Car Race

Course on Computer Graphics
Andrea Lorenzo Polo

August 6, 2020

1 Introduction

The aim of this project is to recreate a car race inspired by the popular children game **Scalextric**. Specifically, the figure 8-shaped that can be observed in Figure 1. To do so, the video is set in a children room and cars move with random velocities along the circuit. At the bottom right of the screen there is a counter of the loops and the first car completing 5 loops wins.

Figure 1: Scalextric Game

2 Program Structure

The program is implemented in `C++` on `Visual Studio 2019` and `OpenGL` is used through the library `freeglut` with version 3.2.1.. In addition, `SOIL` library is used for working with textures.

For a better understanding of the program, Figure 2 shows a simplified diagram of the structure. Class Main keeps track of the time and works with Scene, Car and Menu classes. The vertex points of Car and Scene are computed only once in the initialization and the result is stored. Therefore, at every display the program only have to load the data but not compute it again. Similarly, the textures are only bound once in the initialization, which makes renderization very fast. To bind the textures, Car and Scene classes make use of

Texture class, which receives an image as input and processes it with SOIL to create a texture.

Car class presents additional features like computing the position at each time and translating and rotating the car, so that the cars move along the defined path. It also allows counting the number of loops each car performs and gets their position at each time.

Finally, Menu class is used to control some video features. It changes the position of the camera along the race as time passes. Also, it displays an introduction text, a counter during the race and the winner car when the race is over.

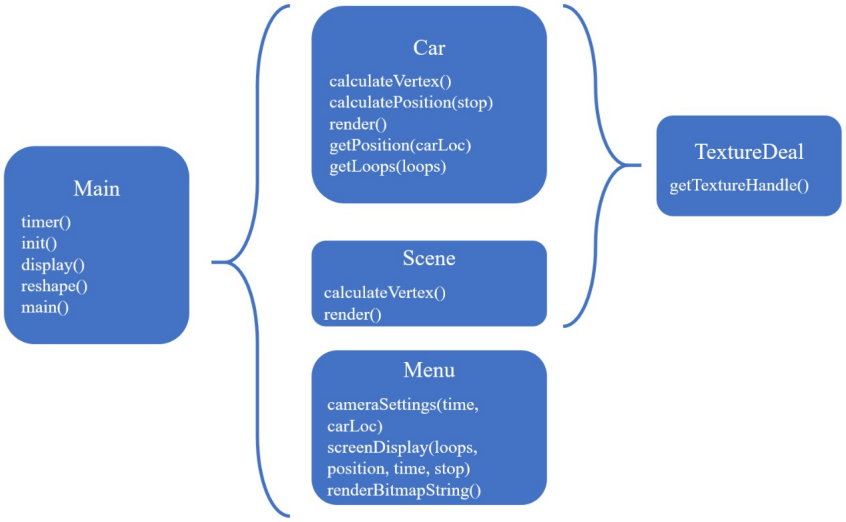

Figure 2: Simple Structure Diagram

3 Key Features

3.1 Scene

To recreate the children's room, the cube map method is used. However, since the project consists of a video and therefore the camera settings are predefined, a ceiling is not needed since is never on screen. The circuit is located at a corner of the room that correspond to the coordinates (0,0,0). The circuit's design consists of a flat path from left to right, the bridge which is comprised of 3 paths (an up ramp, a flat path and a down ramp) and finally two curves that consists of numerous small quads that provide smoothness. The geometry of the circuit is shown in the figures below, which will help understanding the section 3.3. Car Velocities.

3.2 Car Design

The car design is manual and has been done trying to maintain the proportions of real racing cars. Wheels consists of two tori, a small one and a big one. For the body of the car the

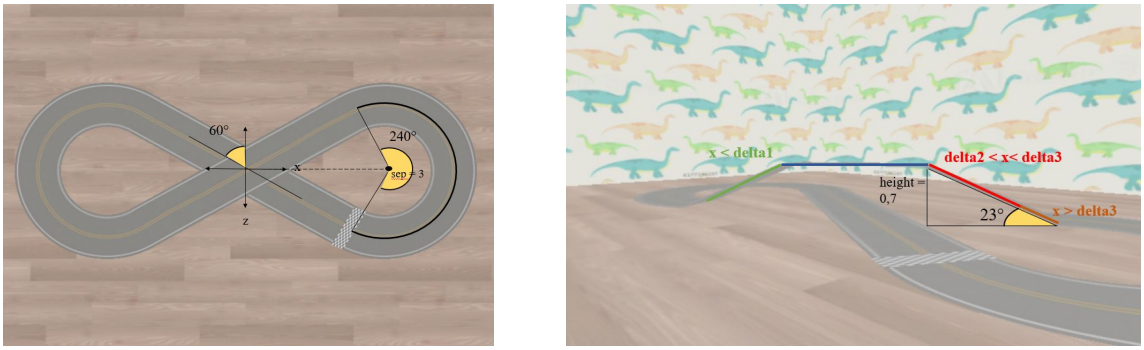

Figure 3: Geometry of the circuit

textures have been designed using PowerPoint. Finally, the windows are blue and have some transparency to make it more realistic.

