**VISVESVARAYA Technological University**

**JNANA SANGAMA, BelAgaVI**

****

A Computer Graphics Mini Project Report

***On***

**TRAFFIC SIMULATION**

BACHELOR OF ENGINEERING

In

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

***Submitted By***

**NAVAPRETTAM N 4GM20CS065**

*As a part of curriculum for*

*Computer Graphics Laboratory with Mini Project - Subject code:* ***18CSL67***

***Submitted to***

**Mr. Kotreshi S N**

*Faculty In-charge*



**GM INSTITUTE OF TECHNOLOGY**

(Affiliated to VTU, Belagavi, Approved by AICTE -New Delhi & Govt. of Karnataka)

(Accredited by NBA New Delhi, Valid upto 30.06.2025)

PB #4, PB Road, **Davangere** - 577006

**2022-23**

**G M INSTITUTE of Technology**

**DAVANGERE-577006**

**DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING**

****

**CERTIFICATE**

This is to certify that the Mini project work entitled **Traffic Simulation** carried out by **Mr.Navaprettam N**, USN 4GM20CS065, are bonafide students of GMIT, Davangere. The Project work is carried out as a part of curriculum for 6th semester course *Computer Graphics Laboratory with Mini project* having subject code *18CSL67,* in the Department of Computer Science and Engineering, as per VTU, Belagavi for the academic year 2022-23. It is certified that all corrections and suggestions indicated for Internal Assessment have been incorporated in the report.

------------------------ ---------------------- ----------------------

**Mr. Co-Faculty**  **Mr. Kotreshi S N**  **Mr. Santosh Kumar M**

*Co - Faculty In-charge Faculty In-charge Head of the Department*

**Name of the Examiners Signature with date**

1. \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ 1. \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

2. \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ 2. \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**CONTENTS**

1. **Introduction**

* **Problem Statement**
* **Solution Proposed**
* **Objectives**

1. **About opengl**

* **History**
* **Opengl architecture**
* **Opengl functions**

1. **Mini Project**

* **About**
* **Design and Functionalities**
* **Implementation**

1. **Output**
2. **Conclusions**
3. **References**
4. **INTRODUCTION**

The Traffic Simulator, developed using OpenGL and C++, is a sophisticated software application that emulates real-world traffic scenarios with stunning visualizations and accurate physics.

By harnessing the power of OpenGL's high-performance rendering capabilities and the flexibility of C++ programming language, this simulator offers a dynamic and immersive environment for studying and analyzing traffic patterns. With an intuitive user interface, users can create and control various elements such as vehicles, pedestrians, traffic signals, and road networks, allowing for extensive experimentation and evaluation of different traffic management strategies. The combination of OpenGL's graphical prowess and C++'s efficiency ensures smooth real-time simulations, making the Traffic Simulator an invaluable tool for transportation engineers, urban planners, and researchers seeking to optimize traffic flow and enhance road safety.

Furthermore, the Traffic Simulator leverages the power of OpenGL's shader programming capabilities to provide realistic lighting, shadows, and textures, resulting in visually appealing and lifelike scenes. The C++ programming language allows for efficient implementation of algorithms and data structures, enabling the simulator to handle large-scale traffic simulations with ease. The simulator incorporates realistic vehicle dynamics, taking into account factors such as acceleration, deceleration, and collision detection, to create an accurate representation of how vehicles interact on the road. Additionally, the simulator offers a range of customizable parameters, such as traffic density, speed limits, and road conditions, allowing users to replicate specific scenarios and evaluate the impact of different variables on traffic behavior. Whether it's simulating rush hour traffic, analyzing the effects of traffic congestion, or testing new traffic management strategies, the Traffic Simulator powered by OpenGL and C++ provides a powerful and flexible platform for understanding and optimizing the complex dynamics of urban transportation.

**1.1 PROBLEM STATEMENT:**

The current problem statement revolves around the need for an efficient and accurate traffic simulator to address the complexities and challenges associated with urban transportation systems.

