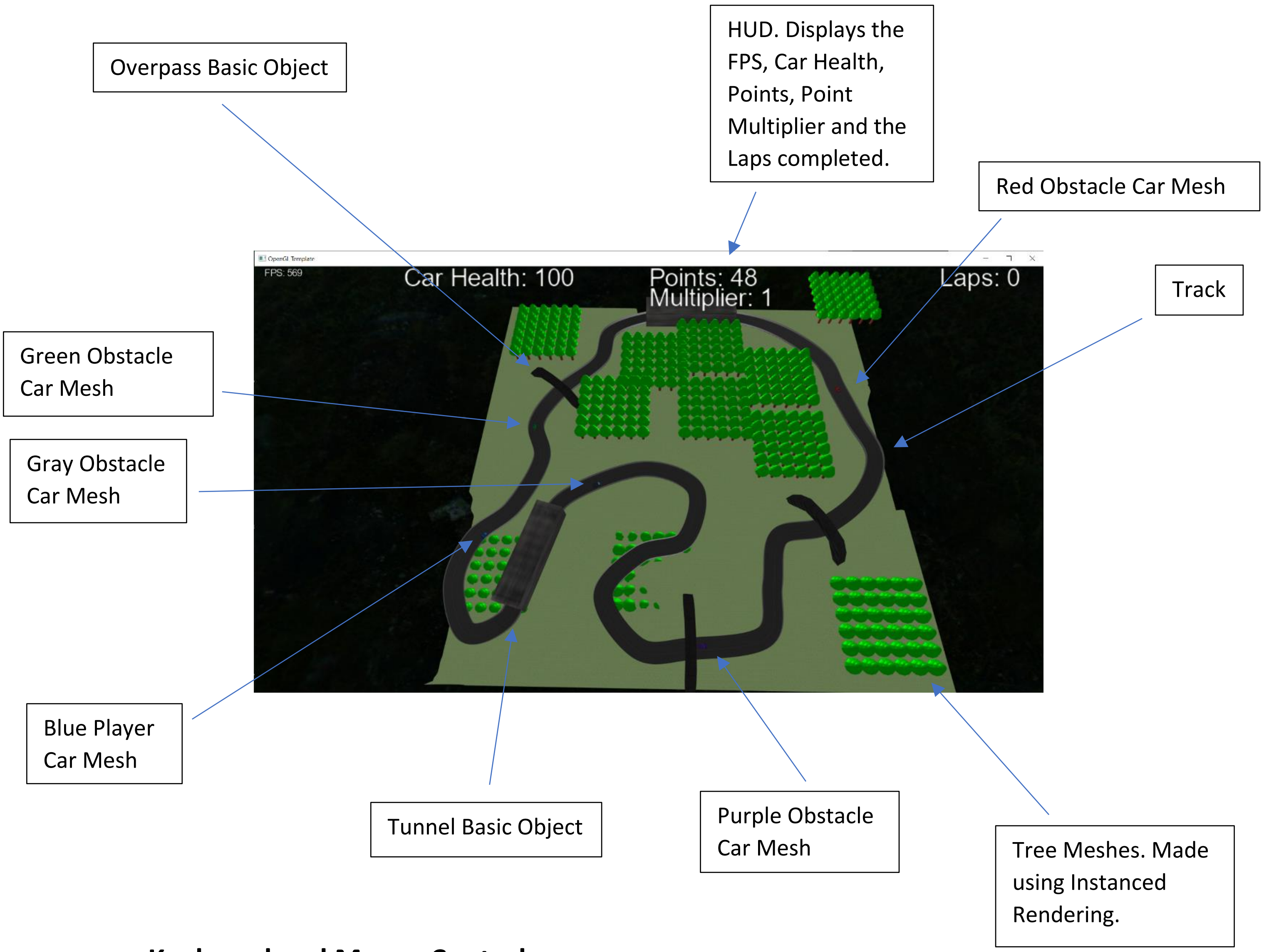
IN3005 Computer Graphics

Coursework Report

Tayyab Hussain

The game for my coursework will be a driving game. In the game, you will control the blue player car and attempt to make it around the track as many times as possible while avoiding the other cars on the track. Your score increases with every second your car is intact(above 0 health), you will increase your score multiplier every time you complete a lap. The cars you must avoid are moving at different speeds along the track and you as the player have to navigate around them and not crash into them so that you don't get damage to your car and lose health. If your health drops below 1, you will die, the car will be destroyed, the game will end and your score will be displayed.

