# OpenGL Car Race Game

## CS352: Computer Graphics & Visualization Lab

Project Report


Course Instructor:
Dr. Somnath Dey

Submitted By:

Gaurav Jain – 200001023
Nishit Singh – 200001056
Vihaan Thora – 200001079

# Introduction

Our project is based on creating a car racing game using OpenGL. For this, we have upgraded an existing car-driving application whose source code can be found at https://github.com/jackgerrits. This code used GLFW and GLM as base libraries and also included some custom external libraries called 'fmt' and 'glad'. This application consisted of a car that could be driven around the terrain. We wanted to modify it such that it turns into a playable game, which can display a score to the player. The main car was included as an object file (.obj and .mtl), and so were the obstacles, such as trees, cones, stumps, etc.

Some of the notable features of the source code were – 1) Physics implementation in car driving, 2) Dynamic Shadow Mapping of the car, 3) Dynamic Particle Generation, and 4) Reflection and Refraction in the water body. Our first task was to create a track that could be used as a lap. For this, we altered the blend map file and defined a square track in the game environment. We also had to make the terrain flat and thus changed the height map file to fit our requirements. We also placed the cones on the left and right edges of the track to make the demarcation clearer. We removed the water body altogether, as the other terrain was not to be driven upon anymore.

Secondly, we wanted to define a set of rules that could be implemented in order to award a score to the player. For this, we based our score calculation on two factors – 1) Time taken by the player to complete a lap, 2) Penalty incurred by the player whilst completing the lap. The penalty is added whenever the player leaves the track. We also wanted to ensure that the scoring system is robust, and hence declared checkpoints at every corner of the track so that a player cannot take shortcuts to complete the lap faster. Players also get a warning when they are going in the wrong way, although no penalty is added if they are. The final display of the game prints the current lap time, score, number of laps completed, speed, fps, and penalty incurred in a given lap. The score and lap count are the only figures that aggregate throughout the gameplay. The text itself was displayed using an external library called 'freetype' and some more files to display the characters on the screen.

We also included keyboard shortcuts to change the camera view of the car (which had to be done by dragging the mouse earlier). There is an option for 6 different views – default (reset), top, left, right, back, and reverse. We also added a night mode to the game, where the visibility gets lower and makes the game more challenging. In the night mode, the headlights of the car turn on automatically, and in the day mode, they are off by default. However, players can toggle them by their own wish by pressing a key. The final task was to create a welcome menu at the beginning of the game, to let the user know about the key controls of the game. The game also plays a background track when the player starts by hitting enter key.

Overall, the project, although built on an existing code, highlighted many challenges for us, such as – 1) Understanding the code flow (how the different textures are mapped onto the display, shadow mappings, game physics, layout, terrain management, etc.), 2) Printing text on the screen (as glut functions were unavailable), 3) Keeping track of the player's position with respect to the track to mark checkpoints and penalty, 4) Implementing the wrong way and lap time function (explicit multithreading), 5) Making the gameplay look and feel more consistent.

## Specifications

1) We first need to download the base libraries, which can be downloaded using the command –
```
$ sudo apt-get install libglfw3-dev libglm-dev
```
2) Then, create a folder called ‘build’ inside the main folder and cd into that.
```
mkdir build
cd build
```
3) Then run the commands
```
cmake ..
make
```
4) You can then run the application using the command
```
./opengl-car-game
```

