PROJECT REPORT

On

“TRAIN CONTROL SYSTEM”

Submitted in partial fulfillment of

Computer Graphics and Laboratory with mini project(17CSL68)

Sixth Semester of the Degree of Bachelor of Engineering

in

Department of Computer Science and Engineering

Visvesvaraya Technological University, Belagavi

during the year 2019-20

Carried out by

Name: ADARSH KUMAR

USN: 1SB17CS0056

Under the guidance of

<MRS.SHARON ROJI PRIYA>

Assistant Professor

Dept. of Computer Science and Engg.

Sri Sairam College of Engineering
Sai Leo Nagar, Guddanahalli Post,
Anekal, Bengaluru – 562106

Sri SAIRAM COLLEGE OF ENGINEERING
Anekal, Bengaluru

Accredited by NAAC
ISO 9001:2015 Certified Institution
Approved by AICTE, New Delhi
Affiliated to Visvesvaraya Technological University
www.sairamce.edu.in

Department of Computer Science and Engineering

CERTIFICATE

Certified that the project entitled <**TRAIN CONTROL SYSTEM>** is a bonafide work carried out by **<ADARSH KUMAR>, <1SB17CS005>** in partial fulfillment of sixth semester of the degree of bachelor of engineering in **Computer Science and Engineering** of the Visvesvaraya Technological University, Belgaum during the year **2019-2020**. It is certified that all corrections/suggestions indicated for Internal Assessment have been incorporated in the report deposited in the department library. The project has been approved as it satisfies the academic requirements in respect of project work prescribed for the Bachelor of Engineering Degree.

Signature of the Guide Signature of the HOD

< MRS.SHARON ROJI PRIYA>

Dr.G.Manjula

Asst.Prof,CSE Dept HOD,CSE Dept

SSCE SSCE

External Viva

Name of the Examiners: 1.________________________

2.______________________

ABSTRACT

The main aim of Train Signal Computer Graphics Mini Project is to illustrate the concepts and usage of pre-built functions in OpenGL. Train signal project simulated the railway crossing where two tracks are connected. When there â€™s just a single railway track then a signal is used to stop one train and let another train to cross over using crossing track. Finally, once the train has crossed over then the signal is turned back to green and the stopped train is allowed to go normally. We have used input devices like mouse and keyboard to interact with the program.

ACKNOWLEDGEMENT

It has been an honor and privilege to do my project on **<TRAIN CONTROL SYSTEM>.**I take this opportunity to convey my sincere thanks and regards to Our

Chairman and Chief Executive Officer **Sri. Sai Prakash Leo Muthu**, and **Dr. Arun Kumar R**, Management Representative for offered me a chance to study in this institute and everyone who has helped me in successful completion of this seminar.

I take immense pleasure in expressing my sincere gratitude to **Dr. B SHADAKSHARAPPA, Principal, Sri Sairam College of Engineering, Bengaluru** for giving me a constant encouragement and support to achieve my goal.

My sincere thanks to **Dr.G.Manjula, Professor & Head, Dept. of Computer Science and Engineering** for permitting us to undertake the project work and for her invaluable guidance.

I would like to thank my guide **<MRS.SHARON ROJI PRIYA>,Asst. Prof**, **Dept. of Computer Science and Engineering,** without her constant motivation and guidance at every stage of this project, this work could not have been materialized.

I perceive as this opportunity as a big milestone in my career development. I will strive to use gained skills and knowledge in the best possible way, and will continue to work on their improvement, in order to attain desired career objectives. Hope to continue cooperation with all of you in the future.

CONTENTS

List of Figures

List of Snapshots

• INTRODUCTION

• COMPUTER GRAPHICS

In this era of computing, computer graphics has become one of the most powerful and interesting fact of computing. It all started with display of data on hardcopy and CRT screen. Now computer graphics is about creation, retrieval, manipulation of models and images.

Graphics today is used in many different areas. Graphics provides one of the most natural means of communicating within a computer, since our highly developed 2D and 3D pattern-recognition abilities allow us to perceive and process pictorial data rapidly and effectively. Interactive computer graphics is the most important means of producing pictures since the invention of photography and television. It has the added advantage that, with the computer, we can make pictures not only of concrete real world objects but also of abstract, synthetic objects, such as mathematical surfaces and of data that have no inherent geometry, such as survey results.

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, coloring, viewing the models. OpenGL offers different coordinate system and frames. OpenGL offers translation, rotation and scaling of objects.

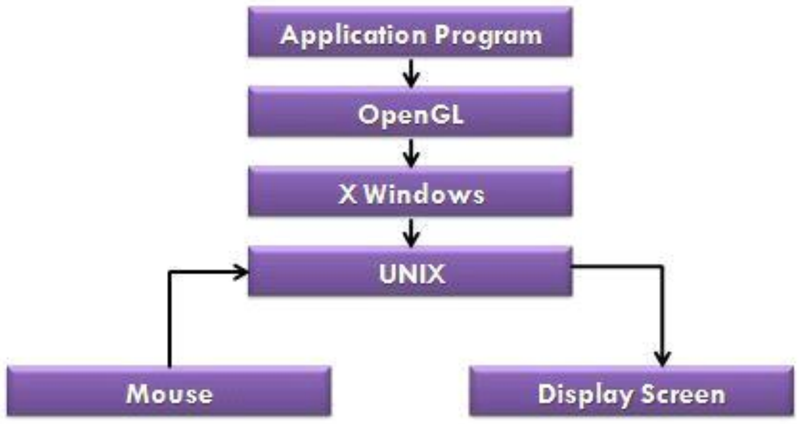


FIG:1.1 OVERVIEW OF COMPUTER GRAPHICS

• <u>OpenGL CONCEPT</u>

1.2.1 <u>INTERFACE</u>

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, coloring, 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 begins 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 color of the objects.

