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: 06

NOMBRE COMPLETO: Lugo Manzano Rodrigo

Nº de Cuenta: 320206968

GRUPO DE LABORATORIO: 03

GRUPO DE TEORÍA: 02

SEMESTRE: 2026-1

FECHA DE ENTREGA LÍMITE: 12/octubre/2025

CALIFICACIÓN: _____

Reporte de Practica No. 6
Texturizado

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.- Crear un dado de 8 caras y texturizarlo por medio de código.

Para iniciar este ejercicio, primeramente, se creo un octaedro por medio de código, fue muy similar a la funcion de crear dado, que venia en la carpeta proporcionada por el profesor para esta práctica. Aunque a continuación se presenta la función completa, los fragmentos correspondientes a las coordenadas UV, se definieron hasta después, ya que primero se realizó la búsqueda de una imagen para texturizar el octaedro y se hizo el cálculo de la posición de cada vértice para saber como acomodarlo en el código.

```
259     void CrearOctaedro()
260     {
261         // 8 caras (triangulos), 3 vertices por cara = 24 vertices
262         // Formato: x, y, z,  S, T,  NX, NY, NZ
263         const float n = 0.57735026919f; // 1/sqrt(3)
264         const float s = 0.5f;           // radio
265
266         // Puntos UV
267         const float U1 = 0.00f, V1 = 0.00f; // 1
268         const float U2 = 0.00f, V2 = 0.40f; // 2
269         const float U3 = 0.33f, V3 = 0.20f; // 3
270         const float U4 = 0.33f, V4 = 0.60f; // 4
271         const float U5 = 0.33f, V5 = 0.99f; // 5
272         const float U6 = 0.66f, V6 = 0.00f; // 6
273         const float U7 = 0.66f, V7 = 0.40f; // 7
274         const float U8 = 0.66f, V8 = 0.79f; // 8
275         const float U9 = 0.99f, V9 = 0.60f; // 9
276         const float U10 = 0.99f, V10 = 0.99f; // 10
277
278         // Indices
279         unsigned int oct_indices[] = {
280             0,1,2,  3,4,5,  6,7,8,  9,10,11,
281             12,13,14,  15,16,17,  18,19,20,  21,22,23
282         };
283
```

Ilustración 1 *Función para crear un Octaedro parte 1.*

```
284     GLfloat oct_vertices[] = {
285         // ---- Cara 1  (triangulo "1" -> puntos 1,2,3)
286         0.0f,  s,   0.0f,   U3 - 0.01f, V3,    n,  n,  n,
287         0.0f,  0.0f, s,     U2 + 0.01f, V2-0.01f,    n,  n,  n,
288         s,     0.0f, 0.0f,  U1 +0.01f, V1 + 0.01f,    n,  n,  n,
289         // ---- Cara 2  (triangulo "2" -> puntos 2,4,3)
290         0.0f,  s,   0.0f,   U3, V3+0.01f,   -n,  n,  n,
291         -s,    0.0f, 0.0f,  U4, V4-0.01f,   -n,  n,  n,
292         0.0f,  0.0f, s,     U2+0.01f, V2,   -n,  n,  n,
293         // ---- Cara 3  (triangulo "3" -> puntos 4,7,3)
294         0.0f,  s,    0.0f,  U3+0.01f, V3 + 0.01f,   -n,  n, -n,
295         0.0f,  0.0f, -s,    U7-0.01f, V7,   -n,  n, -n,
296         -s,    0.0f,  0.0f, U4+0.01f, V4-0.01f,   -n,  n, -n,
297         // ---- Cara 4  (triangulo "4" -> puntos 6,7,3)
298         0.0f,  s,   0.0f,   U3+0.01f, V3,    n,  n, -n,
299         s,     0.0f, 0.0f,  U6-0.01f, V6+0.02f,    n,  n, -n,
300         0.0f,  0.0f, -s,    U7, V7-0.01f,    n,  n, -n,
301         // ---- Cara 5  (triangulo "5" -> puntos 5,4,8)
302         0.0f, -s,   0.0f,   U8 - 0.01f, V8+0.01f,    n, -n,  n,
303         s,     0.0f, 0.0f,  U5+0.01f, V5,    n, -n,  n,
304         0.0f,  0.0f, s,     U4 + 0.01f, V4+0.01f,    n, -n,  n,
305         // ---- Cara 6  (triangulo "6" -> puntos 4,7,8)
306         0.0f, -s,   0.0f,   U8, V8,   -n, -n,  n,
307         0.0f,  0.0f, s,     U4+0.01f, V4,   -n, -n,  n,
308         -s,    0.0f, 0.0f,  U7 -0.01f, V7+0.015f,   -n, -n,  n,
309         // ---- Cara 7  (triangulo "7" -> puntos 7,9,8)
310         0.0f, -s,    0.0f,  U8+0.01f, V8,   -n, -n, -n,
311         -s,    0.0f,  0.0f, U7+0.01f, V7 + 0.01f,   -n, -n, -n,
312         0.0f,  0.0f, -s,    U9, V9,   -n, -n, -n,
313         // ---- Cara 8  (triangulo "8" -> puntos 8,9,10)
314         0.0f, -s,   0.0f,   U8+0.01f,  V8+0.01f,    n, -n, -n,
315         0.0f,  0.0f, -s,    U9,  V9+0.01f,    n, -n, -n,
316         s,     0.0f,  0.0f, U10, V10,   n, -n, -n,
317     };
318
319     Mesh* oct = new Mesh();
320     oct->CreateMesh(oct_vertices, oct_indices, 192, 24);
321     meshList.push_back(oct);
```

