=20;l++)
        {
                glColor3f(0.0,0.5,0.0);
                draw_circle(60,315,l);
        }

//tree 2
glColor3f(0.9,0.2,0.0);
glBegin(GL_POLYGON);
glVertex2f(300,185);
glVertex2f(300,255);
glVertex2f(315,255);
glVertex2f(315,185);
glEnd();


        for(l=0;l<=30;l++)
        {
                glColor3f(0.0,0.5,0.0);
                draw_circle(290,250,l);
                draw_circle(330,250,l);
        }

        for(l=0;l<=25;l++)
        {
                glColor3f(0.0,0.5,0.0);
                draw_circle(300,290,l);
                draw_circle(320,290,l);
        }

        for(l=0;l<=20;l++)
        {
                glColor3f(0.0,0.5,0.0);
                draw_circle(310,315,l);
        }


//tree 5
glColor3f(0.9,0.2,0.0);
glBegin(GL_POLYGON);
glVertex2f(1050,185);
glVertex2f(1050,255);
glVertex2f(1065,255);
glVertex2f(1065,185);
glEnd();


        for(l=0;l<=30;l++)
        {
                glColor3f(0.0,0.5,0.0);
                draw_circle(1040,250,l);
                draw_circle(1080,250,l);
        }

        for(l=0;l<=25;l++)
        {
                glColor3f(0.0,0.5,0.0);
                draw_circle(1050,290,l);
                draw_circle(1070,290,l);
        }

        for(l=0;l<=20;l++)
        {
                glColor3f(0.0,0.5,0.0);
                draw_circle(1060,315,l);
        }


//railway station
glColor3f(0.647059,0.164706,0.164706);
glBegin(GL_POLYGON);
glVertex2f(400,250);
glVertex2f(400,450);
glVertex2f(950,450);
glVertex2f(950,250);
glEnd();

//roof
glColor3f(0.556863,0.419608,0.137255);
glBegin(GL_POLYGON);
glVertex2f(350,450);
glVertex2f(450,500);
glVertex2f(900,500);
glVertex2f(1000,450);
glEnd();
//side window
glColor3f(0.556863,0.419608,0.137255);
glBegin(GL_POLYGON);
glVertex2f(400,400);
glVertex2f(350,350);
glVertex2f(400,350);
glEnd();

//side window
glColor3f(0.556863,0.419608,0.137255);
glBegin(GL_POLYGON);
glVertex2f(950,400);
```

```
glVertex2f(1000,350);
glVertex2f(950,350);
glEnd();


glColor3f(0.847059,0.847059,0.74902);
glBegin(GL_POLYGON);
glVertex2f(425,250);
glVertex2f(425,250);
glVertex2f(425,400);
glVertex2f(450,425);
glVertex2f(550,425);
glVertex2f(575,400);
glVertex2f(575,250);
glEnd();

glBegin(GL_POLYGON);
glVertex2f(600,250);
glVertex2f(600,400);
glVertex2f(625,425);
glVertex2f(725,425);
glVertex2f(750,400);
glVertex2f(750,250);
glEnd();

glBegin(GL_POLYGON);
glVertex2f(775,250);
glVertex2f(775,400);
glVertex2f(800,425);
glVertex2f(900,425);
glVertex2f(925,400);
glVertex2f(925,250);
glEnd();

//window 1
glColor3f(0.196078,0.6,0.8);
glBegin(GL_POLYGON);
glVertex2f(450,300);
glVertex2f(450,375);
glVertex2f(550,375);
glVertex2f(550,300);
glEnd();

glColor3f(0.0,0.0,0.0);
glBegin(GL_LINES);
glVertex2f(450,337.5);
glVertex2f(550,337.5);
glEnd();
```

```
glColor3f(0.0,0.0,0.0);
glBegin(GL_LINES);
glVertex2f(500,375);
glVertex2f(500,300);
glEnd();


//window 2
glColor3f(0.196078,0.6,0.8);
glBegin(GL_POLYGON);
glVertex2f(800,300);
glVertex2f(800,375);
glVertex2f(900,375);
glVertex2f(900,300);
glEnd();

glColor3f(0.0,0.0,0.0);
glBegin(GL_LINES);
glVertex2f(800,337.5);
glVertex2f(900,337.5);
glEnd();

glColor3f(0.0,0.0,0.0);
glBegin(GL_LINES);
glVertex2f(850,375);
glVertex2f(850,300);
glEnd();

//door
glColor3f(0.329412,0.329412,0.329412);
glBegin(GL_POLYGON);
glVertex2f(625,250);
glVertex2f(625,375);
glVertex2f(725,375);
glVertex2f(725,250);
glEnd();


//signal
        glColor3f(1.0,0.0,0.0);
        glBegin(GL_POLYGON);
                glVertex2f(1060,160);
                glVertex2f(1060,350);
                glVertex2f(1070,350);
                glVertex2f(1070,160);
        glEnd();
```