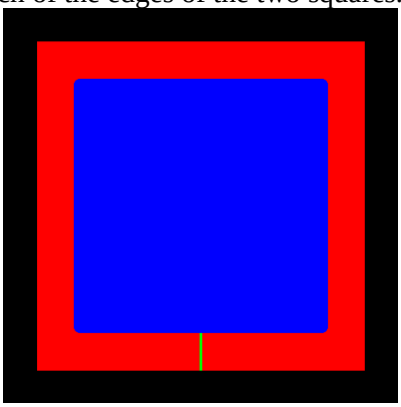
## Instructions to play the game

1) When the game window opens, you will see a menu window, that will show the key controls listed below –
    a. W/A/S/D : Throttle/Left/Right/Brake.
    (Alternatively, you can also use the arrow keys for equivalent control)
    b. Space : Handbrake
    c. I : Reverse View
    d. J : Left View
    e. L : Right View
    f. K : Back View
    g. T : Top View
    h. R : Reset View to Default
    i. N : Toggle Night Mode
    j. H : Toggle Headlights
2) You can then press the Enter key to start the game.
3) The car will spawn at the start-finish line, and the lap timer will start from zero.
4) You then have to start driving along the track, while making sure you don’t leave the track or cut corners while turning.
5) The last checkpoint should increment every time you cross a corner. If it doesn’t, it means you have missed the checkpoint. You have to turn back and complete the checkpoint in order to complete the lap.
6) Avoid taking penalties, or your score will be affected.
7) After you complete the lap, the lap’s score is added to your existing score. It is calculated based on the time taken to finish a lap and is reduced as per the penalty point incurred. The least value for score in a lap can be 0.
8) The lap counter on your screen denotes the laps you have completed.
9) If you want to challenge your friends in this game, you can try emulating a mini grand prix and set a target to complete 10 laps with the highest score possible.
10) Enjoy the game!

## Functionalities Implemented

(existing code base reference)

1) **Map** – The map is a track centered at the origin, with two squares of side lengths 196 and 250. The racing track is the area bound by these two squares. The area outside the bigger square is covered with grass, the track itself is made of asphalt, and the interior region is covered with mud. These textures are mapped onto the map using texture mapping, with the colors mapped to textures in the blend map. The track is surrounded by cones on both sides, which are placed using 8 loops running on each of the edges of the two squares.



(Fig. Texture Blend Map Used. Black – Grass, Red – Asphalt, Blue – Mud, Green – Start Finish Line)

2) **Penalty Detection** – The penalty is detected every time the player goes out of bounds. This is checked in every iteration of the render loop, and the penalty is incremented by one if the out-of-bounds condition is true.
3) **Checkpoint Detection** – If the player gets within a certain range of the 4 checkpoints, we consider that they have crossed the checkpoint. The checkpoints in the implementation are kept as a Boolean array, and if all the checkpoints are visited, then and only then will the lap be counted and the score be incremented.
4) **Score Calculation** – The score is calculated by the formula defined as score = (100000/lap time in seconds) - penalty. Every time the player crosses the start-finish line after crossing all four checkpoints, we add the lap's score to the current score and increment the lap count by 1. We also reset the checkpoints visited and the penalty incurred.
5) **Text** – Since we are not using the standard glut library of OpenGL, displaying text was not as straightforward. In OpenGL, rendering text is achieved through creative techniques since OpenGL has no support for text capabilities. The most common technique is to render character textures onto quads. However, creating relevant character textures could be challenging. The FreeType library is used to render text since it can load fonts, render them to bitmaps, and provide support for several font-related operations. The library can load TrueType fonts that define character glyphs using mathematical equations. Each glyph loaded with FreeType does not have the same size, and FreeType also loads several metrics that specify how large each character should be and how to properly position them. Metrics include the glyph's width, height, horizontal bearing, vertical bearing, and horizontal advance. Text is rendered by storing the generated data somewhere in the application and querying it whenever a character needs to be rendered. The data is defined in a convenient struct and forwarded to the vertex shader, which multiplies the coordinates with a projection matrix and forwards the texture coordinates to the fragment shader. Finally, a VBO and VAO are created for rendering the quads.
6) **Dynamic Display of Parameters** – All the parameters calculated in the above steps are then passed on to the renderer, where we then format the parameters onto the display for the player to see. The list of parameters includes - current lap time, score, number of laps completed, speed, fps, and penalty incurred in a given lap.
7) **Menu** – When the game begins, we call a special renderer that displays the menu, till the user presses enter. The menu displays the instructions on the screen. When enter is pressed, the main renderer is called, and the game begins.

8) **Camera Views** – The player can see the different views of the car while driving by pressing certain keys. This is handled by rotating the view matrix by certain angles according to the view direction required.
9) **Night Mode** – The player can toggle between the day and night modes, which basically reduces the diffuse light coming from the surface, making the overall appearance darker.
10) **Headlights** – These are created as sources of light with a narrow cone radius, which can be turned off (radius 0) and on (radius default).
11) **Sound** – Added a background music track when the game is started. This is done using the 'irrKlang' library that performs implicit multithreading to play sounds. Irrklang is a powerful and easy-to-use sound library for use in OpenGL games. It supports a wide range of audio formats and provides a comprehensive set of features for sound playback, such as 3D sound, multi-channel playback, and spatial audio effects. Irrklang's API is designed to be simple and intuitive, allowing developers to easily add high-quality audio to their games. It also offers seamless integration with other game engines and libraries. Implementing Irrklang in an OpenGL game is a straightforward process that involves loading audio files, creating sound sources, and manipulating sound properties in real-time to enhance the gaming experience.

