



**Escuela Técnica Superior de Ingeniería
Universidad de Huelva**

GRADO EN INGENIERÍA INFORMÁTICA

SIMULADOR DEL JUEGO SCALEXTRIC

Realidad Virtual

Autora:
Alba Márquez-Rodríguez

Profesor:
Francisco José Moreno Velo

Huelva, Mayo 2023

Índice

1. Introducción	2
1.1. Scalextric	2
2. Controles	3
2.1. Cámara	3
2.2. Velocidad	3
3. Descripción de Objetos	4
3.1. Descripción de Objetos que desarrollan la pista	4
3.1.1. Recta Estándar	4
3.1.2. Media Recta	5
3.1.3. Cuarto de Recta	5
3.1.4. Curva Estándar	5
3.1.5. Curva Interior	6
3.1.6. Curva Exterior	6
3.2. Implementación	7
4. Descripción de Clases	8
4.1. Descripción de las clases que desarrollan la escena y el circuito	8
5. Descripción de Formas	9
5.1. Descripción de la forma en la que se calcula la posición instantánea de cada vehículo	9
5.2. Descripción de la forma en la que se desarrolla cada modo de visualización	10
6. Pruebas	11
6.1. Captura del inicio del circuito desde la vista panorámica [F1]	11
6.2. Captura de inicio del circuito	12
6.2.1. Captura de inicio del circuito desde la vista de la primera cámara (Coche 1 ,[F2]) . . .	12
6.2.2. Captura de inicio desde la vista de la segunda cámara (Coche 2 ,[F3])	12
6.3. Coche 2 en la luz (Coche 2 ,[F3])	13
6.4. Panorámica del Coche 1 (Coche 1 ,[F4])	13
6.5. Adelantamiento del Coche 1 al Coche 2 (Coche 2)	14
7. Conclusiones	15

1. Introducción

Esta es la memoria del proyecto final de la asignatura de Realidad Virtual [1]. Dicho proyecto es una aplicación gráfica desarrollada en el lenguaje C++ junto con OpenGL. El objetivo del juego es realizar un simulador del clásico juego *Scalextric*. Para ello se deberán diseñar las pistas y construir un circuito basado en ellas. Los coches se crearán aprovechando modelos externos. El juego consiste en controlar la velocidad de los coches y visualizar la carrera desde diferentes puntos de vista [2].

El programa gráfico se ha implementado utilizando las bibliotecas de OpenGL:

- **GLEW**: Permite determinar que extensiones de OpenGL soporta la plataforma de destino.
- **GLFW**: Permite crear y manejar ventanas.
- **GLM**: Permite utilizar funciones y clases para hacer operaciones matemáticas complejas.
- **FreeImage**: Permite utilizar formatos de imágenes clásicos como PNG, JPG, ...

Además hace uso de dos shaders:

- **VertexShader**: se ejecuta sobre cada vértice y genera como salida la posición de cada uno de los vértices en coordenadas CLIP.
- **FragmentShader**: genera el color de cada píxel en la imagen final.

Haciendo uso de estos dos shaders podemos generar efectos como sombras, reflejos de luz o niebla.

1.1. Scalextric

El origen del juego Scalextric se remonta a mediados del siglo XX. La empresa británica Minimodels Ltd. fabricaba desde 1947 reproducciones a escala en hojalata de objetos cotidianos (como máquinas de escribir) y de algunos automóviles de carreras de la época. En 1952, su director, el ingeniero británico Fred Francis, incluyó un mecanismo de impulsión o “motor” en las reproducciones de coches más representativas: el Scalex. Esta palabra compuesta por ”SCALEc” X” viene a significar .^Escaleta desconocida o variableza que cada coche se fabricaba en un tamaño distinto. En 1957 se modificó el diseño para acoplar un pequeño motor eléctrico a los coches y hacerlos correr sobre un sistema de pistas diseñado a propósito. Los motores utilizados por Minimodels procedían de la adaptación de los que utilizaban unos trenes eléctricos que por entonces fabricaba la empresa TRI-ANG. Paralelamente se diseñó una pista en la que se embutían unos carriles con contactos a ambos lados. La palabra inglesa de estos carriles es ”SLOT”. El lanzamiento del nuevo producto supuso la actualización de su nombre comercial, generando la marca actual Scalextric, resultado de la combinación de Scalex y Electric.