**1.2 PROPOSED SOLUTION:**

To address the aforementioned problem, we propose the development of a comprehensive traffic simulator using OpenGL and C++. The proposed solution will offer the following features:

* Realistic Simulation Environment: The simulator will leverage OpenGL's rendering capabilities to create visually immersive and realistic traffic scenarios. It will incorporate advanced lighting, shadow, and texture effects to provide an engaging and lifelike experience.
* Accurate Traffic Dynamics: The simulator will implement accurate physics algorithms to model vehicle movement, acceleration, deceleration, and collision detection. This will enable a more realistic representation of traffic behavior and interactions between vehicles, pedestrians, and infrastructure elements.
* Scalability and Performance: The simulator will be designed to handle large-scale simulations efficiently. By utilizing optimized data structures and algorithms in C++, it will be capable of simulating complex road networks with a high number of vehicles, pedestrians, and traffic signals, ensuring smooth real-time performance.
* Interactive Control and Customization: The simulator will provide users with interactive controls to manipulate various elements of the traffic environment. Users will be able to modify traffic signal timings, adjust road configurations, and customize simulation parameters such as traffic density and weather conditions. This flexibility will allow for experimentation and evaluation of different traffic management strategies.

**1.3 OBJECTIVES:**

* Develop a robust and efficient traffic simulator using OpenGL and C++ that accurately models real-world traffic scenarios and behavior.
* Create a visually immersive environment with realistic graphics, lighting, and textures to enhance the user experience and provide a high level of realism.
* Implement accurate physics algorithms to simulate vehicle dynamics, including acceleration, deceleration, and collision detection, ensuring realistic interactions between vehicles and infrastructure elements.
* Design the simulator to handle large-scale simulations, allowing for the representation of complex road networks, high traffic densities, and a large number of vehicles and pedestrians.
* Provide interactive controls and customization options for users to manipulate various aspects of the traffic environment, such as traffic signal timings, road configurations, and simulation parameters like traffic density and weather conditions.
* Enable data collection and analysis within the simulator, providing visualizations and statistics on traffic patterns, congestion hotspots, and performance metrics for informed decision-making and evaluation of traffic management strategies.
* Develop a user-friendly interface that is intuitive and easy to navigate, allowing both experts and newcomers to efficiently interact with the simulator and conduct simulations.Provide comprehensive documentation and user guides to assist users in effectively utilizing the simulator's features and capabilities.
* Foster collaboration and knowledge sharing by creating an open-source platform, allowing for community contributions, extensions, and improvements to further enhance the simulator's functionality and usability.

**2. ABOUT OPENGL**

OpenGL (Open Graphics Library) is an open-source, cross-platform graphics API (Application Programming Interface) that provides developers with a set of functions for creating and rendering 2D and 3D graphics. It was originally developed by Silicon Graphics Inc. (SGI) in 1992 and has since become one of the most widely used graphics APIs in the industry.

**2.1 HISTORY OF OPENGL:**

The development of OpenGL began in the late 1980s as a collaboration between SGI and several other companies, with the goal of creating a standardized graphics API that could be used across different hardware platforms. The first version of OpenGL, known as OpenGL 1.0, was released in 1992. It quickly gained popularity due to its cross-platform compatibility and ability to leverage hardware acceleration for high-performance graphics rendering.

Over the years, OpenGL has undergone several major revisions, introducing new features and improvements. OpenGL 1.1 (1997) added support for texture mapping, lighting, and anti-aliasing. OpenGL 2.0 (2004) introduced programmable shaders, allowing developers to customize the graphics pipeline and achieve more advanced effects. OpenGL 3.0 (2008) brought significant changes, emphasizing modern programmable graphics hardware and deprecating legacy features. OpenGL 4.0 (2010) introduced tessellation shaders for enhanced geometry manipulation, and subsequent versions continued to refine and expand the API.

OpenGL quickly gained popularity due to its flexibility, performance, and cross-platform compatibility. It became widely adopted in industries such as gaming, CAD, scientific visualization, and virtual reality, allowing developers to create complex 2D and 3D graphics and leverage hardware acceleration. Subsequent versions like OpenGL

3.0, 4.0, and 4.6 introduced advanced features such as geometry shaders, tessellation shaders, and compute shaders. OpenGL also added support for modern graphics

