VISVESVARAYA TECHNOLOGICAL UNIVERSITY

"Jnana Sangama", Belgaum 590018, Karnataka

`Department of Computer Science & Engineering

A Computer Graphics Mini Project on

“TRAIN ARRIVAL AND DEPARTURE”

In partial fulfilment of COMPUTER GRAPHICS Laboratory**(18CSL67)**

In Computer Science and Engineering for the Academic Year **2021-22**

SUBMITTED BY

D SUPRITH (1SK10CS020)

MITHUN KUMAR GT (1SK19CS023)

UNDER THE GUIDANCE OF

Mrs. Yamuna Raju, BE., MTECH,
Assistant Professor, Govt. SKSJT Institute
Bangalore-560001

GOVT. S.K.S.J TECHNOLOGICAL INSTITUTE

(Affiliated to Visvesvaraya Technological University, Belgaum)

KR circle, Bangalore – 560001

DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING

CERTIFICATE:

This is to certify that **D SUPRITH (1SK19CS010)** and **MITHUN KUMAR G T (1SK19CS023)** of **SIXTH Semester** have successfully completedthe mini project in **“TRAIN ARRIVAL AND DEPARTURE”** in **COMPUTER GRAPHICS LABORATORY (18CSL67)** as prescribed by the **VISVESVARAYA TECHNOLOGICAL UNIVERSITY** for the academic year **2021-2022.**

Signature of the guide	**Signature of the HOD**
YAMUNA R	**Dr . PRADEEP KUMAR**
Department of CSE	**Assoc, Professor & HOD of CSE**

Name of The Examiner	Signature With Date
1.________________	1.________________
2.________________	2.________________

ACKNOWLEDGEMENT

A unique opportunity like this come very rarely. It is indeed a pleasure for me to have worked on this project. The satisfaction that accompanies the successful completion of this project is incomplete without the mention of the people whose guidance and support are made it possible for me to complete this project.

I express my sincere gratitude to **Dr. K.G. CHANDRASHEKAR**, the principal Govt. SKSJTI for his valuable guidance, suggestion and consistent encouragement during the course of my mini project work and timely assistance for completion of the project.

I thank to entire faculty of the Computer Science and Engineering Department, especially **Dr. PRADEEP KUMAR**, Head of the Department CSE, who has given me confidence to believe in myself and complete the project.

I'm grateful to **Mrs. YAMUNA RAJU**, my internal guide, Computer Science and Engineering Department, for his support and encouragement throughout the process.

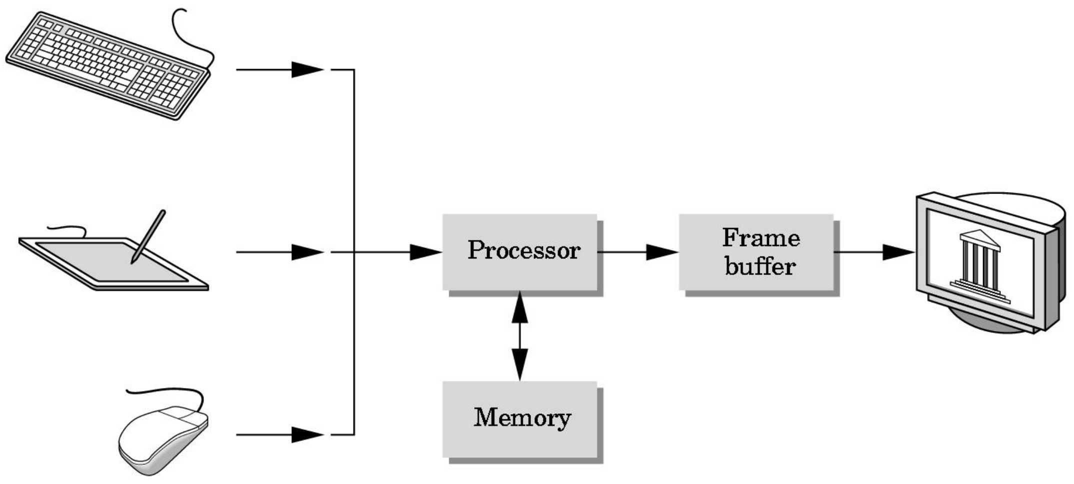
Last but not least I would also like to thank my parents and friends for their moral support.