Figura 1: Logo de *Scalextric*

El sistema del juego es bastante simple. Los mandos del juego permiten aumentar o disminuir el voltaje aplicado a las piezas metálicas de las ranuras. Los coches disponen de escobillas que ponen en contacto el motor eléctrico con estas piezas metálicas, de manera que al aumentar el voltaje aumenta la velocidad de los vehículos. Teniendo en cuenta la escasa profundidad de las ranuras, cuando los coches entran en una curva a una velocidad excesiva se salen de la pista. La diversión consiste en aumentar la velocidad en las rectas y disminuirlas en las curvas para evitar la salidas de pista, compitiendo con el resto de jugadores en una carrera de velocidad.

2. Controles

Los controles permiten modificar diferentes aspectos del juego, siendo los más relevantes los siguientes:

- **Cámara:** Permite cambiar el modo de visualización para observar el circuito y los coches desde diferentes perspectivas.
- **Velocidad:** Permite controlar la velocidad de los coches.

2.1. Cámara

La cámara en el juego tiene tres modos de visualización diferentes, los cuales se pueden cambiar en cualquier momento durante la ejecución del simulador. Los modos de visualización disponibles son los siguientes:

- **F1:** Este modo sitúa la cámara en una posición fija desde la cual se puede observar todo el circuito. Pretende simular el punto de vista de un jugador de Scalextric, permitiendo una visión general del circuito y los coches.
- **F2:** En este modo, la cámara se coloca detrás del primer coche, a una pequeña altura que permite apreciar tanto el coche como los tramos de pista que se encuentran por delante.
- **F3:** Similar al modo anterior, pero centrado en el segundo coche. Permite seguir la carrera desde la perspectiva de este coche.
- **F4:** En este modo, se libera la cámara y el usuario puede controlar la dirección de la misma utilizando los controles adicionales. Esta opción proporciona mayor flexibilidad y libertad al usuario para explorar la escena desde diferentes ángulos.

2.2. Velocidad

Los controles de velocidad permiten ajustar la velocidad de los coches. Cada coche tiene sus propios controles de aceleración y frenado, y se ha establecido un límite superior para la velocidad de los coches. Algunas características relevantes de los controles de velocidad son las siguientes:

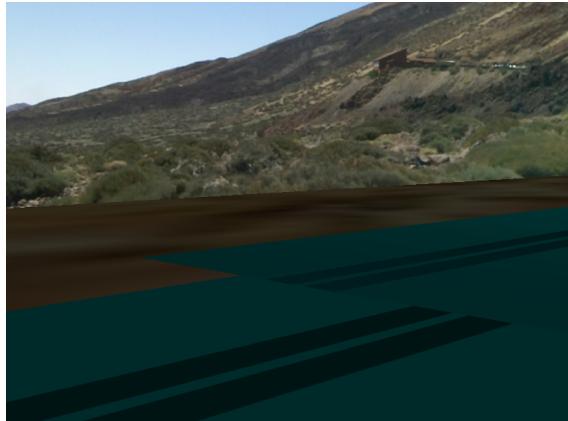
- **Primer coche:**
 - **Q:** Aceleración.
 - **A:** Frenado.
- **Segundo coche:**
 - **O:** Aceleración.
 - **L:** Frenado.
- **Controles generales** (solo en el modo de visualización F4):
 - **Flecha Arriba:** Mueve la dirección hacia arriba.
 - **Flecha Izquierda:** Mueve la dirección hacia la izquierda.
 - **Flecha Derecha:** Mueve la dirección hacia la derecha.
 - **Flecha Abajo:** Mueve la dirección hacia abajo.

Es importante destacar que existen otros controles implementados en el juego, pero no son utilizados en el contexto del simulador de Scalextric.

