VISVESVARAYA TECHNOLOGICAL UNIVERSITY,

BELAGAVI - 590018

Mini Project Report

On

"TRAFFIC SIGNAL"

A report submitted in partial fulfilment of the requirements for

COMPUTER GRAPHICS AND VISUALIZATION LABORATORY (17CSL68)

in

COMPUTER SCIENCE AND ENGINEERING

Submitted by

SHAH DHRUVIL AMIT	**4AL17CS083**
SHETTY SATHVIK RAVINDRA	**4AL17CS089**

DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING

ALVA'S INSTITUTE OF ENGINEERING AND TECHNOLOGY

MOODBIDRI-574225, KARNATAKA

2019 – 2020

ALVA'S INSTITUTE OF ENGINEERING AND TECHNOLOGY

MIJAR, MOODBIDRI D.K. -574225

KARNATAKA

DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING

CERTIFICATE

This is to certify that the Mini Project entitled **"TRAFFIC SIGNAL"** has been successfully completed by

SHAH DHRUVIL AMIT	**4AL17CS083**
SHETTY SATHVIK RAVINDRA	**4AL17CS089**

in the partial fulfillment for the award of Degree of Bachelor of Engineering in Computer and Engineering of the Visvesvaraya Technological University, Belagavi during the year 2019-2020. It is certified that all corrections/suggestions indicated have been incorporated in the report. The Mini project report has been approved dissatisfies the academic requirements in respect of Mini Project work prescribed for the award of Bachelor of Engineering Degree.

Mr. Sayeesh
Mini Project Guide

Dr. Manjunath Kotari,
HOD CSE

External Viva

Name of the Examiners	Signature with Date
1.	
2.	

ACKNOWLEDGEMENT

The satisfaction and euphoria that accompany a successful completion of any task would be incomplete without the mention of people who made it possible, success is the epitome of hard work and perseverance, but steadfast of all is encouraging guidance.

So, with gratitude we acknowledge all those whose guidance and encouragement served as beacon of light and crowned the effort with success.

The selection of this mini project work as well as the timely completion is mainly due to the interest and persuasion of our mini project coordinator **Mr.Sayeesh,** Assistant Professor, Department of Computer Science & Engineering. We will remember his contribution for ever.

We sincerely thank, **Dr.Manjunath Kotari**, Professor and Head, Department of Computer Science & Engineering who has been the constant driving force behind the completion of the project.

We thank our beloved Principal **Dr. Peter Fernandes,** for his constant help and support throughout.

We are indebted to **Management of Alva's Institute of Engineering and Technology, Mijar, Moodbidri** for providing an environment which helped us in completing our mini project.

Also, we thank all the teaching and non-teaching staff of Department of Computer Science & Engineering for the help rendered.

SHAH DHRUVIL AMIT **4AL17CS083**

SHETTY SATHVIK RAVINDRA **4AL17CS089**

ABSTRACT

The aim of the Traffic Signal project is to control the flow of the traffic as well as monitor them through the junctions across roads. They provide smooth motion of the cars in the transportation routes. This system decreases the accident rate. The passage of pedestrian and the passage of emergency vehicles are not implemented here.

The traffic signal project provides a simulation of traffic signal being done using computer graphics. The car in this project are built using cubes and can be moved using arrow keys. Based on the traffic signal light the user can obey the traffic rules. We have used input devices like mouse and keyboard to navigate and interact with the program. Here in this project we illustrate the concept and usage of pre- built functions in OpenGL.

TABLE OF CONTENTS

LIST OF FIGURES

CHAPTER 1

INTRODUCTION

Computer graphics is the creation and manipulation of the picture with the aid of computers. It is divided into two broad classes. It is also called passive graphics. Here the user has no control over the pictures produced on the screen. Interactive graphics provide extensive user-computer interaction. It provides a two-way communication between computer and user. It provides a tool called “motion dynamics” using which the user can move objects. It is also able to produce audio feedback to make the stimulated environment more realistic. C language helps to implement different graphics objects which are interactive and non-interactive.

1.1 Computer Graphics

Computer graphics is a sub-field of computer science and is concerned with digitally synthesizing and manipulating visual content. Although the term refers to three-dimensional computer graphics, it also encompasses two-dimensional graphics and image processing. Computer graphics is often differentiated from the field of visualization, although the two have same similarities. Graphics are visual presentation on some surface like wall, canvas, computer screen. Graphics often combine text, illustration and color.

Computer graphics started with the display of data on hardcopy plotters and cathode ray tube (CRT) screens soon after the introduction of computers. It has grown to include the creation, storage and manipulation of models and images of objects. These models come from a diverse and expanding set of fields, and include physical, mathematical, engineering, architectural, and even conceptual structures. Computer graphics today is largely interactive. The user controls the contents, structure and appearance of objects and of their displayed images by using input devices, such as keyboard, mouse, or touch- sensitive panel on the screen.