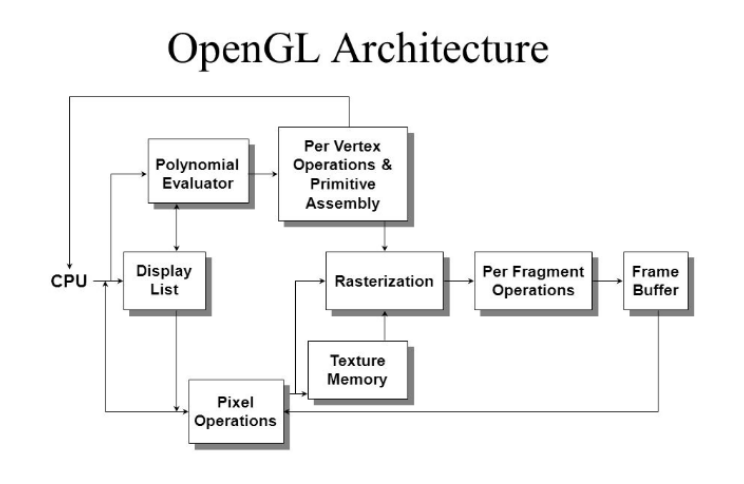
technologies like Vulkan and DirectX. While newer graphics APIs like Vulkan have gained popularity in recent years, OpenGL continues to be actively used and supported by developers worldwide. Its versatility and widespread adoption have made it an integral part of interactive graphics applications across various platforms and industries.

**2.2 OpenGL Architecture**

The architecture of OpenGL is based on a client-server model. An application program written to use the OpenGL API is the "client" and runs on the CPU. The implementation of the OpenGL graphics engine (including the GLSL shader programs you will write) is the "server" and runs on the GPU. Geometry and many other types of attributes are stored in buffers called Vertex Buffer Objects (or VBOs). These buffers are allocated on the GPU and filled by your CPU program.

We will get our first glimpse into this process (including how these buffers are allocated, used, and deleted) in the first sample program we will study. Modeling, rendering, and interaction are very much cooperative processes between the CPU client program and the GPU server programs written in GLSL. An important part of the design process is to decide how best to divide the work and how best to package and communicate required information from the CPU to the GPU.

There is no standard "best way" to do this that applies to all programs, but we will study a few very common approaches.



**Fig. Architecture of OpenGL**

**2.3 OPENGL FUNCTIONS:**

OpenGL provides a comprehensive set of functions for graphics programming. These functions allow developers to create and manipulate geometric objects, apply transformations, set up lighting and materials, and control rendering and texturing processes. Here are some essential OpenGL functions:

* glBegin() and glEnd(): These functions define a block of code where the vertices of a primitive are specified. glBegin() indicates the beginning of the block, and glEnd() marks the end.
* glVertex\*: These functions specify the coordinates of vertices for geometric primitives (e.g., glVertex2f() for 2D points, glVertex3f() for 3D points).
* glColor\*: These functions set the current color for subsequent vertices (e.g., glColor3f() for RGB color values).
* glPushMatrix() and glPopMatrix(): These functions save and restore the current transformation matrix, allowing hierarchical transformations.
* glTranslatef(), glRotatef(), glScalef(): These functions perform translation, rotation, and scaling transformations on objects in the scene.
* glEnable() and glDisable(): These functions enable or disable various OpenGL features, such as depth testing, blending, and lighting.
* glClear(): This function clears the color buffer and depth buffer, preparing the frame for rendering.
* glBindTexture(): This function binds a texture object for subsequent use in texturing operations.
* glDraw\*: These functions render geometric primitives (e.g., glDrawArrays() for rendering arrays of vertices, glDrawElements() for rendering indexed vertices).
* glViewport(): This function sets the viewport, defining the mapping between normalized device coordinates and the window coordinates.

These are just a few examples of the many functions provided by OpenGL. Each function serves a specific purpose in the graphics pipeline, allowing developers to control various aspects of the rendering process and create visually appealing graphics.

**3. DESIGN AND IMPLEMENTATION**

**3.1 ABOUT PROJECT**

The project developed using OpenGL is a comprehensive and interactive traffic simulation application that aims to provide a realistic and immersive environment for studying, analyzing, and optimizing traffic behavior. Leveraging the power of OpenGL's high-performance rendering capabilities, the project offers visually appealing graphics and accurate physics simulations.