Ilustración 2 *Función para crear un Octaedro parte 2.*

Una vez creada la función, se busco una imagen para texturizar el octaedro, se eligió la siguiente imagen:

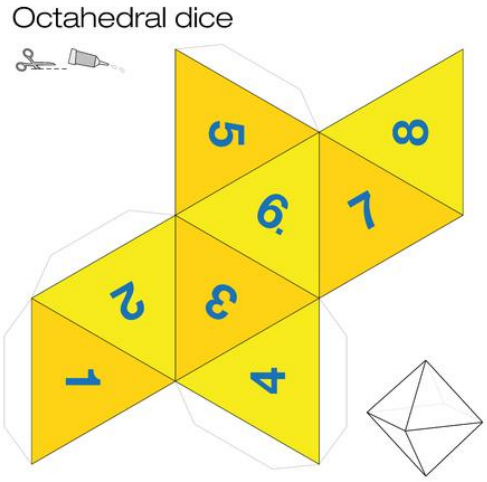

Ilustración 3 *Imagen para texturizar octaedro, antes de optimizar en GIMP*

Una vez que en GIMP optimizamos la imagen, quedo de la siguiente manera, aquí eliminamos bordes innecesarios, escalamos la imagen a 512x512 pixeles y le añadimos todos los canales de RGBA:

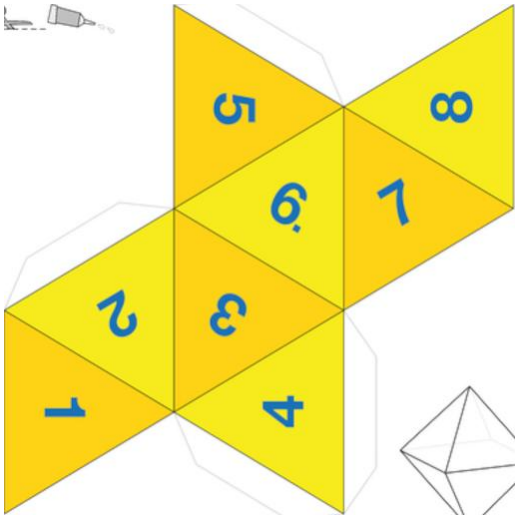

Ilustración 4 *Imagen para texturizar octaedro - Optimizada en GIMP*

Ahora antes de texturizar el octaedro por medio d código, fue importante definir las coordenadas de cada vértice en la imagen, las cuales quedaron de la siguiente manera:

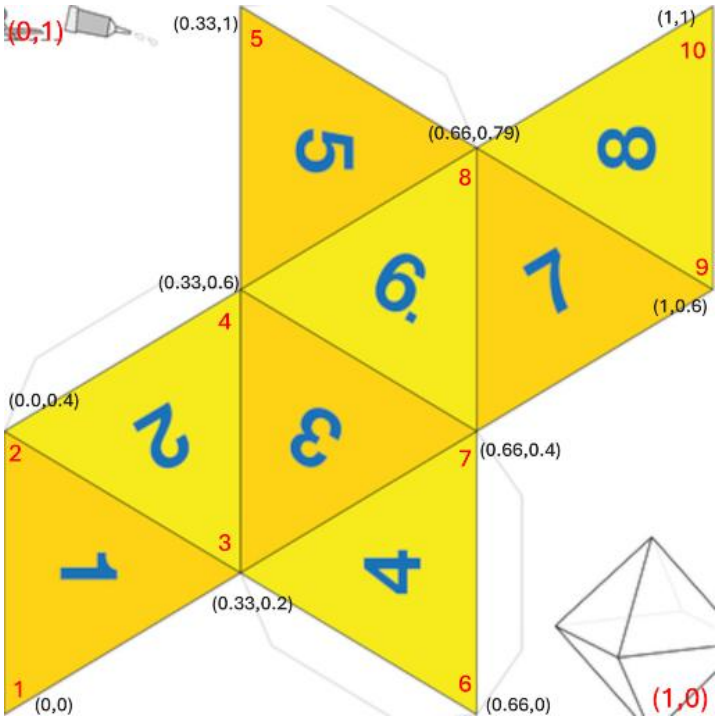

Ilustración 5 *Coordenadas de los vértices para texturizar el octaedro*

Estas coordenadas se obtuvieron de la siguiente manera, según los puntos marcados con color rojo en la imagen y sabiendo que en OpenGL la imagen es de 1x1, y nuestra imagen es de 512px por 512px:

- **Punto 1:** Para este punto, como esta en el origen podemos observar que es la coordenada (0,0)
- **Punto 2:** Podemos ver que en el eje x se encuentra sobre la coordenada 0, pero para el eje y no se puede ver a ojo. Así que, en GIMP, al poner el curso sobre es punto nos da que estamos sobre el píxel (0, 307), recordando que en GIMP las coordenadas en Y son a la inversa primero hacemos una resta:

  $$512 - 307 = 205$$

  Esto quiere decir que es como si estuviéramos en el pixel (0,205). Para sacar la equivalencia en coordenadas, necesitaríamos hacer una regla de 3, la cual se resume a una división:

  $$\frac{205}{512} = 0.4003$$

  Con esto ya sabemos que nuestra coordenada del punto 2 entonces es equivalente a (0, 0.4).
- **Punto 3:** Para este punto, respecto a la coordenada x podemos observar que es como si dividiéramos la imagen en 3 partes y tomáramos la primera porción, por lo tanto, la coordenada en x será 0.33. Para la coordenada Y, no se puede ver a ojo, entonces hacemos un proceso similar al anterior, con la ayuda de GIMP, que nos dice que es el pixel (170, 409), entonces:

  $$512 - 409 = 103$$

  Ahora la regla de 3:

  $$\frac{103}{512} = 0.2011$$

  Por lo tanto, nos referimos a la coordenada (0.33, 0.20)
- **Punto 4:** En este caso, a ojo sabemos que la coordenada en x es 0.33. Y con ayuda de GIMP, que nos arrojo estar en el píxel (170, 204).

  $$512 - 204 = 308$$

  Ahora la regla de 3:

  $$\frac{308}{512} = 0.6015$$

  Por lo tanto, nos referimos a la coordenada (0.33, 0.60)

- **Punto 5:** Para este punto, con solo observar la imagen podemos decir que se trata de la coordenada (0.33, 1)
- **Punto 6:** Igualmente con solo observar la imagen podemos decir que se trata de la coordenada (0.66, 0)
- **Punto 7:** Observando la imagen sabemos que la coordenada x es 0.66. Y como esta a la altura del punto 2 anteriormente calculado, la coordenada en Y será de 0.40 Entonces es (0.66, 0.40)
- **Punto 8:** En este caso, a ojo sabemos que la coordenada en x es 0.66. Y con ayuda de GIMP, que nos arrojó estar en el píxel (340, 103).

  $$512 - 103 = 409$$

  Ahora la regla de 3:

  $$\frac{409}{512} = 0.7988$$

  Por lo tanto, nos referimos a la coordenada (0.66, 0.79)
- **Punto 9:** Observando la imagen sabemos que la coordenada x es 1, por estar en el extremo derecho. Y como está a la altura del punto 4 anteriormente calculado, la coordenada en Y será de 0.60 Entonces es (1, 0.60)
- **Punto 10:** Al encontrarse en la esquina superior derecha, sabemos que es la coordenada (1,1)

Ahora si con esta información analizada podemos poner nuestros puntos en la función:

```
// Puntos UV
const float U1 = 0.00f, V1 = 0.00f; // 1
const float U2 = 0.00f, V2 = 0.40f; // 2
const float U3 = 0.33f, V3 = 0.20f; // 3
const float U4 = 0.33f, V4 = 0.60f; // 4
const float U5 = 0.33f, V5 = 0.99f; // 5
const float U6 = 0.66f, V6 = 0.00f; // 6
const float U7 = 0.66f, V7 = 0.40f; // 7
const float U8 = 0.66f, V8 = 0.79f; // 8
const float U9 = 0.99f, V9 = 0.60f; // 9
const float U10 = 0.99f, V10 = 0.99f; // 10
```

Ilustración 6 *Puntos o coordenadas UV colocadas en la función*

Y analizando la figura con sus caras y vértices, procedemos a colocar las coordenadas ST para cada vértice, de modo que no queden en espejo y todas sus caras queden sin los bordes de la imagen, para esto en algunos casos se suma o resta 0.01f:

```
GLfloat oct_vertices[] = {
    // ---- Cara 1  (triangulo "1" -> puntos 1,2,3)
    0.0f,  s,    0.0f,    U3 - 0.01f, V3,     n,  n,  n,
    0.0f,  0.0f, s,       U2 + 0.01f, V2-0.01f,     n,  n,  n,
    s,     0.0f, 0.0f,    U1 +0.01f, V1 + 0.01f,     n,  n,  n,
    // ---- Cara 2  (triangulo "2" -> puntos 2,4,3)
    0.0f,  s,    0.0f,    U3, V3+0.01f,    -n,  n,  n,
    -s,    0.0f, 0.0f,    U4, V4-0.01f,    -n,  n,  n,
    0.0f,  0.0f, s,       U2+0.01f, V2,    -n,  n,  n,
    // ---- Cara 3  (triangulo "3" -> puntos 4,7,3)
    0.0f,  s,     0.0f,   U3+0.01f, V3 + 0.01f,   -n,  n, -n,
    0.0f,  0.0f, -s,      U7-0.01f, V7,    -n,  n, -n,
    -s,    0.0f,  0.0f,   U4+0.01f, V4-0.01f,    -n,  n, -n,
    // ---- Cara 4  (triangulo "4" -> puntos 6,7,3)
    0.0f,  s,    0.0f,    U3+0.01f, V3,     n,  n, -n,
    s,     0.0f, 0.0f,    U6-0.01f, V6+0.02f,     n,  n, -n,
    0.0f,  0.0f, -s,      U7, V7-0.01f,     n,  n, -n,
    // ---- Cara 5  (triangulo "5" -> puntos 5,4,8)
    0.0f, -s,    0.0f,    U8 - 0.01f, V8+0.01f,     n, -n,  n,
    s,     0.0f, 0.0f,    U5+0.01f, V5,     n, -n,  n,
    0.0f,  0.0f, s,       U4 + 0.01f, V4+0.01f,     n, -n,  n,
    // ---- Cara 6  (triangulo "6" -> puntos 4,7,8)
    0.0f, -s,    0.0f,    U8, V8,    -n, -n,  n,
    0.0f,  0.0f, s,       U4+0.01f, V4,    -n, -n,  n,
    -s,    0.0f, 0.0f,    U7 -0.01f, V7+0.015f,    -n, -n,  n,
    // ---- Cara 7  (triangulo "7" -> puntos 7,9,8)
    0.0f, -s,     0.0f,   U8+0.01f, V8,    -n, -n, -n,
    -s,    0.0f,  0.0f,   U7+0.01f, V7 + 0.01f,    -n, -n, -n,
    0.0f,  0.0f, -s,      U9, V9,    -n, -n, -n,
    // ---- Cara 8  (triangulo "8" -> puntos 8,9,10)
    0.0f, -s,    0.0f,    U8+0.01f,  V8+0.01f,     n, -n, -n,
    0.0f,  0.0f, -s,      U9,  V9+0.01f,     n, -n, -n,
    s,     0.0f,  0.0f,   U10, V10,    n, -n, -n,
};
```