```
            glColor3f(0.7,0.7,0.7);
            glBegin(GL_POLYGON);
                        glVertex2f(1040,350);
                        glVertex2f(1040,500);
                        glVertex2f(1090,500);
                        glVertex2f(1090,350);
            glEnd();

            for(l=0;l<=20;l++)
            {
                        glColor3f(0.0,0.0,0.0);
                        draw_circle(1065,475,l);
                        glColor3f(1.0,1.0,0.0);
                        draw_circle(1065,425,l);
                        glColor3f(0.0,0.0,0.0);
                        draw_circle(1065,375,l);
            }
if(train==1)
{
//train carrier 3

glColor3f(0.258824,0.435294,0.258824);
glBegin(GL_POLYGON);
glVertex2f(-600+i-xm,50);
glVertex2f(-600+i-xm,300);
glVertex2f(-1000+i-xm,300);
glVertex2f(-1000+i-xm,50);
glEnd();

//top

glColor3f(0.309804,0.184314,0.184314);
glBegin(GL_POLYGON);
glVertex2f(-590+i-xm,300);
glVertex2f(-590+i-xm,310);
glVertex2f(-1010+i-xm,310);
glVertex2f(-1010+i-xm,300);
glEnd();

// Windows

glColor3f(1.0,1.0,1.0);
glBegin(GL_POLYGON);
glVertex2f(-825+i-xm,175);
glVertex2f(-825+i-xm,285);
glVertex2f(-985+i-xm,285);
glVertex2f(-985+i-xm,175);
```

```
glEnd();

glBegin(GL_POLYGON);
glVertex2f(-615+i-xm,175);
glVertex2f(-615+i-xm,285);
glVertex2f(-775+i-xm,285);
glVertex2f(-775+i-xm,175);
glEnd();

// carrier 3 Wheels

for(l=0;l<50;l++)
  {
        glColor3f(0.35,0.16,0.14);
        draw_circle(-675+i-xm,50,l);
        draw_circle(-925+i-xm,50,l);
  }

//train carrier 2

glColor3f(0.258824,0.435294,0.258824);
glBegin(GL_POLYGON);
glVertex2f(-150+i-xm,50);
glVertex2f(-150+i-xm,300);
glVertex2f(-550+i-xm,300);
glVertex2f(-550+i-xm,50);
glEnd();

//top

glColor3f(0.309804,0.184314,0.184314);
glBegin(GL_POLYGON);
glVertex2f(-140+i-xm,300);
glVertex2f(-140+i-xm,310);
glVertex2f(-560+i-xm,310);
glVertex2f(-560+i-xm,300);
glEnd();

// Windows

glColor3f(1.0,1.0,1.0);
glBegin(GL_POLYGON);
glVertex2f(-375+i-xm,175);
glVertex2f(-375+i-xm,285);
glVertex2f(-535+i-xm,285);
glVertex2f(-535+i-xm,175);
glEnd();
```

```
glBegin(GL_POLYGON);
glVertex2f(-165+i-xm,175);
glVertex2f(-165+i-xm,285);
glVertex2f(-325+i-xm,285);
glVertex2f(-325+i-xm,175);
glEnd();

//connecter

glColor3f(0.309804,0.184314,0.184314);
glBegin(GL_POLYGON);
glVertex2f(-550+i-xm,75);
glVertex2f(-550+i-xm,95);
glVertex2f(-600+i-xm,95);
glVertex2f(-600+i-xm,75);
glEnd();

// carrier 2 Wheels

for(l=0;l<50;l++)
  {
        glColor3f(0.35,0.16,0.14);
        draw_circle(-225+i-xm,50,l);
        draw_circle(-475+i-xm,50,l);
  }

// train carrier 1


glColor3f(0.258824,0.435294,0.258824);
glBegin(GL_POLYGON);
glVertex2f(300+i-xm,50);
glVertex2f(300+i-xm,300);
glVertex2f(-100+i-xm,300);
glVertex2f(-100+i-xm,50);
glEnd();

//top

glColor3f(0.309804,0.184314,0.184314);
glBegin(GL_POLYGON);
glVertex2f(310+i-xm,300);
glVertex2f(310+i-xm,310);
glVertex2f(-110+i-xm,310);
glVertex2f(-110+i-xm,300);
glEnd();

// Windows
```

```
glColor3f(1.0,1.0,1.0);
glBegin(GL_POLYGON);
glVertex2f(75+i-xm,175);
glVertex2f(75+i-xm,285);
glVertex2f(-85+i-xm,285);
glVertex2f(-85+i-xm,175);
glEnd();

glBegin(GL_POLYGON);
glVertex2f(285+i-xm,175);
glVertex2f(285+i-xm,285);
glVertex2f(125+i-xm,285);
glVertex2f(125+i-xm,175);
glEnd();

//connecter

glColor3f(0.309804,0.184314,0.184314);
glBegin(GL_POLYGON);
glVertex2f(-100+i-xm,75);
glVertex2f(-100+i-xm,95);
glVertex2f(-150+i-xm,95);
glVertex2f(-150+i-xm,75);
glEnd();