1.INTRODUCTION

1.1 Computer Graphics

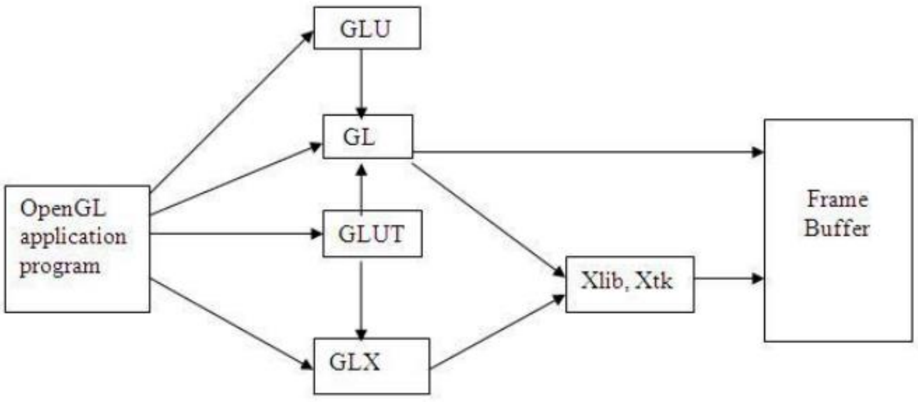
- Graphics provides one of the most natural means of communicating with a computer, since our highly developed 2D Or 3D pattern-recognition abilities allow us to perceive and process pictorial data rapidly.
- Computers have become a powerful medium for the rapid and economical production of pictures.
- There is virtually no area in which graphical displays cannot be used to some advantage.
- Graphics provide a so natural means of communicating with the computer that they have become widespread.
- Interactive graphics is the most important means of producing pictures since the invention of photography and television .
- We can make pictures of not only the real world objects but also of abstract objects such as mathematical surfaces on 4D and of data that have no inherent geometry.
- A computer graphics system is a computer system with all the components of the general purpose computer system. There are five major elements in system: input devices, processor, memory, frame buffer, output devices.

- A computer graphics system is a computer system with all the components of the general purpose computer system. There are five major elements in system: input devices, processor, memory, frame buffer, output devices.



1.2 <u>OpenGL Technology</u>

OpenGL is the premier environment for developing portable, interactive 2D and 3D graphics applications. Since its introduction in 1992, OpenGL has become the industry's most widely used and supported 2D and 3D graphics application programming interface (API), bringing thousands of applications to a wide variety of computer platforms. OpenGL fosters innovation and speeds application development by incorporating a broad set of rendering, texture mapping, special effects, and other powerful visualization functions. Developers can leverage the power of OpenGL across all popular desktop and workstation platforms, ensuring wide application deployment. OpenGL Available Everywhere: Supported on all UNIX® workstations, and shipped standard with every Windows 95/98/2000/NT and MacOS PC, no other graphics API operates on a wider range of hardware platforms and software environments. OpenGL runs on every major operating system including Mac OS, OS/2, UNIX, Windows 95/98, Windows 2000, Windows NT, Linux, OPEN Step, and BeOS; it also works with every major windowing system, including Win32, MacOS, Presentation Manager, and X-Window System. OpenGL is callable from Ada, C, C++, Fortran, Python, Perl and Java and offers complete independence from network protocols and topologies. The OpenGL interface Our application will be designed to access OpenGL directly through functions in three libraries namely: gl,glu,glut.



2. REQUIREMENTS AND SPECIFICATIONS

2.1 Hardware Requirements

The standard output device is 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 i.e. used to give input in the game. A keyboard is used for controlling and inputting data in the form of characters, numbers i.e. to change the user views . Apart from these hardware requirements there should be sufficient hard disk space and primary memory available for proper working of the package to execute the program. Pentium III or higher processor, 16MB or more RAM. A functional display card.

Minimum Requirements expected are cursor movement, creating objects like lines, squares, rectangles, polygons, etc. Transformations on objects/selected area should be possible. Filling of area with the specified color should be possible.

2.2 Software Requirements

The editor has been implemented on the OpenGL platform and mainly requires an appropriate version of eclipse compiler to be installed and functional in ubuntu. Though it is implemented in OpenGL, it is very much performed and independent with the restriction, that there is support for the execution of C and C++ files. Text Modes is recommended.

Developed Platform

Windows 11

Language Used In Coding

C-language

Tool Used In Coding

Eclipse

3. SOFTWARE DESIGN

3.1 SYSTEM DESIGN

Existing System:

Existing system for a graphics is the TC++ . This system will support only the 2D graphics. 2D graphics package being designed should be easy to use and understand. It should provide various option such as free hand drawing , line drawing , polygon drawing , filled polygons, flood fill, translation , rotation , scaling , clipping etc. Even though these properties were supported, it was difficult to render 2D graphics cannot be . Very difficult to get a 3Dimensional object. Even the effects like lighting , shading cannot be provided. So we go for Eclipse software.