Ilustración 7 *Colocación de coordenadas UV para texturizar correctamente el octaedro*

En el main llamamos a nuestra funcion CrearOctaedro() y cargamos nuestra imagen de texturizado:

```
333     CrearOctaedro();
334     CreateShaders();
335
336     camera = Camera(glm::vec3(0.0f, 0.0f, 0.0f), glm::vec3(0.0f, 1.0f, 0.0f), -60.0f, 0.0f, 0.3f, 0.5f);
337
338     brickTexture = Texture("Textures/brick.png");
339     brickTexture.LoadTextureA();
340     dirtTexture = Texture("Textures/dirt.png");
341     dirtTexture.LoadTextureA();
342     plainTexture = Texture("Textures/plain.png");
343     plainTexture.LoadTextureA();
344     pisoTexture = Texture("Textures/piso.tga");
345     pisoTexture.LoadTextureA();
346     dadoTexture = Texture("Textures/dado_animales2.png");
347     dadoTexture.LoadTextureA();
348     logofiTexture = Texture("Textures/escudo_fi_color.tga");
349     logofiTexture.LoadTextureA();
350     octaedroTexture = Texture("Textures/octaedro_numeros_gimp.png");
351     octaedroTexture.LoadTextureA();
```

Ilustración 8 *Carga de la imagen de texturizado*

Por último, dibujamos el octaedro con su textura, ubicado en el índice 5 de la MeshList.

```
453         //Reporte de práctica :
454         //Ejercicio 1: Crear un dado de 8 caras y texturizarlo por medio de código
455         model = glm::mat4(1.0);
456         model = glm::translate(model, glm::vec3(-1.5f, 4.5f, -2.0f));
457         model = glm::scale(model, glm::vec3(2.0f, 2.0f, 2.0f));
458         glUniformMatrix4fv(uniformModel, 1, GL_FALSE, glm::value_ptr(model));
459         octaedroTexture.UseTexture();
460         meshList[5]->RenderMesh();
```

Ejecución Final del Ejercicio 1:

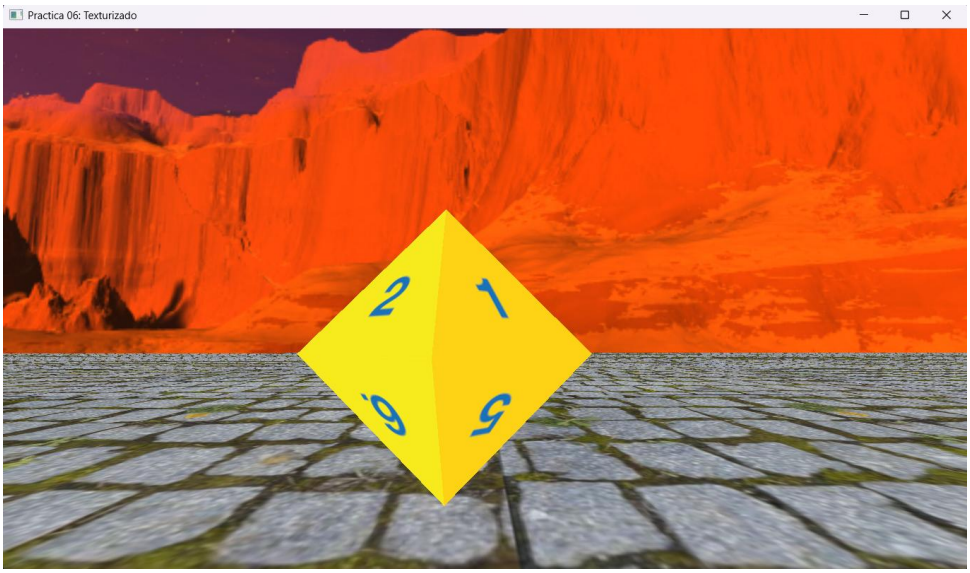

Ilustración 9 *Vista 1 del octaedro Texturizado*

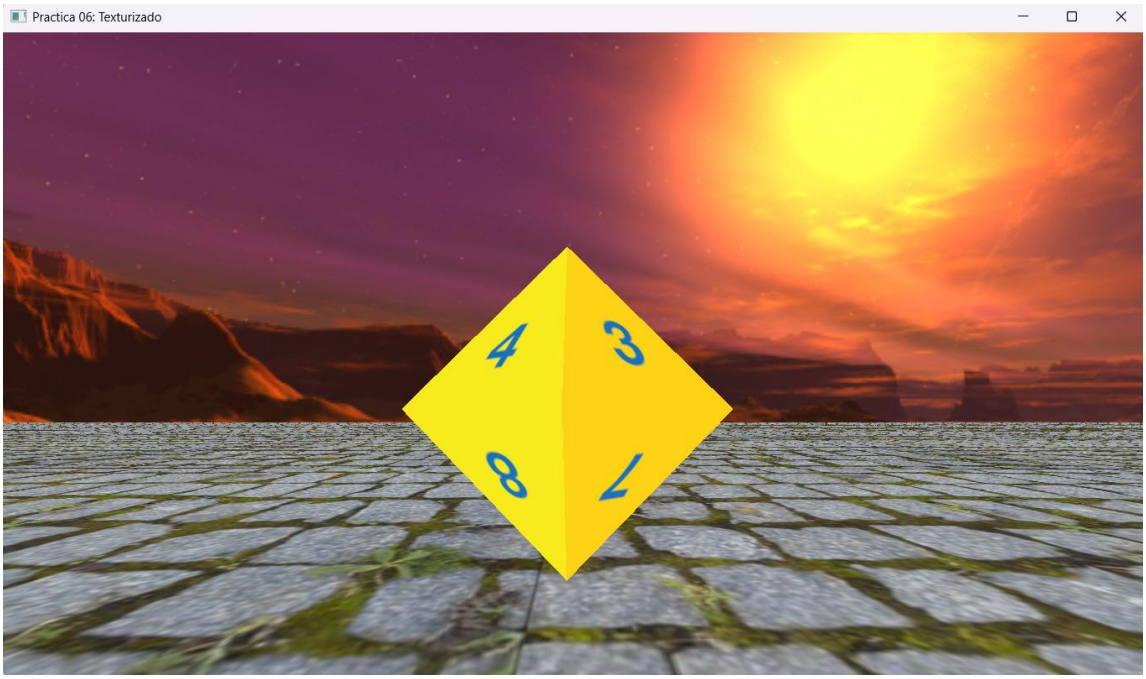

Ilustración 10 *Vista 2 del octaedro Texturizado*

2.- Importar el modelo de su coche con sus 4 llantas acomodadas y tener texturizadas las 4 llantas (diferenciar caucho y rin)

Para comenzar esta actividad, primero se separaron todas las llantas del coche y posteriormente el rin de cada una de las 4 llantas.

