INM376: Computer Graphics using OpenGL

Amir Badash

PROJECT REPORT

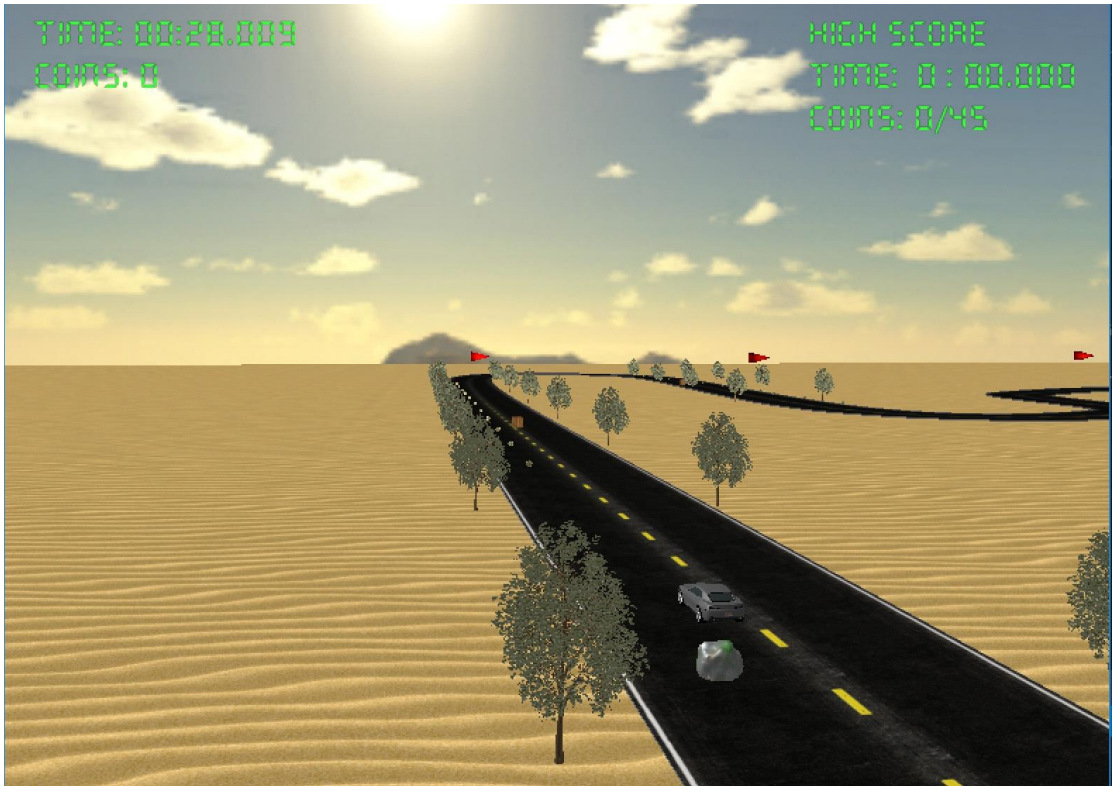
1. Overview
2. Interface controls
3. Implementation
4. References
5. Final considerations

1. Overview

The task was to create a 3D graphics scene using OpenGL that included a non-linear path, different camera views, primitive shapes and some mesh models.