// carrier 1 Wheels

for(l=0;l<50;l++)
  {
        glColor3f(0.35,0.16,0.14);
        draw_circle(-25+i-xm,50,l);
        draw_circle(225+i-xm,50,l);
  }

//train base

glColor3f(0.196078,0.6,0.8);
glBegin(GL_POLYGON);
glVertex2f(350+i-xm,50);
glVertex2f(350+i-xm,125);
glVertex2f(755+i-xm,125);
glVertex2f(820+i-xm,50);
glEnd();

//train control chamber

glColor3f(1.0,0.25,0.0);
```

```
glBegin(GL_POLYGON);
glVertex2f(360+i-xm,125);
glVertex2f(360+i-xm,325);
glVertex2f(560+i-xm,325);
glVertex2f(560+i-xm,125);
glEnd();

//window

glColor3f(1.0,1.0,1.0);
glBegin(GL_POLYGON);
glVertex2f(375+i-xm,175);
glVertex2f(375+i-xm,315);
glVertex2f(545+i-xm,315);
glVertex2f(545+i-xm,175);
glEnd();

//train top

glColor3f(0.309804,0.184314,0.184314);
glBegin(GL_POLYGON);
glVertex2f(350+i-xm,325);
glVertex2f(350+i-xm,350);
glVertex2f(570+i-xm,350);
glVertex2f(570+i-xm,325);
glEnd();

//tain engine
glColor3f(1.0,0.5,0.0);
glBegin(GL_POLYGON);
glVertex2f(560+i-xm,125);
glVertex2f(560+i-xm,225);
glVertex2f(755+i-xm,225);
glVertex2f(755+i-xm,125);
glEnd();

glColor3f(0.0,0.0,0.0);
glBegin(GL_POLYGON);
glVertex2f(580+i-xm,125);
glVertex2f(580+i-xm,225);
glVertex2f(590+i-xm,225);
glVertex2f(590+i-xm,125);
glEnd();

glColor3f(0.0,0.0,0.0);
glBegin(GL_POLYGON);
glVertex2f(600+i-xm,125);
glVertex2f(600+i-xm,225);
```

```
glVertex2f(610+i-xm,225);
glVertex2f(610+i-xm,125);
glEnd();

glColor3f(0.0,0.0,0.0);
glBegin(GL_POLYGON);
glVertex2f(735+i-xm,125);
glVertex2f(735+i-xm,225);
glVertex2f(745+i-xm,225);
glVertex2f(745+i-xm,125);
glEnd();

//tain smoke

glColor3f(0.196078,0.6,0.9);
glBegin(GL_POLYGON);
glVertex2f(650+i-xm,225);
glVertex2f(650+i-xm,275);
glVertex2f(700+i-xm,275);
glVertex2f(700+i-xm,225);
glEnd();

glColor3f(0.309804,0.184314,0.184314);
glBegin(GL_POLYGON);
glVertex2f(640+i-xm,275);
glVertex2f(640+i-xm,300);
glVertex2f(710+i-xm,300);
glVertex2f(710+i-xm,275);
glEnd();

//train head-light

glColor3f(1.0,1.0,0.0);
glBegin(GL_POLYGON);
glVertex2f(755+i-xm,225);
glVertex2f(765+i-xm,225);
glVertex2f(765+i-xm,185);
glVertex2f(755+i-xm,185);
glEnd();

glColor3f(0.0,0.0,0.0);
glBegin(GL_POLYGON);
glVertex2f(755+i-xm,225);
glVertex2f(785+i-xm,225);
glVertex2f(755+i-xm,205);

glEnd();
```

```
// train connector

glColor3f(0.309804,0.184314,0.184314);
glBegin(GL_POLYGON);
glVertex2f(350+i-xm,75);
glVertex2f(350+i-xm,95);
glVertex2f(300+i-xm,95);
glVertex2f(300+i-xm,75);
glEnd();

//train wheels

for(l=0;l<50;l++)
  {
        glColor3f(0.35,0.16,0.14);
        draw_circle(425+i-xm,50,l);
        draw_circle(700+i-xm,50,l);
  }
}
  //Railway Station Board

glColor3f(0.5,0.5,0.5);
glBegin(GL_POLYGON);
glVertex2f(580,500);
glVertex2f(580,520);
glVertex2f(590,520);
glVertex2f(590,500);
glEnd();

glColor3f(0.5,0.5,0.5);
glBegin(GL_POLYGON);
glVertex2f(770,500);
glVertex2f(770,520);
glVertex2f(780,520);
glVertex2f(780,500);
glEnd();

glColor3f(0.435294,0.258824,0.258824);
glBegin(GL_POLYGON);
glVertex2f(560,510);
glVertex2f(560,540);
glVertex2f(580,550);
glVertex2f(780,550);
glVertex2f(800,540);
glVertex2f(800,510);
glEnd();

glColor3f(1.0,1.0,1.0);
```

```
    glRasterPos2f(570,520);
    declare("RAILWAY STATION");

glFlush();
}



void traffic_light()
{
         int l;
if(light==1)
           {
for(l=0;l<=20;l++)
                  {

glColor3f(0.0,0.0,0.0);
                  draw_circle(1065,475,l);

                  glColor3f(0.0,0.7,0.0);
                  draw_circle(1065,375,l);
                  }
           }

         else
           {

for(l=0;l<=20;l++)
                  {
                  glColor3f(1.0,0.0,0.0);
                  draw_circle(1065,475,l);

                  glColor3f(0.0,0.0,0.0);
                  draw_circle(1065,375,l);
                  }
           }
}


void idle()
{
glClearColor(1.0,1.0,1.0,1.0);

if(light==0 && (i>=0 && i<=1150))
 {

          i+=SPEED/10;
    m+=SPEED/150;
```

```
            n-=2;
             o+=0.2;
            c+=2;