• **OPENGL ARCHITECTURE:**

• **Pipeline Architectures**

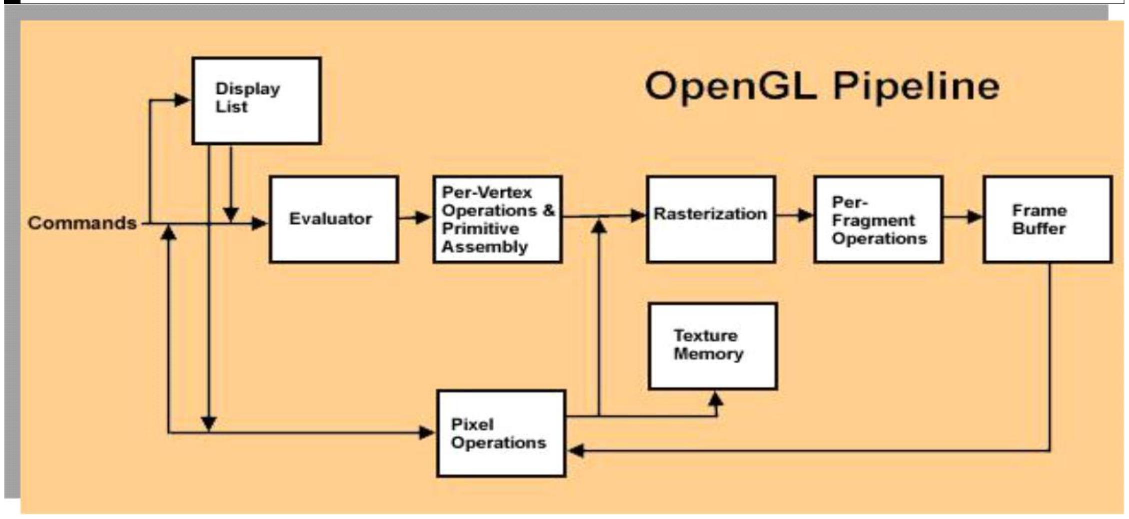


FIG:1.2.3.1 OPENGL PIPELINE ARCHITECTURE

2) <u>OpenGL Engine And Drivers</u>

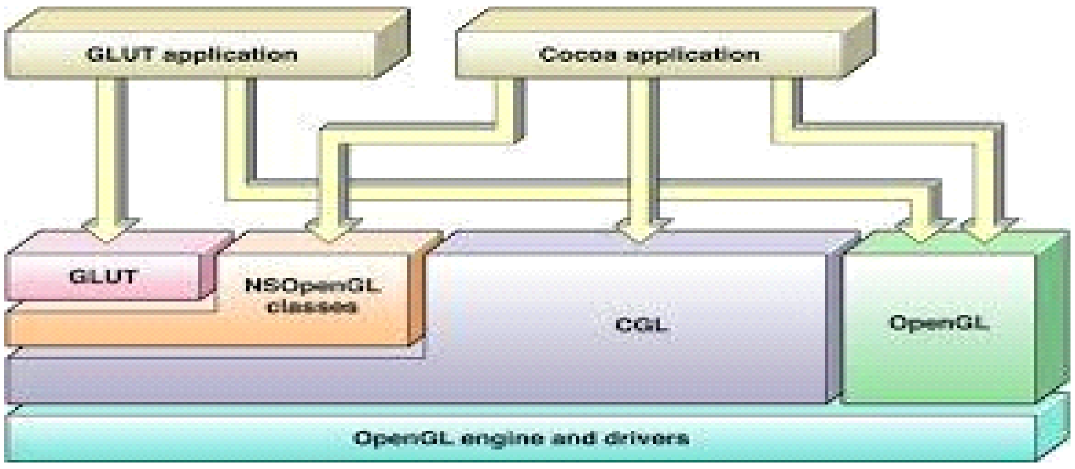


FIG:1.2.3.2 OPENGL ENGINE AND DRIVERS

3) <u>Application Development-API's</u>

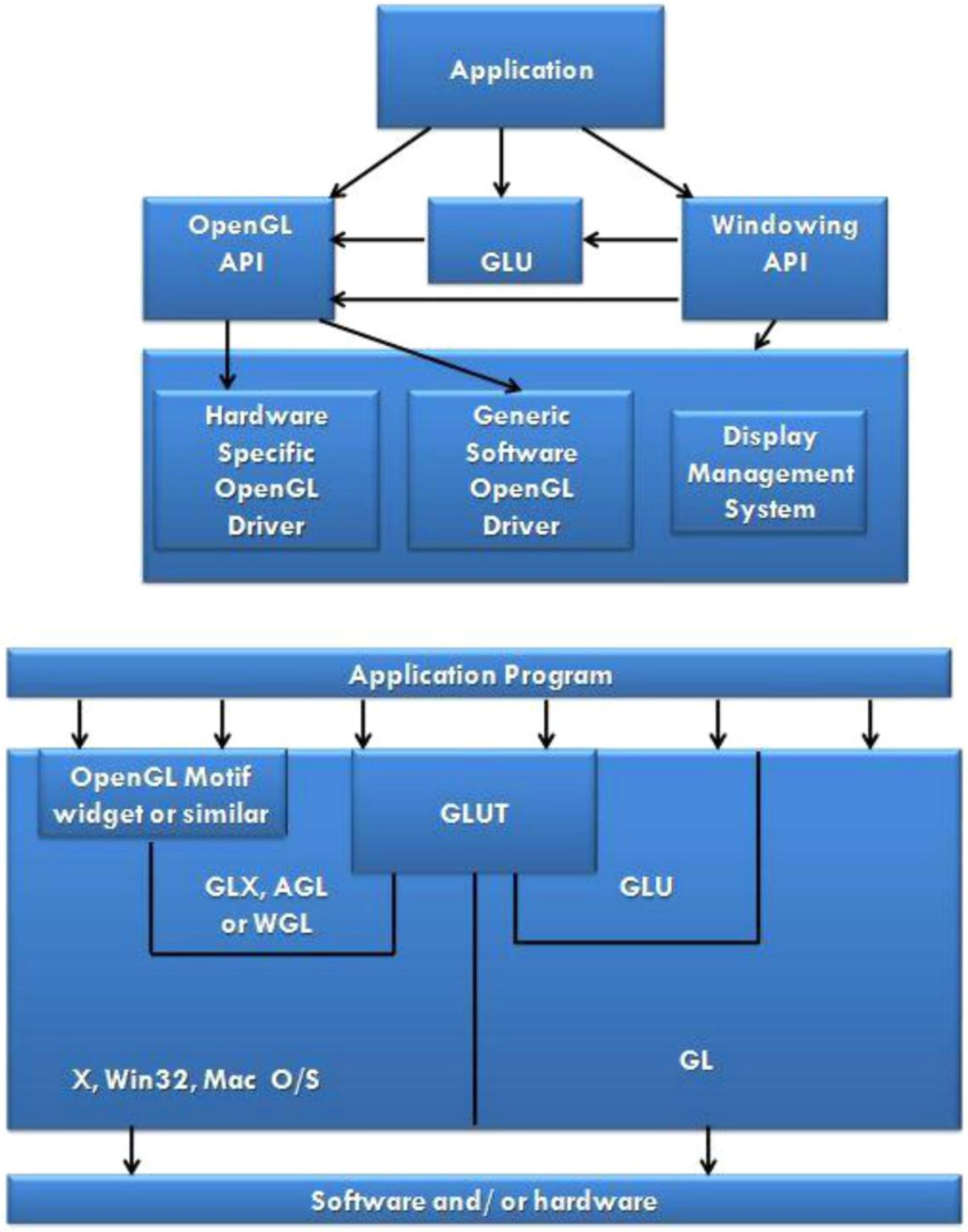

Application
OpenGL API
GLU
Windowing API
Hardware Specific OpenGL Driver
Generic Software OpenGL Driver
Display Management System
Application Program
OpenGL Motif widget or similar
GLUT
GLX, AGL or WGL
GLU
X, Win32, Mac O/S
GL
Software and/ or hardware

FIG:1.2.3.3 APPLICATIONS DEVELOPMENT(API'S)

The above diagram illustrates the relationship of the various libraries and window system components.

Generally, applications which require more user interface support will use a library designed to support those types of features (i.e. buttons, menu and scroll bars, etc.) such as Motif or the Win32 API.

Prototype applications, or ones which do not require all the bells and whistles of a full GUI, may choose to use GLUT instead because of its simplified programming model and window system independence.

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 displaylist 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, colors, 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 color 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 color, depth, and sometimes texture-coordinate 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.REQUIREMENT SPECIFICATION

- **SOFTWARE REQUIREMENTS SPECIFICATION**

This section attempts to bring out the requirements and specifications as given out by the Visvesvaraya Technological University for the completion of the package. Minimum requirements expected are cursor movement, editing picture objects like point, line, circle, ellipse and polygons. Transformations on objects/selected area should be possible. User should be able to open the package do the required operations and exit from the environment.

2.2 EXTERNAL INTERFACE REQUIREMENTS

User Interface:

The interface for the 2D package requires for the user to have a mouse connected, and the corresponding drivers software and header files installed. For the convenience of the user, there are menus and sub -menus displayed on the screen.

Menus:

The Menus consists of various operations related to drawing on the area specified. It also has the 'clear' option for clearing the screen and also changes the background color.

Hardware Interface:

The standard output device, as mentioned earlier has been assumed to be a color monitor. It is quite essential for any graphics package to have this, as provision of color options to the user is a must. The mouse, the main input device, has to be functional. A keyboard is also required. .

Apart from these hardware requirements, there should be sufficient hard disk space and primary memory available for proper working of the package.