3.3 Car Velocities

Movements of the car are control in `computePosition()` function. This function divides divides the circuit into 4 paths: 0 is the flat path, 1 is the left curve, 2 is the bridge and 3 is the right curve. For cases 0 and 2 the increase in velocity is the same, and the same happens in cases 1 and 3. The formulas followed at both cases are:

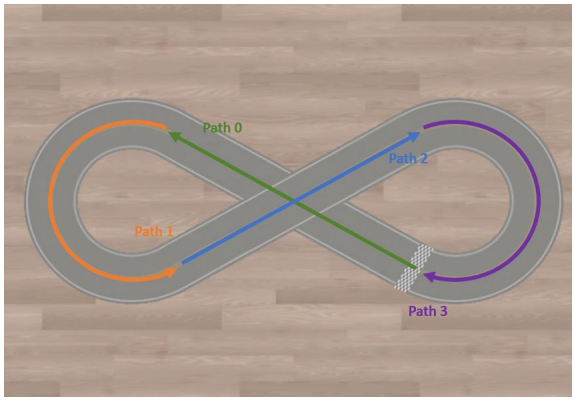

Figure 4: Paths of the circuit

Path 0 . For both terms the starting point is calculated knowing the position corresponds to half of the angle 240º translated by the separation constant in the x axis. Then, d_0 increases with constant velocity, moving with the same orientation of the path (60º + 90º). Note for z the velocity term is negative since the car moves towards -z. Finally, the car is rotate by an angle of -30º.

$$
\begin{aligned}
x &= R\cos(120) + \text{sep} + d_0\cos(150) \\
z &= R\sin(120) - d_0\sin(150) \\
\text{angle}_{x_z} &= -30
\end{aligned}
$$

Path 1 . The starting points are obtained knowing the car will be in the position corresponding to 60º of the circle translated minus the separation in the x axis. However, since for the sin value in z the sign should change, we use -60º. The angle will increase approximately up to +60º, that is the end of Path 1. Note that radius changes from r to R, since the car that was in the exterior part at the starting of Path 0 will be know in the interior of the circle and the other way around. Finally, the car keeps rotating as it moves through the circle, with the same velocity.

$$\begin{aligned} x &= r\cos(-60 - d_1) - \text{sep} \\ z &= r\sin(-60 - d_1) \\ \text{angle}_{x_z} &= -30 + d_1 \end{aligned}$$

Path 2 . Here the principles are the same as in Path 0 but the other way around. The addition is the up and down paths, therefore the position in the y-axis also changes. The path is divided in 4 part that depend in the x-axis: smaller than $delta_1$ when goes up; among $delta_2$ and $delta_3$ when goes down; bigger than $delta_3$ for a smoother transition when going down; and otherwise it is in the flat path at height 0,7. In the same way, the car is also rotated in the y-angle, when going up by -23º, when going down by 23º and finally 10º for the transition part. At the top, that angle is 0º since it goes through the flat path.

$$x = r\cos(60) - \text{sep} + d_2\cos(-30)$$

$$y = \begin{cases} +d_y, & \text{if } x \leq -delta_1 \\ -d_y, & \text{if } x \geq delta_2 \ \&\ x \leq delta_3 \\ -d_y, & \text{if } x \geq delta_3 \\ height, & \text{otherwise} \end{cases}$$

$$z = r\sin(60) + d_2\sin(30)$$

$$\text{angle}_{x_z} = 30 + 180$$

$$\text{angle}_y = \begin{cases} -23, & \text{if } x \leq -delta_1 \\ 23, & \text{if } x \geq delta_2 \ \&\ x \leq delta_3 \\ 10, & \text{if } x \geq delta_3 \\ 0, & \text{otherwise} \end{cases}$$

Path 3 . The equations used in this path are equivalent to the ones used in Path 1. Note we start using R again since the car that was in the outer side for the other curve will now be in the interior one.

$$\begin{aligned} x &= R\cos(240 + d_3) + \text{sep} \\ y &= R\sin(240 + d_3) \\ \text{angle}_{x_z} &= 30 + 180 - d_3 \end{aligned}$$

The values used for this formulas can be taken from the geometry of the circuit by simply looking at it. However, other values like the 23^o of the ramps have been obtained by simple geometric equations. Some of the values showed can be obtained straightforward and others have been obtained after simple mathematical calculations. The velocities, that is, the increase of each d, is 0.05 for d_0 and d_2 and 2^o for d_1 and d_3. This numbers have been obtained so that the rate of movement pixels/frame advanced is similar in the straight paths and the curves.

In order to create an actual race and give different velocities to the cars, these velocities are multiplied by a random number that ranges between 0.6 and 2. This random number is change every time the car changes path so that there are over takings during the race.

4 Conclusions and future work

As a first contact with OpenGL, C++ and 3D Design, the final result meets the targets presented for the submission of the project. However, due to time constraints, there are some aspects that could have been improved:

- The use of a 3D Car object designed with a more suitable application like Blender would make the project more realistic. Although there are several models that can be downloaded from the internet, the process of uploading it to OpenGL is not straightforward and most tutorials use Modern OpenGL.

- There is a small gap when the car in the transition among Path 2 and Path 3. Although the path going down is divided into 2 parts to make the transition smoother, at the very end of the ramp the car is hidden for a few milliseconds.

- Finally, velocities of the cars change every time the car changes path. This is because changing velocities constantly would produce a lumpy movement. More time would help in finding a way to change velocities constantly but in a smoother way.