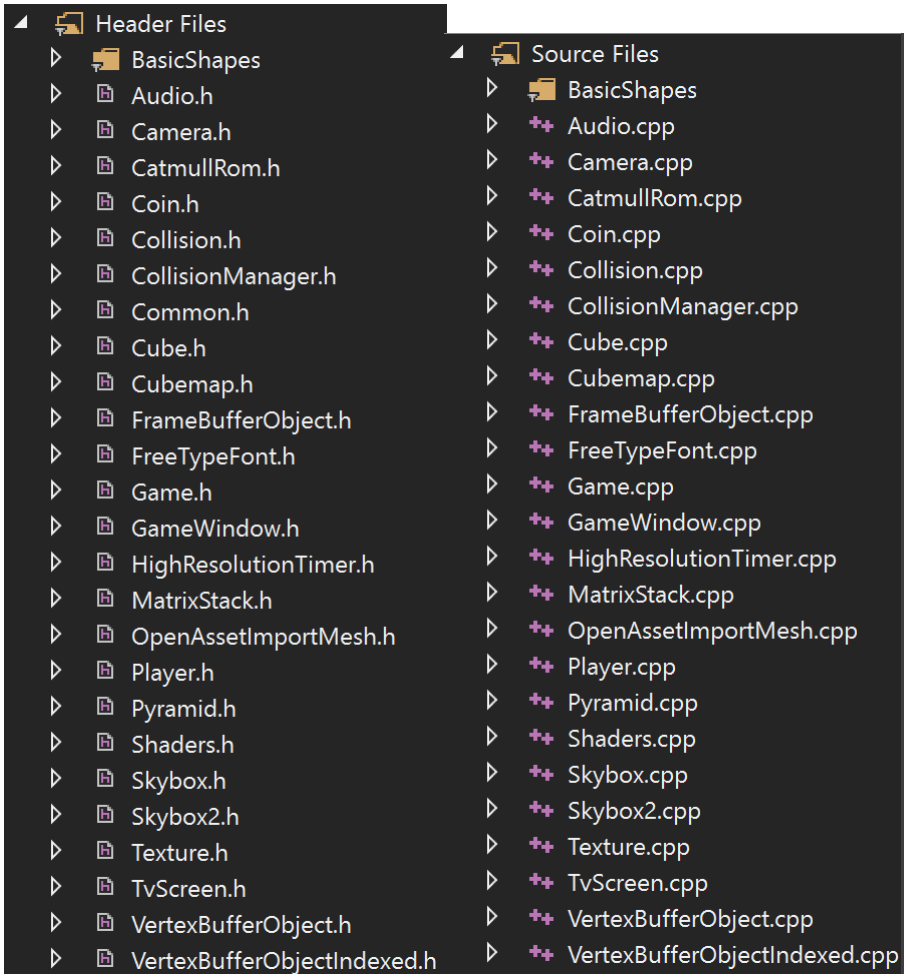
More in detail my game shows a car which moves around the path trying to finish the lap in the shortest time possible.

At the same time the player can collects coins but need to avoid rocks that make him bouncing back or boxes that make him lose coin.



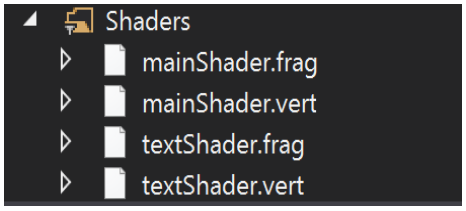
As is possible to see the scene present a car which the player moves around the path, a rock, a box, pyramids that give a hint to the player to where he will turn, some coins and the trees at regular intervals that demarcated the edge of the path.

The project is structured into three folders that contain all the code needed for the application to work properly.



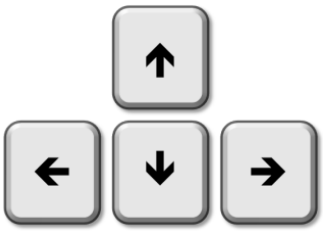
Everything important will be discussed more in detail later on but now we can just see that there are classes (and consequentially header file) for every part of the project such as the player, the coin, the cube, the pyramid, etc. and a main class called "Game" that will include all of them and will execute the program.

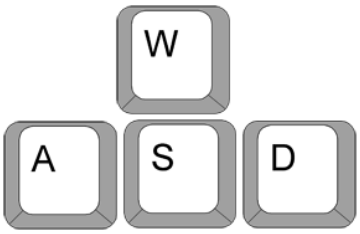
There is also a folder that include all the shader files. More precisely there is are the text shader - which handle all the text that appear into the game - and the main shader - which handle all the lighting part that it will be discussed later on.



2. Interface controls

The game can be played only using the keyboard, more in detail only the arrow keys or the corresponding WASD keys.





As we can see inside the player class, all the movement made by the player is checked with the function *"TranslateByKeyboard"* that controls which button is pressed and acts consequently.

Every move - increase or decrease speed, move left or right - as a control point so that for example the speed cannot by more than 4.0f or less than 0.0f - when the car breaks - and cannot move left or right more than 6.0f.

About the left or right movement there is another control that check if the T vector is positive or negative, so that when the player curves the controls are not flipped.