Software Interface:

The editor has been implemented on the DOS platform and mainly requires an appropriate version of the compiler to be installed and functional. Though it has been implemented on DOS, it is pretty much platform independent with the

Hardware Requirements:

System	**:**	**Intel**

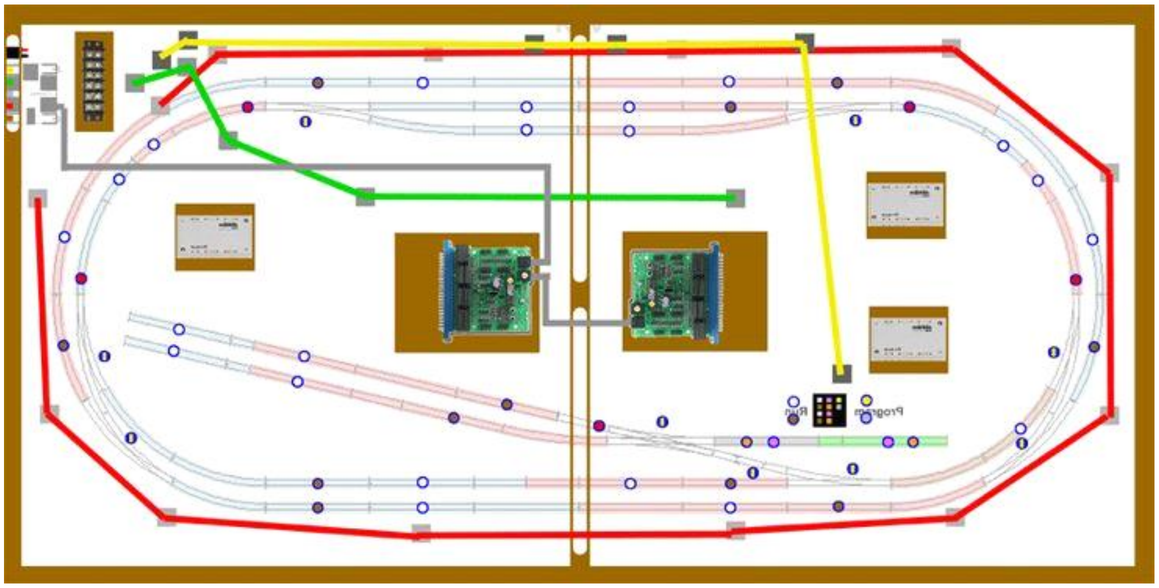
Frequency	**:**	**3.0 GHz**
RAM	**:**	**4 GB**
Disk Capacity	**:**	**500 GB**

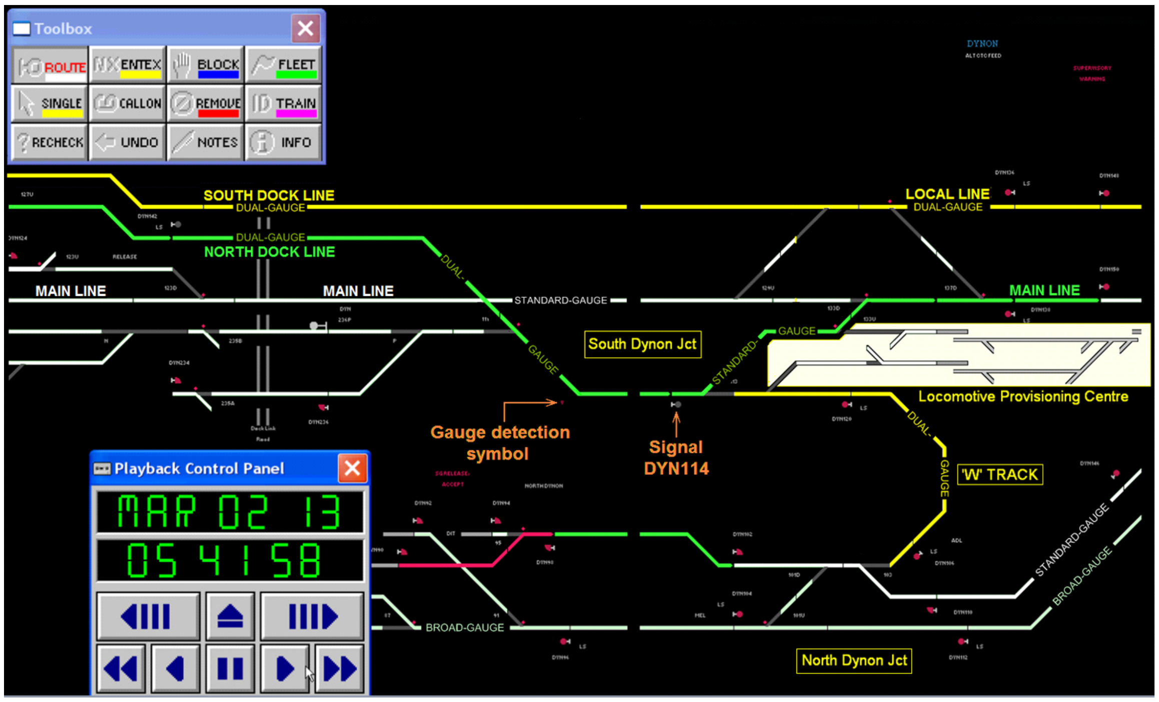
Software Requirements:

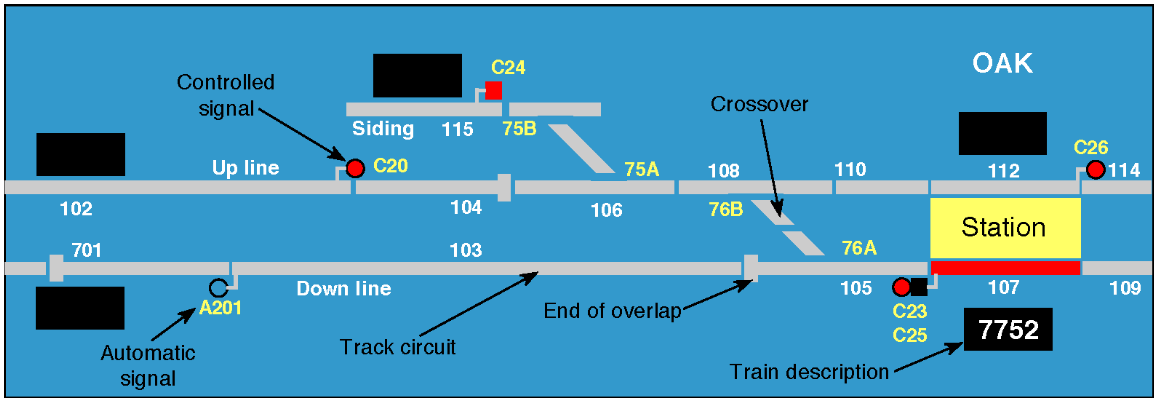
Operating System : WINDOWS XP/7/8

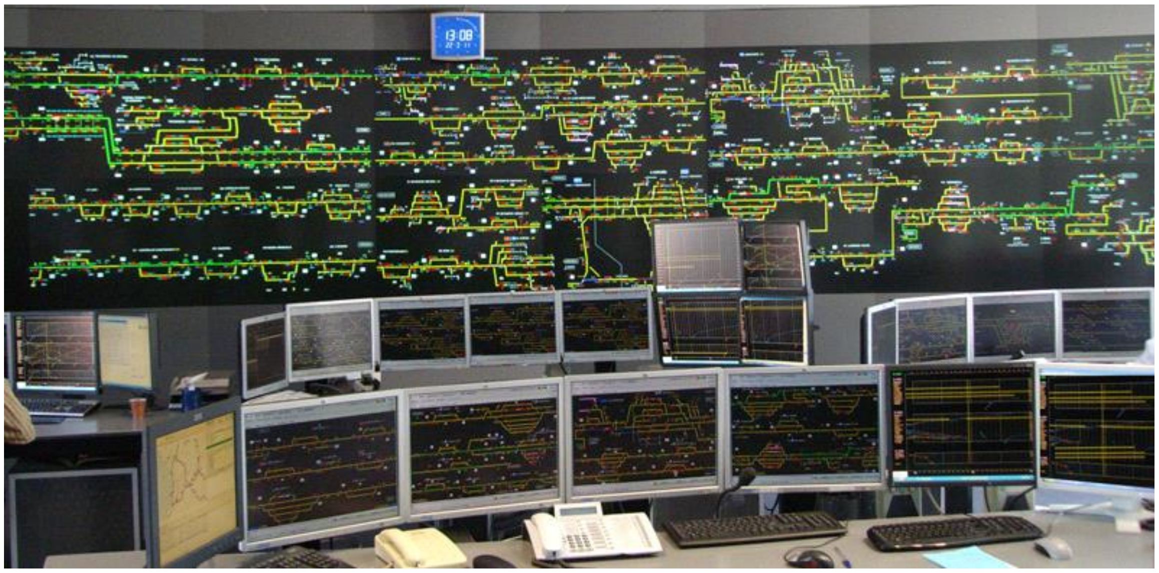
Compiler, IDE : MS Visual Studio 2013 or later

3.SYSTEM DESIGN




Toolbox
ROUTE
ENTEX
BLOCK
FLEET
SINGLE
CALLON
REMOVE
TRAIN
RECHECK
UNDO
NOTES
INFO
DYNON
SOUTH DOCK LINE
DUAL-GAUGE
DUAL-GAUGE
NORTH DOCK LINE
MAIN LINE
MAIN LINE
STANDARD-GAUGE
DUAL-GAUGE
LOCAL LINE
DUAL-GAUGE
MAIN LINE
South Dynon Jct
STANDARD-GAUGE
Locomotive Provisioning Centre
Gauge detection symbol
Signal DYN114
DUAL-GAUGE
'W' TRACK
Playback Control Panel
MAR 02 13
05 41 58
BROAD-GAUGE
STANDARD-GAUGE
BROAD-GAUGE
North Dynon Jct


OAK
Controlled signal
C24
Siding
115
75B
Crossover
Up line
C20
75A
108
110
112
C26
114
102
104
106
76B
Station
701
103
76A
Down line
105
107
109
A201
C23
C25
7752
End of overlap
Automatic signal
Track circuit
Train description



4.IMPLEMENTATION SOURCE CODE

```
#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)
 {
```