PROPOSED SYSTEM :

To achieve three dimensional effects, OpenGL software is proposed . It is software which provides a graphical interface. It is a interface between application program and graphics hardware. the advantages are:

1. OpenGL is designed as a streamlined.

2. It is a hardware independent interface i.e. it can be implemented on many different hardware platforms.

3. With OpenGL, we can draw a small set of geometric primitives such as points, lines and polygons etc.

4. Its provides double buffering which is vital in providing transformations.

5. It is event driven software.

6. It provides call back function.

3.2 DETAILED DESIGN

TRANSFORMATION FUNCTIONS

Matrices allow arbitrary linear transformations to be represented in a consistent format, suitable for computation. This also allows transformations to be concatenated easily (by multiplying their matrices).

Linear transformations are not the only ones that can be represented by matrices. Using homogenous coordinates, both affine transformation and perspective projection on R n can be represented as linear transformations on RPn+1 (that is, n+1- dimensional real projective space). For this reason, 4x4 transformation matrices are widely used in 3D computer graphics.

3-by-3 or 4-by-4 transformation matrices containing homogeneous coordinates are often called, somewhat improperly, "homogeneous transformation matrices". However, the transformations they represent are, in most cases, definitely non-homogeneous and nonlinear (like translation, roto-translation or perspective projection). And even the matrices themselves look rather heterogeneous, i.e. composed of different kinds of elements (see below). Because they are multi-purpose transformation matrices, capable of representing both affine and projective transformations, they might be called "general transformation matrices", or, depending on the application, "affine transformation" or "perspective projection" matrices. Moreover, since the homogeneous coordinates describe a projective vector space, they can also be called "projective space transformation matrices".

Finding the matrix of a transformation

If one has a linear transformation T(x) in functional form, it is easy to determine the transformation matrix A by simply transforming each of the vectors of the standard basis by T and then inserting the results into the columns of a matrix. In other words,

$$\mathbf{A} = \begin{bmatrix} T(\vec{e}_1) & T(\vec{e}_2) & \cdots & T(\vec{e}_n) \end{bmatrix}$$

For example, the function T(x) = 5x is a linear transformation. Applying the above process (suppose that n = 2 in this case) reveals that

$$T(\vec{x}) = 5\vec{x} = \begin{bmatrix} 5 & 0 \\ 0 & 5 \end{bmatrix} \vec{x}$$

Examples in 2D graphics

Most common geometric transformations that keep the origin fixed are linear, including rotation, scaling, shearing, reflection, and orthogonal projection; if an affine transformation is not a pure translation it keeps some point fixed, and that point can be chosen as origin to make the transformation linear. In two dimensions, linear transformations can be represented using a 2×2 transformation matrix.

Rotation

For rotation by an angle θ anticlockwise about the origin, the functional form is x' = xcosθ − ysinθ and y' = xsinθ + ycosθ. Written in matrix form, this becomes:

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix}$$

Similarly, for a rotation clockwise about the origin, the functional form is x' = xcosθ + ysinθ and y' = − xsinθ + ycosθ and the matrix form is:

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} \cos\theta & \sin\theta \\ -\sin\theta & \cos\theta \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix}$$

Scaling: For scaling (that is, enlarging or shrinking), we have $x' = s_x \cdot x$ and $y' = s_y \cdot y$. The matrix form is: $\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} s_x & 0 \\ 0 & s_y \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix}$ When $s_x s_y = 1$, then the matrix is a squeeze mapping and preserves areas in the plane.

4. IMPLEMENTATION

4.1 FUNCTIONS USED

Void glColor3f(float red, float green, float blue);

This function is used to mention the color in which the pixel should appear. The number 3 specifies the number of arguments that the function would take. ‘f ’ gives the type that is float. The arguments are in the order RGB(Red, Green, Blue). The color of the pixel can be specified as the combination of these 3 primary colors.

Void glClearColor(int red, int green, int blue, int alpha);

This function is used to clear the color of the screen. The 4 values that are passed as arguments for this function are (RED, GREEN, BLUE, ALPHA) where the red green and blue components are taken to set the background color and alpha is a value that specifies depth of the window. It is used for 3D images.

Void glutKeyboardFunc();

void glutKeyboardFunc(void (*func)(unsigned char key, int x, int y));

where func is the new keyboard callback function. glutKeyboardFunc sets the keyboard callback for the current window. When a user types into the window, each key press generating an ASCII character will generate a keyboard callback. The key callback parameter is the generated ASCII character. The x and y callback parameters indicate the mouse location in window relative coordinates when the key was pressed. When a new window is created, no keyboard callback is initially registered, and ASCII key strokes in the window are ignored. Passing NULL to glutKeyboardFunc disables the generation of keyboard callbacks.