 }

 if(light==0 && (i>=2600 && i<=3000))
 {

             i+=SPEED/10;
             m+=SPEED/150;
            n-=2;
             o+=0.2;
            c+=2;

 }

if(light==0)
 {
             i=i;
             m+=SPEED/150;
            n-=2;
             o+=0.2;
            c+=2;

 }

else
 {

     i+=SPEED/10;
     m+=SPEED/150;
            n-=2;
             o+=0.2;
            c+=2;
 }
if(i>3500)
             i=0.0;
if(m>1100)
             m=0.0;
if( o>75)
 {
            plane=0;
 }
if(c>500)
 {
            comet=0;
 }
```

```
glutPostRedisplay();

}

void mouse(int btn,int state,int x,int y)
{
        if(btn==GLUT_LEFT_BUTTON && state==GLUT_UP)
exit(0);
}


void keyboardFunc( unsigned char key, int x, int y )
{
switch( key )
   {
case 'g':
case 'G':
light=1;
break;

        case 'r':
        case 'R':
                light=0;
                break;

case 'd':
        case 'D':
                day=1;
                break;

        case 'n':
case 'N':
                day=0;
                break;

        case 't':
case 'T':
                train=1;
                i=0;
                break;

   };

}


void main_menu(int index)
{
        switch(index)
        {
        case 1:
        if(index==1)
         {
        plane=1;
                o=n=0.0;
         }
        break;

        case 2:
        if(index==2)
         {
                comet=1;
                 c=0.0;
         }
        break;
        }
}



void myinit()
{
glClearColor(1.0,1.0,1.0,1.0);
glColor3f(0.0,0.0,1.0);
glPointSize(2.0);
glMatrixMode(GL_PROJECTION);
glLoadIdentity();
gluOrtho2D(0.0,1100.0,0.0,700.0);
}



void display()
{

glClear(GL_COLOR_BUFFER_BIT);
draw_object();
traffic_light();
glFlush();
}


int main(int argc,char** argv)
{
int c_menu;
        printf("Project by CSEMiniProjects.com\n");
```

```
    printf("------------------------------------------------------------------------------");
    printf("                    ARRIVAL AND DEPARTURE OF TRAIN                              ");
    printf("------------------------------------------------------------------------------");
            printf("Press 'r' or 'R' to change the signal light to red. \n\n");
            printf("Press 'g' or 'G' to change the signal light to green. \n\n");
    printf("Press 'd' or 'D' to make it day. \n\n");
            printf("Press 'n' or 'N' to make it night. \n\n");
            printf("Press 't' or 'T' Train arrive at station.\n\n");
            printf("Press RIGHT MOUSE BUTTON to display menu. \n\n");
            printf("Press LEFT MOUSE BUTTON to quit the program. \n\n\n");
            printf("Press any key and Hit ENTER.\n");
            scanf("%s",&ch);

