Imagen que contiene alimentos, taza

Descripción generada automáticamenteImagen que contiene dibujo, alimentos, taza

Descripción generada automáticamente

UNIVERSIDAD NACIONAL AUTÓNOMA DE MÉXICO

FACULTAD DE INGENIERÍA

DIVISIÓN DE INGENIERÍA ELÉCTRICA

INGENIERÍA EN COMPUTACIÓN

LABORATORIO DE COMPUTACIÓN GRÁFICA e

INTERACCIÓN HUMANO COMPUTADORA

**REPORTE DE PRÁCTICA Nº**: 05

**NOMBRE COMPLETO:** Lugo Manzano Rodrigo

**Nº de Cuenta:** 320206968

**GRUPO DE LABORATORIO:** 03

**GRUPO DE TEORÍA:** 02

**SEMESTRE:** 2026-1

**FECHA DE ENTREGA LÍMITE:** 28/septiembre/2025

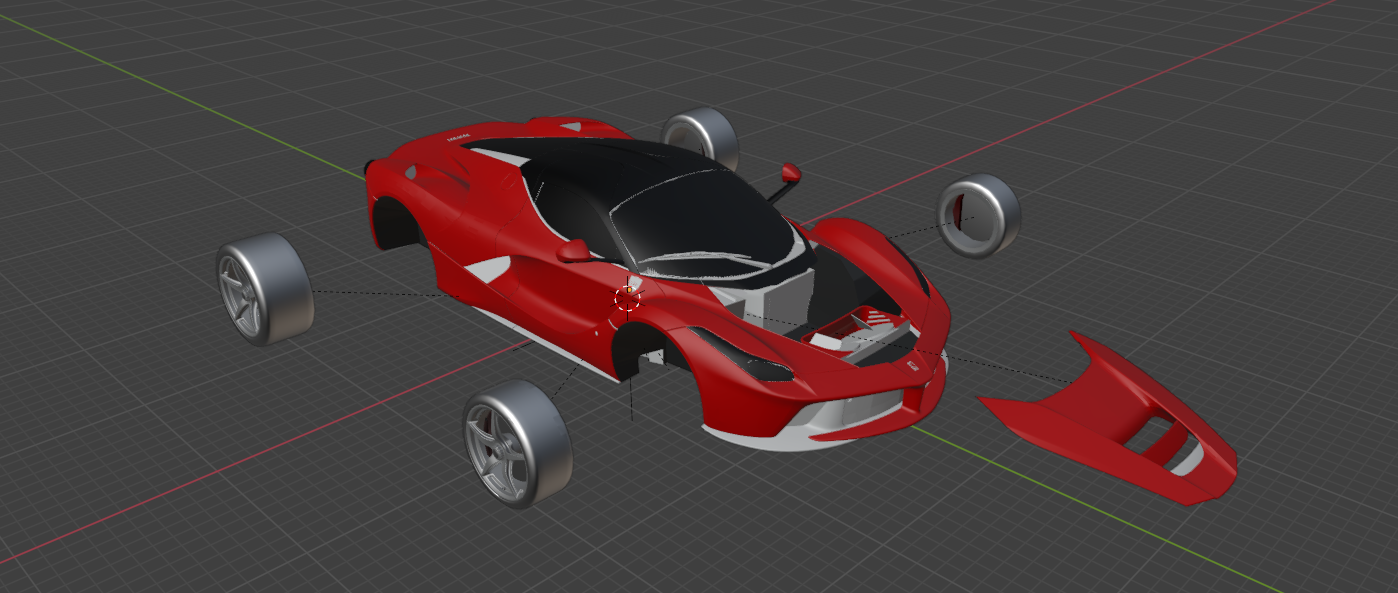
**CALIFICACIÓN: \_\_\_\_\_\_\_\_**

**Reporte de Practica No. 5**

***Modelado jerárquico***

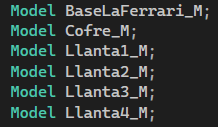
**1.- Ejecución de los ejercicios que se dejaron, comentar cada uno y capturas de pantalla de bloques de código generados y de ejecución del programa.**

***1.- Importar su modelo de coche propio dentro del escenario a una escala adecuada.***



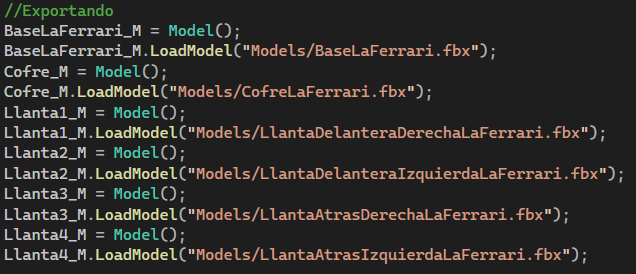
**Ilustración 1** Modelo auto LaFerrari "desarticulado"

Para esto lo primero que hicimos fue importar nuestro modelo en Blender del auto solicitado en el cuestionario previo 3. A partir de este lo que hicimos fue desarticularlo para poder tener la base del auto, las llantas y el cofre por separado, ya nos servirá para los ejercicios posteriores. En este caso los importamos en formato .FBX, igual que el original que descargamos.