Void glFlush();

Different GL implementations buffer commands in several different locations, including network buffers and the graphics accelerator itself. glFlush empties all of these buffers, causing all issued commands to be executed as quickly as they are accepted by the actual rendering engine. Though this execution may not be completed in any particular time period, it does complete in finite time.

Void glMatrixMode(GLenum mode);

where mode Specifies which matrix stack is the target for subsequent matrix operations. Three values are accepted: GL_MODELVIEW, GL_PROJECTION, and GL_TEXTURE. The initial value is GL_MODELVIEW.

glMatrixMode sets the current matrix mode. mode can assume one of three values:

GL_MODELVIEW: Applies subsequent matrix operations to the modelview matrix stack.

GL_PROJECTION: Applies subsequent matrix operations to the projection matrix stack.

void glViewport(GLint x, GLint y, GLsizei width, GLsizei height)

where x, y Specify the lower left corner of the viewport rectangle, in pixels. The initial value is (0, 0).

width, height Specify the width and height of the viewport. When a GL context is first attached to a surface (e.g. window), width and height are set to the dimensions of that surface. glViewport specifies the affine transformation of x and y from normalized device coordinates to window coordinates. Let (xnd, ynd) be normalized device coordinates. Then the window coordinates (xw, yw) are computed as follows:

$$x_w = (x_{nd} + 1)\, {}^{width}/_2 + x$$

$$y_w = (y_{nd} + 1)\, {}^{height}/_2 + y$$

Viewport width and height are silently clamped to a range that depends on the implementation. To query this range, call **glGetInteger** with argument GL_MAX_VIEWPORT_DIMS.

void glutInit(int *argcp, char **argv);

glutInit will initialize the GLUT library and negotiate a session with the window system. During this process, glutInit may cause the termination of the GLUT program with an error message to the user if GLUT cannot be properly initialized. Examples of this situation include the failure to connect to the window system, the lack of window system support for OpenGL, and invalid command line options.glutInit also processes command line options, but the specific options parse are window system dependent.

void glutReshapeFunc(void (*func)(int width, int height));

glutReshapeFunc sets the reshape callback for the current window. The reshape callback is triggered when a window is reshaped. A reshape callback is also triggered immediately before a window's first display callback after a window is created or whenever an overlay for the window is established. The width andheight parameters of the callback specify the new window size in pixels. Before the callback, the current window is set to the window that has been reshaped. If a reshape callback is not registered for a window or NULL is passed to glutReshapeFunc (to deregister a previously registered callback), the default reshape callback is used. This default callback will simply

void glutMainLoop(void);

glutMainLoop enters the GLUT event processing loop. This routine should be called at most once in a GLUT program. Once called, this routine will never

glutPostRedisplay():

glutPostRedisplay, glutPostWindowRedisplay — marks the current or specified window as needing to be redisplayed.

5.APPENDIX(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();

    //train 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();