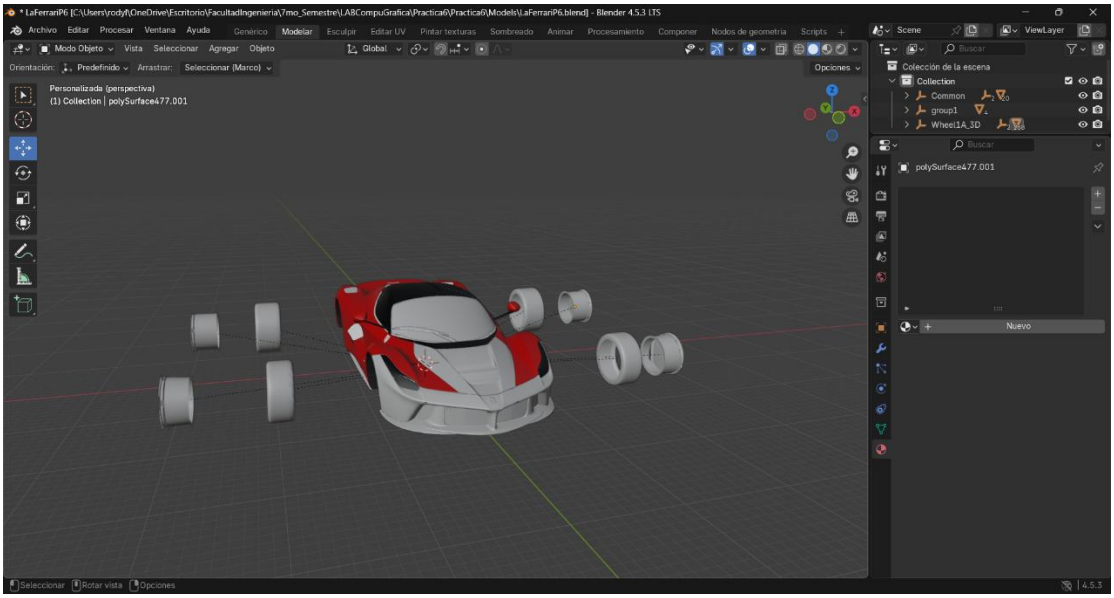

Ilustración 11 *Rin y llantas separadas*

Posteriormente se eligieron imágenes para texturizar llantas y rines

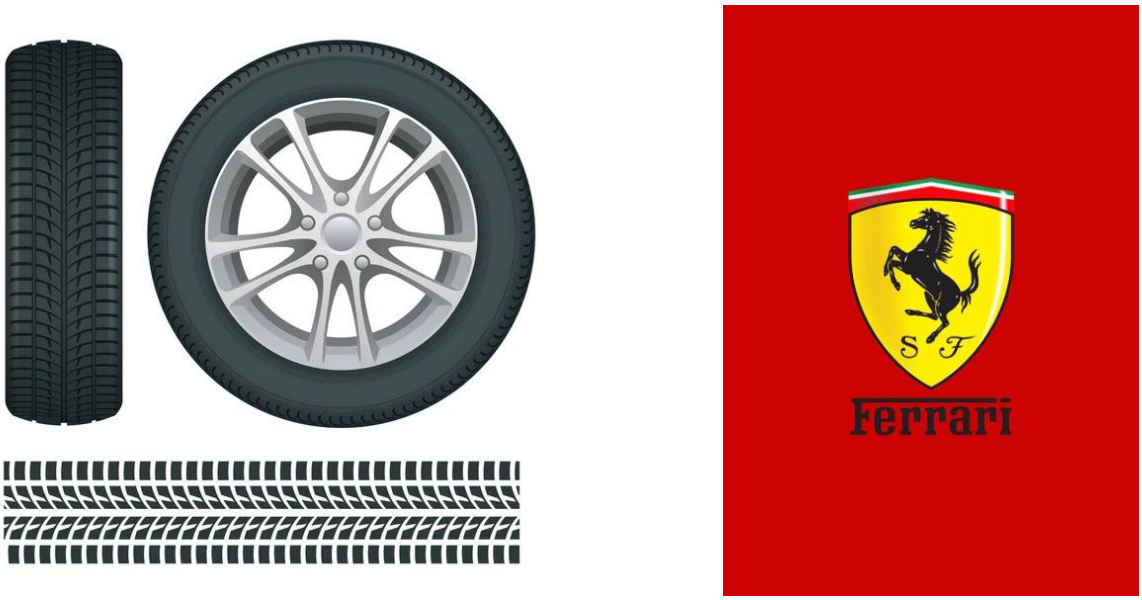

Ilustración 12 *Imágenes para texturizar llantas y rines*

Luego se texturizo cada rin y cada llanta de forma similar a la que ya sabemos, para esto primero se tuvo que eliminar todas las texturas de las llantas que tenía el modelo original, ya que si no causaban errores. Después en la sección de material, y la opción de color base agregábamos la imagen con la que íbamos a dar textura a cada parte del carro.

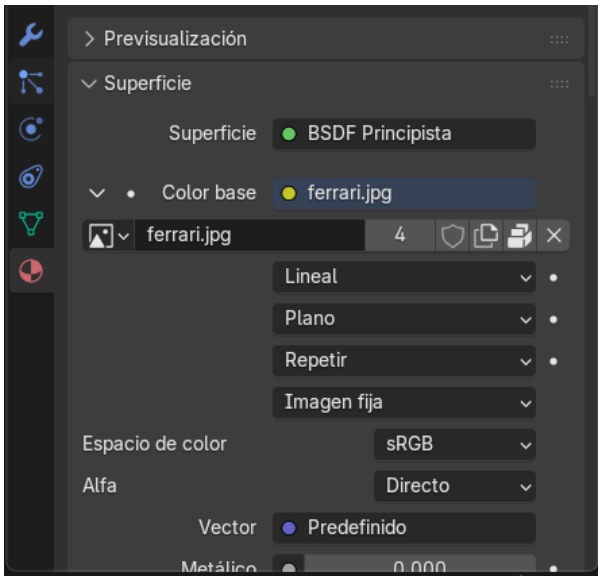

Ilustración 13 *Ejemplo de agregar imagen para texturizado en Blender*

Después en la sección de Editar UV, con nuestro objeto seleccionado en Modo edición, podíamos ir ajustando las caras de nuestro objeto para ir acomodando la textura en nuestro objeto.