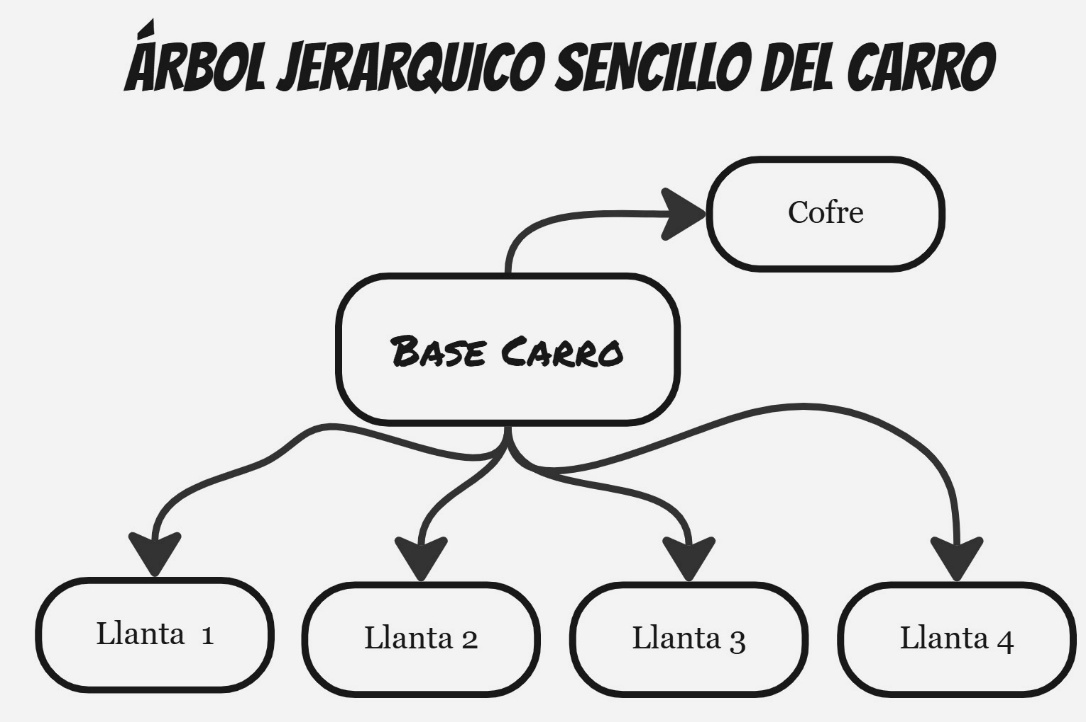
**Ilustración 2** Instancias de Model necesarias para el carro

Ya en OpenGL / C++ primero se definen seis instancias de Model que representan cada subensamblaje del coche, una para la carrocería o base del auto, otra para el cofre y otras para cada una de las cuatro llantas. Cada Model encapsula la malla, materiales/texturas y los buffers de GPU asociados al FBX correspondiente.



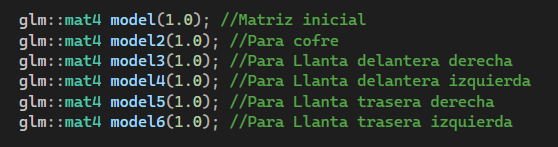
**Ilustración 3** Exportación de modelos

Después lo que hacemos es crear 6 objetos Model, cada uno con su propio archivo FBX para cargarlo en memoria. Aquí lo que hacemos es preparar los recursos para que, más adelante, se pueda dibujar cada pieza con su matriz model correspondiente.



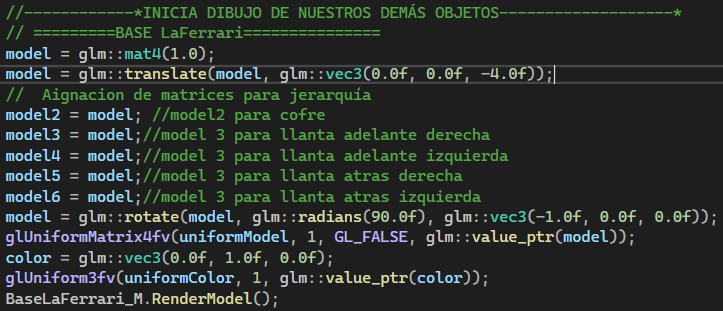
**Ilustración 4** Arbol jerárquico sencillo del auto

Este fue un boceto previo que se realizó de cómo iba a quedar la jerarquía de cada parte del carro, a partir de él también se identificaron las matrices necesarias para poder llevar a cabo el dibujado de este, cada matriz es una matriz de transformación de modelo, es decir, la que define posición, rotación y escala de cada parte del auto en el espacio 3D y se ocuparon las que se mencionan a continuación.