Other buttons are:

'1' - change to "BackCamera"

'2' - change to "FrontCamera"

'3' - change to "TopCamera"

'4' - change to "FreeCamera"

'6' - add or delete fog

'8' - change to "Night and Spotlight mode"

The only way to use the mouse - that is not useful for the game itself - is when we are using the *"FreeCamera"* mode which allow the player to move around the game with both keyboards and mouse.

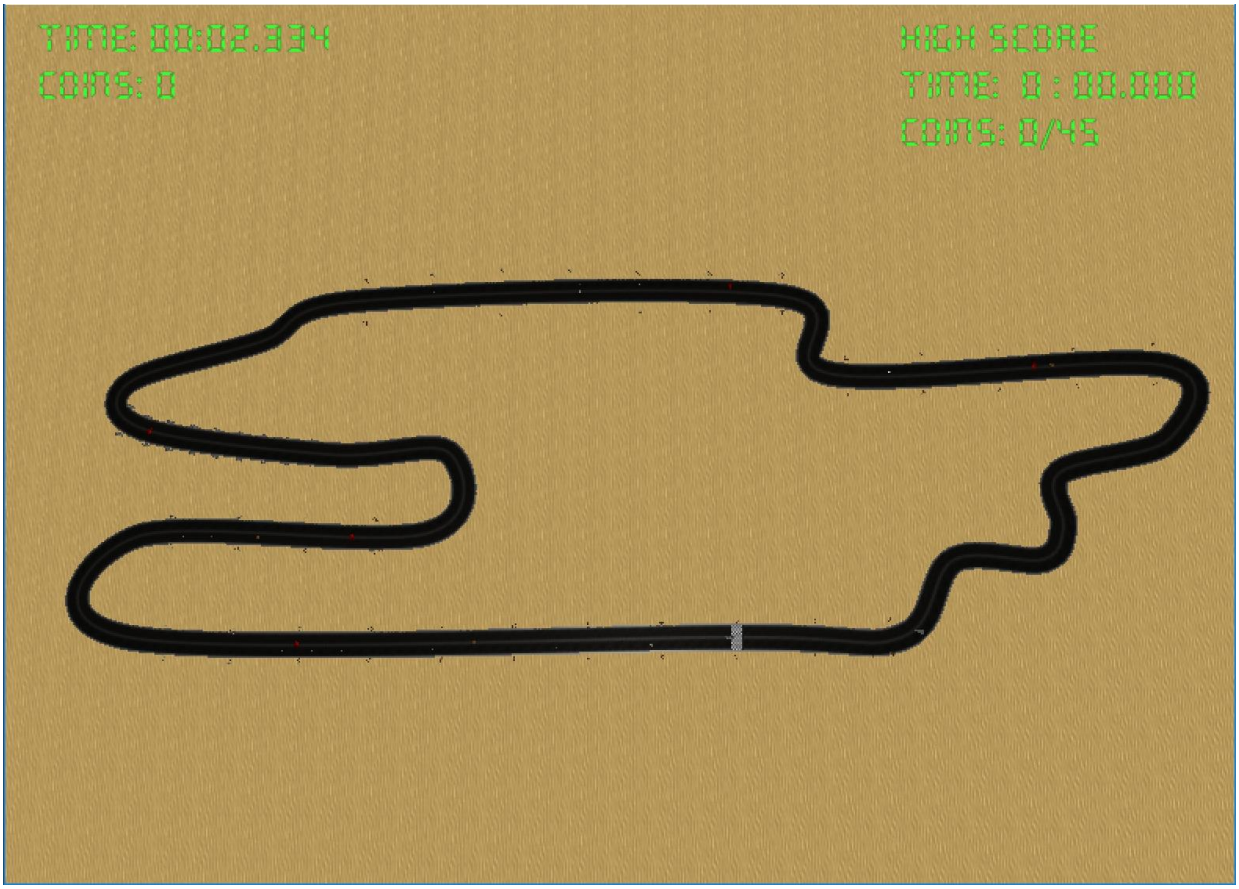
3. Implementation

3.1 Path

The non-linear path based on splines is created by the "*CatmullRom*" class which set the control points, create the centreline, the offset curves and the track. All these three methods are called inside the "*Game*" class - inside the *"Initialise"* method - as well as all the rendering methods (RenderCentreline, RenderOffsetCurves and RenderTrack) - inside the *"RenderScene"* method.