            glutInit(&argc,argv);
            glutInitDisplayMode(GLUT_SINGLE|GLUT_RGB);
            glutInitWindowSize(1100.0,700.0);
            glutInitWindowPosition(0,0);
            glutCreateWindow("Traffic Control");
            glutDisplayFunc(display);
    glutIdleFunc(idle);
            glutKeyboardFunc(keyboardFunc);
            glutMouseFunc(mouse);
            myinit();
            c_menu=glutCreateMenu(main_menu);
            glutAddMenuEntry("Aeroplane",1);
            glutAddMenuEntry("Comet",2);
            glutAttachMenu(GLUT_RIGHT_BUTTON);
            glutMainLoop();
            return 0;
}
```

Commonly used Functions of OpenGL includes:

glBegin, glEnd	The glBegin and glEnd functions delimit the vertices of a primitive or a group of like primitives.
glClear	The glClear function clears buffers to preset values.
glColor	These functions set the current color
glFlush	The glFlush function forces execution of OpenGL functions in finite time.
glLoadIdentity	The glLoadIdentity function replaces the current matrix with the identity matrix.
glMatrixMode	The glMatrixMode function specifies which matrix is the current matrix.
glVertex	These functions specify a vertex.
gluOrtho2D	The gluOrtho2D function defines a 2-D orthographic projection matrix.

CHAPTER 4
RESULTS AND SNAPSHOTS

Design preview:

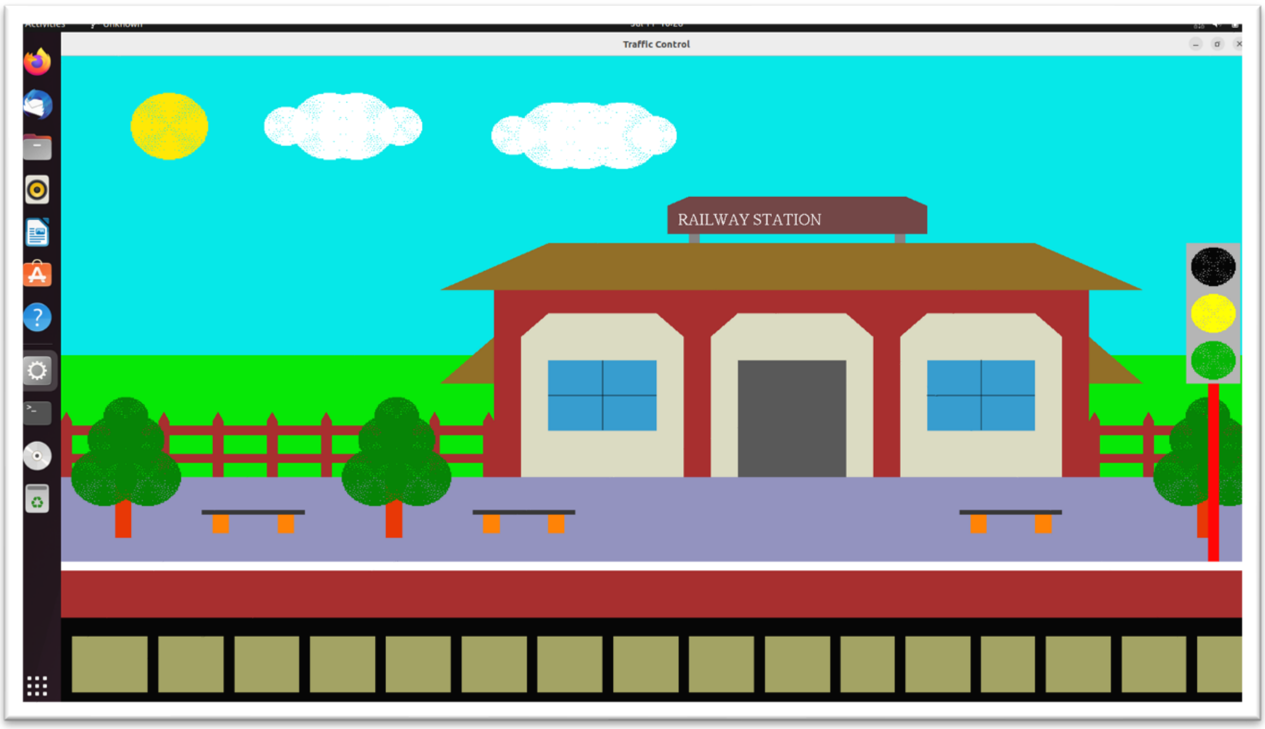

Figure 4.1: Railway station background in day theme

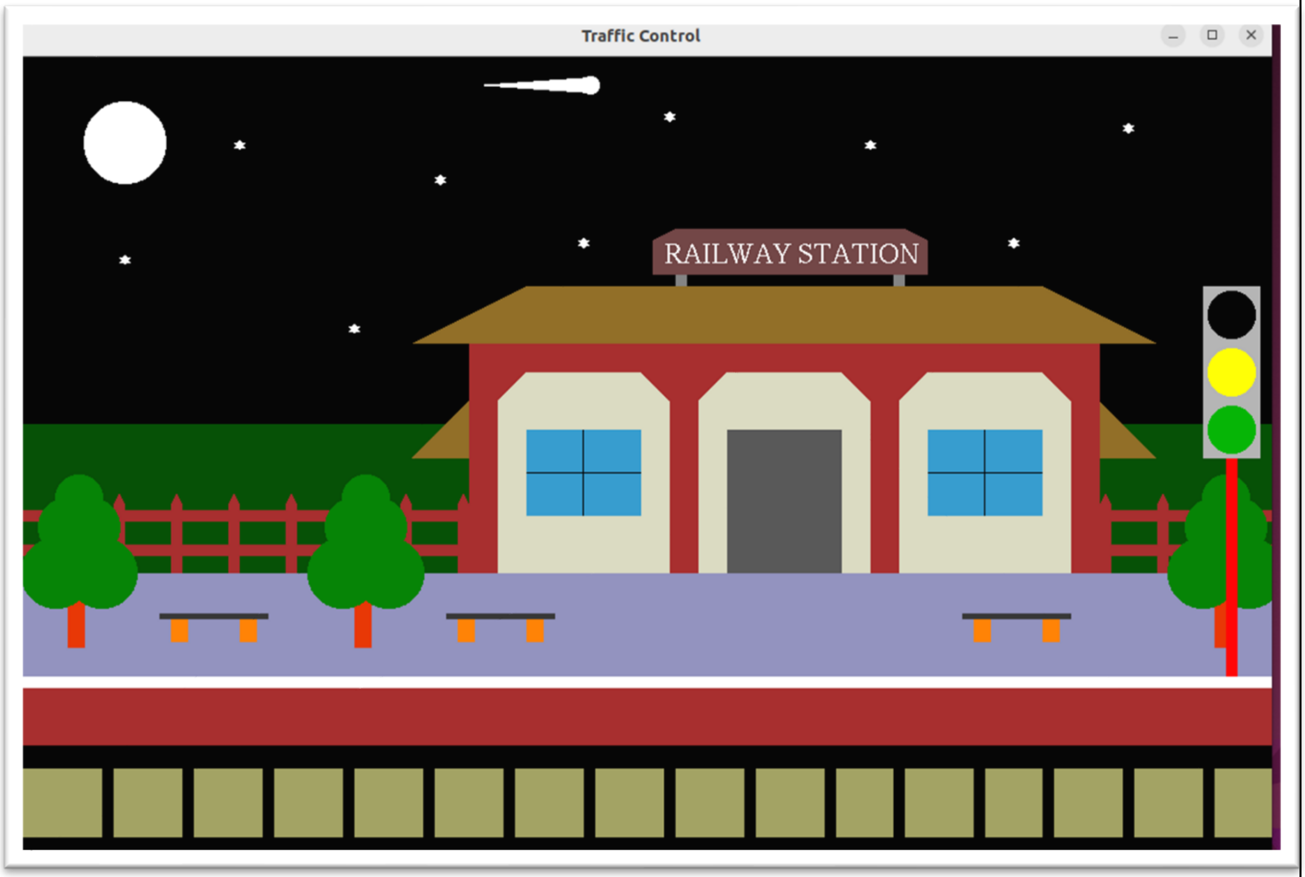

Figure 4.2: Movement of comet in night theme