//train 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)
```

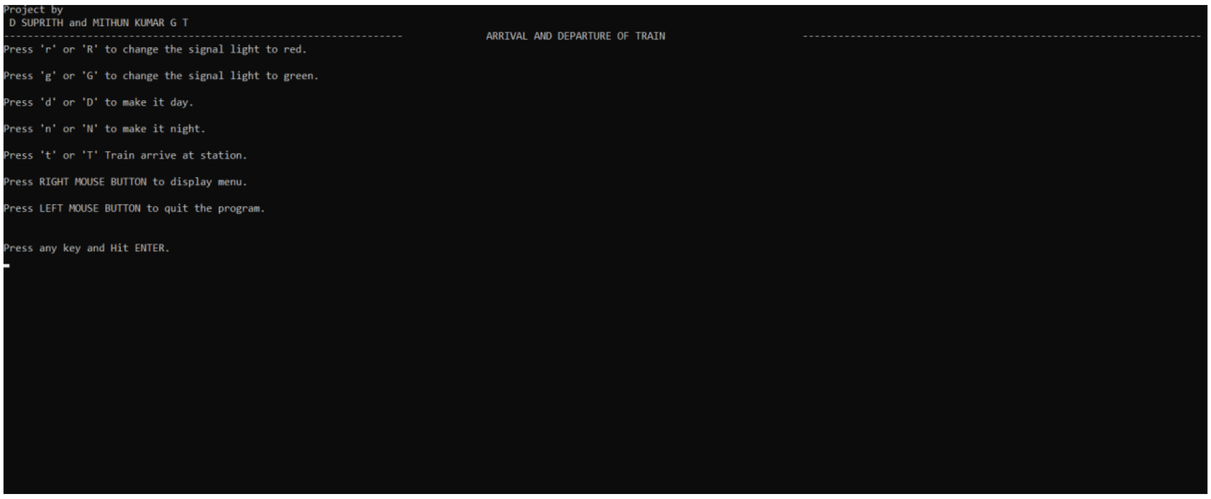
```
        m=0.0;
    if( o>75)
     {
     plane=0;
     }
    if(c>500)
     {
        comet=0;
     }
    glutPostRedisplay();
}
void mouse(int btn,int state,int x,int y)
    {
        if(btn==GLUT_LEFT_BUTTON && state==GLUT_UP)
    exit(0);
    }
void keyboardFunc( unsigned char key, int x, int 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 \n D SUPRITH and MITHUN KUMAR G T \n");