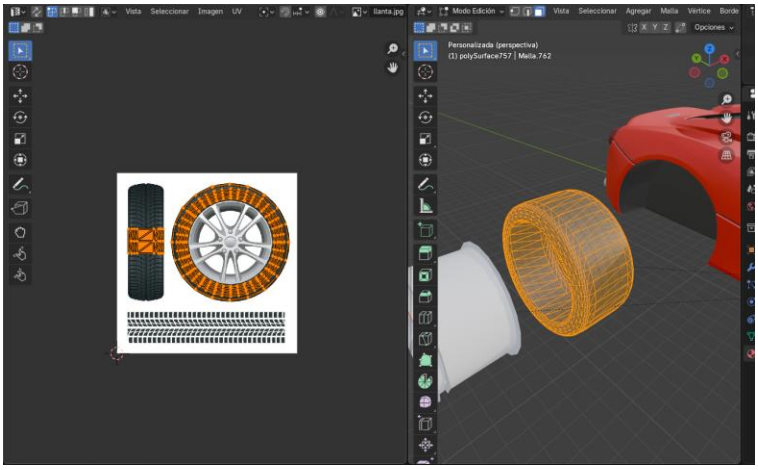

Ilustración 14 *Ejemplo de texturizado en Blender*

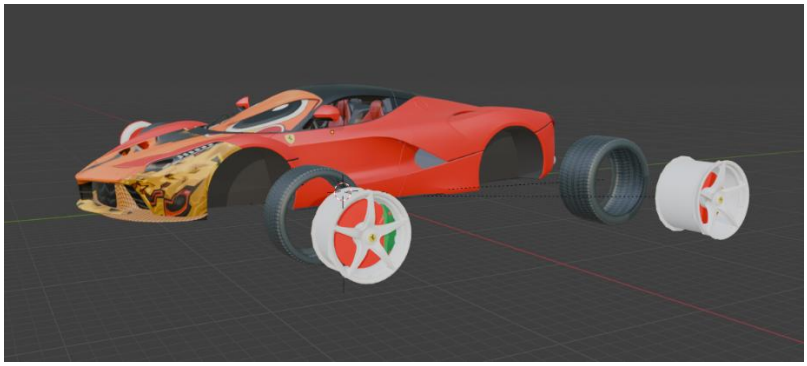

Ilustración 15 *Llantas y Rines texturizados*

Una vez texturizados tantos rines como llantas, los volvimos a juntar para poder exportarlos como un solo objeto posteriormente.

Ilustración 16 *Llantas y Rines texturizados juntos*

Una vez terminado en Blender el texturizado y haber importado nuestros modelos pasamos a OpenGL, para dibujar la parte del carro (base y llantas).

```
Model BaseCarro_M;
Model Llanta_AdelanteD_M;
Model Llanta_AdelanteI_M;
Model Llanta_AtrasD_M;
Model Llanta_AtrasI_M;
```

Ilustración 17 *Objetos tipo Model para base y llantas del carro*

```
356         BaseCarro_M = Model();
357         BaseCarro_M.LoadModel("Models/BaseCarro.fbx");
358         Llanta_AdelanteD_M = Model();
359         Llanta_AdelanteD_M.LoadModel("Models/LlantaAdelanteD.fbx");
360         Llanta_AdelanteI_M = Model();
361         Llanta_AdelanteI_M.LoadModel("Models/LlantaAdelanteI.fbx");
362         Llanta_AtrasD_M = Model();
363         Llanta_AtrasD_M.LoadModel("Models/LlantaAtrasD.fbx");
364         Llanta_AtrasI_M = Model();
365         Llanta_AtrasI_M.LoadModel("Models/LlantaAtrasI.fbx");
```

Ilustración 18 *Carga de modelos de la base del carro y sus 4 llantas*

```
467         */
468         //Instancia del coche
469         model = glm::mat4(1.0);
470         model = glm::translate(model, glm::vec3(0.0f + mainWindow.getmuevex() , -0.5f, -3.0f));
471         modelaux = model;
472         model = glm::scale(model, glm::vec3(2.0f, 2.0f, 2.0f));
473         model = glm::rotate(model, 90 * toRadians, glm::vec3(-1.0f, 0.0f, 0.0f));
474         model = glm::rotate(model, 180 * toRadians, glm::vec3(0.0f, 0.0f, 1.0f));
475         glUniformMatrix4fv(uniformModel, 1, GL_FALSE, glm::value_ptr(model));
476         color = glm::vec3(0.0f, 0.0f, 1.0f);//color azul para el coche
477         glUniform3fv(uniformColor, 1, glm::value_ptr(color));
478         BaseCarro_M.RenderModel();
479
480         //Llanta delantera izquierda
481         model = modelaux;
482         model = glm::translate(model, glm::vec3(-2.6f, 0.4f, 1.7f));
483         model = glm::rotate(model, -90 * toRadians, glm::vec3(0.0f, 1.0f, 0.0f));
484         model = glm::scale(model, glm::vec3(2.0f, 2.0f, 2.0f));
485         color = glm::vec3(0.5f, 0.5f, 0.5f);//llanta con color gris
486         glUniform3fv(uniformColor, 1, glm::value_ptr(color));
487         glUniformMatrix4fv(uniformModel, 1, GL_FALSE, glm::value_ptr(model));
488         Llanta_AdelanteI_M.RenderModel();
489
490         //Llanta trasera izquierda
491         model = modelaux;
492         model = glm::translate(model, glm::vec3(2.6f, 0.4f, 1.7f));
493         model = glm::rotate(model, -90 * toRadians, glm::vec3(0.0f, 1.0f, 0.0f));
494         model = glm::scale(model, glm::vec3(2.0f, 2.0f, 2.0f));
495         glUniformMatrix4fv(uniformModel, 1, GL_FALSE, glm::value_ptr(model));
496         Llanta_AtrasI_M.RenderModel();
497
498         //Llanta delantera derecha
499         model = modelaux;
500         model = glm::translate(model, glm::vec3(-2.6f, 0.4f, -1.7f));
501         model = glm::rotate(model, 90 * toRadians, glm::vec3(0.0f, 1.0f, 0.0f));
502         model = glm::scale(model, glm::vec3(2.0f, 2.0f, 2.0f));
503         glUniformMatrix4fv(uniformModel, 1, GL_FALSE, glm::value_ptr(model));
504         Llanta_AdelanteD_M.RenderModel();
505
506         //Llanta trasera derecha
507         model = modelaux;
508         model = glm::translate(model, glm::vec3(2.6f, 0.4f, -1.7f));
509         model = glm::rotate(model, 90 * toRadians, glm::vec3(0.0f, 1.0f, 0.0f));
510         model = glm::scale(model, glm::vec3(2.0f, 2.0f, 2.0f));
511         glUniformMatrix4fv(uniformModel, 1, GL_FALSE, glm::value_ptr(model));
512         Llanta_AtrasD_M.RenderModel();
```

Ilustración 19 *Instancia de cada parte del auto*

3.- Texturizar la cara del personaje de la imagen tipo cars en el espejo (ojos) y detalles en cofre y parrilla de su propio modelo de coche

Para llevar a cabo esta parte primero identificamos nuestra imagen original tipo cars del previo 4, basada en nuestro personaje del proyecto.