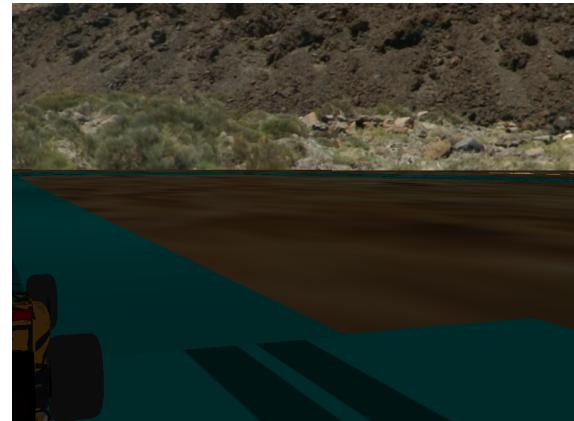
3. Descripción de Objetos

3.1. Descripción de Objetos que desarrollan la pista

Como ejemplo de circuito a desarrollar se propone el circuito de Jerez que se muestra a continuación. El circuito está formado por un total de 31 rectas estándar, 3 medias rectas, 1 cuarto de recta, 8 curvas interiores, 17 curvas estándar y 14 curvas exteriores. Al construir el circuito a base de pistas hay una pequeña desviación que provoca que la ultima pista no coincida exactamente con la pista de comienzo pero esto no debe afectar al funcionamiento de la aplicación.



Pistas inicial y final no coinciden



Pistas inicial y final no coinciden

Las pistas de Scalextric tienen una anchura de 156 mm. Las ranuras de alimentación se encuentran a un cuarto de distancia de cada extremo, es decir, a 39 mm de cada lado.

3.1.1. Recta Estándar

La Recta Estándar es una pieza rectangular con dimensiones de 350 mm de largo y 156 mm de ancho. Esta pieza se genera en el archivo *Scene3D.cpp* y se declara en *Recta.cpp* con los siguientes parámetros:

- **Longitud total:** definida como *LongRecta* con valor de 35 cm.
- **Mitad de la anchura:** 15.6/2.
- **Contador:** utilizado para llevar el orden de construcción de las piezas.
- **Variables (vx1, vx2, vz1, vz2):** coordenadas del plano en las que se representarán las piezas (en este proyecto, las coordenadas X e Z representan el largo y el ancho, respectivamente, y la coordenada Y define la altura).

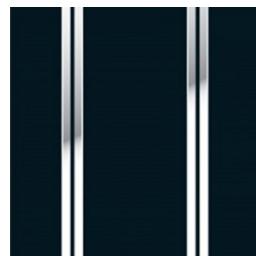


Figura 2: Pieza Recta Estándar

3.1.2. Media Recta

La Media Recta tiene las mismas características que la Recta Estándar, y utiliza su mismo constructor para crearlas, pero su longitud es diferente. Los parámetros de construcción son los siguientes:

- **Longitud total:** definida como *LongMediaRecta* con valor de 17.5 cm.
- **Mitad de la anchura:** 15.6/2.
- **Contador:** utilizado para llevar el orden de construcción de las piezas.
- **Variables (vx1, vx2, vz1, vz2):** coordenadas del plano en las que se representarán las piezas.

3.1.3. Cuarto de Recta

El Cuarto de Recta tiene las mismas características que la Recta Estándar y utiliza su mismo constructor para crearlas, pero su longitud es diferente. Los parámetros de construcción son los siguientes:

- **Longitud total:** definida como *LongMediaRecta* con valor de 8.75 cm.
- **Mitad de la anchura:** 15.6/2.
- **Contador:** utilizado para llevar el orden de construcción de las piezas.
- **Variables (vx1, vx2, vz1, vz2):** coordenadas del plano en las que se representarán las piezas.

3.1.4. Curva Estándar

La Curva Estándar tiene dimensiones definidas con un radio interior de 214 mm, un radio exterior de 370 mm y un ángulo de 45°. Para crearla, se utiliza el constructor declarado en *CurvaStd.cpp*, que tiene los siguientes parámetros:

- **Número de capas.**
- **Número de sectores.**
- **Radio interior:** 2.14 cm.
- **Radio exterior:** 3.70 cm.
- **Contador:** utilizado para llevar el orden de construcción de las piezas.
- **Variables (vx1, vx2, vz1, vz2):** coordenadas del plano en las que se representarán las piezas.