Graphical interfaces have replaced textual interfaces as the standard means for user-computer interaction. Graphics has also become a key technology for communicating ideas, data and trends in most areas of commerce, science, engineering and education. Much of the task of creating effective graphic communication lies in modeling the objects whose images we want to produce.

Computer graphics is no more a rarity. Even people who do not use computers in their daily work encounter computer graphics in television commercials and as cinematic special

effects. Computer graphics is an integral part of all user interfaces and is indispensable for visualizing objects.

1.2 OpenGL

OpenGL is a low level graphics library specification. It makes available to the programmer a small set of geometric primitives - points, lines, polygons, images, and bitmaps. OpenGL provides a set of commands that allow the specification of geometric objects in two or three dimensions, using the provided primitives, together with commands that control how these objects are rendered (drawn). Since OpenGL drawing commands are limited to those that generate simple geometric primitives (points, lines, and polygons), the OpenGL Utility Toolkit (GLUT) has been created to aid in the development of more complicated three-dimensional objects such as a sphere, a torus and even a teapot.

GLUT is designed to fill the need for a window system independent programming interface for OpenGL programs. The interface is designed to be simple yet still meet the needs of useful OpenGL programs. Removing window system operations from OpenGL is a sound decision because it allows the OpenGL graphics system to be retargeted to various systems including powerful but expensive graphics workstations as well as mass-production graphics systems like video games, set-top boxes for interactive television, and PCs. GLUT simplifies the implementation of programs using OpenGL rendering. The GLUT application programming interface (API) requires very few routines to display a graphics scene rendered using OpenGL. The GLUT routines also take relatively few parameters.

CHAPTER 2

REQUIREMENT SPECIFICATION

2.1 Functional Requirements

Here, we have made an attempt to design a 3D gear using OpenGL. The software has been written using C language and data structure like simple structure type are used.

The project requires the access of OpenGL utility toolkit through the use of the header file “gl/glut.h”. This header file, in addition to the usual header files is needed for the working of the project. For running the program, any basic PC running compatible version of Microsoft Visual Studio is sufficient.

2.2 Non-Functional Requirements

The software should produce the information error messages, if any errors are found in the input program. It should use memory as less as possible, dynamic memory allocation is preferabble to accomplish this task.

2.3 Hardware Requirements

The minimum/recommended hardware configuration required for developing the proposed software is given below:

- PENTIUM-2 and above compatible systems.
- 512 MB RAM
- Approximately 170MB free space in the hard disk.
- Hard disk access time must be less than 19 millisecond.

2.4 Software Requirements

API - OpenGL

Platform -Windows

Language Tool -C++

Complier -Microsoft Visual Studio 2010

Documentation Tool -MS-Word

2.4.1 Why C Language for this project?

C is a minimalist programming language. Among its design goals were that it could be complied in a straight forward manner using a relatively simple complier, provided low-level access to memory, generated only a few machine language instructions for each of its core language elements and did not require extensive run-time support. As a result, Ccode is suitable for many system programming applications that had traditionally been implemented in assembly language.

Despite its low-level capabilities, the language was designed to encourage machine independent programming. A standard compliant and portably written C program can be compiled for a very wide variety of computer platforms and operating systems with minimal changes to its source code.

2.4.2 Graphics in C

C itself doesn't support any graphics library. In order for C to be completely portable from platform to platform it has focused on providing platform independent functions, such as file access, texting string manipulation, mathematical functions etc. So in order to get graphics in C, one needs to do one of four things:

- Write a 16 bit DOS program and use assembly language for graphics.
- Make calls to the Windows API
- Make calls to the OpenGL subsystem
- Use a graphics library

2.4.3 OpenGL subsystem

OpenGL provides a powerful but primitive set of rendering commands, and all higher- level drawing must be done in terms of these commands. Also, OpenGL programs have to use the underlying mechanisms of windowing system. A number of libraries exist to simplify the programming tasks, including the following:

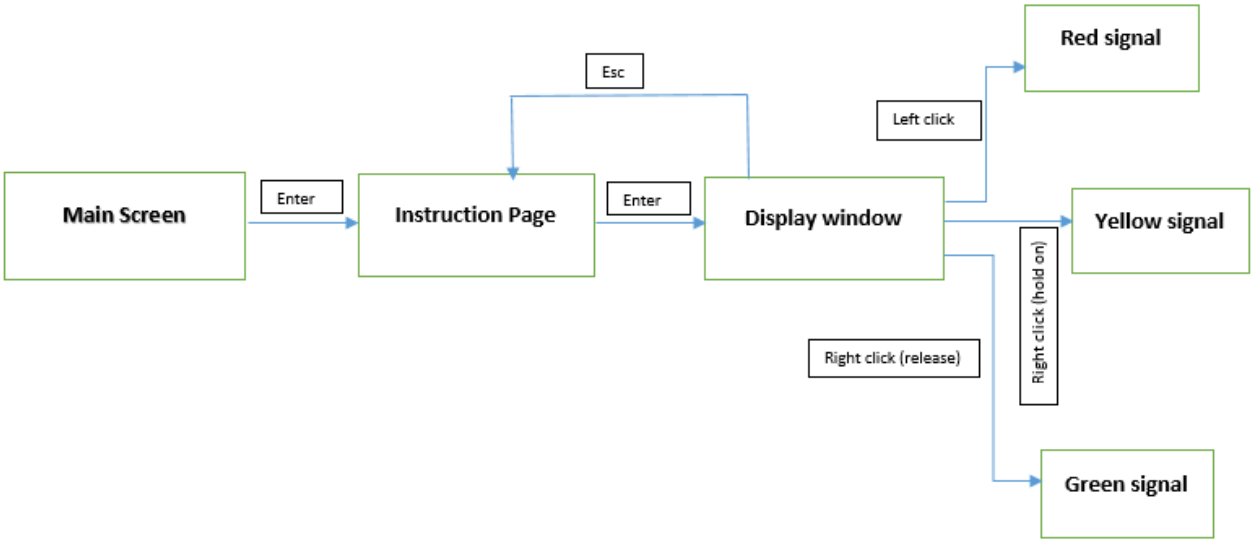
- The OpenGL Utility Library (GLU) contains several routines that use the lower-level OpenGL commands to perform such tasks as setting up matrices for specific viewing orientations and projections, performing polygon tessellation, and

rendering surfaces. The library is provided as part of every OpenGL implementation.

- The OpenGL Utility Toolkit (GLUT) is a window system-independent toolkit. GLUT routines use prefix glut.

CHAPTER 3

SYSTEM DESIGN



This project has been created using the OpenGL interface along with the GLUT (OpenGL Library Toolkit), using the platform Visual Studio as the compiling tool. This visualizer has been designed in a simple and lucid manner so that further developments can be made, and run on many platforms with a few changes in the code.

Initially the "Main Menu" screen is displayed where we have the details about the Project name and the users. Press 'enter' to move to the "Tips and Tricks Menu" where the specific key options for further progress in the execution is displayed. Press 'enter' to move to the display window and press 'esc' to revert back. Use 'left click' to display red signal, 'right click(hold)' to display yellow signal and 'right click(release)' to display green signal. The changes between different traffic signals can be made by repeating the above mentioned operations .

CHAPTER 4

IMPLEMENTATION

We have implemented our program using OpenGL software. OpenGL is a software interface to graphics hardware. This interface consists of about 150 distinct commands that you use to specify the objects and operations needed to produce interactive three-dimensional applications. OpenGL is designed as a streamlined, hardware-independent interface to be implemented on many different hardware platforms.

To achieve these qualities, no commands for performing windowing tasks or obtaining user input are included in OpenGL; instead, you must work through whatever windowing system controls the particular hardware you're using. Similarly, OpenGL doesn't provide high-level commands for describing models of three-dimensional objects. Such commands might allow you to specify relatively complicated shapes such as automobiles, parts of the body, airplanes, or molecules. With OpenGL, you must build up your desired model from a small set of geometric primitives - points, lines, and polygons sophisticated library that provides these features could certainly be built on top of OpenGL.

The OpenGL Utility Library (GLU) provides many of the modeling features, such as quadric surfaces and NURBS curves and surfaces. GLU is a standard part of every OpenGL implementation. Also, there is a higher-level, object-oriented toolkit, Open Inventor, which is built atop OpenGL, and is available separately for many implementations of OpenGL.

The functions implemented in the project are:

1. **void glClearColor(GLclampf r, GLclampf g,GLclampf b,GLclampf a)** sets the present RGBA clear color used when clearing the color buffer.
2. For CMY system a combination from the RGB system is only used to get the desired results.
3. **void glutInit(int argc,char **argv)** initializes GLUT. The arguments from main are passed in can be used by the applications