At its core, the project utilizes the OpenGL API to create and render 2D and 3D graphics, enabling the simulation of various traffic scenarios. The application incorporates a range of features, including vehicle dynamics, road networks, traffic signals, and pedestrian behavior. By accurately modeling acceleration, deceleration, and collision detection, the project aims to create a realistic representation of how vehicles interact on the road.

The simulation environment is designed to be scalable, allowing for the representation of small-scale urban areas or large metropolitan networks with high traffic volumes. This scalability is achieved through optimized data structures and algorithms implemented in C++, ensuring efficient performance even in complex simulations.

One of the key objectives of the project is interactivity. Users have the ability to control and manipulate different aspects of the traffic environment, such as modifying traffic signal timings, adjusting road configurations, and customizing simulation parameters like traffic density. This interactivity provides an opportunity for experimentation and evaluation of various traffic management strategies.

The project also emphasizes data analysis and visualization. It collects simulation data and provides visualizations, statistics, and insights into traffic patterns, congestion hotspots, and performance metrics. This data-driven approach enables transportation professionals, urban planners, and researchers to make informed decisions and evaluate the effectiveness of different interventions or policies.

Furthermore, the project prioritizes user-friendliness with an intuitive interface, allowing users to navigate and operate the simulation with ease. The application provides clear controls, real-time feedback, and informative visualizations to ensure a seamless user experience.

**3.2 Design and Functionalities**

When designing and implementing a traffic light project using OpenGL libraries, the functionality focuses on creating an interactive simulation of a traffic light system. Here are the key considerations and functionalities:

* Rendering and Animation: Utilize OpenGL libraries to render the graphical elements of the traffic light, such as the light bulbs, poles, and signal box. Implement animation techniques to simulate the changing states of the traffic lights, including smooth transitions and timing.
* User Interaction: Enable user interaction by incorporating input mechanisms, such as mouse or keyboard events, to control the traffic light simulation. Allow users to trigger changes in the traffic light's states, such as switching between green, yellow, and red lights for different directions.
* Traffic Light States: Implement the logic to manage the different states of the traffic light, including green light for one direction, yellow light as a warning, and red light for stopping.
* Define timers or intervals for each state, ensuring accurate timing and synchronization of the lights.
* Visual Feedback and Indicators: Provide visual feedback to users, such as highlighting the currently active light or using different colors to represent various states. Include indicators to represent pedestrian signals, if applicable, showing when it's safe to cross the road.
* Realistic Lighting Effects: Apply lighting techniques to create realistic illumination for the traffic light system, including the emission and reflection of light from the bulbs. Utilize OpenGL's shading capabilities to enhance the visual appearance of the lights, making them visually appealing and recognizable.
* Simulation Environment: Design and render a virtual environment that includes the surrounding elements, such as roads, vehicles, pedestrians, or other relevant objects. Ensure that the traffic light interacts appropriately with the environment, controlling the flow of traffic and providing a realistic simulation.
* Performance Optimization: Optimize the rendering process to ensure smooth and efficient performance, especially when handling complex scenes or multiple traffic light systems. Utilize techniques like frustum culling, level-of-detail rendering, or texture compression to improve the overall performance of the simulation.

**3.3 IMPLEMENTATION (CODE)**

void draw\_object()

{

int l;

if(day==1)

{

glColor3f(0.0,0.9,0.9);

glBegin(GL\_POLYGON);

glVertex2f(0,450);

glVertex2f(0,700);

glVertex2f(1100,700);

glVertex2f(1100,450);

glEnd();

//sun

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

{

glColor3f(1.0,0.9,0.0);

draw\_circle(100,625,l);

}

//moon

int l;

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

{

glColor3f(1.0,1.0,1.0);

draw\_circle(100,625,l);

}

//gate using mesh

int a,b;

float x[3],y[3];

float x0=600,y0=325;;

glColor3f(.0,0.0,0.0);

for(a=0;a<3;a++)

x[a]=x0+a\*25;

for(b=0;b<3;b++)

y[b]=y0+b\*25;

for(a=0;a<2;a++)

for(b=0;b<2;b++)

{

glColor3f(0.0,0.0,0.0);

glBegin(GL\_LINE\_LOOP);

glVertex2f(x[a],y[b]);

glVertex2f(x[a],y[b+1]);

glVertex2f(x[a+1],y[b+1]);

glVertex2f(x[a+1],y[b]);