Figura 3: Pieza Curva Estándar

3.1.5. Curva Interior

La Curva Interior está declarada en *CurvaInterior.cpp* y tiene dimensiones definidas con un radio interior de 58 mm, un radio exterior de 214 mm y un ángulo de 45°. Los parámetros de construcción son los siguientes:

- **Número de capas.**
- **Número de sectores.**
- **Radio interior:** 0.58 cm.
- **Radio exterior:** 2.14 cm.
- **Contador:** utilizado para llevar el orden de construcción de las piezas.
- **Variables (vx1, vx2, vz1, vz2):** coordenadas del plano en las que se representarán las piezas.



Figura 4: Pieza Curva Interior

3.1.6. Curva Exterior

La Curva Exterior está declarada en *CurvaExterior.cpp* y tiene dimensiones definidas con un radio interior de 370 mm, un radio exterior de 526 mm y un ángulo de 22.5°. Los parámetros de construcción son los siguientes:

- **Número de capas.**
- **Número de sectores.**
- **Radio interior:** 3.70 cm.
- **Radio exterior:** 5.26 cm.
- **Contador:** utilizado para llevar el orden de construcción de las piezas.
- **Variables (vx1, vx2, vz1, vz2):** coordenadas del plano en las que se representarán las piezas.



Figura 5: Pieza Curva Exterior

3.2. Implementación

En lugar de aplicar las texturas de las curvas, se ha aplicado la textura de la recta a cada triángulo de la curva. Al aplicarla triángulo a triángulo, da la sensación de curva con la textura de recta.

Todos los cálculos se encuentran en los respectivos constructores al aplicar las texturas en:

- *Pieza_CurvaStd.cpp*
- *Pieza_CurvaInt.cpp*
- *Pieza_CurvaExt.cpp*

Por lo tanto, en sus constructores se realiza una llamada a los constructores de "Pieza_" donde se les asigna la textura *RectaStd.png*.

4. Descripción de Clases

4.1. Descripción de las clases que desarrollan la escena y el circuito

La clase `CGScene` se encarga de crear los objetos que componen la escena del circuito, el circuito se encuentra dentro de esta clase. A continuación se describen las clases principales utilizadas en esta clase:

- `CGGround`: Esta clase genera el terreno sobre el cual se mueve el coche. Se crea un plano horizontal y se asigna un material con luces especulares, difusas y ambientales. Además, se utiliza una textura de piedras (`stone.jpg`) para el terreno.
- `CGSkybox`: Esta clase se encarga de generar el skybox de la escena. Un skybox es un cubo que rodea el escenario y se utiliza para mostrar una imagen de fondo y dar la sensación de un entorno más realista. Se deben asociar las coordenadas de cada cara del cubo con las imágenes del skybox descargadas. En este caso, se ha seleccionado un skybox del Teide.



Vista Izquierda



Vista de Frente



Vista Derecha



Vista de atrás

- `CGModel`: Esta es la clase principal que llama a las demás clases y genera la simulación del Scalextric. Realiza una llamada a la clase `CGScene` para crear la escena del circuito.
- `CGLight`: Objeto que representa la luz del juego. Está compuesta por luz difusa, luz specular y luz ambiental.

En la clase `CGScene`, el proceso de construcción de la escena se realiza de la siguiente manera:

1. **Inicializar el Skybox**: Se crea el skybox de la escena, asignando las imágenes del skybox descargadas a las coordenadas adecuadas.
2. **Inicializar el Ground**: Se genera el terreno del circuito utilizando la clase `CGGround`. Se crea un plano horizontal y se le asigna un material con luces especulares, difusas y ambientales, así como una textura de piedras.
3. **Inicializar las Luces**: Se crean las luces de la escena, incluyendo la luz specular, la luz ambiental y la luz difusa.
4. **Array pista**: Se crea un array llamado "pista" que define el orden de las pistas del circuito. Cada elemento del array tiene la forma `x, y`, donde:

- **x**: Representa el tipo de pista, que puede ser:

- Recta Estándar (1)
- Media Recta (2)
- Cuarto de Recta (3)
- Curva Interior (4)
- Curva Exterior (5)
- Curva Estándar (6)

- **y**: Indica el sentido de la curva

- Giro a la izquierda
- Giro a la derecha

5. Descripción de Formas

5.1. Descripción de la forma en la que se calcula la posición instantánea de cada vehículo

Para el estudio de la forma en la que se calcula la posición instantánea de cada vehículo hay que estudiar la clase *CGModel*, concretamente la función *Update()* que se ejecuta cada 0,2segundos.