4. **int glutCreateWindow(char *title)** creates a window on the display. The string title can be used to label the window.
5. **void glutInitDisplayMode(unsigned int mode)** requests a display with the properties in mode. The value of mode is determined by the logical OR.
6. **void glutInitWindowSize(int width, int height)** specifies the initial height and weight of the window in pixels.
7. **void glutInitWindowPosition(int x, int y)** specifies the initial position of the top-left corner of the window in pixels
8. **void glViewport(int x, int y, GLsizei width, GLsizei height)** specifies a widthxheight viewport in pixels whose lower left corner is at (x,y) measured from the origin of the window.
9. **void glutMainLoop()** cause the program to enter an event processing loop.
10. **void glutDisplayFunc(void (*func)(void))** registers the display function func that is executed when the window needs to be redrawn.
11. **void glutPostRedisplay()** requests that the display callback be executed after the current callback returns.
12. **void glutSwapBuffers()** swaps the front and back buffers.
13. **void glutMouseFunc(void *f(int button, int state, int x, int y)** the callback function returns the button(GLUT_LEFT_BUTTON,GLUT_RIGHT_BUTTON, GLUT_MIDDLE_BUTTON), the state of the button after the event (GLUT_UP, GLUT_DOWN) and the position of the mouse relative to the top-left corner of the window.
14. **void glutReshapeFunc(void *f(int width, int height)** the callback function returns the height and width of the new window.
15. **void glutCreateMenu(void (*f)(int value))** returns an identifier for a top-level menu and registers the callback function f that returns an integer value corresponding to the menu entry selected.
16. **void glutAddMenuEntry(char *name, int value)** adds an entry with the string name displayed to the current menu.
17. **void glutAttachMenu(int button)** attaches the current menu to the specified mouse button.
18. **void glutAddSubMenu(char *name, int menu)** adds a submenu entry name to the current menu.

19. **void glutMotionFunc(void (*f)(int x, int y))** the position of the mouse is returned by the callback when the mouse is moved at least one of the mouse buttons is pressed.
20. **void glEnable(GLenum feature)** enables an OpenGL feature. Features that can be enables includes GL_DEPTH_TEST, GL_LIGHTING, GL_LIGHTi, GL_POLYGON_STIPPLE etc…
21. **void glDisable(GLenum feature)** disables an OpenGL feature.
22. **void glMatrixMode(GLenum mode)** specifies which matrix will be affected by subsequent transformations.
23. **void glLoadIdentity()** sets the current transformations matrix to an identity matrix.
24. **void glPushMatrix() and void glPopMatrix()** pushes to and pops from the matrix stack corresponding to the current matrix mode.
25. **void glTranslated [fd] (TYPE x, TYPE y,TYPE z)** alters the current matrix by a displacement of (x,y,z). TYPE is either GLflaot or GLdouble.
26. **void glRotate [fd] (TYPE angle, TYPE dx, TYPE dy, TYPE dz)** alters the current matrix by a rotation of angle degrees about the axis (dx,dy,dz).
27. **void glScale [fd] (TYPE sx, TYPE sy, TYPE sz)** alters the current matrix by a scaling of (sx,sy,sz).
28. **void glOrtho(GLdouble left, GLdouble right, GLdouble top, GLdouble bottom, GLdouble near, GLdouble far)** defines an orthographic viewing volume with all parameters measured from the center of the projection plane.
29. **void glLightfv(GLenum light, GLenum param, TYPE *v)** sets scalar and vector parameter param for light source light.

GLUT Callback functions

1. **Event-driven:** Programs that use windows
   a. Input/output
   b. Wait until an event happens and then execute some pre-defined functions according to the user's input
   c. Events – key press, mouse button press and release, window resize, etc.

1. **Callback function** : Routine to call when an event happens
    1. Window resize or redraw
2. U**Callback function** : Routine to call when an event happens
    a. Window resize or redraw
    b. User input (mouse, keyboard)
    c. Animation (render many frames)