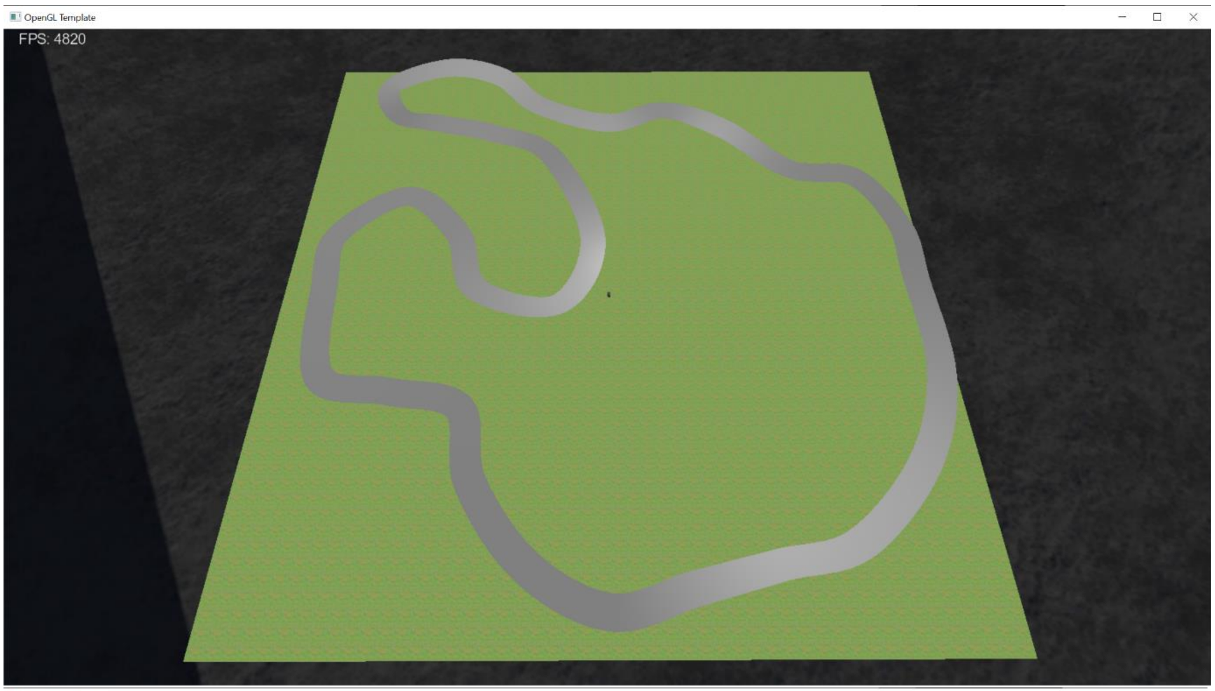


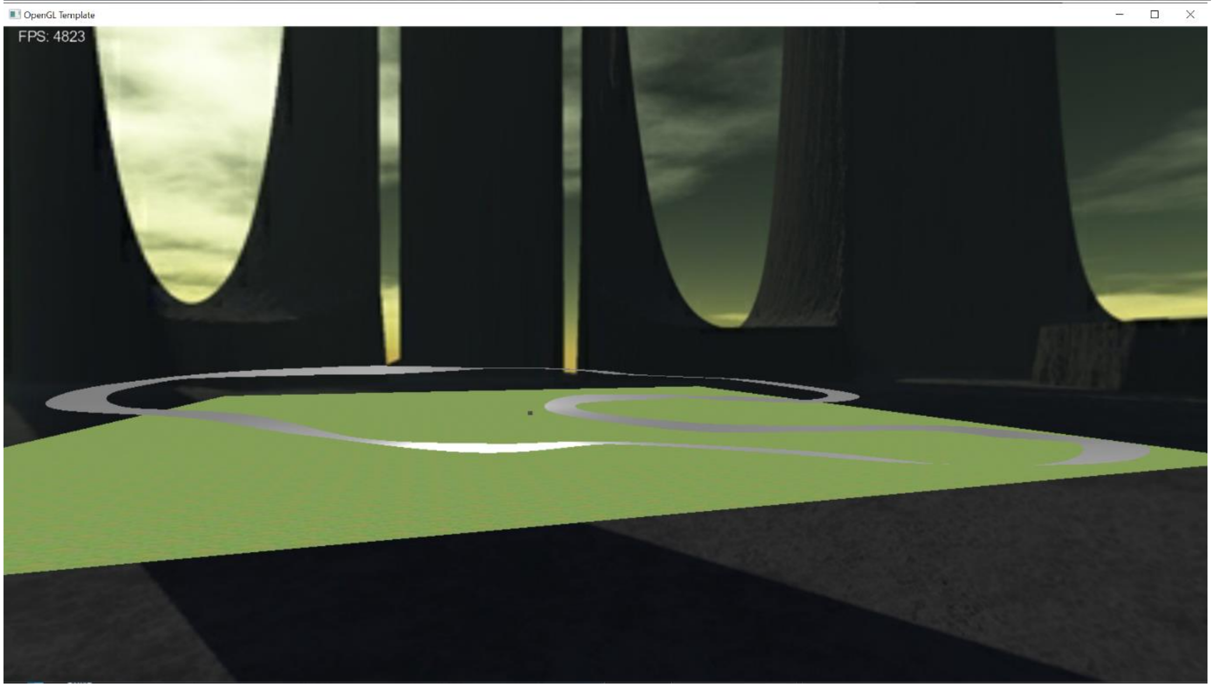
Keyboard and Mouse Controls:

Accelerate	'W'
Brake	'S'
Turn Left	'A'
Turn Right	'D'
First-Person Camera	'1'
Third-Person Camera	'2'
Free Camera	'3'
Quit	'ESC'

Route and Camera

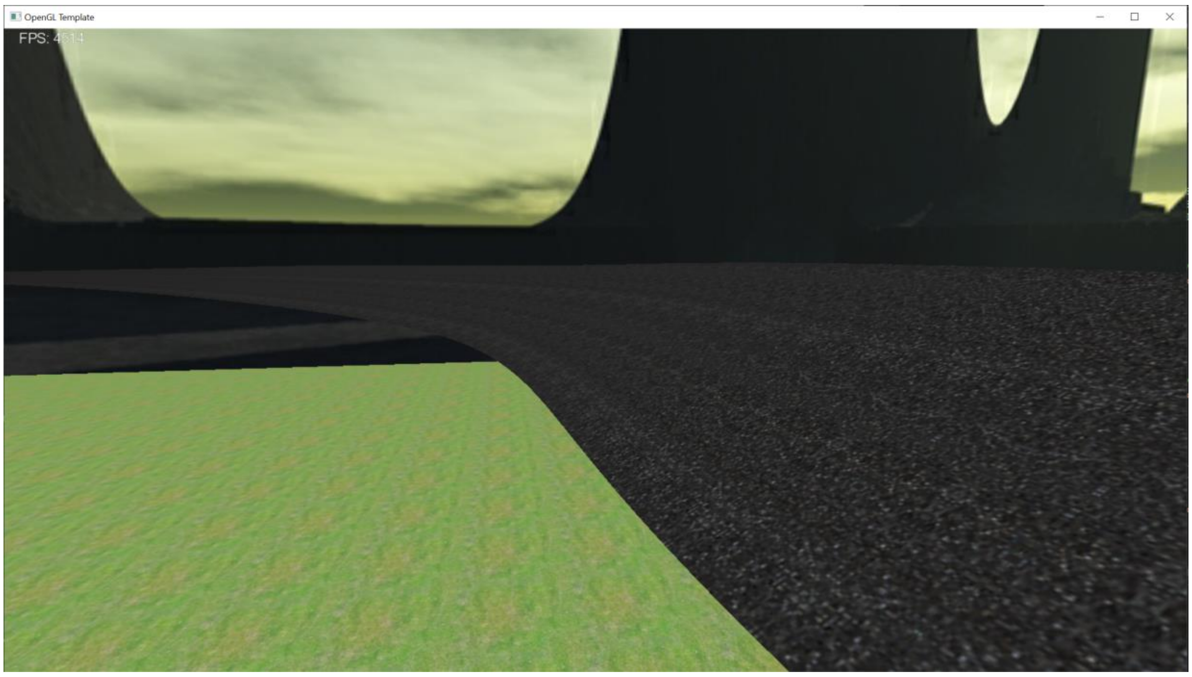
Route:





This is the track I have created for my game. I implemented it using the Catmull-Rom spline. There are 33 points along the track, each with its own individual up-vector to create banked or off-camber corners.

After applying a texture to it, here is what the track looks like now:

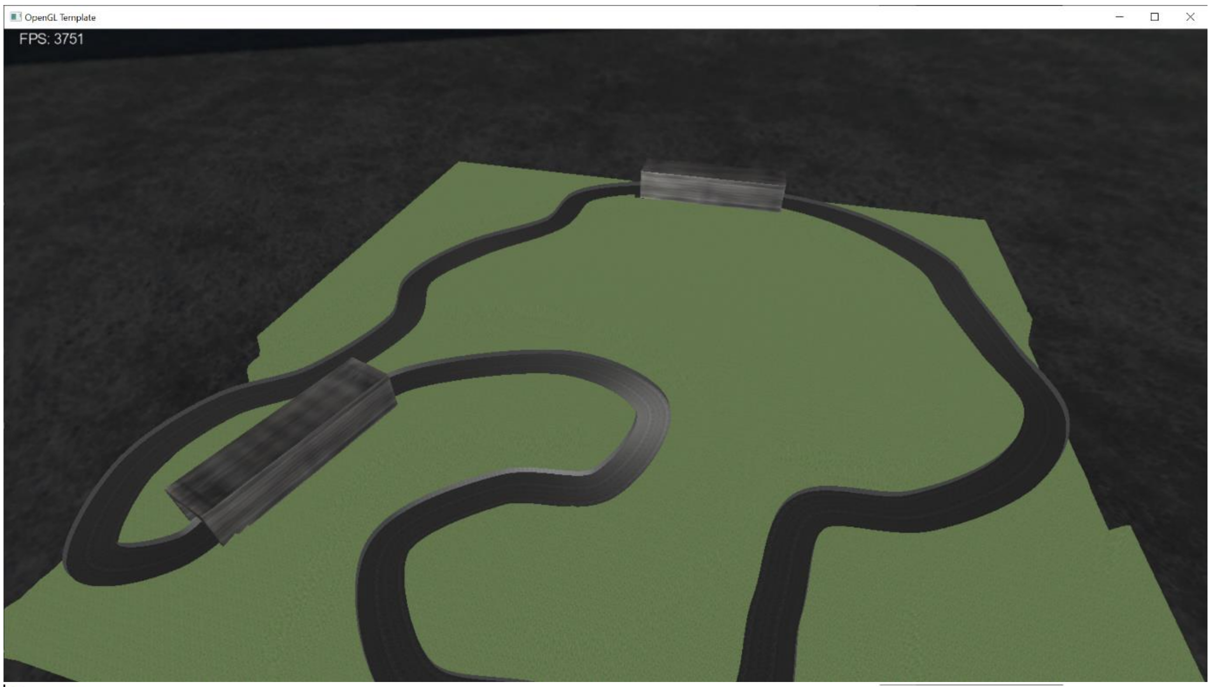


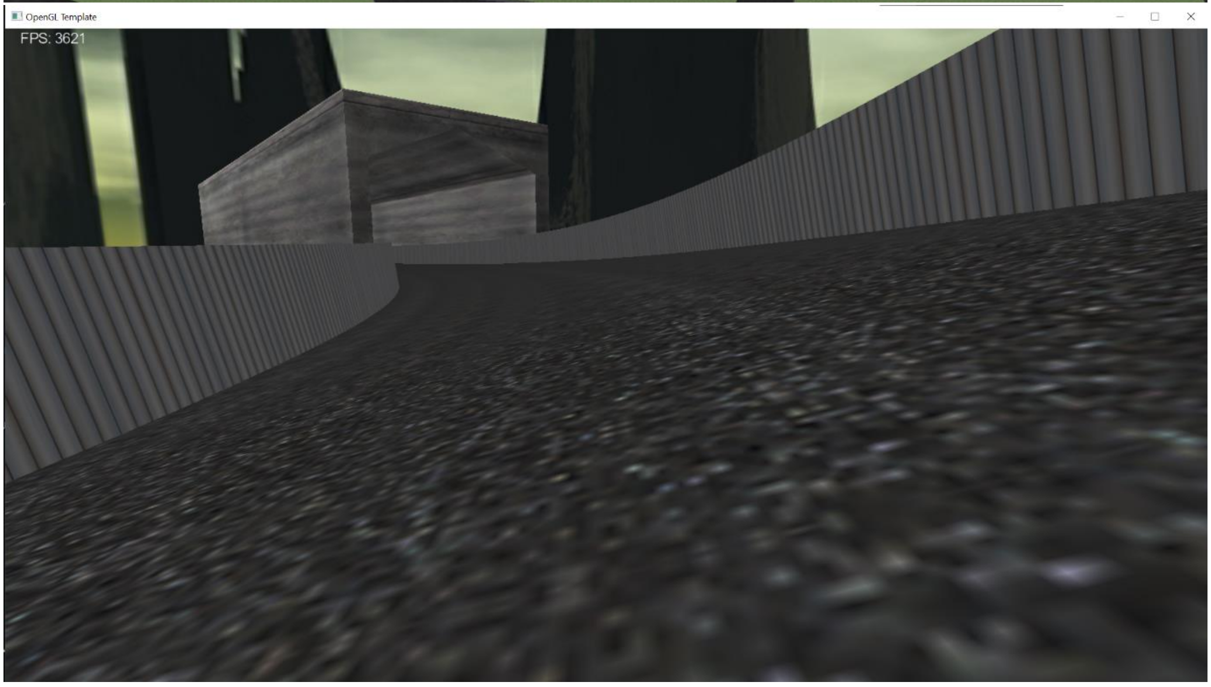
I added height-mapped terrain though it is very rudimentary. I designed it using Microsoft Paint 3D, so it doesn’t blend in well. I’m hoping that once I add some more meshes I can hide the rough edges of my heightmap.

I then placed my Tunnel primitive in 2 locations along the track and used a Catmull-Rom spline again to create 2 barricades on either side of the track. The barricades use the offset points (that create the track width) and a TNB frame from each of these points to position a new point above the current offset point in the direction of the centre line`s up vector.

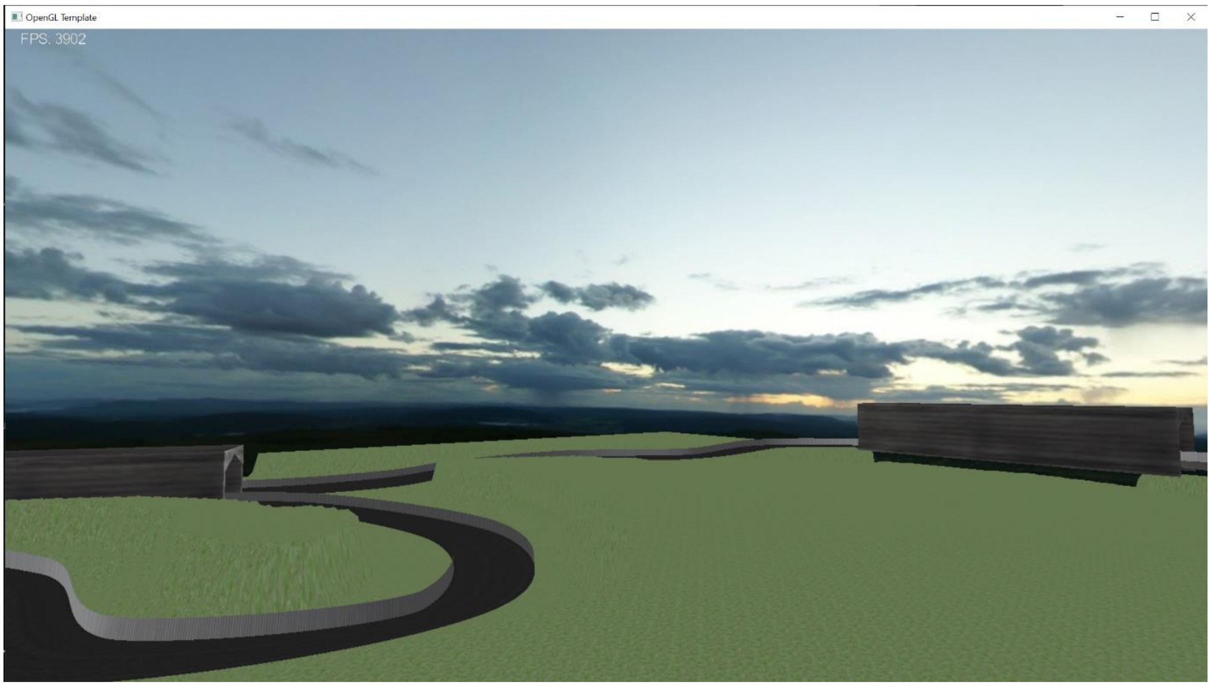
I did have a few issues with the texturing of the barricades. Since both the track and the barricades had different textures, but they were part of the same class, only one texture would be applied to both objects. I could not find an elegant solution to this problem. I could’ve simply not applied a texture to the track or barricade, but I didn’t like how this looked. So instead, I created a new Catmull-Rom spline class and renamed it BarricadeSpline. This class is identical to the original in every way except that it doesn’t render anything other than the barricades and therefore I can use the main Catmull-Rom class to render the track and track texture and the barricade class to render the barricades and barricade texture.