Al haber dos coches hay dos subvariables de cada variable, así para la variable Coche está *Coche1* y *Coche2*. A continuación se nombran las diferentes variables y se describen.

Coche1 y *Coche2* que son la representación de cada coche dentro del circuito, a los que se le aplicaran los cambios de posición y orientación para generar los movimientos de estos.

Las variables *Dist_Recorrida1* y *Dist_Recorrida2* almacenan la distancia recorrida de cada coche.

Para los coches están *Dist_CocheX1* y *Dist_CocheX2* son las coordenadas *X* de los coches y *Dist_CocheZ1* y *Dist_CocheZ2* son las coordenadas *Z* de los coches.

También está *velocidad* es la velocidad con las que ambos coches se van a desplazar por el circuito.

Y *Indice_1* e *Indice_2* almacenan en que pista se sitúan cada uno de los coches.

Para el movimiento de los coches se arrastra la distancia recorrida que se inicializa a 0 para que vaya incrementando con respecto a la velocidad que tiene. Se comienza por la pista 1 y el coche se va desplazando hasta que alcanza la distancia máxima de esa pista. Cuando se termina la pista actual reinicia la distancia recorrida a 0 y se incrementa el índice para saber que se ha desplazado a la siguiente pista, y así hasta que llega al final.

Ambos coches tienen las mismas variables y realizan lo mismo pero tienen una posición inicial diferente, así en el caso de las curvas se alteran su giro y su desplazamiento. Digamos que hay dos radios, el exterior y el interior. El coche que se encuentre en el radio exterior tendrá una curva de desplazamiento mayor y una rotación menor mientras que el que se sitúe en el radio interior tendrá un desplazamiento menor y una rotación mayor.

Por esta mecánica hay que estudiar el tipo de pista en el que se encuentra el coche, ya que la longitud no es la misma para una recta que para una curva ni para los diferentes tipos de rectas ni curvas. Por ejemplo, para las rectas la longitud cambia por lo que cambia de pista antes o después. Por otra parte las curvas también varían con la curvatura y nos encontramos con la misma variación que hay que estudiar y tener en cuenta.

```
1 // si es recta
2 if (Pistas[Indice_1][0] == 3 || Pistas[Indice_1][0] == 2 || Pistas[Indice_1][0] == 1){
3     if (Pistas[Indice_1][0] == 1){
4         if (velocidad < 7 && velocidad != 0) velocidad++;
5         GLdouble dx = 0, dz = 0;
6         Dist_CocheX1 = Dist_CocheX1 + Dist_Recorrida1;
7
8         if (Dist_CocheX1 <= LongRecta){
9             Coche1->Translate(glm::vec3(0, -Dist_Recorrida1, 0));
10            posBeg -= glm::vec3(0, -Dist_Recorrida1, 0);
11        }
12    } else{
13        Dist_Recorrida1 = 0, Dist_CocheX1 = 0; Indice_1++;
14        Coche1->Translate(glm::vec3(0, -Dist_Recorrida1, 0));
15        posBeg -= glm::vec3(0, -Dist_Recorrida1, 0);
16    }
17 }
18 if (Pistas[Indice_1][0] == 2){
19     ...
}
```

Listing 1: Extracto del código que calcula según el tipo de pista

5.2. Descripción de la forma en la que se desarrolla cada modo de visualización

La aplicación a desarrollar debe incluir tres modos de visualización. El primero consiste en situar la cámara en una posición fija desde la que se pueda observar todo el circuito. Este modo de visualización pretende simular el punto de vista de un jugador de Scalextric. El segundo modo de visualización consiste en situar la cámara detrás del primer coche, a una pequeña altura que permita apreciar el coche y los tramos de pista que se encuentren delante. El tercer modo de visualización es similar al anterior pero centrado en el segundo coche.