Small mention for the *"CreateWall"* and *"RenderWall"*:
These functions are useful in order to have a wall around both sides of the path, especially when the path goes uphill.

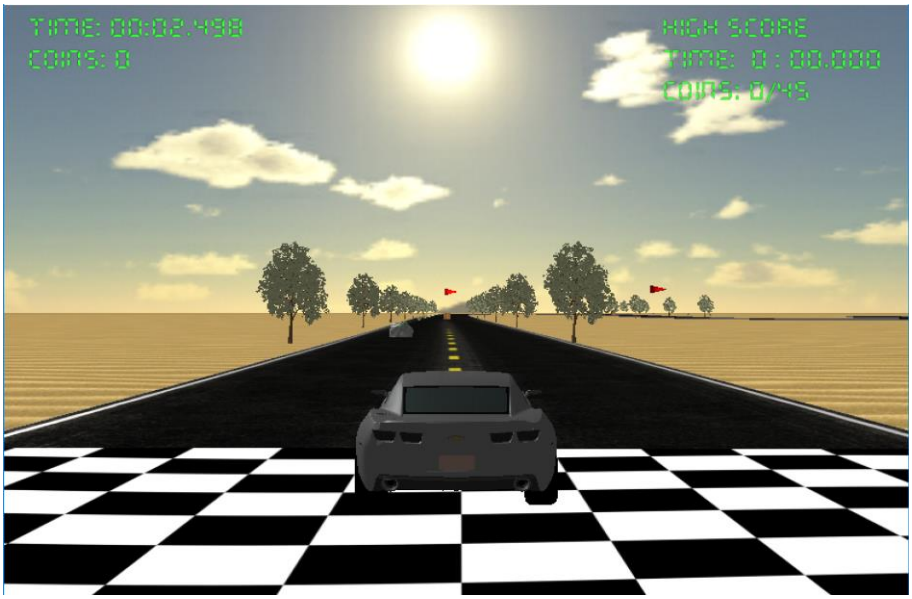
The result as shown in the image below is a street with many curves that it is even going up and consequently down at some point.



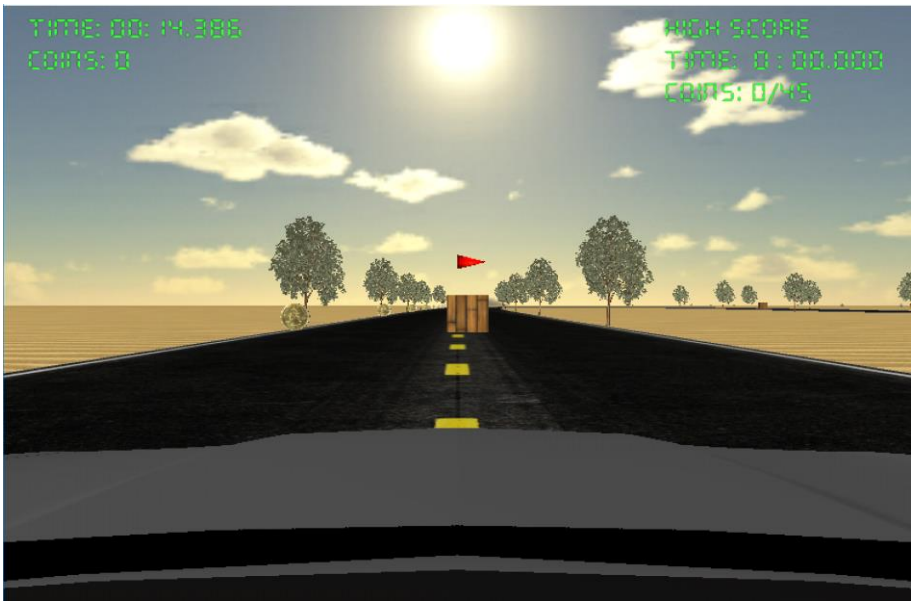
On the sketch picture is possible to see that at the path I added also a texture. More in detail it happened inside the *"CreateTrack"* method where the texture is loaded - and the anisotropic filtering is used - and then binded in the *"RenderTrack"* method.