**Ilustración 5** Matrices para aplicar la jerarquía

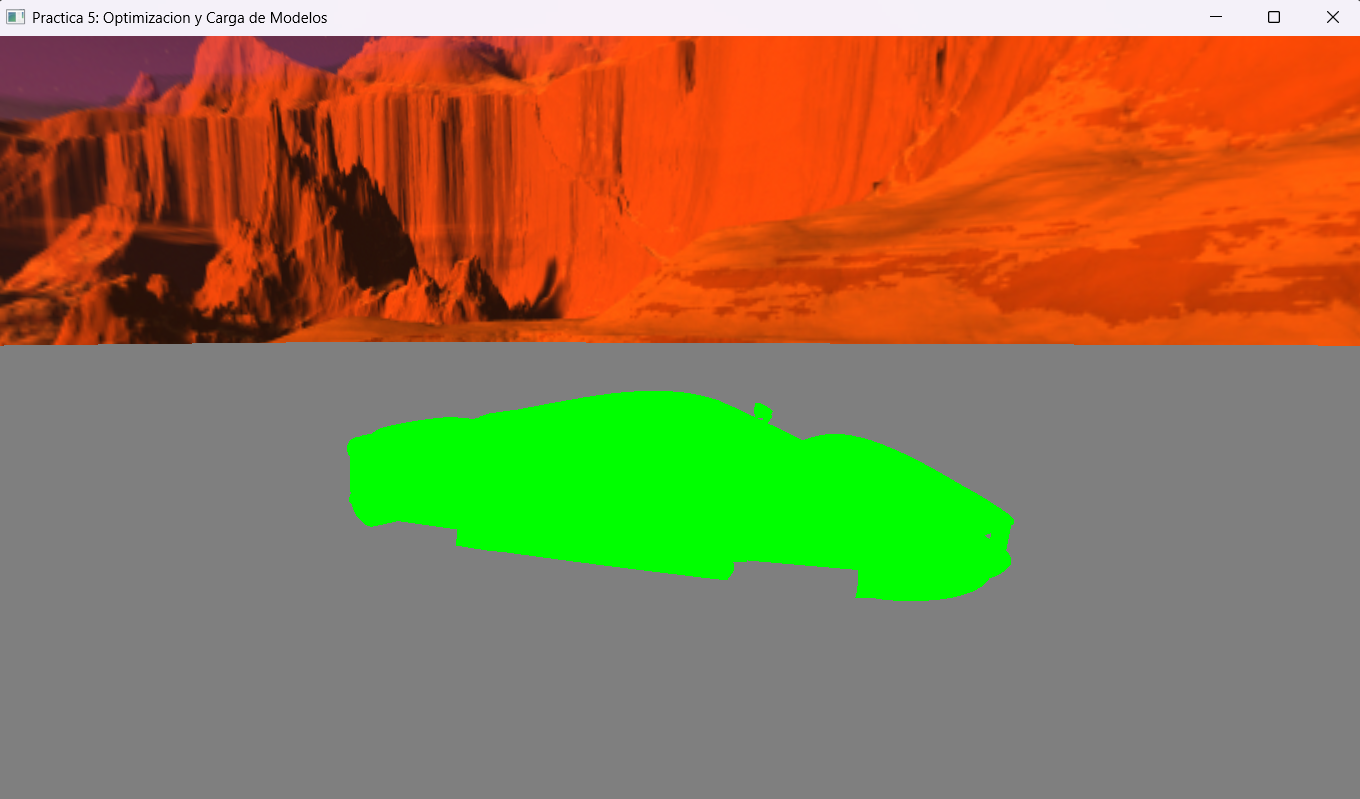
* Primero cabe recalclar que con glm::mat4(1.0) creamos la matriz identidad, que es punto de partida limpio para componer transformaciones.
* La matriz **model** es la matriz padre o nodo central de la jerarquía, la usamos principalmente para iniciar con la base del coche chasis y de ahí empezar a derivar las otras partes.
* De las matrices **model2 a model6** son matrices independientes por pieza (cofre y llantas), son las matrices hijas que empiezan copiando model para heredar la transformación de la base del auto y luego se les aplica su transformación local.
* Tener una matriz por pieza nos evita acumulaciones indeseadas y permite organización y separar cofre y rodado de llantas sin afectar a las demás partes.



**Ilustración 6** Código para dibujar la base del auto

Este bloque de código lo que hace es dibujar nuestra base del auto en la escena. Primero reiniciamos la matriz del chasis a identidad, luego colocamos a nuestra base del auto a cierta distancia de la cámara trasladándolo a z=−4. Aprovechamos para colocar la copia de esta transformación global ya construida, en las matrices hijas model2 a model6 para que el cofre y las cuatro llantas hereden exactamente la misma posición del chasis y se muevan con él.

Posteriormente como se tuvo un problema al exportar los modelos, ya que, aunque cambiáramos lo ejes de referencia el coche aparecía siempre en una posición que hacía verlo como si estuviera rotado aplicamos al chasis un giro de 90° alrededor del eje X para corregir la orientación del FBX respecto al sistema de ejes del render. Finalmente, enviamos esa matriz y le asignamos color verde al shader para distinguirla mejor y dibujamos la carrocería base con BaseLaFerrari\_M.RenderModel().