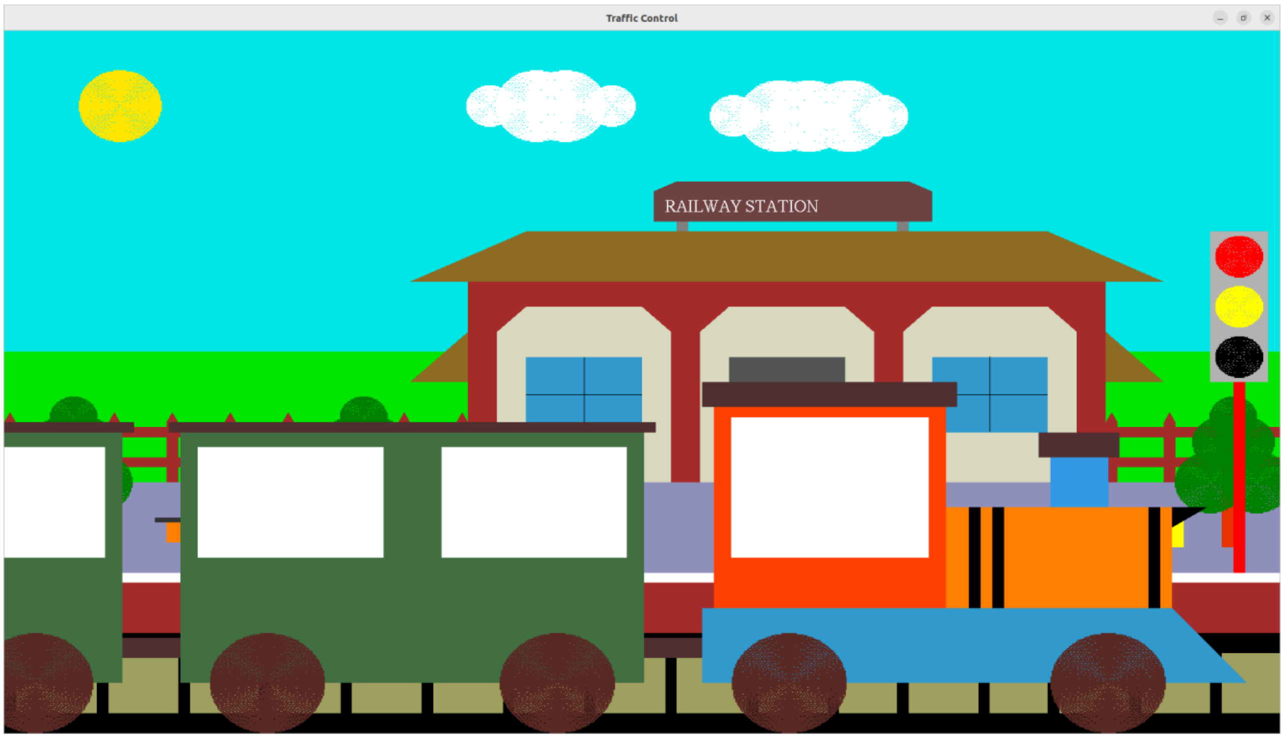

Figure 4.3: Train stops when r key is pressed through keyboard

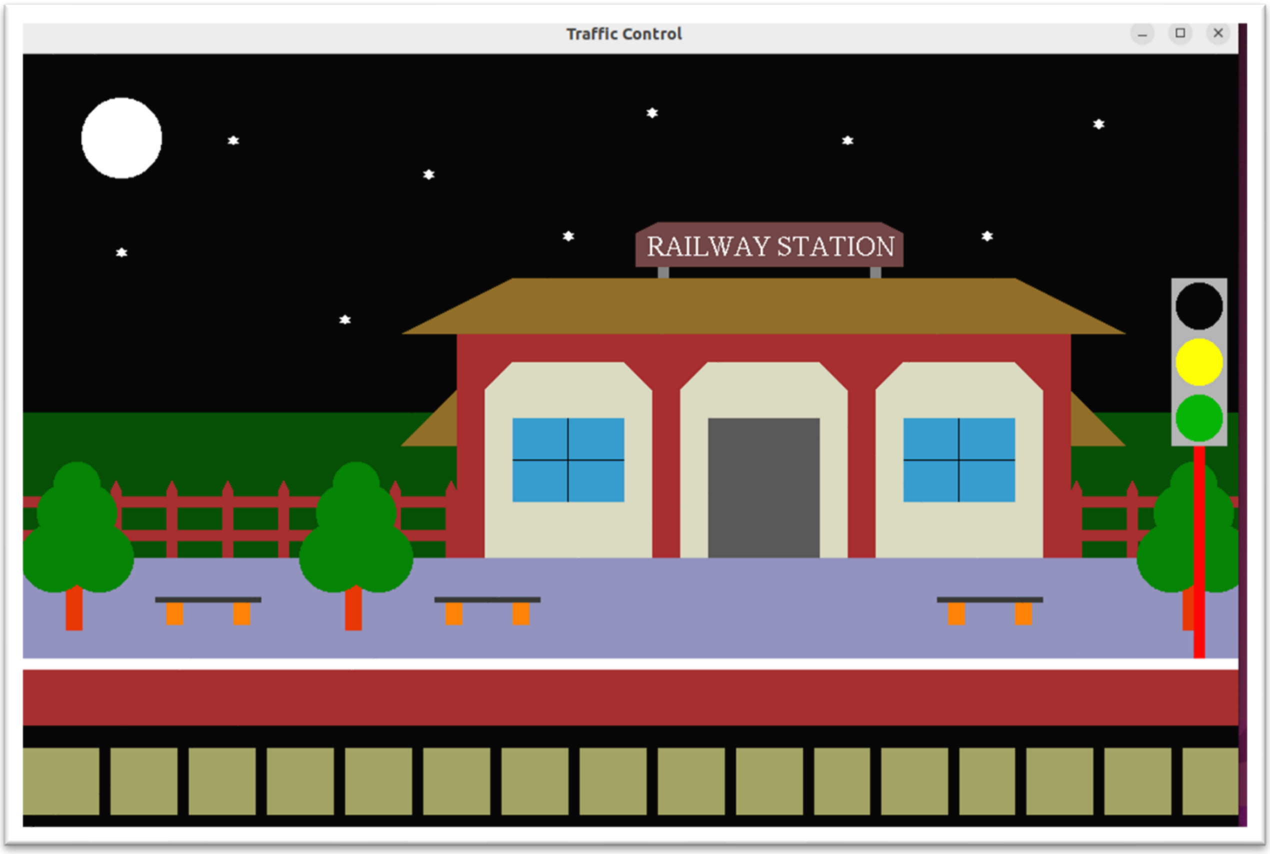

Figure 4.2: Night theme of railway station

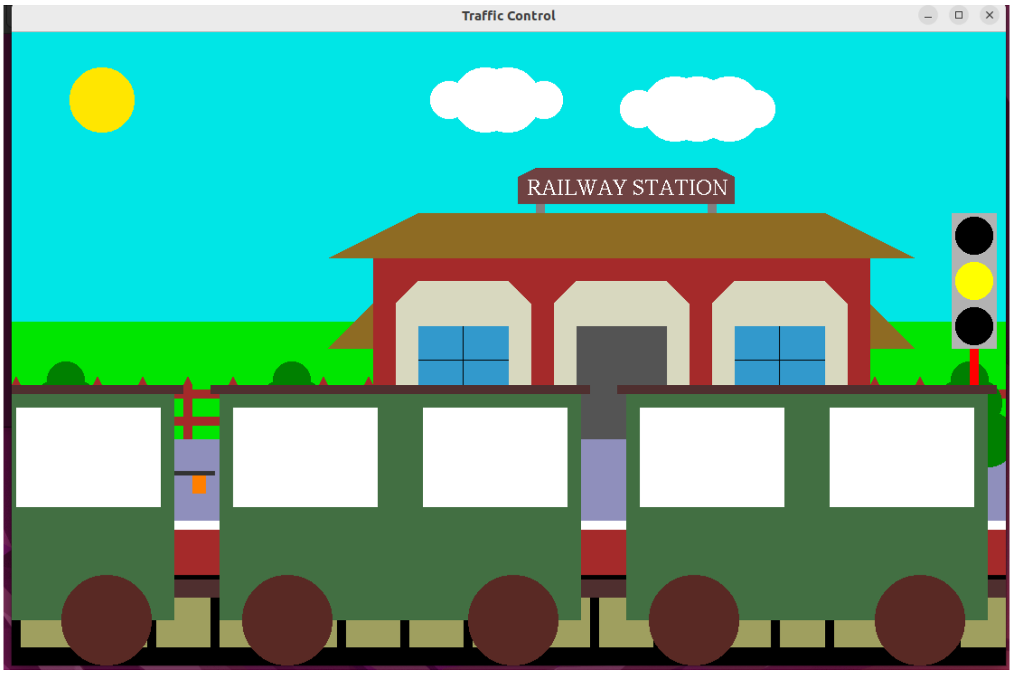

Figure 4.2: Moving of train

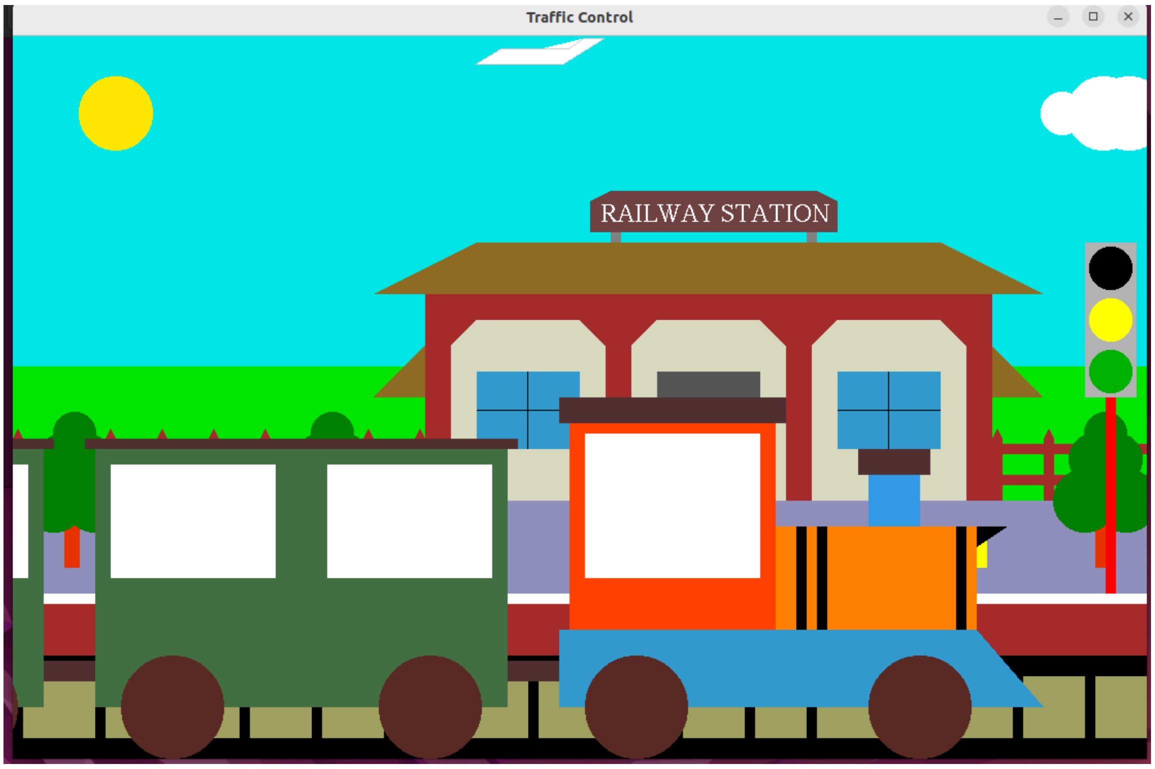

Figure 4.2: Aeroplane movement in day theme

Summary:

We are able to successfully implement the movement of train, aeroplane, comet using opengl function and primitives. The movement of aeroplane and comet is successfully implemented as a menu driven process. The theme of the railway station can be changed using keyboard controls.

CHAPTER 5
CONCLUSION

CONCLUSION

In this project we were able to complete the implementation of **“Railway Station”** problem using Graphics library. The graphics concepts whatever we learnt in theory, we applied in this project.

Using different graphics functions for the windows, shapes and bringing text on the screen using the concept of ASCII value was the most challenging task we faced during the project implementation phase.

Now we have a proper understanding of how to use different graphics library functions for different shapes, display, text etc.

REFERENCES

[1] The OpenGL Utility Toolkit (GLUT) documentation: https://www.opengl.org/resources/libraries/glut/spec3/spec3.html/

[2] Donald D. Hearn, M. Pauline Baker - Computer Graphics with OpenGL (3rd Edition)-Prentice Hall (2003)

[3] Edward Angel - Interactive Computer Graphics_ A Top-Down Approach Using OpenGL (5th Edition) (2008)