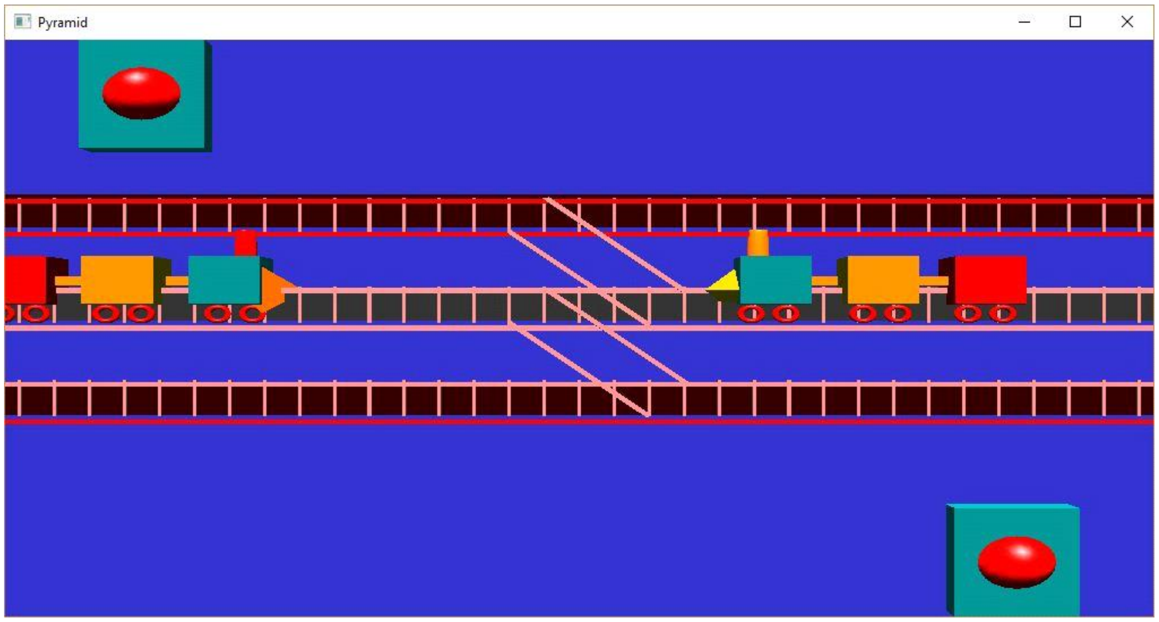
```
 i=i;
 m+=SPEED/150;
n-=2;
 o+=0.2;
c+=2;

 }

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;
```

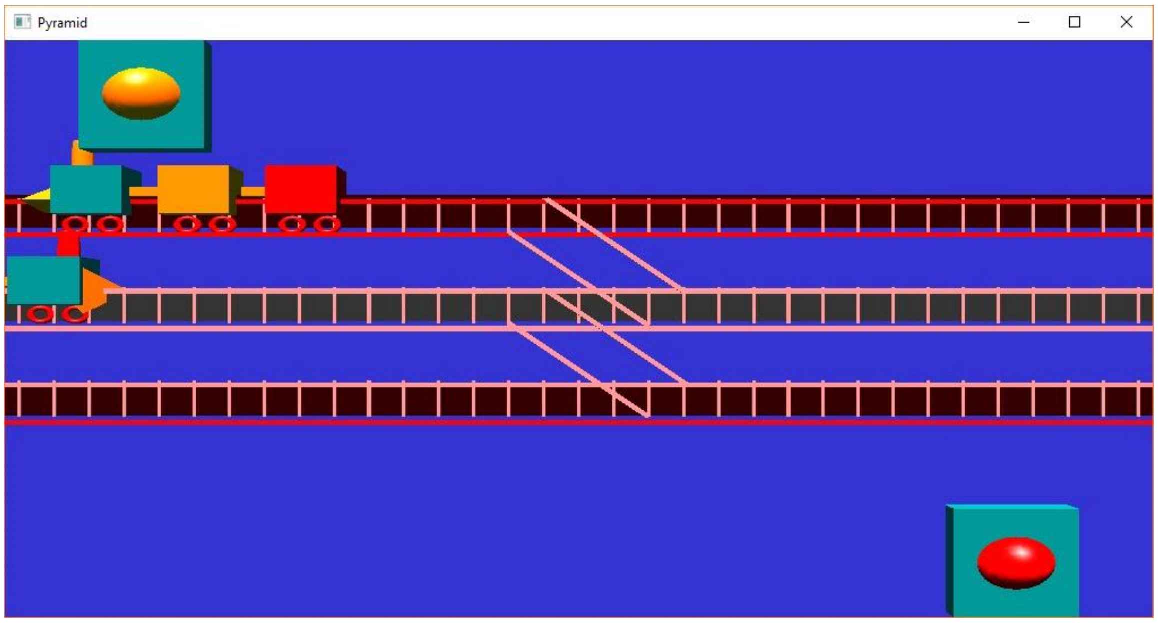
```
 case 'r':
 case 'R':
                                         light=0;
                                         break;

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;
 }
}
```