This is what the track looks like as of now:





I also changed the skybox at this stage, to match the theme of my game more closely. This was the outcome:

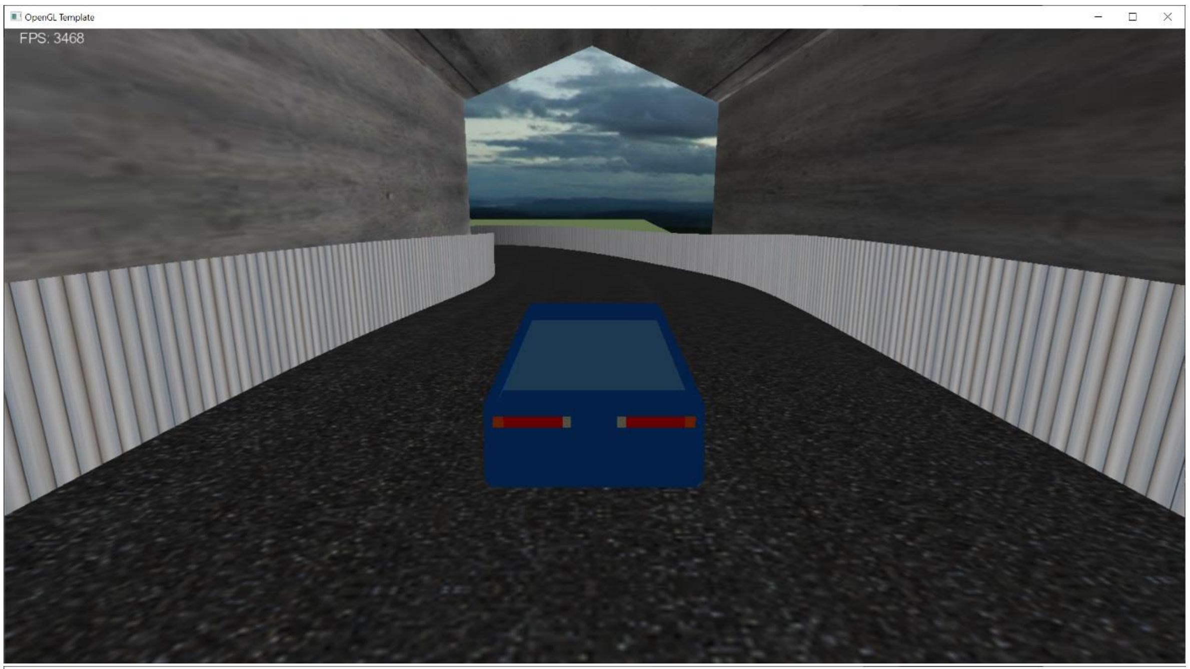


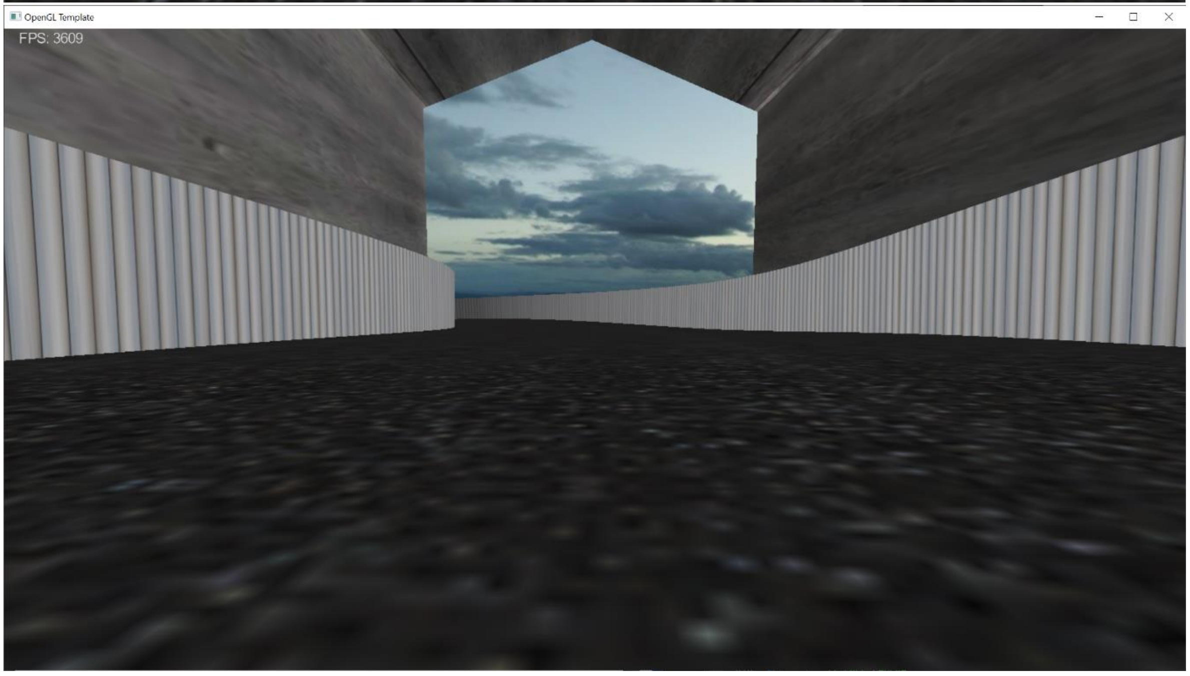
Camera:

The 2 camera modes I decided to add to my game were third-person and first-person. I have implemented keyboard controls to change between these 2 camera modes and the free-view camera (The keyboard controls have been detailed above).

The way I implemented this is by first adding a mesh(player car) to my game and specifying its position and orientation using a TNB frame calculated from the track spline. I then set the cameras relative to the player's car, e.g. behind and above the car for the third-person camera. In terms of the keyboard controls, I used 3 boolean values for each of the cameras, third-person, first-person and free-view. In the code, once the corresponding key was pressed, I would set the related boolean value to true and set the others to false. Then in the update() method, the code checks to see which boolean is set to true and sets the camera accordingly.

These are what the 2 camera modes look like in my game:

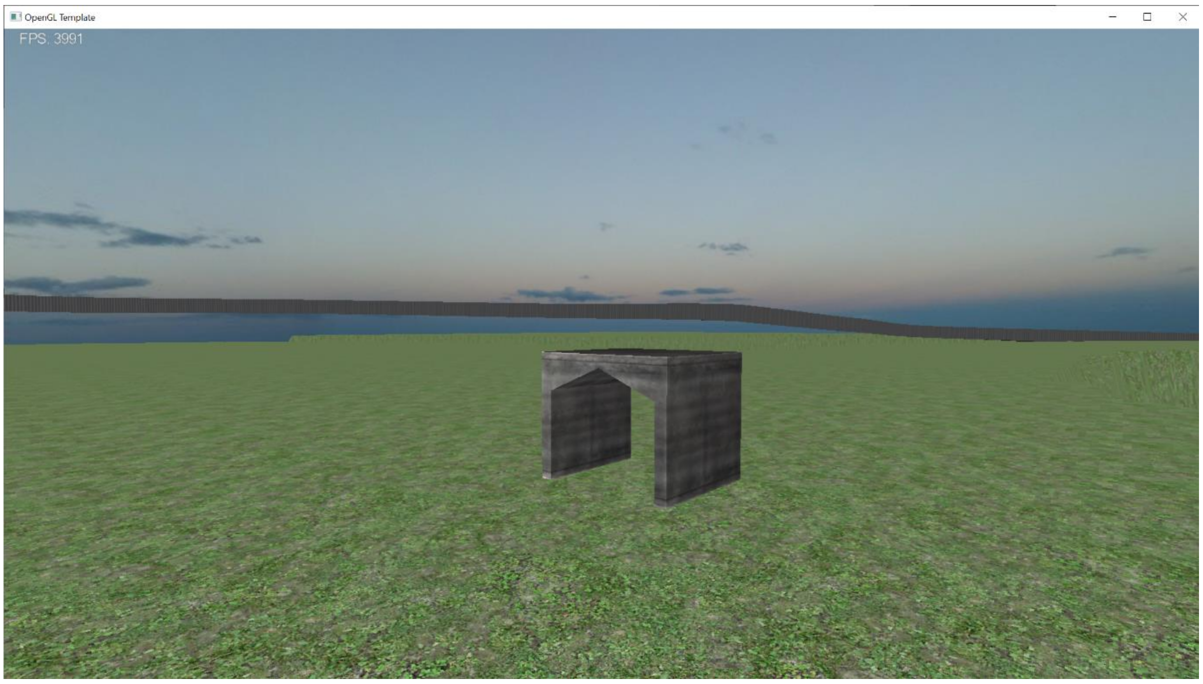




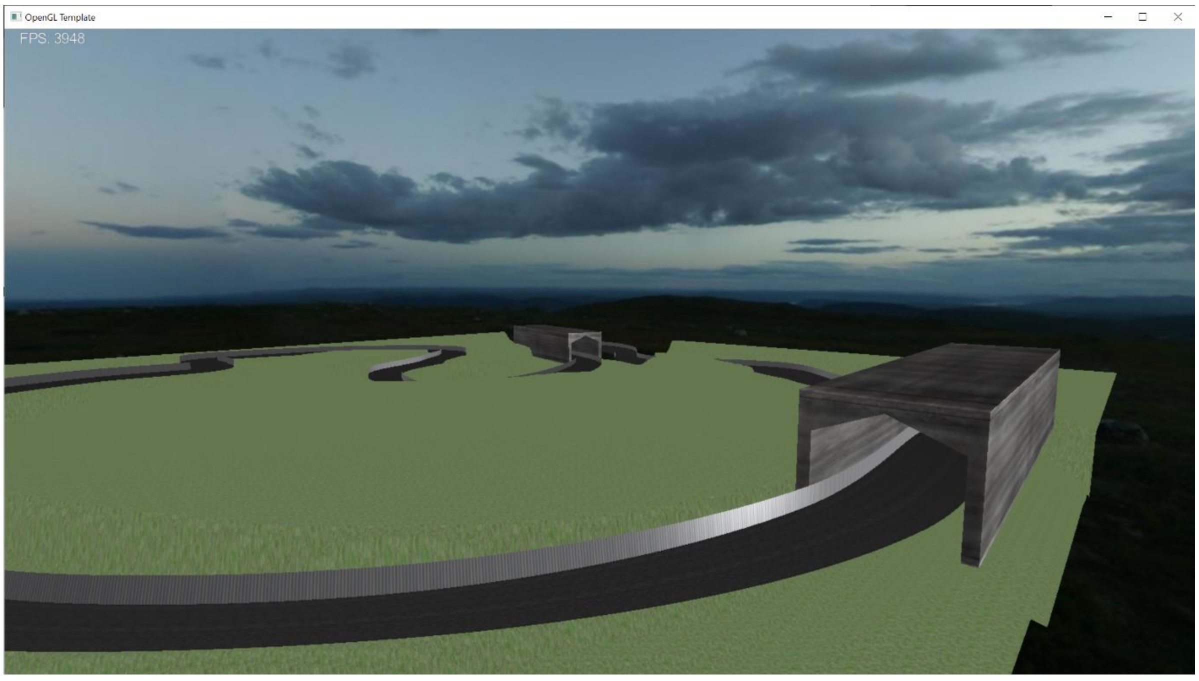
Basic Objects, Meshes and Lighting

Basic Objects:

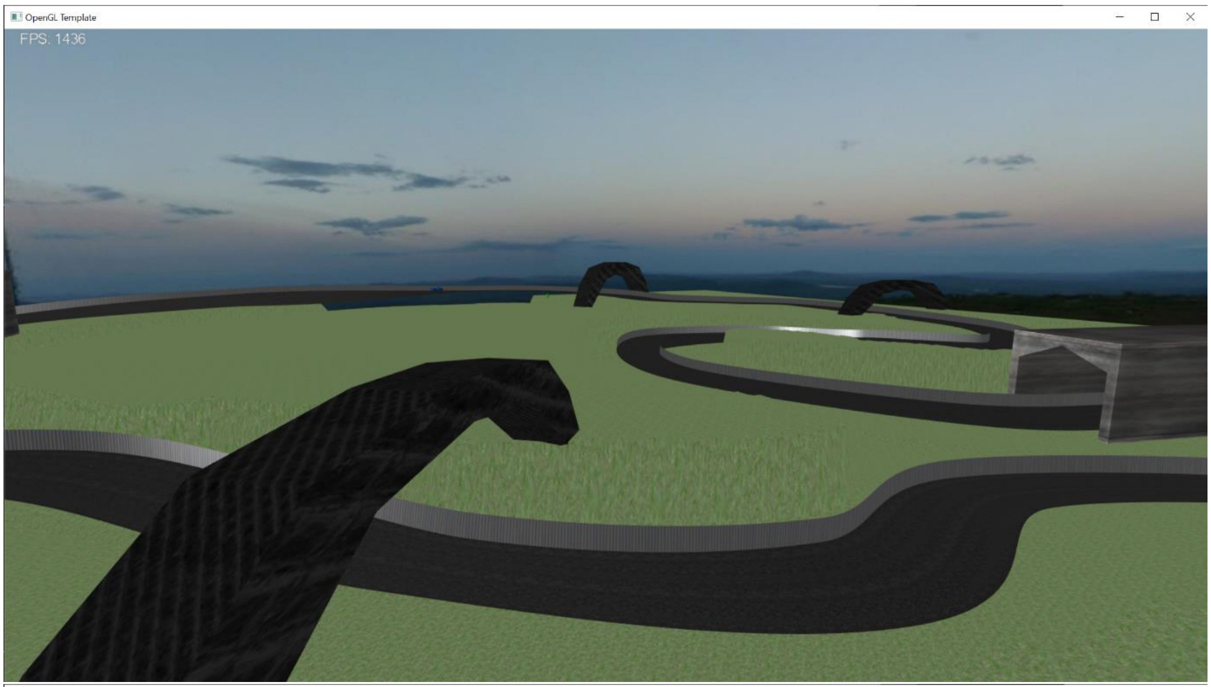
The first basic object I have designed is a tunnel. The tunnel is defined in its own class and has its vertex attributes pushed into a vertex buffer object or VBO. The tunnel object also has its own shader program which applies the Phong lighting model. The lighting is calculated per-fragment as opposed to per-vertex as this results in higher quality shading. Also, within the fragment shader, I have mixed the colour of my object, which was stored in the VBO, with the texel colour. This allows me to have a bit more control over how the final texture of my object looks.
Putting all this together, here is what my object looks like:

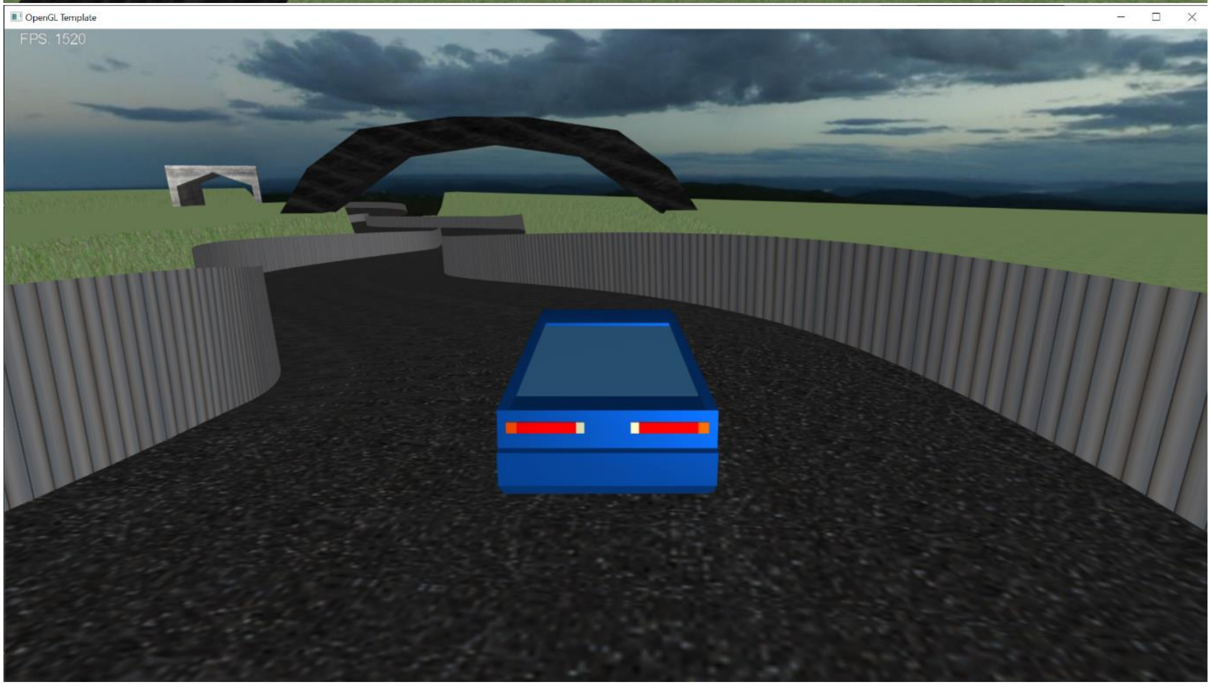


After placing 2 transformed instances of my object into the game, here is what it looks like now:



The second basic object I have added to my game is an overpass. My thinking was to create something that looked like a walkway to allow people at racetracks to cross the track safely. I wanted to create a tessellation shader for this object which would smooth out the rough edges and give it a more gradual slope. I did not manage to implement this within the allotted time. Although it would`ve unquestionably improved the look of the object, I think for this game, the current implementation looks ok. Since the tessellation shader implementation was unsuccessful, this object is currently being rendered using the same shader as the Tunnel object. After applying a texture and placing multiple instances of the overpass into the scene, here is what the game looks like now:

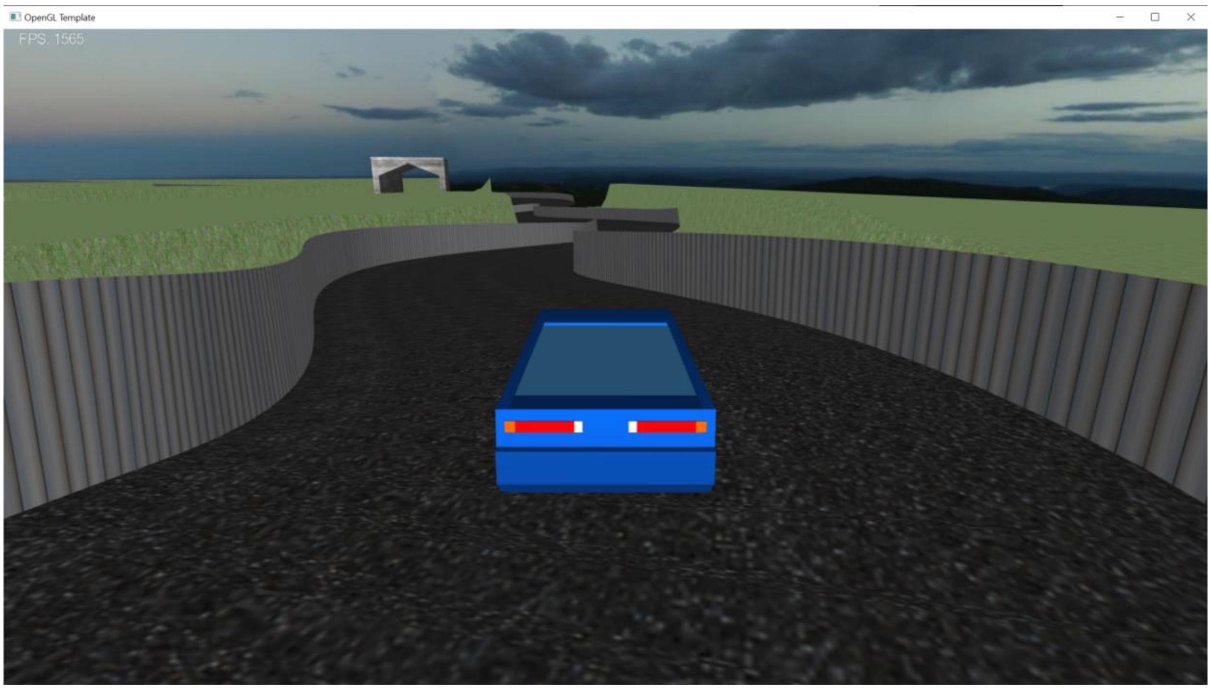




Meshes:

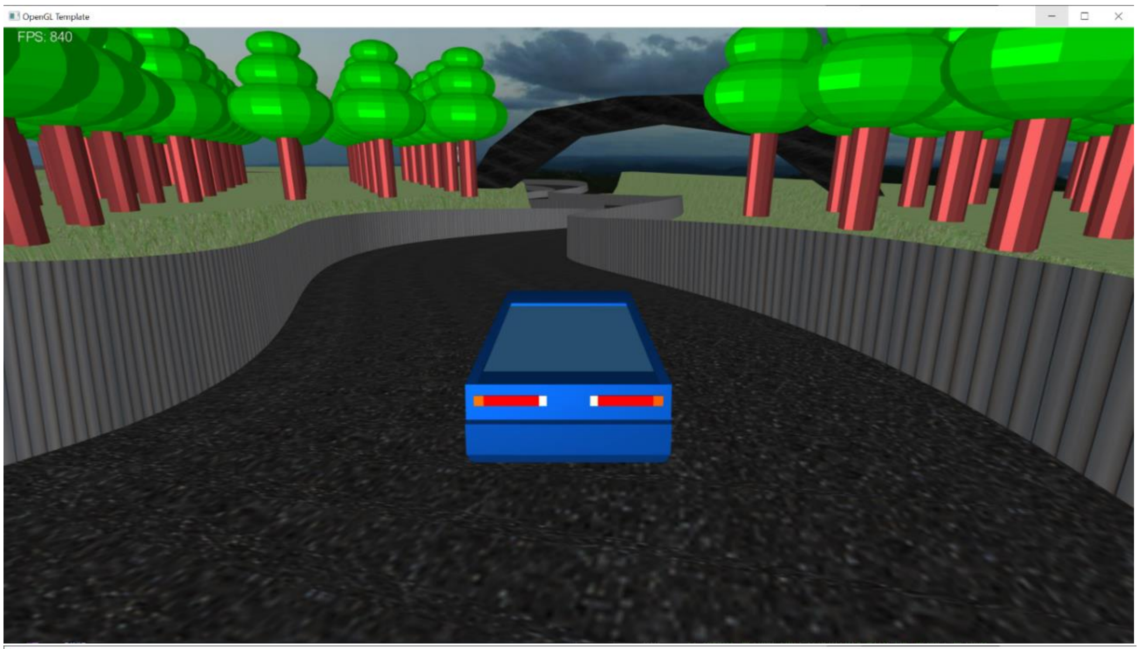
The first mesh I have implemented into my game is a car. This is the car that can be controlled by the player to play the level. The car is placed onto my track by setting its position based on the track's centre line. The first-person camera and third-person camera are then set relative to the car's current position. The car can then be controlled by the WASD keys which can move the car forward and slow the car down and move left to right on the track. This is all done relative to the centreline of the track. To stop the car from leaving the track, I implemented a rudimentary collision system between the car and the barricade. The way this works is by simply checking, how far the car is from the centreline and not allowing it to go any further than a set limit. Lastly, I have set the mesh to rotate when the player presses either 'A' or 'D' to turn the car. This creates a very basic but effective animation to give visual feedback to the player when they are turning.