**Ilustración 7** Base del auto en la escena 3D

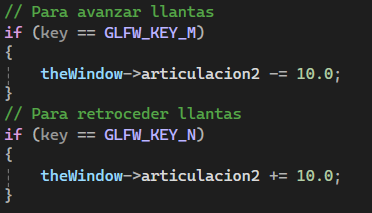
***2.- Importar sus 4 llantas y acomodarlas jerárquicamente, agregar el mismo valor de rotación a las llantas para que al presionar puedan rotar hacia adelante y hacia atrás.***



**Ilustración 8** Dibujado jerárquico de las 4 llantas

Ese bloque de código dibuja las cuatro llantas siguiendo el mismo patrón.

* Primero para cada una se parte de la matriz copiada del chasis la cual ya incluye la traslación global.
* Luego aplicamos una traslación local para colocarla en su posición correspondiente y se realiza el rodado con una rotación por articulacion2 alrededor del eje Z (todas comparten ese valor para girar hacia adelante o atrás al mismo tiempo).
* Como mencionamos se tuvo el problema con los ejes de referencia al exportar el modelo, por ello añadimos una rotación fija de 90° en Y para alinear el eje del FBX con el de la escena.
* Por último, enviamos la matriz al shader, le otorgamos el color rojo para distinguir mejor las llantas y se renderiza la malla de la llanta. Gracias a iniciar desde la matriz del chasis, todas heredan el movimiento del coche y solo su giro de cada una es independiente.

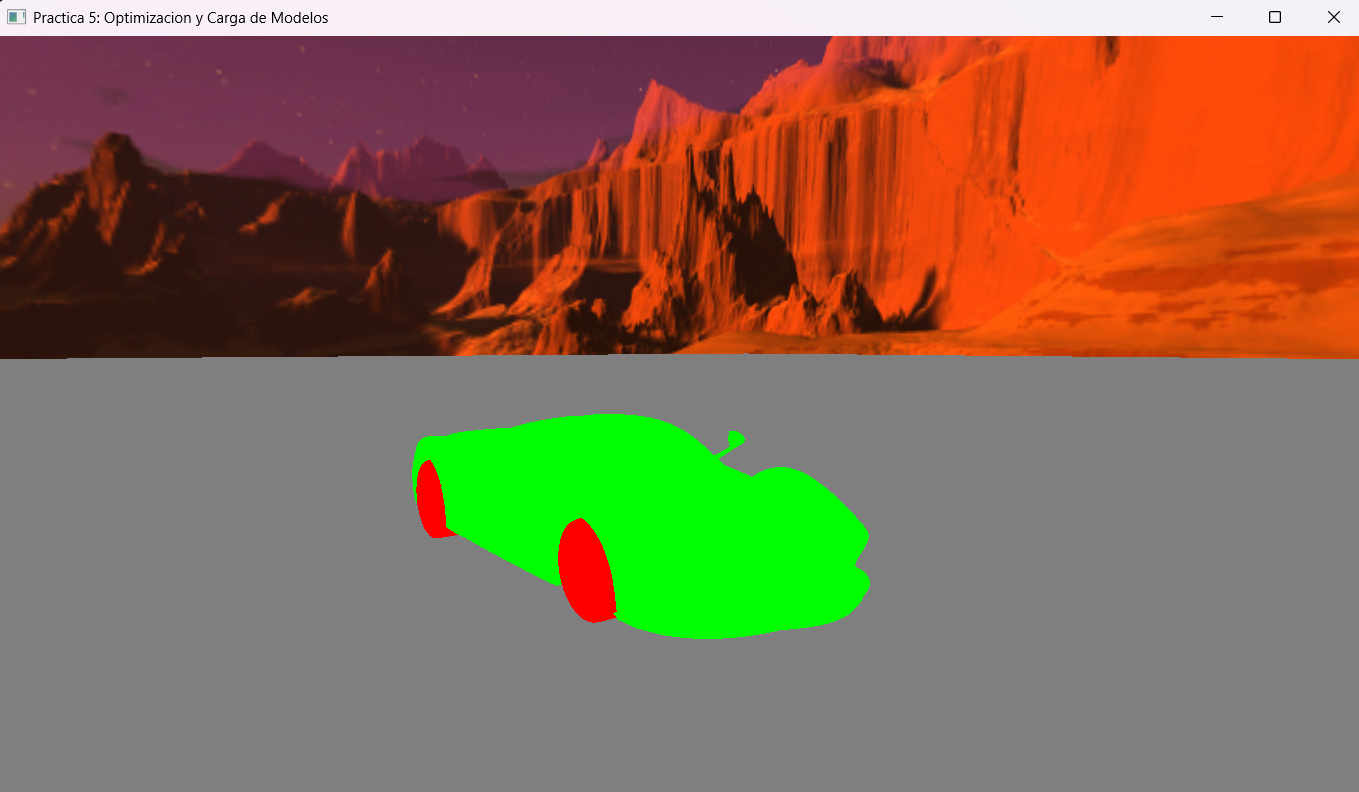


**Ilustración 9** Teclas para rotación de llantas

Para poder realizar el movimiento en las llantas en el archivo Window.cpp primero definimos las letras, la articulación y el valor que iban a ocupar, en este caso se ocupan las letras M y N:

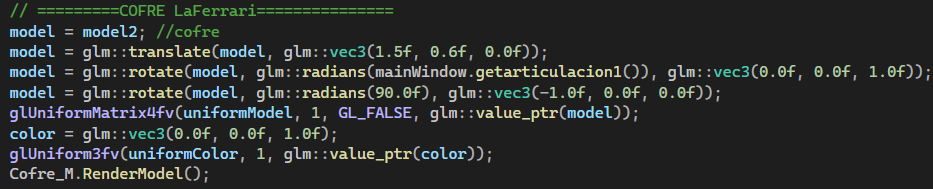
* **M:** disminuye articulacion2 en 10°, haciendo girar todas las llantas en un sentido de avanzar.
* **N:** aumenta articulacion2 en 10°, girándolas en el sentido contrario, es decir, para retroceder.

Como articulacion2 se usa en el glm::rotate(..., articulacion2, vec3(0,0,1)) de cada llanta, las cuatro comparten el mismo ángulo y giran sincronizadas.



**Ilustración 10** Base del auto y llantas en la escena 3D

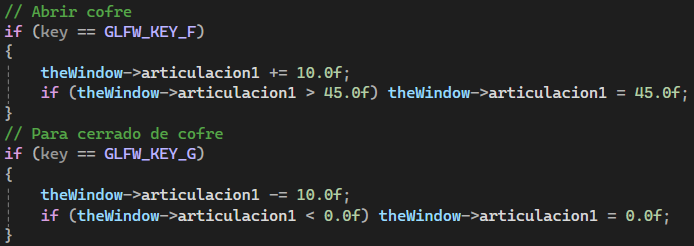
***3.- Importar el cofre del coche, acomodarlo jerárquicamente y agregar la rotación para poder abrir y cerrar.***



**Ilustración 11** Dibujado del cofre

Hacer esto es muy similar a lo anteriormente explicado con las llantas, con este bloque dibujamos el cofre con jerarquía y su articulación, partimos de model2 que ya hereda la transformación global del chasis, y lo trasladamos hasta la posición deseada.

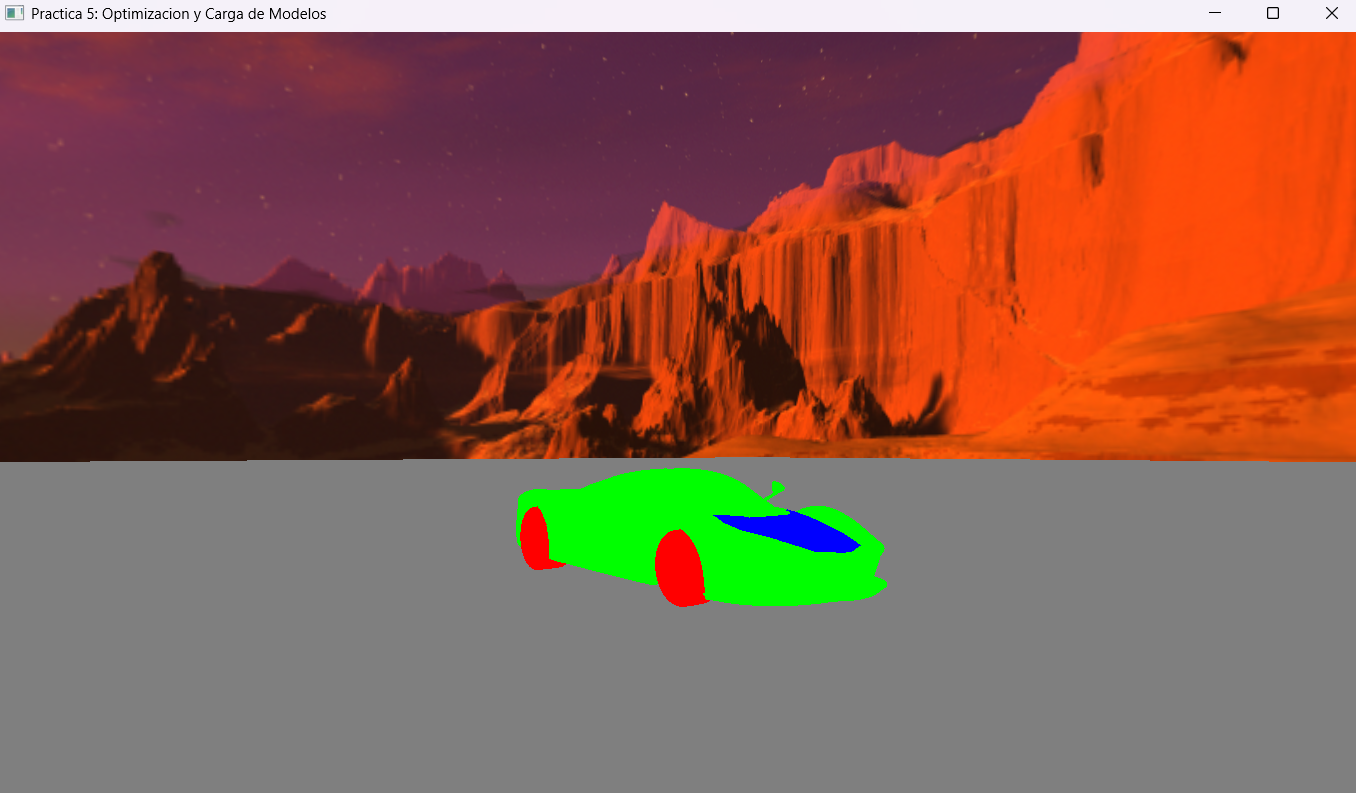
Luego aplicamos la rotación de apertura con el ángulo articulacion1 alrededor del eje Z, y hacemos un ajuste fijo de orientación de 90° en X para alinear el FBX con la escena. Por último, enviamos la matriz y le otorgamos color azul al shader para distinguir el cofre y renderizamos la malla del cofre con Cofre\_M.RenderModel().