     printf("----------------------------------------------------------------");
     printf("            ARRIVAL AND DEPARTURE OF TRAIN                    ");
     printf("----------------------------------------------------------------\n");
     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;
}
```

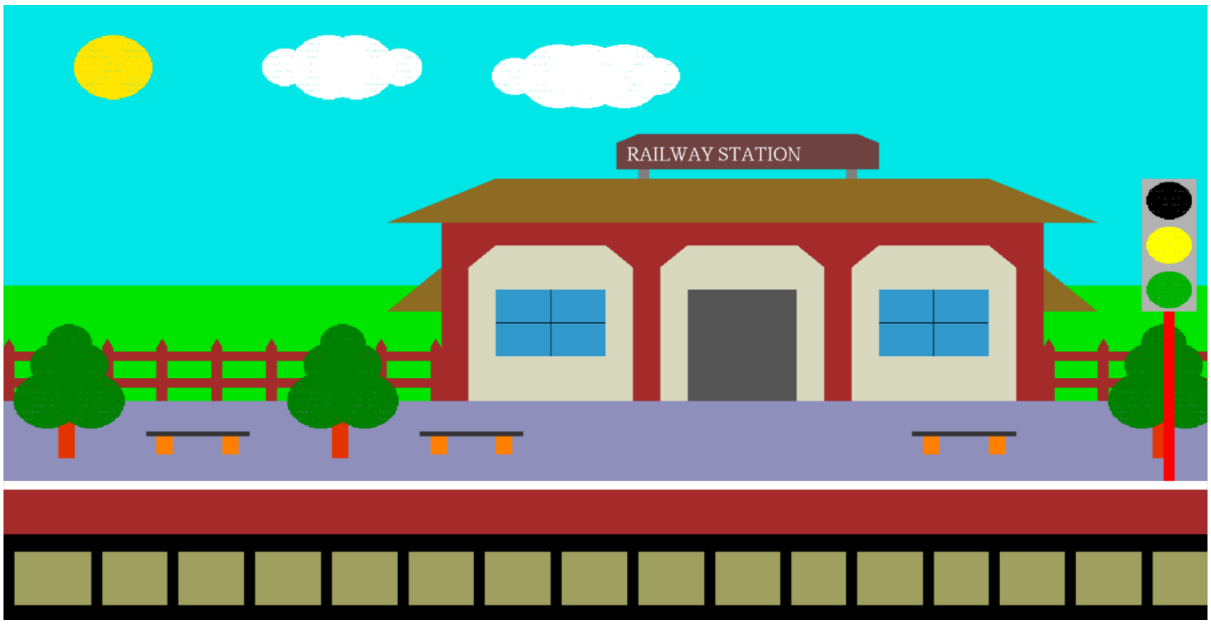
6. Snapshots

1. Snap Shot Showing Instructions Before Starting:

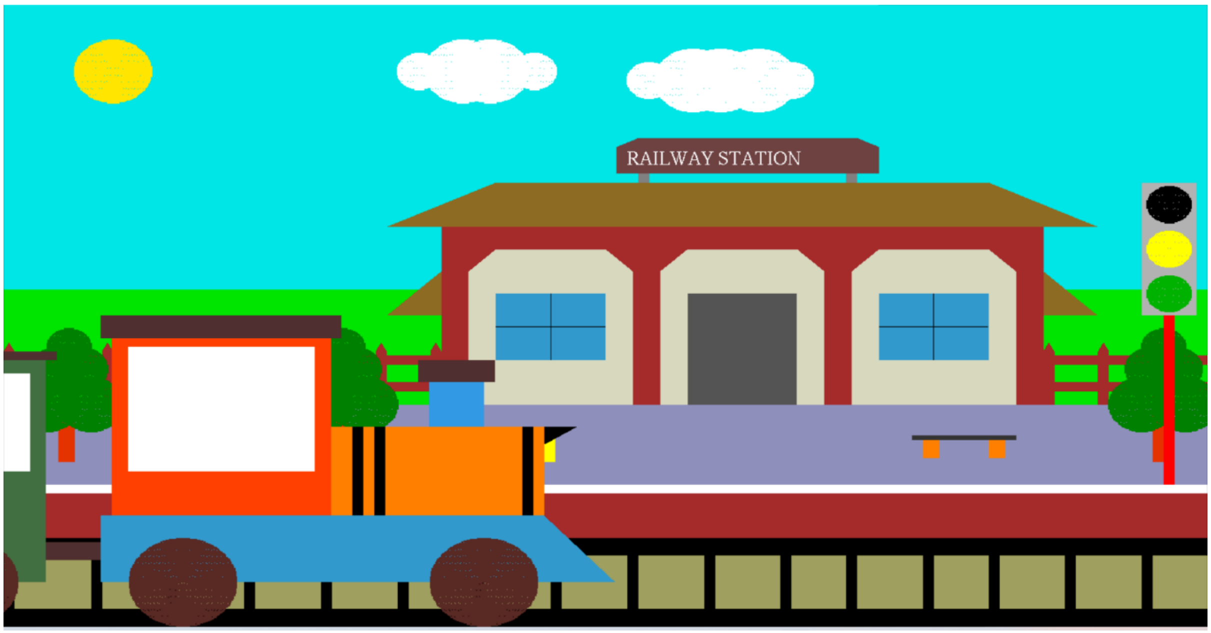