Tal y como se ha indicado en el enunciado, se han incorporado 3 modos de visualización:

- **F1:** situar la cámara en una posición fija desde la que se pueda observar todo el circuito. Este modo de visualización pretende simular el punto de vista de un jugador de Scalextric.
- **F2:** situar la cámara detrás del primer coche, a una pequeña altura que permita apreciar el coche y los tramos de pista que se encuentren delante.
- **F3:** similar al anterior pero centrado en el segundo coche.
- **F4:** a partir de la cámara anterior, en la que se encontraba, se libera y se puede controlar su dirección.

Para las dos visualizaciones de los coches se ha puesto una altura más elevada al configurarlas para evitar tener un punto de visión a la altura del suelo. También se ha retrasado la cámara respecto a la posición del coche para así tener una visión del coche.

En el código para las vistas se emplea la variable auxiliar *selección* que dentro de la función *Update()* realiza el seguimiento y desplaza con la asociación a la cámara dependiendo del valor que tenga esta variable auxiliar que indica el coche a seguir. A la cámara se le asocia la matriz de seguimiento *matseg* para que en cada iteración se desplace a la vez que el coche seleccionado. Esta selección se realiza con pulsar sus respectivas teclas que solo cambian el valor de selección a 1, 2 o 3.

Por ello en el caso de la cámara panorámica del circuito al ser el valor de selección distinto de 1 y 2 la cámara se mantiene fija durante cualquier iteración del circuito.

Por lo tanto solo tenemos una cámara a la que le vamos asociando distinto objetivos de seguimiento o una manera estática (Visión Panorámica del circuito).

6. Pruebas

Para las pruebas de funcionamiento se mostrarán capturas del estado final de la simulación en diferentes etapas y formas.

6.1. Captura del inicio del circuito desde la vista panorámica [F1]

Como podemos observar en esta captura diferenciamos el circuito (conjunto de todas las pistas), el ground de gravilla puesto y el skybox elegido por detrás.



Figura 6: Imagen al inicio desde la vista panorámica

Podemos observar la generación de una sombra en la esquina del circuito. Esto se debe a una incorrecta generación de los shaders.

Además, se pueden observar los coches sobre la pista. Pero al tamaño de los coches y los colores no permiten diferenciarlos bien, haciendo un poco de zoom se pueden ver mejor:



Figura 7: Coche sobre la pista

6.2. Captura de inicio del circuito

6.2.1. Captura de inicio del circuito desde la vista de la primera cámara (Coche 1 ,[F2])

Como se ha explicado en la imagen 7 se genera una sombra incorrecta. Esta sombra afecta al inicio del circuito por lo que los coches comienzan en una zona a la sombra.



Figura 8: Imagen al inicio desde el coche 1

6.2.2. Captura de inicio desde la vista de la segunda cámara (Coche 2 ,[F3])



Figura 9: Imagen al inicio desde el coche 2

6.3. Coche 2 en la luz (Coche 2 ,[F3])

En la imagen 7 se puede observar que se genera una sombra sobre el circuito, esta sombra coincide con el inicio por lo que los coches se ven en sombra. Los coches iluminados quedan de la siguiente manera:



Figura 10: Coche 2 en la luz

6.4. Panorámica del Coche 1 (Coche 1 ,[F4])



Panorámica del coche 1 trasera



Panorámica del coche 1 delantera

6.5. Adelantamiento del Coche 1 al Coche 2 (Coche 2)

En teoría, si los dos coches van a la misma velocidad, al tener el Coche 1 una curvatura de radio menor, debe tomar la ventaja tras alguna curva. Es lo que ocurre en la primera curva, donde el Coche 1 adelanta al coche 2:



Coche 1 adelantando al Coche 2 desde el Coche 2



Coche 1 adelantando al Coche 2 panorámica



Figura 11: Coche 1 adelantando al Coche 2: Caso 2

Aún así, el Coche 2 puede también podrían adelantar al Coche 1 en otra curva en donde la curva sea exterior para el Coche 1 e interior para el Coche 2. Pero si el Coche 1 ya ha adelantado varias veces al Coche 2 entonces deben haber varias curvas interiores para el Coche 2 consecutivas. También dependerá de la curvatura de las curvas acumuladas para cada Coche.

Por último para observar que todos los cálculos conllevan a un error al redondear o truncar decimales, los coches no siempre se mantienen en el centro del ralí, pero si se mantienen dentro del circuito. Además a mayor velocidad mas error se cometen en los cálculos. (Esto podría llegar a simular la realidad puesto que a mayor velocidad mas posibilidad de descarrilarse).