3.2 Camera(s)

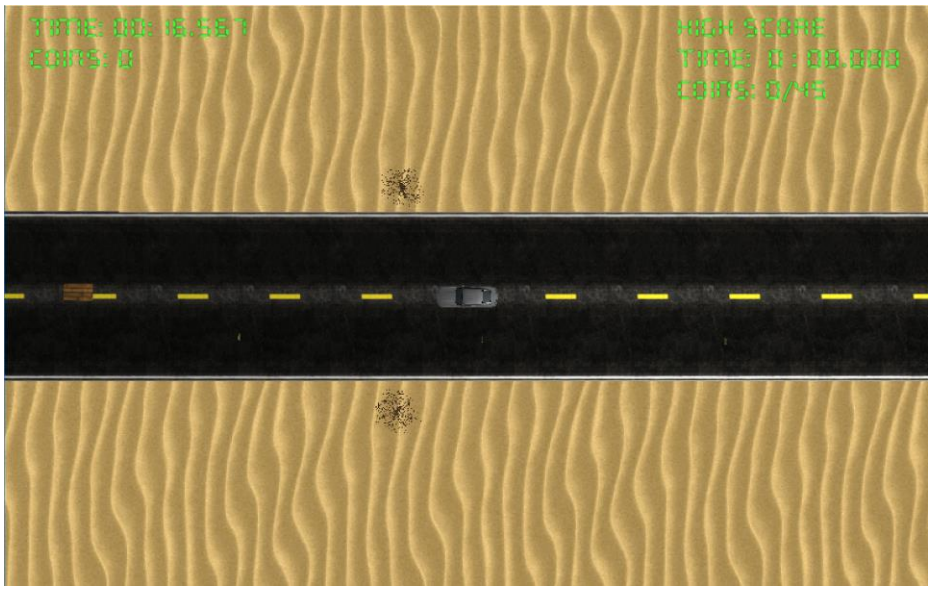
As seen before, with the keyboard buttons '1','2','3' and '4' we can swap from a camera to another.
This is possible due to some Boolean values that are checked into the *"Game::Update"* method continuously and that will set the cameras in the right place using the player position and the TNB frame.



Back Camera



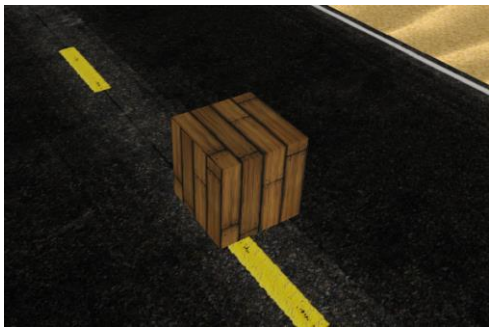
Front Camera



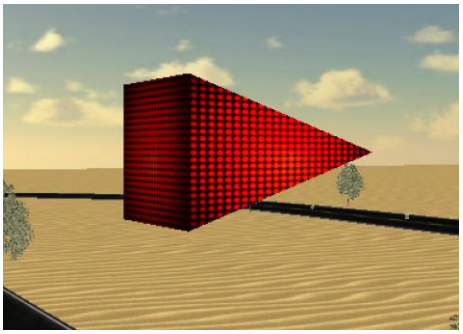
Top Camera

3.3 Basic Objects

The basic object that I used in this project are a cube and a pyramid.



Box

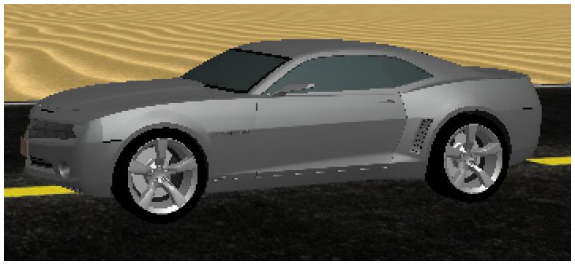


Pyramid

Both have an own class that create the object with the *"Create"* method. The latter add at the VBO all the points in the right position and with the right normal (also for the lights) so that when in the *"RenderScene"* method the primitives GL_TRIANGLE_STRIP - for the square faces - or GL_TRIANGLES are called the objects are perfectly created.