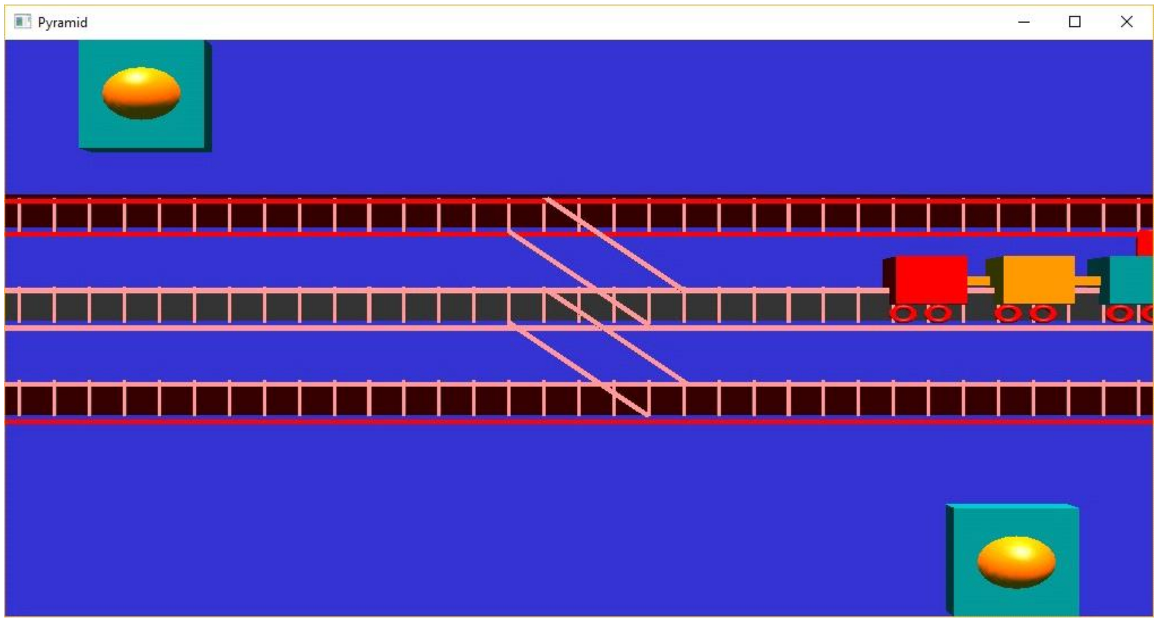
```
void myinit()
{
glClearColor(1.0,1.0,1.0,1.0);
glColor3f(0.0,0.0,1.0);
glPointSize(2.0);
glMatrixMode(GL_PROJECTION);
glLoadIdentity();
gluOrtho2D(0.0,1100.0,0.0,700.0);
}



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;
}
```

5.RESULTS







6.<u>TESTING</u>

- **LIFE CYCLE OF TESTING**

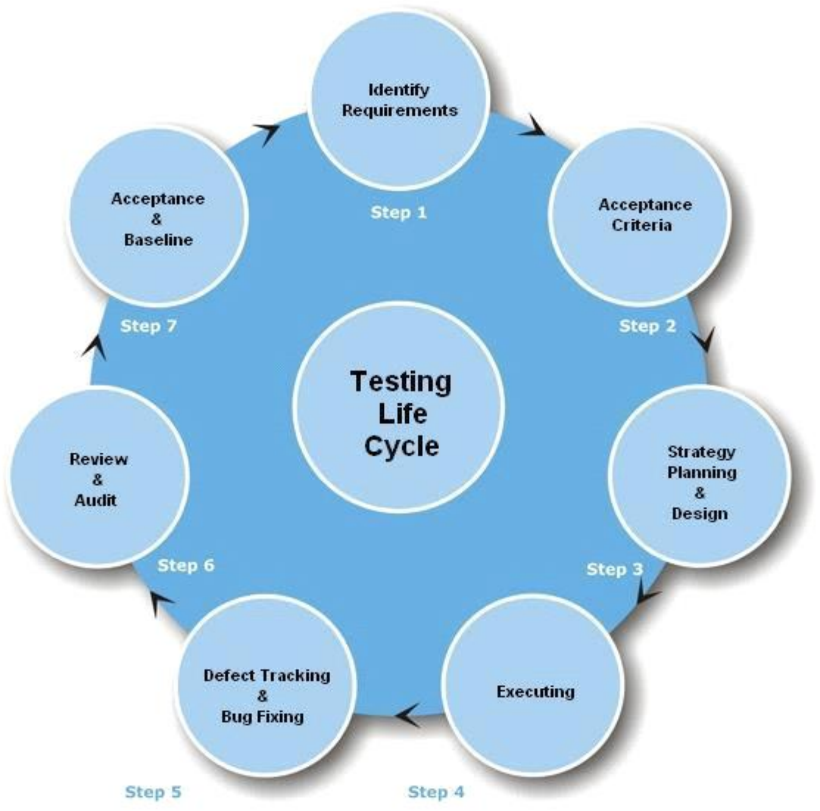


FIG 6.1 TESTING LIFE CYCLE

6.2 TYPES OF TESTING

6.2.1 MANUAL TESTING

Manual testing includes testing a software manually, i.e., without using any automated tool or any script. In this type, the tester takes over the role of an end-user and tests the software to identify any unexpected behavior or bug. There are different stages for manual testing such as unit testing, integration testing, system testing, and user acceptance testing.