7. Conclusiones

En este proyecto, nos hemos enfrentado al desafío de desarrollar un programa gráfico utilizando C++ con diversas librerías de OpenGL. El proceso ha consistido en diferentes fases y etapas, desde cargar un skybox a generando las pistas del Scalextric. También se ha importado un modelo externo, en este caso el del coche, utilizando un parser de .obj a C++.

La creación de gráficos utilizando OpenGL y librerías como FreeImage y GLEW ha sido un desafío por la falta de experiencia previa en el desarrollo de gráficos en la GPU. La complejidad y desconocimiento de este lenguaje de programación y del entorno de desarrollo han sido la primera barrera a superar.

La dificultad es aún mayor al intentar generar un modelo de *Scalextric* y llevar a cabo los cálculos matemáticos necesarios para la creación de la pista y la simulación del movimiento de los vehículos. La necesidad de precisión en los cálculos y el dominio de conceptos geométricos implicados para conseguir un seguimiento correcto de la pista han sido una de las mayores dificultades del proyecto. Esto se puede observar en los resultados finales en los que finalmente no se ha podido terminar de solucionar ya que el redondeo de los cálculos crea un error que se va acumulando y a partir de cierta acumulación se hace muy evidente la desviación de los coches.

Por otro lado, la posibilidad de partir de código ya existente proveniente de las prácticas de la asignatura ha facilitado la programación. Ya que muchas funciones estaban ya creadas y se han podido mantener intactas. Mientras que otras han tenido que ser modificadas sin tener ningún otro código en el que basarse. Las prácticas utilizadas para este proyecto han sido la práctica 6 [3] y la práctica 11 de sombras [4].

De la misma manera, el proyecto ha tenido otras partes más sencillas como la generación de los modelos de los vehículos que se ha simplificado gracias al empleo del Parser Object, una herramienta que automatiza este proceso generando a partir de un .obj un objeto de C++. Esto ha permitido una creación rápida, eficiente y correcta de los modelos, reduciendo tanto el tiempo como el esfuerzo requeridos y consiguiendo resultados óptimos.

Sin embargo, tras haber terminado el primer objetivo del proyecto que es un *Scalextric* que genere los gráficos correctamente y que actúe como tal, se procedió a la creación de sombras, donde se presentaron desafíos adicionales al modificar el código con el fin de implementar los shaders.

A partir de las explicaciones teóricas de la asignatura sobre el sombreado y la práctica 11 se pudieron sombras, pero al tener que modificar diferentes partes del código que ya funcionaba correctamente, surgieron errores. A pesar de estas dificultades, se logró finalmente generar los shaders de manera correcta. Aún así, hay una sombra generada de forma incorrecta lo que no permite conseguir un resultado realmente satisfactorio.

En resumen, la creación de gráficos mediante OpenGL ha sido un desafío por la inexperiencia en la programación gráfica en la GPU. La creación de un modelo de *Scalextric* y los cálculos para la construcción de la pista y la simulación del movimiento de los vehículos han sido de las tareas más complejas. Aún así, han habido recursos que han facilitado el proyecto como las prácticas de la asignatura o el uso del Parser Object. Pero finalmente se han logrado superar estos desafíos y obtener resultados satisfactorios.

Referencias

- [1] Francisco José Moreno Velo, “Realidad Virtual (GII-RV).” http://www.uhu.es/francisco.moreno/gii_rv/. Accedido en junio de 2023.
- [2] Francisco José Moreno Velo, “Realidad Virtual (GII-RV) - Trabajo.” http://www.uhu.es/francisco.moreno/gii_rv/trabajo/trabajo.htm. Accedido en junio de 2023.
- [3] Francisco José Moreno Velo, “Realidad Virtual (GII-RV) - Práctica 6.” http://www.uhu.es/francisco.moreno/gii_rv/practicas/practica06.htm. Accedido en junio de 2023.
- [4] Francisco José Moreno Velo, “Realidad Virtual (GII-RV) - Práctica 11.” http://www.uhu.es/francisco.moreno/gii_rv/practicas/practica11.htm. Accedido en junio de 2023.