CHAPTER 5

SNAPSHOTS

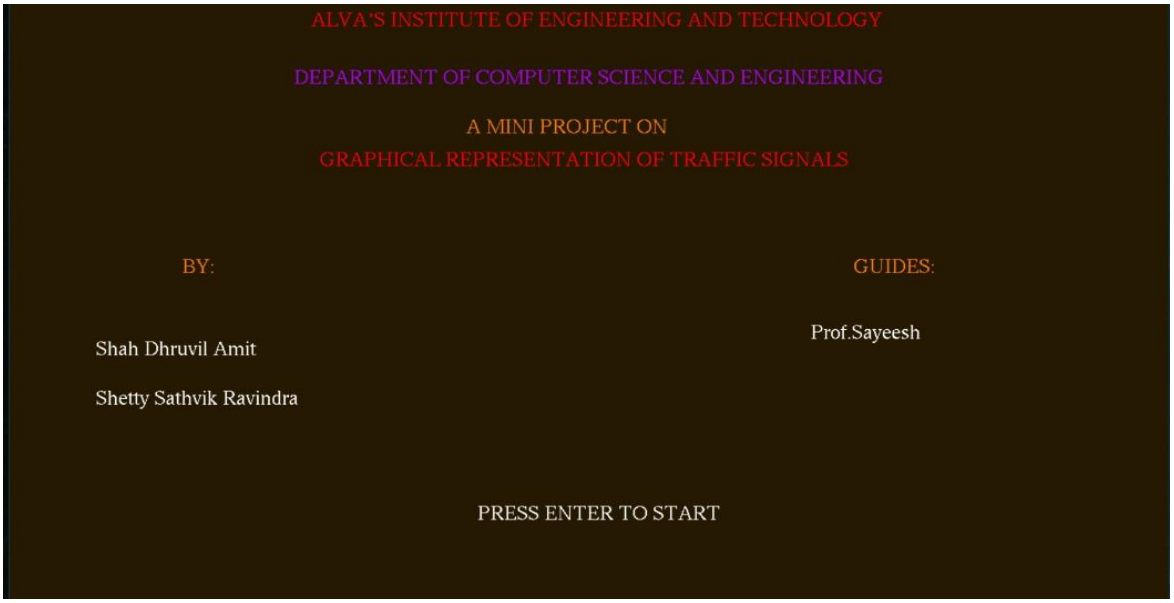


Fig 5.1: Main menu

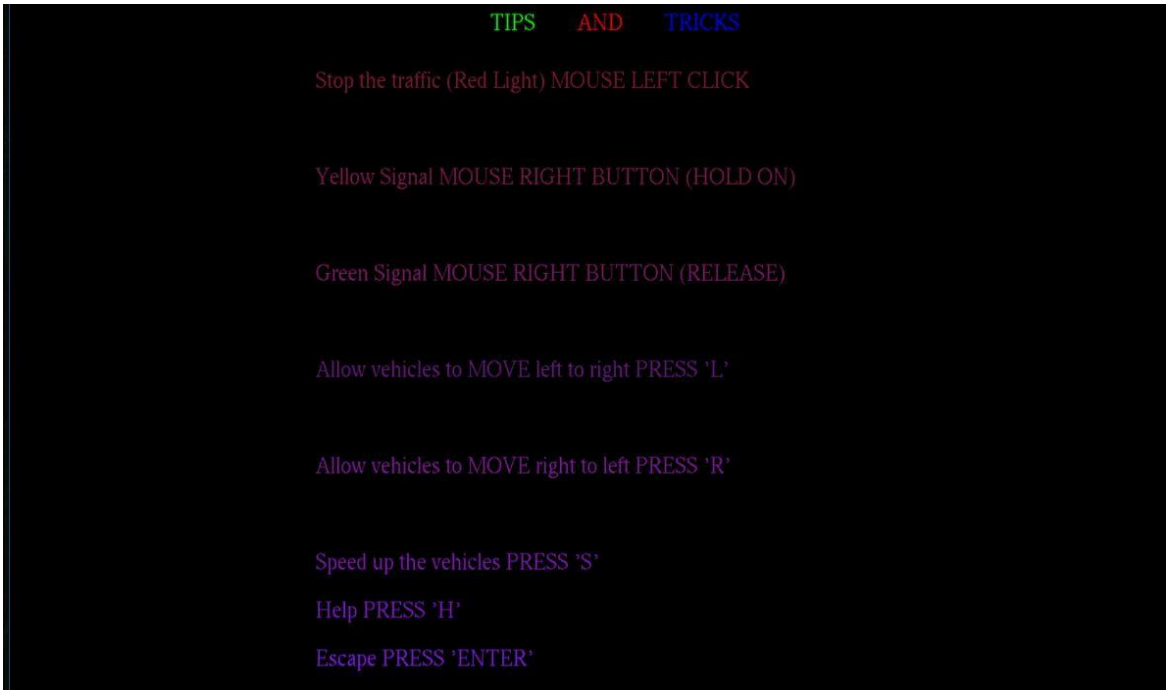


Fig 5.2: Tips and Tricks Menu

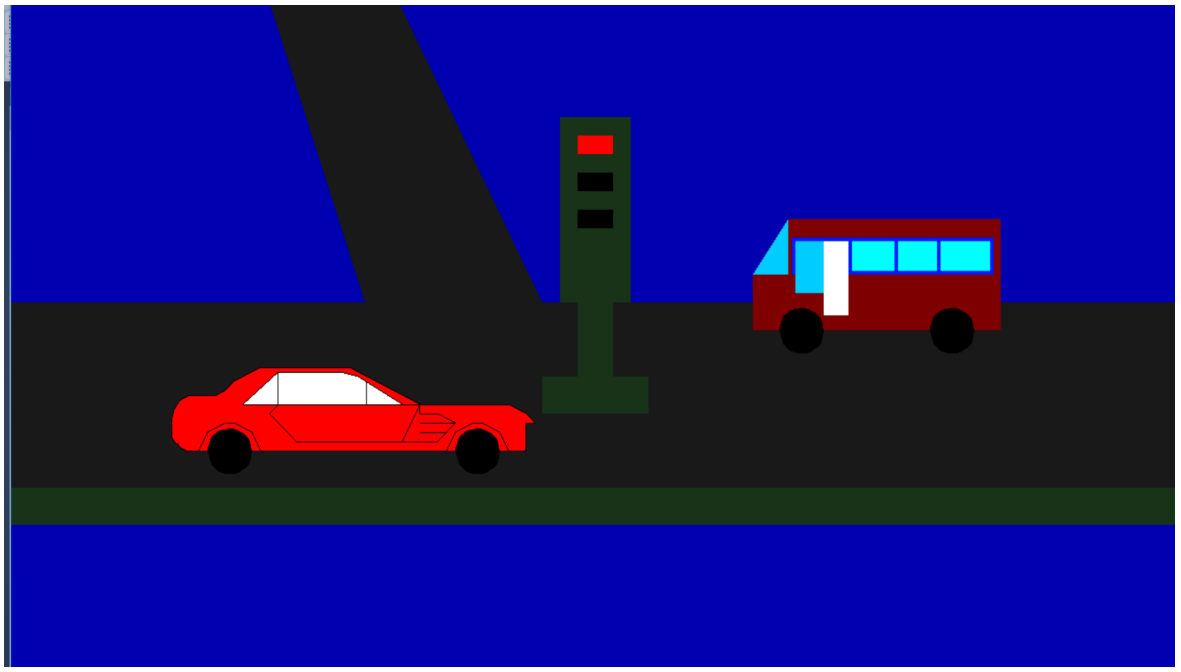


Fig 5.3: Red Signal

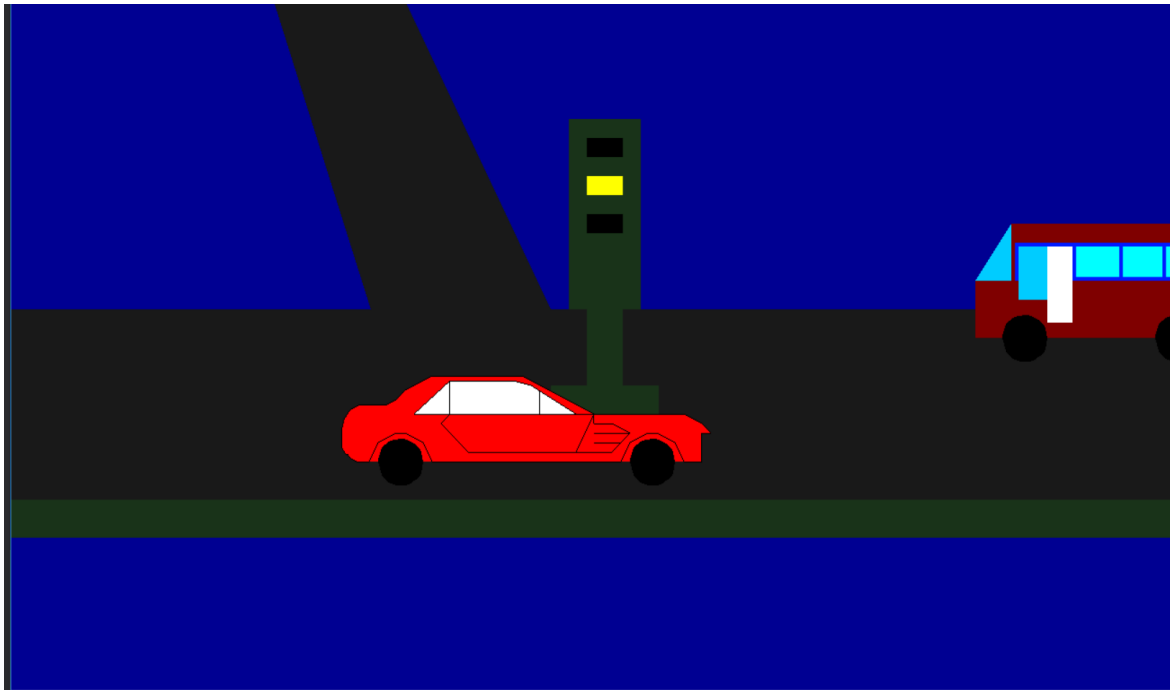


Fig 5.4: Yellow Signal

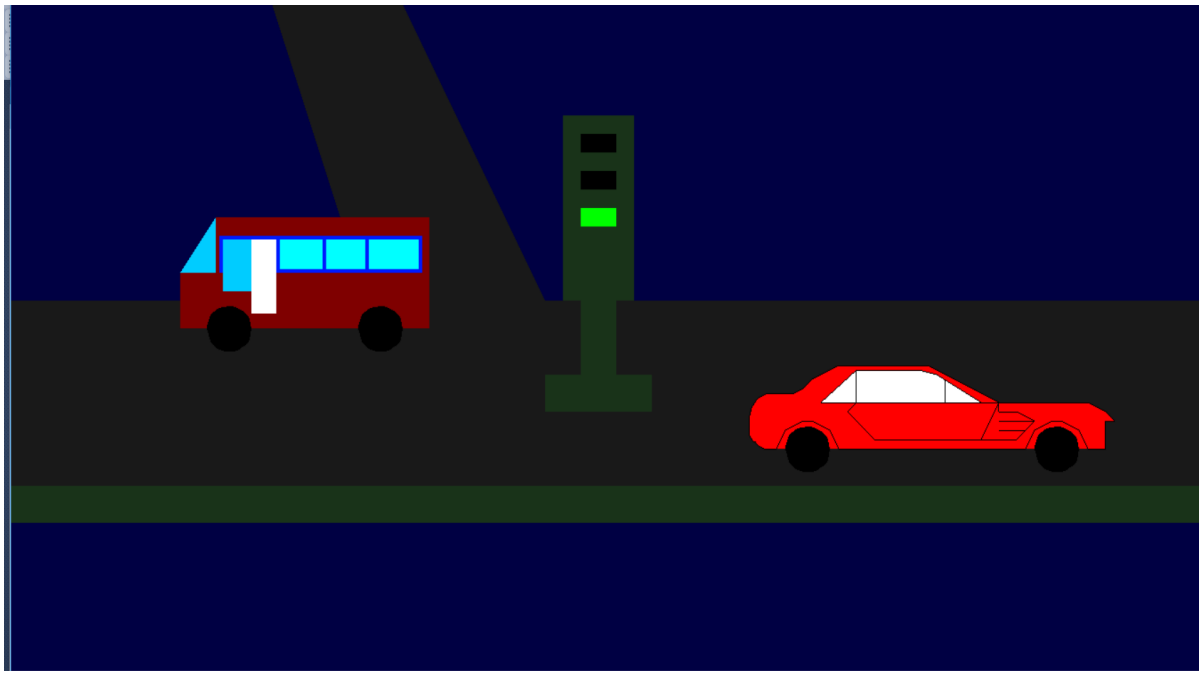


Fig 5.5: Green Signal

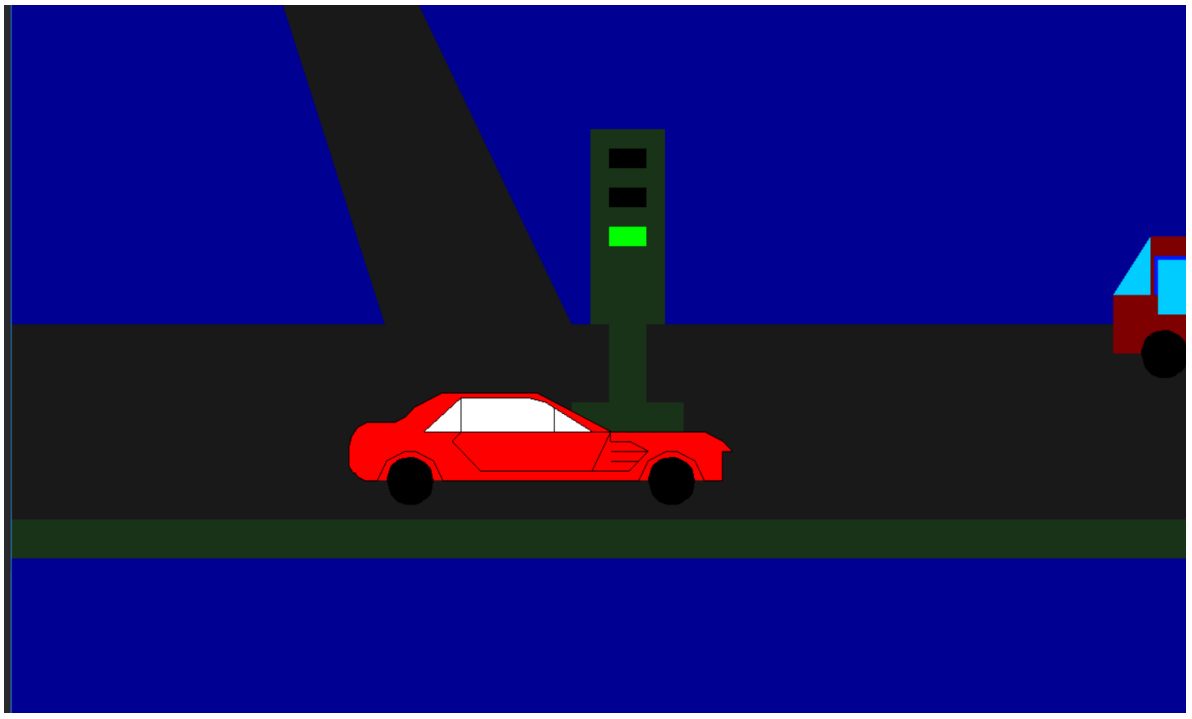