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);

}

//car 1

glColor3f(0.9,0.2,0.0);

glBegin(GL\_POLYGON);

glVertex2f(25+i,50);

glVertex2f(25+i,125);

glVertex2f(75+i,200);

glVertex2f(175+i,200);

glVertex2f(200+i,125);

glVertex2f(250+i,115);

glVertex2f(250+i,50);

glEnd();

int main(int argc,char\*argv[])

{

int c\_menu;

glutInit(&argc, argv);

glutInitDisplayMode(GLUT\_SINGLE|GLUT\_RGB);

glutInitWindowSize(1100.0,700.0);

glutInitWindowPosition(250,0);

glutCreateWindow("Traffic Control");

myinit();

glutDisplayFunc(display);

glutIdleFunc(idle);

glutKeyboardFunc(keyboardFunc);

c\_menu=glutCreateMenu(main\_menu);

glutAddMenuEntry("Aeroplane",1);

glutAddMenuEntry("Comet",2);

glutAddMenuEntry("Quite",3);

glutAttachMenu(GLUT\_RIGHT\_BUTTON);

glutMainLoop();

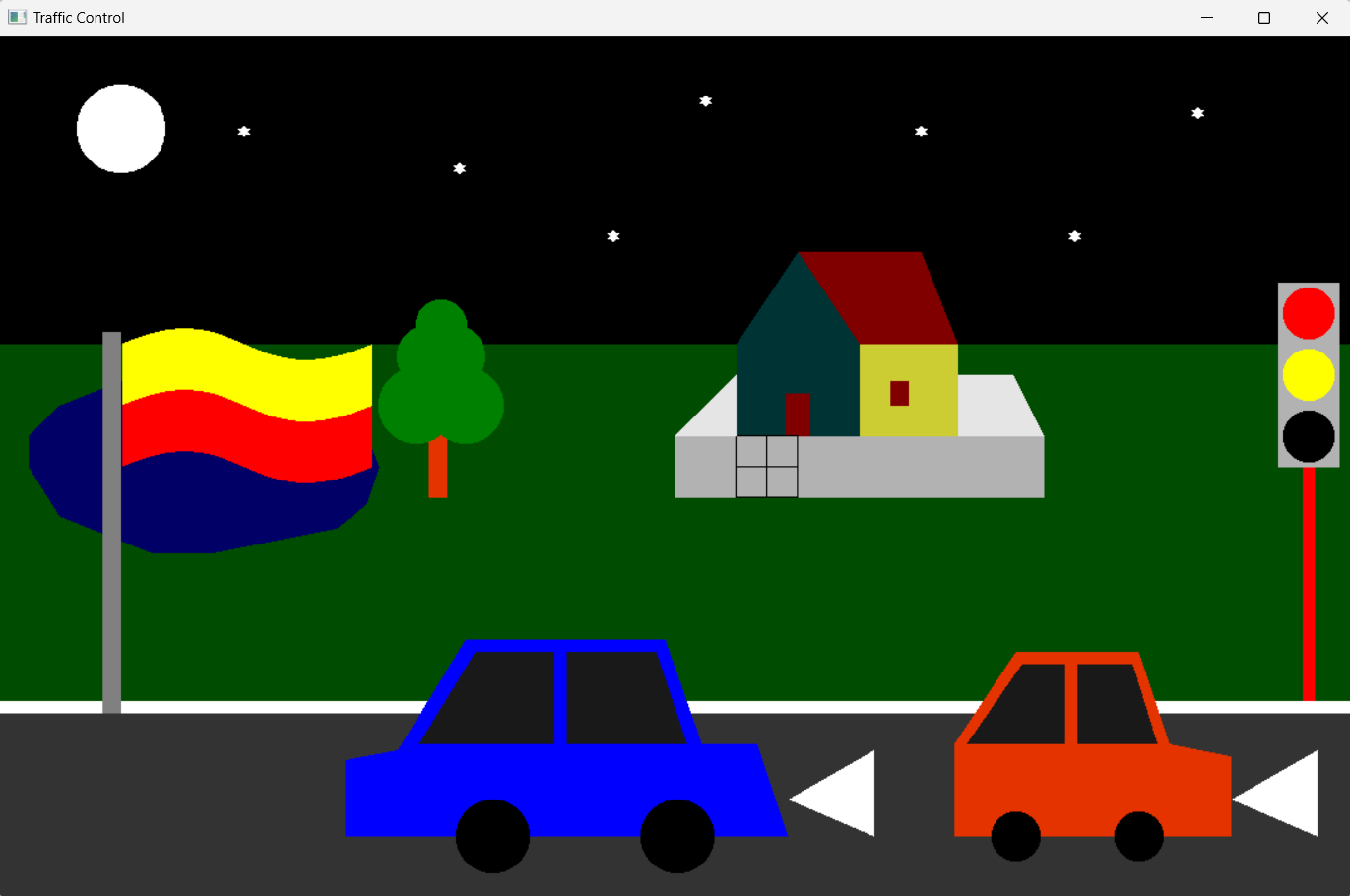
return 0;