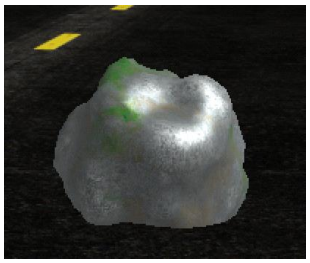
3.4 Meshes

Five different mesh objects are used in this game:



Car

Coin



Rock

Palm Tree

Column

Even if the car and the coins have their own class, all the meshes are loaded in the *"Initialise"* method and rendered inside the *"RenderScene"* method. Apart from the car and the columns, all the other objects are loaded more the once with an easy for-loop. Even though, all the models are transformed, orientated and scaled inside a Model View Matrix Stack.

3.5 Lighting

For the lighting there is just a Boolean variable that is set pressing the button '7' - for the sunlight - or '8' - for the night/spotlights.

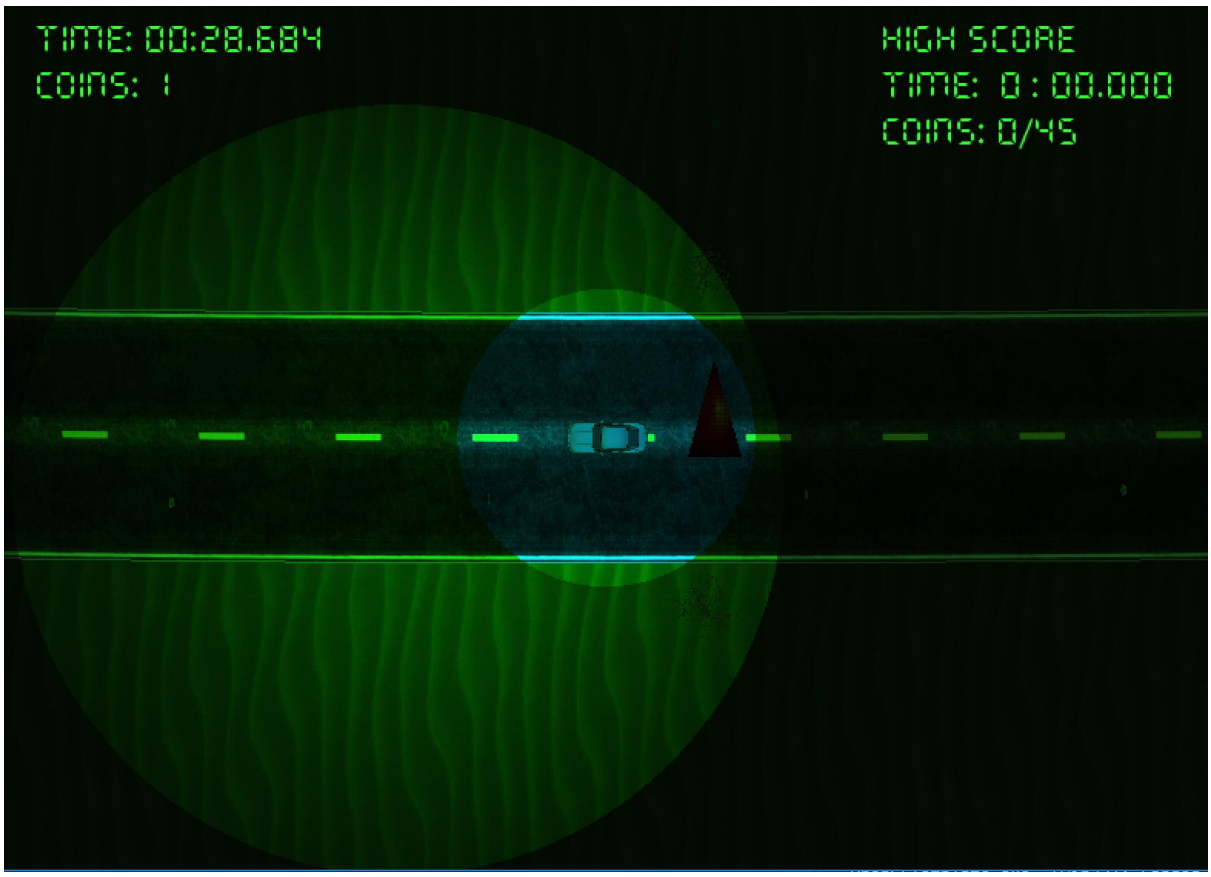
In fact, at the beginning of the "*RenderScene*" method there are two different parts of code that communicate with the main shader:

- The first one set the variable inside the shader "useSpotlight" to false so that the PhongModel function is called and the light set to white at a given position.