**Ilustración 12** Teclas para rotación de cofre

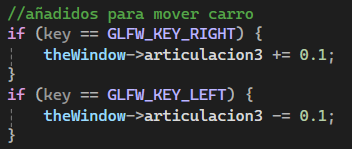
Con este fragmento asignamos las teclas para abrir/cerrar el cofre controlando la articulacion1:

* **F:** aumenta articulacion1 en 10° hasta un máximo de 45° que simula el cofre abierto).
* **G:** reduce articulacion1 en 10° hasta un mínimo de 0° para cerrar el cofre.



**Ilustración 13** Base del auto, llantas y cofre en la escena 3D

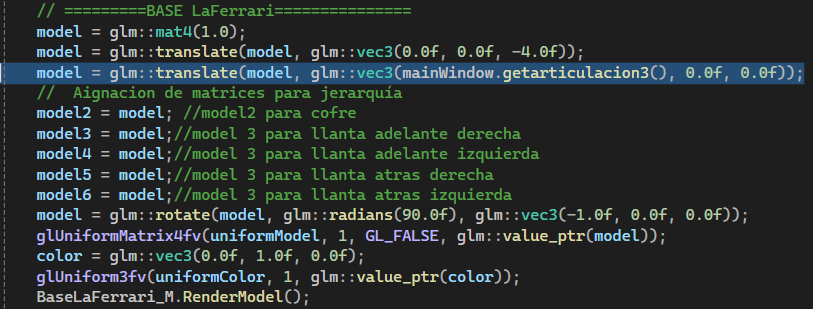
***4.- Agregar traslación con teclado para que pueda avanzar y retroceder de forma independiente***



**Ilustración 14** Teclas para mover el carro

Con este bloque de código asigna las flechas →/← del teclado para modificar la variable articulacion3, que usas luego usaremos para trasladar el coche en X:

* **→ (RIGHT):** articulacion3 += 0.1 para que el coche se desplace +X, es decir, a la derecha).
* **← (LEFT):** articulacion3 -= 0.1 para que el coche se desplace −X, es decir, a la izquierda.



**Ilustración 15 Translate con articulación 3 para mover el carro**

Se añade esta línea en el apartado del dibujado del auto model = glm::translate(model, vec3(mainWindow.getarticulacion3(), 0, 0)) para aplicar una traslación global en X al chasis, logrando que el coche avance o retroceda de forma independiente, y que todas sus piezas lo sigan por jerarquía.

Como resumen, esto es lo que hace cada articulación:

|  |  |  |
| --- | --- | --- |
| **No. Articulación** | **Parte que maneja** | **Tecla asignada** |
| **Articulación 1** | Rotación de las llantas | **M** para girar hacia adelante  **N** para girar hacia atrás |
| **Articulación 2** | Rotación del cofre | **F** para abrir  **G** para cerrar |
| **Articulación 3** | Desplazamiento del auto | **→ (RIGHT):** para avanzar  **← (LEFT):** para retroceder |

**Tabla 1** Articulaciones ocupadas para movimiento del auto

**Ejecución Final:**

**

**Ilustración 16** Ejecución Final

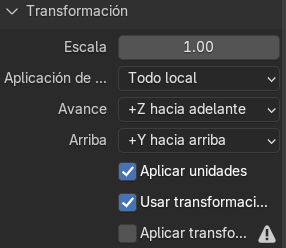
***LINK DEL VIDEO PARA VER LA EJECUCION FINAL:***

A continuación, se deja un video para poder visualizar mejor los movimientos del auto:

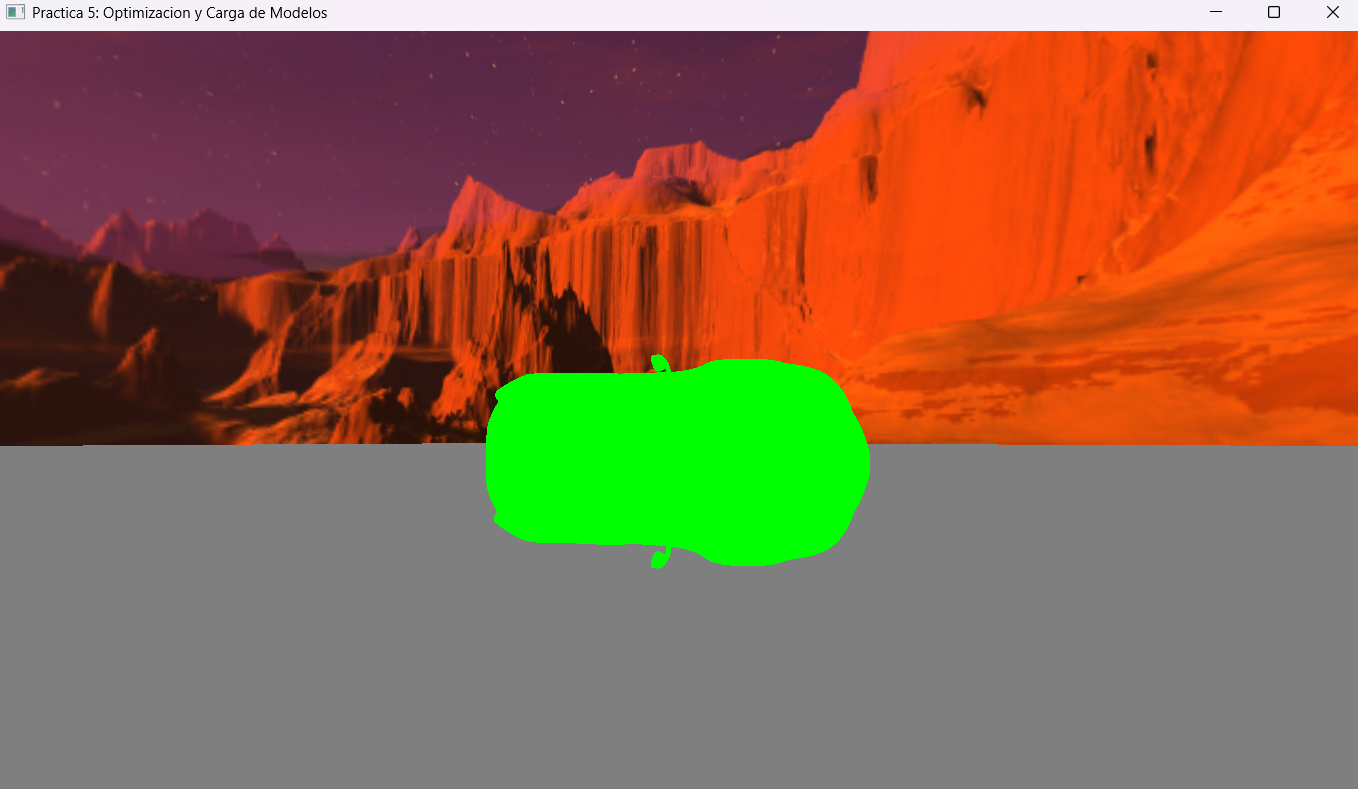
<https://drive.google.com/file/d/1VHujmYdzH-xAFVonqZyAXf3l1gxZkOhB/view?usp=sharing>

**2.- Liste los problemas que tuvo a la hora de hacer estos ejercicios y si los resolvió explicar cómo fue, en caso de error adjuntar captura de pantalla.**

Para esta práctica hubo un problema y es que el auto aparecía como si estuviera rotado, aunque en Blender se aplicara la transformación correcta o cualquier otra al archivo para exportar, al momento de mostrarla en la ventana de OpenGl/C++ el auto seguía apareciendo de la misma manera.