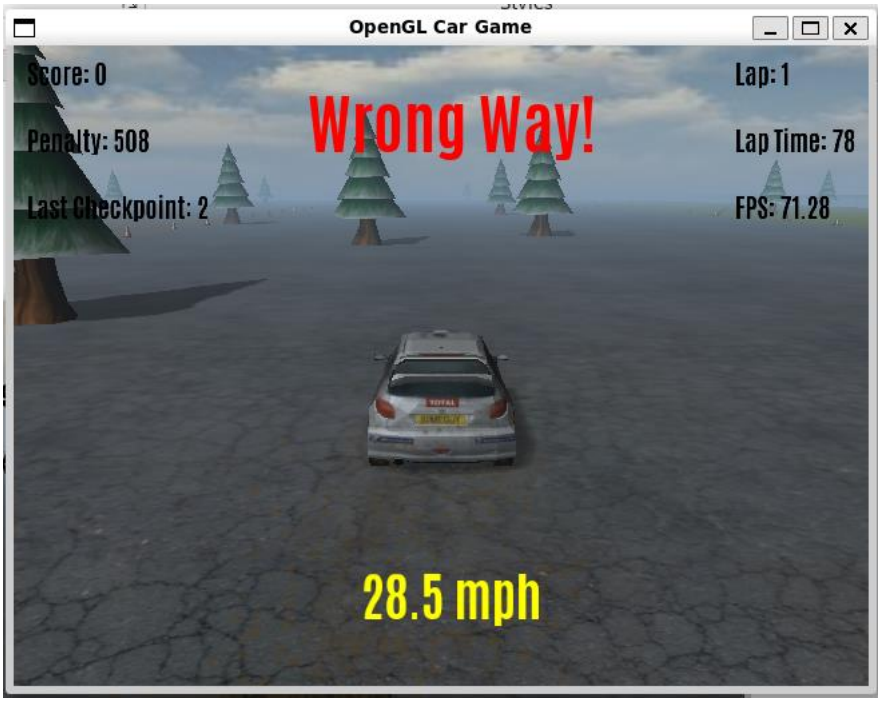
## Output

1) Game Menu


2) After Hitting Enter (Spawn Position)

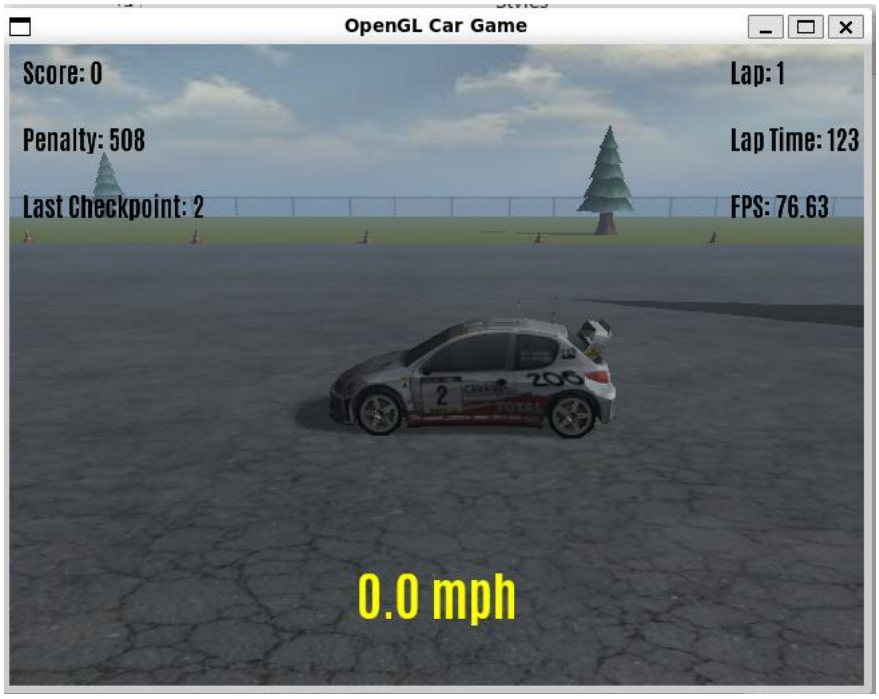
3) Driving in the game (on the track)