- The second one set the variable inside the shader "useSpotlight" to true so that the BlinnPhongSpotlightModel shader method is called and three spotlights are rendered:

1. one blue upon the player,
2. one green still upon the player but bigger and with a different 'z' value,
3. one white as a front lights of the car.

Back Camera – Spotlight ON



Top Camera – Spotlights ON

3.6 Gameplay & HUD

As written in the beginning the game sees a car moved by a player that need to finish a lap as fast as possible and at the same time trying to collects as much coins as he can. Rocks around the path make the car bounce back while if the boxes are hit the player will lose some coins.

Because at the end of every lap the time and the coins are compared with the high score, the aim of the game is obviously to beat the high score - which depends previously by the time and then by the coins.

All the collisions are made by two class, *"Collision"* and *"CollisionManager"*.

The former create a sphere that is added by the latter around every object that need to be interactive, then the collisions are checked continuously. For every collision the tag of the object will determine the right behaviour that will occur during the game.

As shown previously the HUD is very simple:

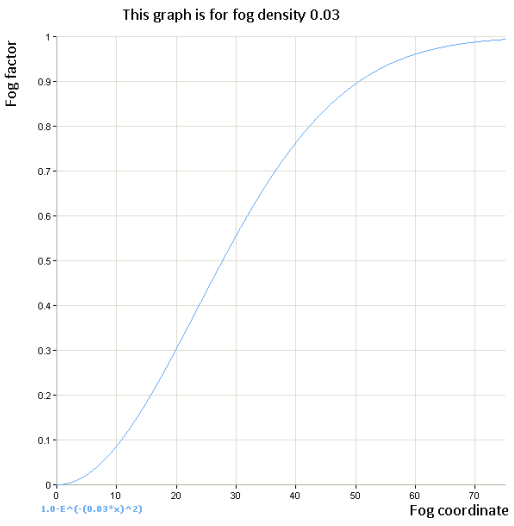
it shows only the timer and the coin's counter on the left side and the update high score on the right side. The font is set at the beginning - *"Game::Initialise"* method - while the colour sent to the text shader and all the controls are inside the *"Game::DisplayFrameRate"* method. Finally, under the high score there is a mini map that shows a top view of the gameplay which is explain later on in the advance rendering.

3.7 Advanced Rendering

As advanced rendering I implemented:

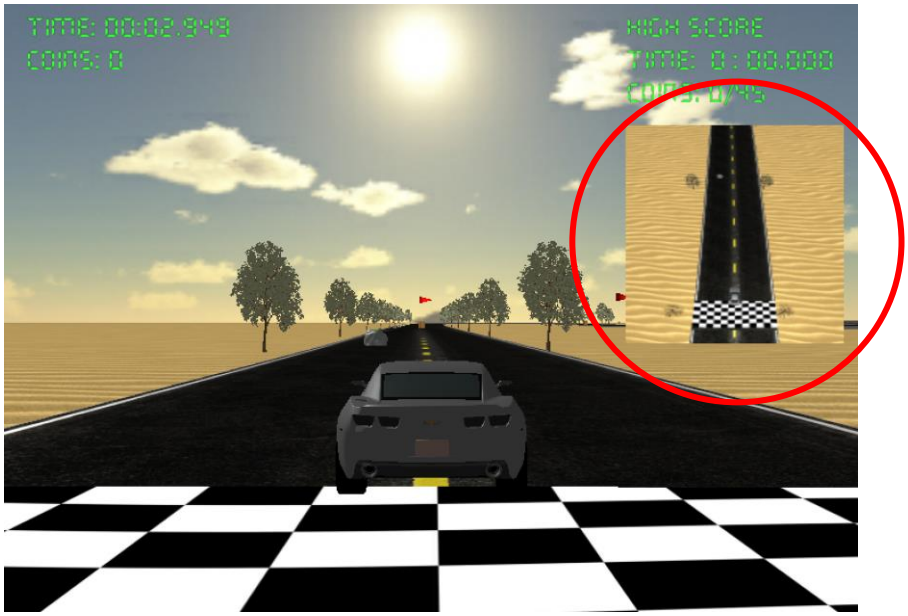
- Camera shaking:
  When the player collides with the boxes there will be a little shaking of the camera made by the method *"RotateViewPoint"* that move the camera around an offset - create by a sin function - with the view that point to the *z* axis. Everything will last for some seconds.