```
Project by
 D SUPRITH and MITHUN KUMAR G T
-------------------------------------------------------------------          ARRIVAL AND DEPARTURE OF TRAIN          -------------------------------------------------------------------
Press 'r' or 'R' to change the signal light to red.

Press 'g' or 'G' to change the signal light to green.

Press 'd' or 'D' to make it day.

Press 'n' or 'N' to make it night.

Press 't' or 'T' Train arrive at station.

Press RIGHT MOUSE BUTTON to display menu.

Press LEFT MOUSE BUTTON to quit the program.


Press any key and Hit ENTER.
_
```

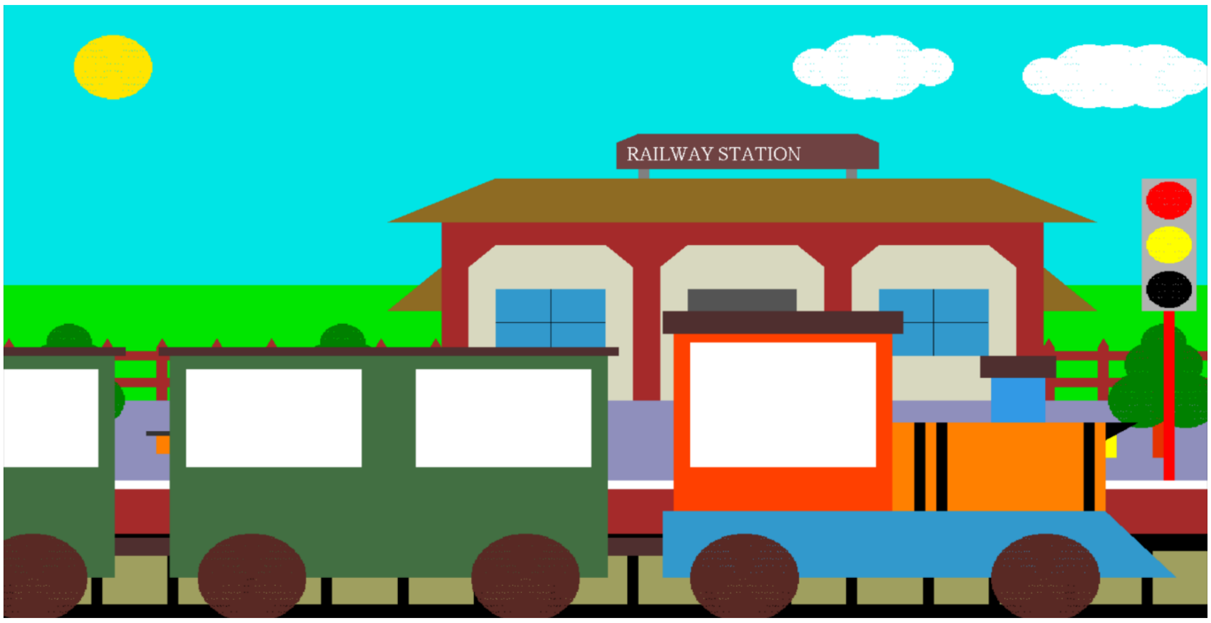
2. Snap Shot Showing Beginning of the project Day Time :



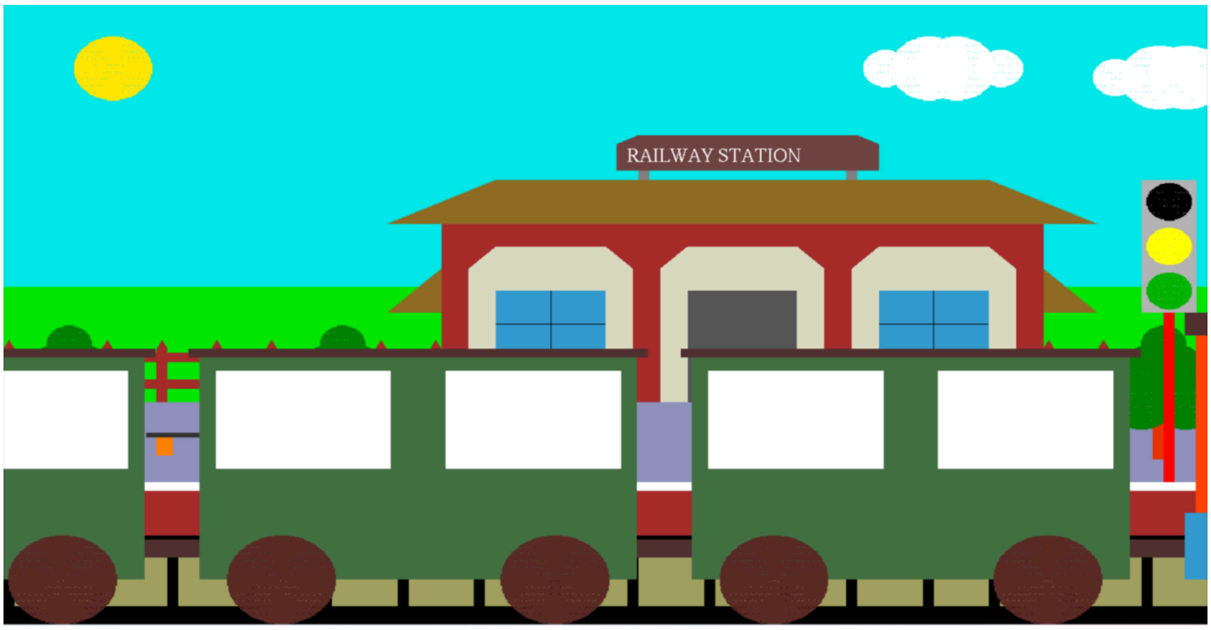
3. Snap Shot Showing Arrival of the Train :



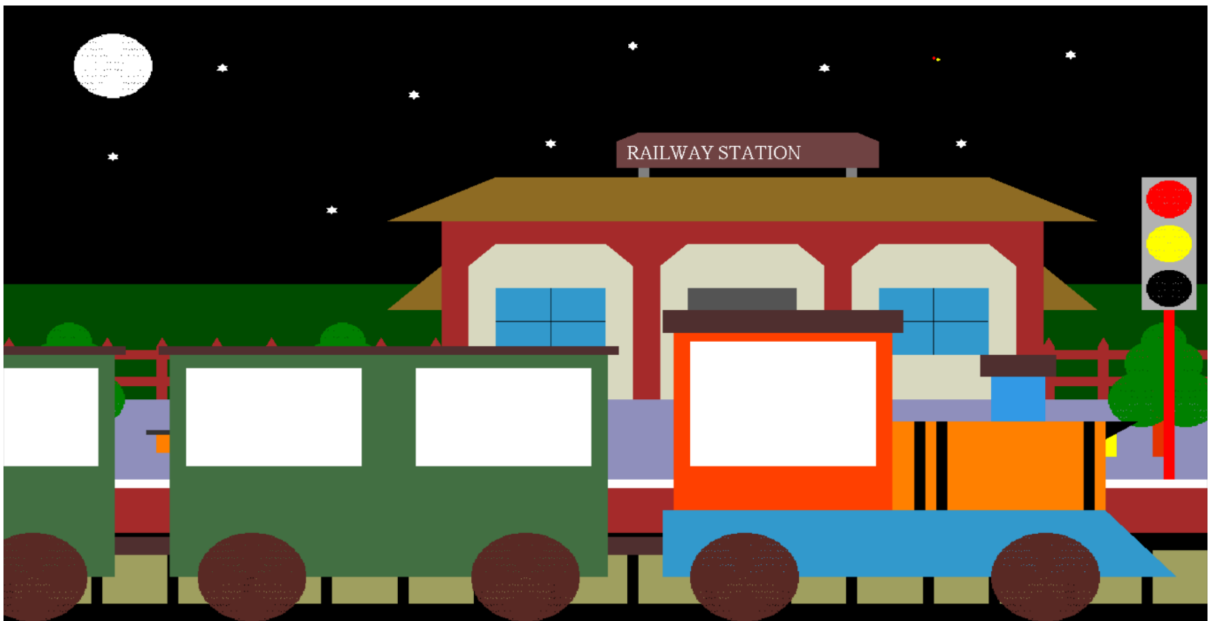
4. Snap Shot Showing Train stopped at station :



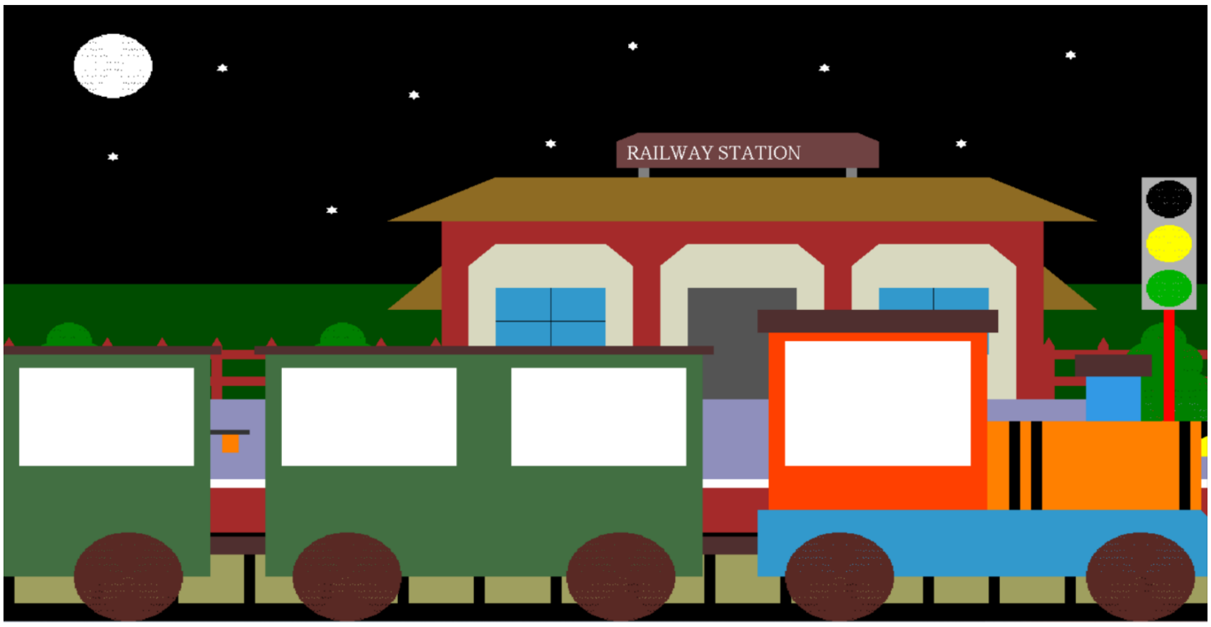
5. Snap Shot Showing Departure of Train from the station:



6. Snap Shot Showing Night Time view:



6. Snap Shot Showing Night Time train Departure :



7. CONCLUSION :

Arrival and departure of the train and motion of some objects were included in this project. Some other options are also provided. You can implement some other objects by keeping the reference of this according to your requirements to enhance the project.

These operations will be done by using some keys in the keyboard

1. Press 'r' or 'R' to change the signal light to red.

2. Press 'g' or 'G' to change the signal light to green.

3. Press 'd' or 'D' to make it day.

4. Press 'n' or 'N' to make it night.

5. Press 't' or 'T' Train arrive at station.

6. Press RIGHT MOUSE BUTTON to display menu

7. Press LEFT MOUSE BUTTON to quit the program.

8. BIBLIOGRAPHY

1. Reference from Interactive Computer Graphics by EDWARD ANGEL

2. Internet source

- www.angelfire.com
- Wikipedia.org.
- Google.
- Opengl.org.