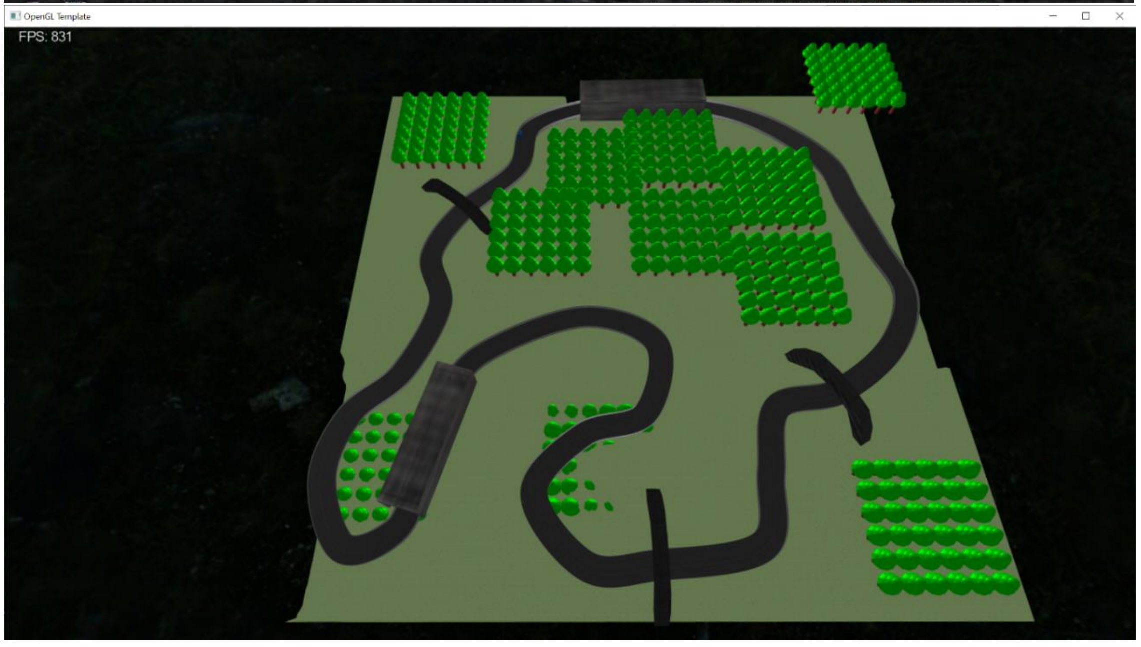
This is what the mesh looks like in the game from the third-person camera:



The second mesh I have implemented is a low poly tree that I used to improve the look of the terrain. Since I wanted to have hundreds of trees in my game, I used instanced rendering. This means all the information about an object and its various positions are sent to the GPU in one go. I will talk more about this in the Advanced Rendering Techniques section of the report.

This is what the game looks like with the new trees:



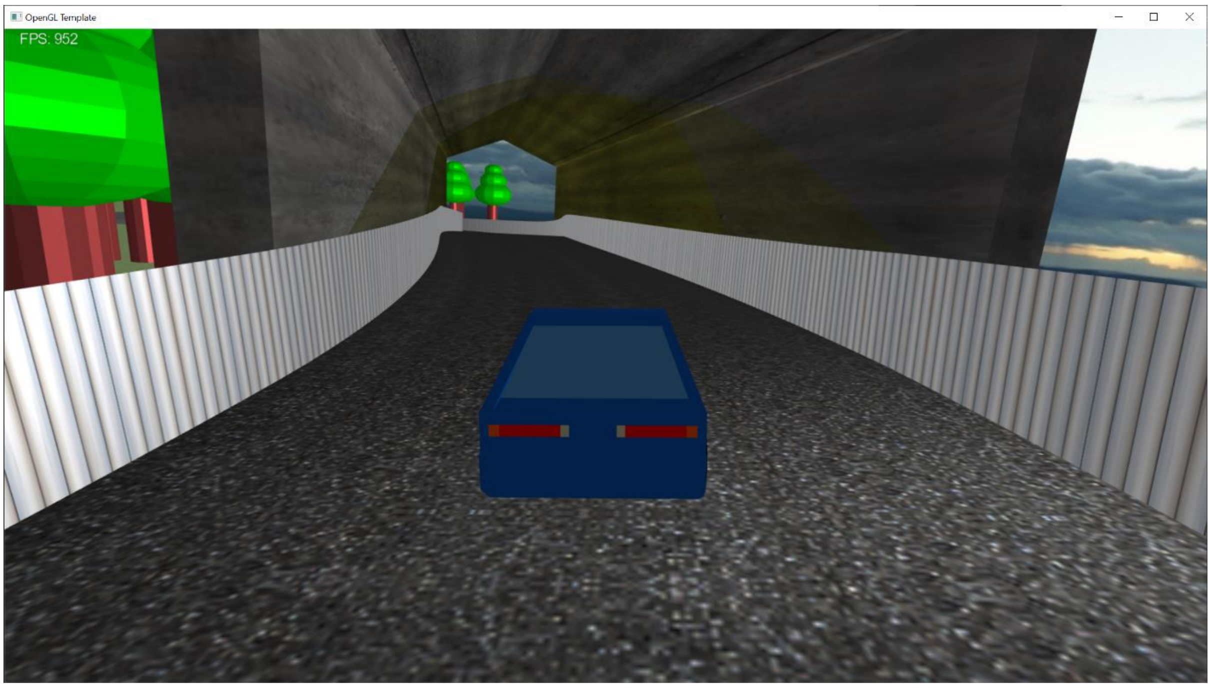


Lighting:

I have implemented headlamps for the player's car for the lighting component of the coursework. I implemented a Blinn Phong spotlight model in my tunnel shader to be able to calculate the lighting colour. One problem I did have initially was that the texture and colour of my tunnels looked good when lit by the spotlight, but they were not very aesthetically pleasing when outside of the spotlight. To fix this I retained the Phong model I initially used and I called the Phong model to work out the colour of a fragment if it wasn't within the spotlight cut-off angle.

Lastly, I used the TNB frame that I previously used to set the car's position and orientation, to set the position and direction of the spotlight.