- Fog:
  As for the spotlight, the fog is added in the shader, more precisely in the main fragment shader and call it in the *"Game::RenderScene"* method when the button '6' is pressed – setting the Boolean *"fog"* to true or vice versa.
  If the latter is true, then the output colour are mixed with the fog – created by the function *"getFogFactor"* which use the Exp2 equation in order to put the fog inside the right coordinate as shown on the picture on the right – and its colour set by the client in the Game class.



The image in the next page shows the final result of the HUD when the fog is rendered.

- Render to texture:
  A FBO is created in the *"Initialise"* method and the binded in the *"Render"* method in order to create a mini map on the right side of the HUD.
  To do that the *"RenderScene"* method is called two times:
  The first one – where the *'pass'* value is 0 - is used to create the scene for the mini map. The camera is only positioned upon the player as a top view – it never changes like in the regular scene -, the light is following the player and the fog and spotlights are not used.
  The second one – where the *'pass'* value is 1 – is used to create the regular scene and it add a 2D TV Screen (the mini map) that used the previous rendered scene as texture but only when the camera used are the front one or back one.



4. References

Every object that was downloaded has a *"free use"* or *"personal use only"* license.

Mesh objects:

1 – Car: http://tf3dm.com/3d-model/camaro-2009-99347.html

2 – Palm Tree: http://tf3dm.com/download-page.php?url=palm-tree-66044

3 – Coin: http://tf3dm.com/3d-model/coin-4532.html

4 – Rock: http://www.turbosquid.com/FullPreview/Index.cfm/ID/451171

5 – Column: http://www.turbosquid.com/FullPreview/Index.cfm/ID/333497

Textures:

1 – Desert Sand: http://jb1992.deviantart.com/art/Desert-sand-texture-323267113

2 – Asphalt: http://bgfons.com/upload/asphalt_texture441.jpg

3 – Checked Terrain: https://i.ytimg.com/vi/MzdzE7wwT78/maxresdefault.jpg

4 – Box: http://orig11.deviantart.net/1614/f/2009/353/9/5/wood_study_3_by_devin_busha.jpg

5 – Pyramid: http://m.rgbimg.com/cache1p7S2t/users/i/ia/iammi-z/600/mFgXB1Q.jpg

The Skyboxes were downloaded from this website:

http://www.custommapmakers.org/skyboxes.php

and then converted into jpg file using this website:

http://image.online-convert.com/convert-to-jpg

then finally flipped using this website:

http://flipapicture.com/

The Exp2 function for the fog were download from:

http://www.mbsoftworks.sk/content/tutorials/opengl3_3/articles/12_opengl3_3/exp2FogGraph.png

At the end, the arrow and WASD keys were download from:

http://www.101computing.net/wp/wp-content/uploads/arrowKeys.png

http://s3.amazonaws.com/gifpumper/ui/wasd.png

5. Final Considerations

This kind of project shows how to create a simple 3D game using OpenGL but more important it teach us how to implement everything related to graphics – e.g. splines, meshes, basic objects, lighting, etc.

I think that this project is well structured because almost every part of the game has its own class and the main class (*Game.cpp*) which include all of them, has a good separation of every aspect that are useful for the proper functioning of the game as the initialisation method, the rendering method and the update method – where the last two are called continuously by the "*GameLoop*" function. It also accomplished every requirement, adding in some part even something extra – e.g. three spotlights instead of two, five meshes instead of four, etc.

Obviously there are thing that can be improved and where I have not spent too much time in order to finish what the coursework really asked. For example, I choose to do not make the coins disappear when the player collides with them because the structure of the game is that the coin is just one object that it is rendered 45 times. This means that I cannot use some value on the coin itself to make it disappear after a collision but a value for every representation of it which is quite long and useless. The same happened for the boxes but in that case I set those Boolean value for two main reason: there were only three boxes and to show that I know how it could be done.

In order to expand this project, I think that other cars and a slightly different gameplay like a race competition could entertain more than now. At the same time there are a lot of other different graphic effects that can be added but that I found not completely right or useless for this game idea.