Ilustración 20 *Imagen original carro Pikachu tipo Cars*

A partir de esa imagen, la optimizamos en GIMP, para ocupar cada parte solicitada y poder texturizar nuestros objetos.

Ilustración 21 *Imagen optimizada en GIMP para texturizar ojos*

Ilustración 22 *Imagen optimizada en GIMP para texturizar cofre*

Ilustración 23 *Imagen optimizada en GIMP para texturizar parrilla*

Para texturizar en Blender, fue similar al proceso anterior explicado:

- Se separaron cristal de enfrente, parrilla y cofre del auto para poder texturizar por aparte cada uno
- Se añadían las imágenes en el apartado de material, según el objeto por texturizar
- Se acomodaban las caras del objeto en la edición de coordenadas UV

Ilustración 24 *Texturizado de Ojos, Cofre y Parrilla*

Una vez terminado en Blender el texturizado y haber importado nuestros modelos pasamos a OpenGL, para dibujar las partes del carro (ojos, cofre y parrilla).

```
Model OjosCarro_M;
Model CofreCarro_M;
Model ParrillaCarro_M;
```

Ilustración 25 *Objetos tipo Model para ojos, cofre y parrilla del carro*

```
OjosCarro_M = Model();
OjosCarro_M.LoadModel("Models/OjosCarro.obj");
CofreCarro_M = Model();
CofreCarro_M.LoadModel("Models/CofreCarro.obj");
ParrillaCarro_M = Model();
ParrillaCarro_M.LoadModel("Models/ParrillaCarro.obj");
```

Ilustración 26 *Carga de modelos de ojos, cofre y parrilla del carro*

```
514         //Ojos del coche
515         model = modelaux;
516         model = glm::translate(model, glm::vec3(-0.0f, -0.2f, 0.0f));
517         model = glm::rotate(model, 90 * toRadians, glm::vec3(0.0f, 1.0f, 0.0f));
518         model = glm::scale(model, glm::vec3(0.2f, 0.2f, 0.2f));
519         glUniformMatrix4fv(uniformModel, 1, GL_FALSE, glm::value_ptr(model));
520         OjosCarro_M.RenderModel();
521
522         //Cofre del coche
523         model = modelaux;
524         model = glm::translate(model, glm::vec3(-0.0f, -0.15f, 0.0f));
525         model = glm::rotate(model, 90 * toRadians, glm::vec3(0.0f, 1.0f, 0.0f));
526         model = glm::scale(model, glm::vec3(0.2f, 0.2f, 0.2f));
527         glUniformMatrix4fv(uniformModel, 1, GL_FALSE, glm::value_ptr(model));
528         CofreCarro_M.RenderModel();
529
530         //Parrilla del coche
531         model = modelaux;
532         model = glm::translate(model, glm::vec3(-0.0f, -0.15f, 0.0f));
533         model = glm::rotate(model, 90 * toRadians, glm::vec3(0.0f, 1.0f, 0.0f));
534         model = glm::scale(model, glm::vec3(0.2f, 0.2f, 0.2f));
535         glUniformMatrix4fv(uniformModel, 1, GL_FALSE, glm::value_ptr(model));
536         ParrillaCarro_M.RenderModel();
```

Ilustración 27 *Instancia de ojos, cofre y parrilla*

Ejecución Final del Ejercicio 2 y 3:

Ilustración 28 *Diferencia entre rin y llanta por texturizado*

Ilustración 29 *Vista 1 Auto*

Ilustración 30 *Vista 2 auto*

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 practica solo hubo un pequeño problema al trabajar en Blender, pues no se veían las imágenes importadas para poder texturizar los objetos pero era debido a las texturas originales del modelo, así que eliminando las texturas de esos objetos se resolvió el problema.

3.- Conclusión:

a. Los ejercicios del reporte: Complejidad, Explicación.

Me parecieron ejercicio que no tenían una alta complejidad técnica, pero si era un poco tardado para poder acomodar tanto los vértices en el texturizado por código, y en Blender al trabajar con objetos de muchísimas caras y vértices era mucho más tardado acomodar sus caras para texturizar, así como estar separando los objetos.

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, la explicación del código fue rápido pero debido a que es mucha información y cosas por hacer, como editar imágenes, texturizar ya sea por código o por blender, exportar modelos, acomodarlos, etc. Pero me gusta tener video de apoyo en la plataforma.

c. Conclusión

Como conclusión de la práctica 6 puedo decir que resultó bastante interesante el texturizado por código, ya que desconocía que se podía hacer de esa manera, yo pensé que solo por medio de un software de modelado como en Blender. Igualmente me resulto interesante el modelado por esta plataforma, además de que puede ser más rápido. También aprendí a usar GIMP y optimizar imágenes, para añadir canales RGBA, recortarlas para trabajar solo con la parte importante de la imagen, escalarla, así como analizarla para poder obtener las coordenadas de cada punto. La práctica me permitió seguir aprendiendo nuevas cosas sobre la computación gráfica y esto aporta bastante al aprendizaje y poder seguir reforzando mis habilidades.