This is what the result looked like within the tunnel:

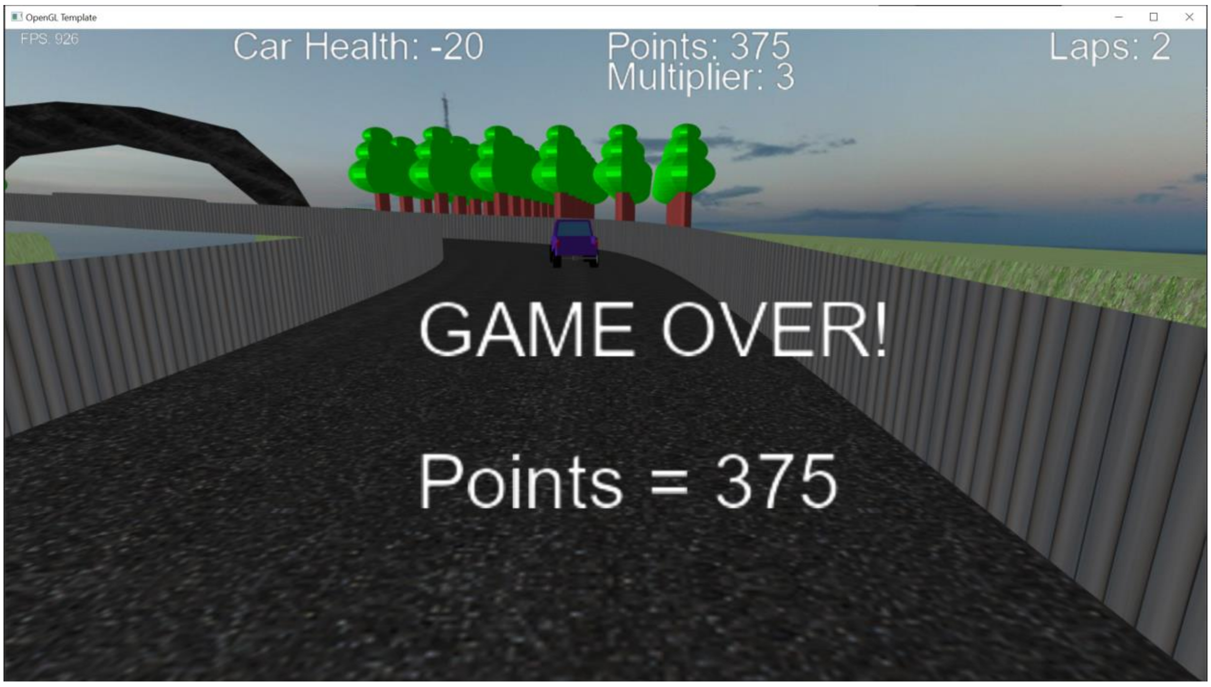


HUD, Gameplay and Advanced Rendering

HUD:

I used the given font class within the template to create a simple HUD for my game. The HUD consists of 4 main components, including points and point multiplier, player health, current lap and lastly a game over screen. The points are governed by the amount of time the player can stay alive and they are multiplied by the total laps the player has completed. The health starts at 100 and goes down by 30 on every collision with another car. The current lap is determined by the corresponding function in the Catmull-Rom class. The game over screen is called when the player's health is at or below 0 and simply stops the points being counted and displays "GAME OVER" with the current number of points.

This is what the HUD looks like:



Gameplay:

The premise of the game is to get around the track as many times as possible while avoiding the other cars on the track, racking up points along the way. To implement this, I first added 4 other cars on the track and I did this the same way I implemented the players' car mesh by setting the positions of the cars relative to the track and their orientation was calculated by a TNB frame for each of the cars. The cars were set at slightly different speeds and at slightly different offsets to the centreline.

The player can move left and right across the track and speed up or slow down to a stop based on keyboard inputs. The player movement is also rudimentarily animated. When the player moves left or right on the track, the player's car rotates to face the direction of the turn. The animation is very simple but it does help the player understand if the car is turning or not and in what direction.

The gameplay is points based. You get points every second while the player's health is above 0. The points are then multiplied by a multiplier. The multiplier is set based on how many laps the player has currently completed. The player can die if they hit too many obstacle cars. This is checked using collision detection.

The collision detection was done by checking the distance between the player car and the obstacle car on every frame, if the distance was below a threshold a collision would be registered. Upon a collision, the player's car would lose health. If the player's health dropped below 1, the player's car would de-spawn and the score would be presented to the player along with a message saying game over.

Advanced Rendering:

I attempted to implement 3 advanced rendering techniques.

The first advanced rendering technique I wanted to implement was instanced rendering for the tree meshes. This was implanted successfully. I chose this technique because I wanted to have hundreds of trees in my scene. Instanced rendering takes all the information about an object and its various positions and sends it to the GPU in one go, which is much quicker than the CPU sending information to the GPU on every render of a tree. However, I did run into a problem with this, due to the way meshes are rendered into the game using a single class(COpenAssetImportMesh). When I attempted to use instanced rendering for my trees, the GPU was also rendering x number of instances of all the objects in my scene. This dropped the frame rate significantly. To get around this, I created a separate class to load the trees, called CTreeMesh. This class is identical to COpenAssetImportMesh but doing it this way allows me to only use instanced rendering for the trees and nothing else, which improved performance.

The second advanced rendering technique I wanted to implement was motion blur. Unfortunately, I could not get this to work. I wanted to have the obstacle cars in my game appear blurry when they moved around in front of the player and I chose motion blur to do this. The way I was attempting to do this was with an FBO or a Frame Buffer Object. An FBO essentially takes the 2d texture of any given frame and stores it for later use. The way I wanted to implement motion blur was by first binding an FBO using the current frame and then rendering the frame again on the next pass. The difference between the 2 images would then be sent into a shader program where the shader would calculate the difference between the 2 images and apply a blur between them to make it look like motion blur. However, I simply could not get this implemented into my game.

The third advanced rendering technique I wanted to implement into my game was shadows by using a shadow map. I was also unsuccessful in implementing this rendering technique. I attempted to use an FBO again for this feature but the FBO for this implementation would only use the depth buffer. The way the shadow map would work is the FBO would be created at the start of the game by capturing a depth buffer from the perspective of the world light in the scene. The depth buffer would only store the closest value to the light in any given direction. This means parts of the scene that were obstructed from the light would not be shown in the depth buffer. This depth buffer could then be used to set a texture on top of objects in the scene based on if they were directly visible to the light or if they were obstructed. Despite spending the majority of a week on this feature, I could not get it working.

Reflection

Overall, I am content with what I have produced during the course of this module. I think the game could definitely have a lot more polish when it comes to things such as the height map and the placement of the trees in the scene. I am also disappointed that the advanced rendering techniques were largely unsuccessfully implemented. But I am really proud of the track and the track barricades. I think the objects and the meshes have been implemented well even if I couldn’t get the tessellation shaders working for the overpass. Also, I think the dynamic lighting with the spotlights and the HUD were good implementations based on the criteria. The overall deliverable looks like a decent first attempt at an OpenGL game and has a lot of room to build on the work already done.

To expand the project into a more complete game or simulation, I think at a bare minimum in terms of advanced rendering techniques, the shadows have to be implemented. It would make the game look so much less flat when playing. And for the C++ side of things, I think the next big step would be to implement a physics engine and have all of the objects in the scene be physics objects so that the movement and collisions especially could be sharper.

Asset Listings:

Tunnel Texture

URL: https://opengameart.org/node/7651

Date of Download: 11/02/2023

License: Creative Commons

Track Texture

URL: https://opengameart.org/content/black-asphalt-tilling-256px

Date of Download: 31/03/2023

License: Creative Commons

Barricade Texture

URL: https://opengameart.org/content/seamless-metal-door-texture

Date of Download: 01/04/2023

License: Creative Commons

Grass Texture

URL: https://opengameart.org/content/grass-texture-0

Date Of Download: 03/04/2023

License: Creative Commons

Skybox

URL: https://opengameart.org/content/mountain-skyboxes

Date of Download: 03/04/2023

License: Creative Commons

Cars

URL: https://opengameart.org/content/vehicles-assets-pt1

Date of Download: 03/04/2023

License: Creative Commons

Overpass Texture

URL: https://opengameart.org/node/49685

Date of Download: 05/04/2023

License: Creative Commons

Tree Model

Url: https://opengameart.org/content/lowpoly-tree

Date of Download: 06/04/2023

License: Creative Commons