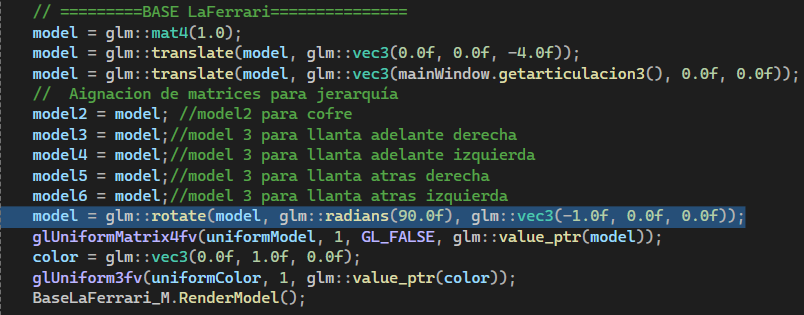


**Ilustración 17** Transformación - Exportación Blender

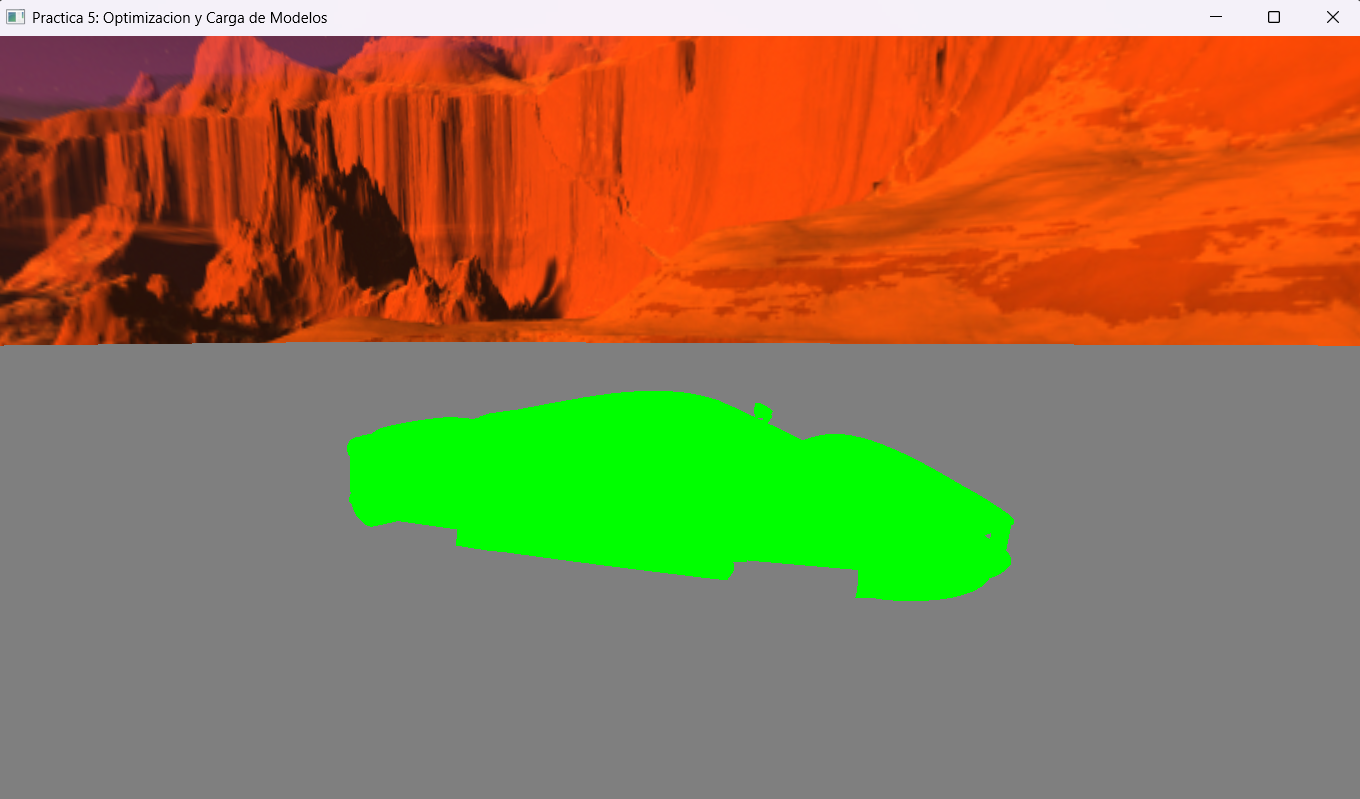


**Ilustración 18** Auto antes de aplicar la rotacion en OpenGl/C++

Para solucionarlo se aplicó una rotación directa en OpenGl/C++ a los objetos para posicionarlos correctamente



**Ilustración 19** Ejemplo rotación a base del auto



**Ilustración 20** Auto antes de aplicar la rotacion en OpenGl/C++

**3.- Conclusión:**

**a. Los ejercicios del reporte: Complejidad, Explicación.**

Me parecieron ejercicio que no tenían una alta complejidad técnica, es similar a lo que veníamos viendo de jerarquización, pero implementando la exportación de modelos desde Blender, y no precisamente dibujado o instanciados desde OpenGL.

**b. Comentarios generales: Faltó explicar a detalle, ir más lento en alguna explicación, otros comentarios y sugerencias para mejorar desarrollo de la práctica**

La práctica en clase la sentí bastante bien, como en las clases anteriores, el profesor se toma el tiempo adecuado para explicar el código lo mejor posible para que no queden dudas y además me gusta que no solo explique el código, también explica con ejemplos y teoría en el pizarrón.

**c. Conclusión**

Como conclusión de la práctica 5 puedo decir que resultó bastante interesante el modelado en Blender, ya que permite recrear modelos y trabajar con ellos más rápido que si se tuvieran que hacer desde 0 acomodando cada primitiva en OpenGL. Además, se siguió trabajando con jerarquía para hacer posible que los objetos se vayan enlazando o relacionando para poder llevar a cabo una correcta relación y movimiento entre el nodo padre y sus derivados. La práctica me permitió seguir aprendiendo nuevas cosas sobre la computación gráfica y esto aporta bastante al aprendizaje y poder seguir reforzando mis habilidades.