6.2.2 AUTOMATION TESTING

Automation testing, which is also known as Test Automation, is when the tester writes scripts and uses another software to test the product. This process involves automation of a manual process. Automation Testing is used to re-run the test scenarios that were performed manually, quickly, and repeatedly.

6.2.3 BLACK-BOX TESTING

The technique of testing without having any knowledge of the interior workings of the application is called black-box testing. The tester is oblivious to the system architecture and does not have access to the source code. Typically, while performing a black-box test, a tester will interact with the system's user interface by providing inputs and examining outputs without knowing how and where the inputs are worked upon.

6.2.4 WHITE-BOX TESTING

White-box testing is the detailed investigation of internal logic and structure of the code. White-box testing is also called **glass testing** or **open-box testing**. In order to perform **white-box** testing on an application, a tester needs to know the internal workings of the code. The tester needs to have a look inside the source code and find out which unit/chunk of the code is behaving inappropriately.

6.2.5 GREY-BOX TESTING

Grey-box testing is a technique to test the application with having a limited knowledge of the internal workings of an application. In software testing, the phrase the more you know, the better carries a lot of weight while testing an application.

6.2.6 FUNCTIONAL TESTING

This is a type of black-box testing that is based on the specifications of the software that is to be tested. The application is tested by providing input and then the results are examined that need to conform to the functionality it was intended for. Functional testing of a software is conducted on a complete, integrated system to evaluate the system's compliance with its specified requirements.

6.2.7 UNIT TESTING

This type of testing is performed by developers before the setup is handed over to the testing team to formally execute the test cases. Unit testing is performed by the respective developers on the individual units of source code assigned areas. The developers use test data that is different from the test data of the quality assurance team. The goal of unit testing is to isolate each part of the program and show that individual parts are correct in terms of requirements and functionality.

6.2.8 <u>INTEGRATION TESTING</u>

Integration testing is defined as the testing of combined parts of an application to determine if they function correctly. Integration testing can be done in two ways: Bottom-up integration testing and Top-down integration testing.

6.2.9 <u>SYSTEM TESTING</u>

System testing tests the system as a whole. Once all the components are integrated, the application as a whole is tested rigorously to see that it meets the specified Quality Standards. This type of testing is performed by a specialized testing team.

6.3.0 <u>REGRESSION TESTING</u>

Whenever a change in a software application is made, it is quite possible that other areas within the application have been affected by this change. Regression testing is performed to verify that a fixed bug hasn't resulted in another functionality or business rule violation. The intent of regression testing is to ensure that a change, such as a bug fix should not result in another fault being uncovered in the application.

6.3.1 <u>ACCEPTANCE TESTING</u>

This is arguably the most important type of testing, as it is conducted by the Quality Assurance Team who will gauge whether the application meets the intended specifications and satisfies the client's requirement. The QA team will have a set of pre-written scenarios and test cases that will be used to test the application. By performing acceptance tests on an application, the testing team will deduce how the application will perform in production. There are also legal and contractual requirements for acceptance of the system.

6.3.2 <u>ALPHA TESTING</u>

This test is the first stage of testing and will be performed amongst the teams (developer and QA teams). Unit testing, integration testing and system testing when combined together is known as alpha testing. During this phase, the following aspects will be tested in the application:

- Spelling Mistakes
- Broken Links
- Cloudy Directions

- The Application will be tested on machines with the lowest specification to test loading times and any latency problems.

- **BETA TESTING**

This test is performed after alpha testing has been successfully performed. In beta testing, a sample of the intended audience tests the application. Beta testing is also known as **pre-release testing**. Beta test versions of software are ideally distributed to a wide audience on the Web, partly to give the program a "real-world" test and partly to provide a preview of the next release. In this phase, the audience will be testing the following:

- Users will install, run the application and send their feedback to the project team

- Typographical errors, confusing application flow, and even crashes.

- Getting the feedback, the project team can fix the problems before releasing the software to the actual users.

- **NON-FUNCTIONAL TESTING**

This section is based upon testing an application from its non-functional attributes. Non-functional testing involves testing a software from the requirements which are nonfunctional in nature but important such as performance, security, user interface, etc. Some of the important and commonly used non-functional

- Load Testing

- Performance Testing

- Stress Testing

7.CONCLUSION

The project “Train signal” clearly demonstrates the usage of OpenGL library. Train signal simulation has been shown clearly and train crossing has been demonstrated using OpenGL.
Finally we conclude that this program clearly illustrate the concepts of openGL and has been completed successfully and is ready to be demonstrated.

8. BIBILIOGRAPHY

WE HAVE OBTAINED INFORMATION FROM MANY RESOURCES TO DESIGN AND
IMPLEMENT OUR PROJECT SUCCESSIVELY. WE HAVE ACQUIRED MOST OF THE KNOWLEDGE FROM RELATED WEBSITES. THE FOLLOWING ARE SOME OF THE
RESOURCES.

REFERENCE BOOKS:

INTERACTIVE COMPUTER GRAPHICS A TOP-DOWN APPROACH

-By Edward Angel.

➢ COMPUTER GRAPHICS,PRINCIPLES &amp; PRACTICES

- Foley van dam
- Feiner hughes

Web Sites:

http://jerome.jouvie.free.fr/OpenGl/Lessons/Lesson3.php

http://google.com

http://opengl.org