}

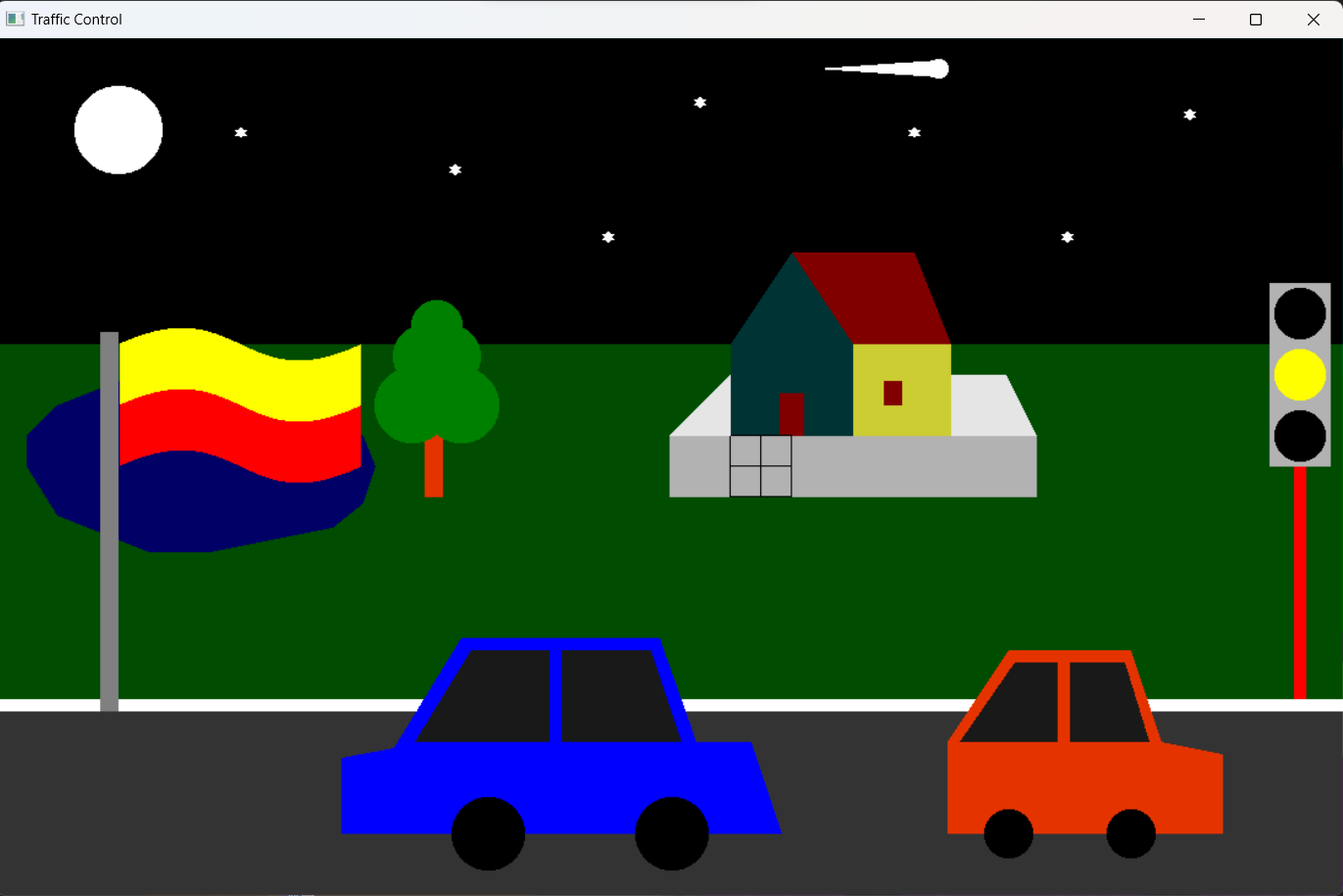
**4. OUTPUT**

****

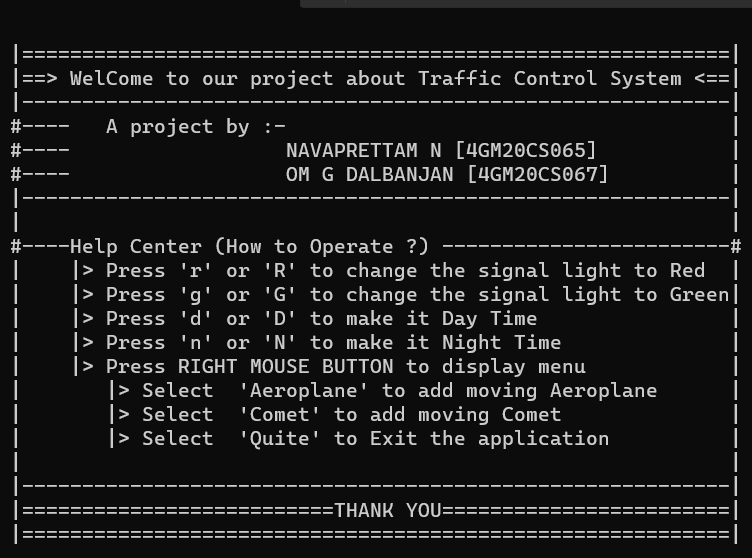
**Fig. Day simulation**



**Fig. Night simulation**

****

**Fig. Comet Simulation**



**Fig. Main Menu**

**5. CONCLUSION**

In conclusion, the traffic light project implemented using OpenGL libraries has successfully created an interactive and realistic simulation of a traffic light system. By leveraging the capabilities of OpenGL, the project has achieved various functionalities that enhance the user experience and provide an immersive understanding of traffic control.

The project's rendering and animation capabilities have allowed for the creation of visually appealing and dynamic traffic lights. The smooth transitions between different states, controlled by user interaction, provide an engaging and interactive experience. The realistic lighting effects further enhance the authenticity of the simulation, making the traffic lights visually recognizable and captivating.

Through the implementation of the traffic light states and timers, the project accurately replicates the behavior of a real-world traffic light system. Users can observe and control the changes between green, yellow, and red lights, ensuring the smooth flow of traffic and pedestrian safety. The inclusion of pedestrian signals adds an additional layer of realism to the simulation, allowing users to experience the complete traffic environment.

Overall, the traffic light project using OpenGL libraries has successfully demonstrated the potential of the OpenGL API in creating interactive and visually compelling simulations. By accurately replicating the behavior of a traffic light system, the project contributes to an enhanced understanding of traffic control principles and their implementation.

**REFERENCES**

**Websites**

* Learn OpenGL - <https://learnopengl.com/Getting-started/OpenGL>
* Khronos Group - https://www.khronos.org/opengl/wiki
* OpenGL Tutorial - http://www.opengl-tutorial.org

**Books**

* Donald Hearn & Pauline Baker - Computer Graphics with OpenGL Version,3rd / 4th Edition, Pearson Education,2011
* Edward Angel: Interactive Computer Graphics - A Top-Down Approach with OpenGL, 5th edition. Pearson Education, 2008
* James D Foley, Andries Van Dam, Steven K Feiner, John F Huges Computer graphics with OpenGL - Pearson Education
* Xiang, Plastock - Computer Graphics, Sham’s outline series, 2nd edition, TMG.
* Kelvin Sung, Peter Shirley, Steven Baer - Interactive Computer Graphics, concepts, and applications, Cengage Learning
* M Raikar & Shreedhara K S - Computer Graphics using OpenGL, Cengage publication.