Fig 5.6: Moves Right Side Vehicles

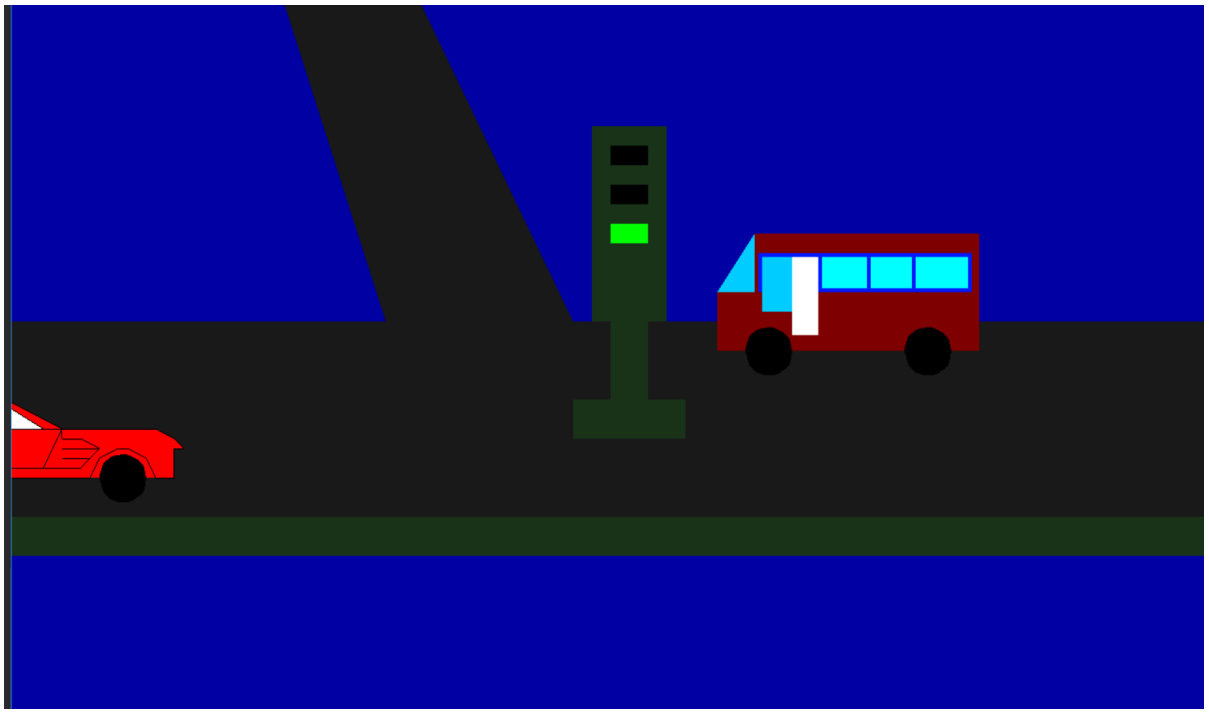


Fig 5.7: Moves Left Side Vehicles

CHAPTER 6

CONCLUSION AND FUTURE ENHANCEMENT

6.1 CONCLUSION

Using OpenGL and C++ we thus created and implemented Traffic Signal visualizer virtually. This project witnesses the fact that using OpenGL we can demonstrate how Traffic signals operates and the movement of the vehicles according to the current state of the traffic signal. This project makes a clear understandability of how these algorithms. It makes the student easier to study how the traffic signal works.

6.2 FUTURE ENHANCEMENT

Further we can add a detection mechanism for the accident occurring at the traffic signals, this may in turn reduce the accident rate. Finally, we can create programs with higher complexity using more of the in-built functions supported by OpenGL.

REFERENCES

1. Interactive Computer Graphics: A Top Down Approach with OpenGL- Edward Angel, 5th Edition, Addison-Wellesley,2008

2. Online tutorials for game development at NeHe productions.

3. OpenGL Red Book and Blue Book for reference.

4. www.opengl.org for OpenGL tutorials