4) Going in the wrong direction

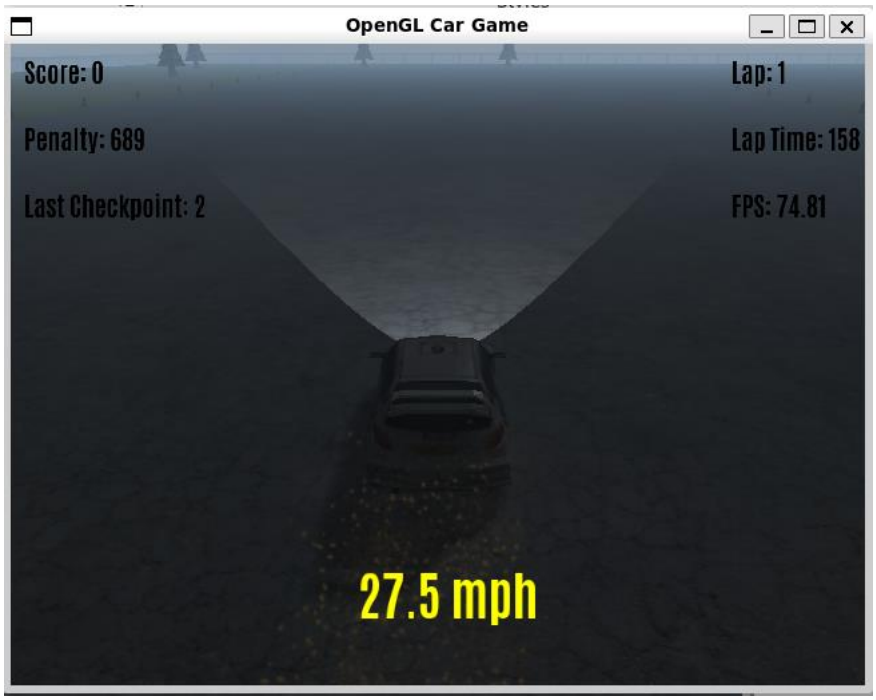


5) Top View (on pressing ‘T’)

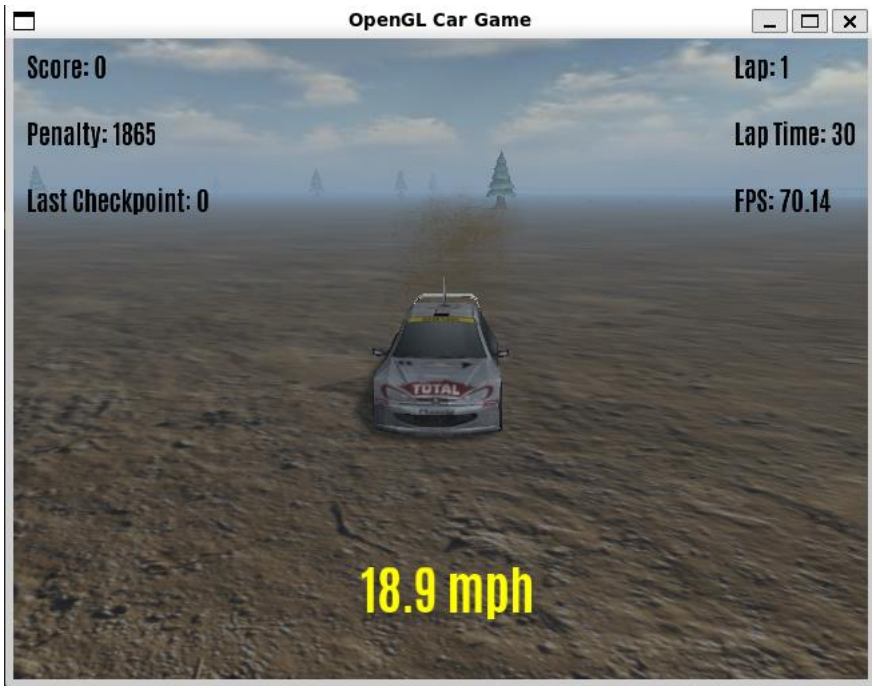

6) Left View (on pressing ‘J’)



7) Night mode (on pressing ‘N’)




8) Rear view (on pressing ‘I’)



9) Zoomed-out view (on scrolling down)