



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: CARBAJAL REYES IRVIN JAIR

Nº de Cuenta: 422042084

GRUPO DE LABORATORIO: 11

GRUPO DE TEORÍA: 04

SEMESTRE 2025-2

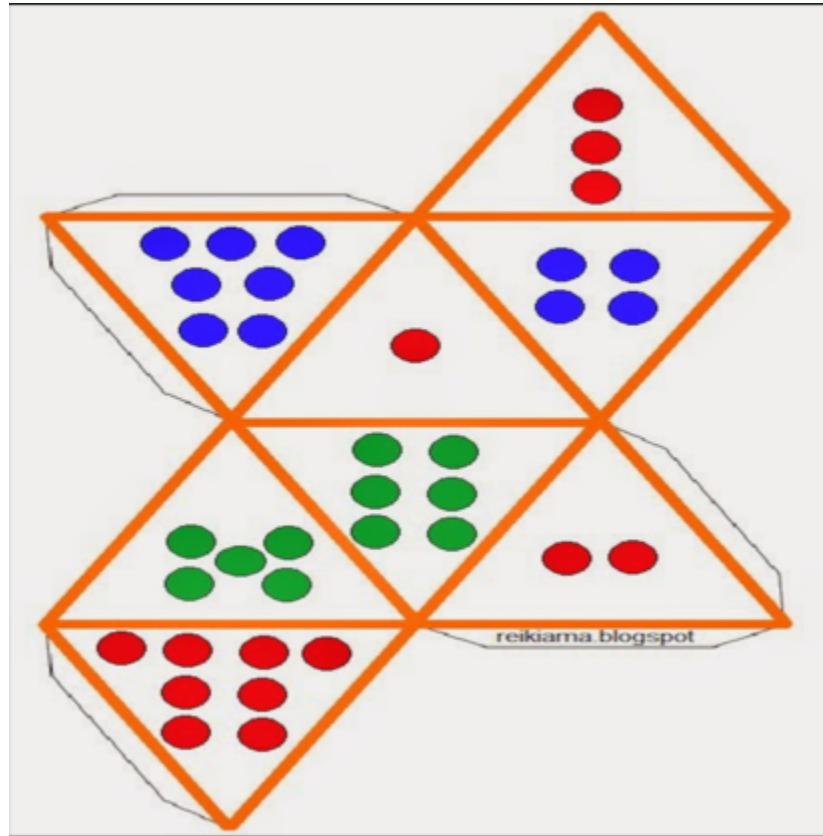
FECHA DE ENTREGA LÍMITE: 26 DE MARZO DE 2025

CALIFICACIÓN: _____

Actividades

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

Para la elaboración de este ejercicio primero encontramos una imagen para poder texturizar nuestro dado de ocho caras.



Luego, con ayuda de GIMP optimizamos nuestra imagen para que tenga una resolución de 2 a la n, donde se usó 1024x1024. La imagen debe estar en modo RGB y en formato JPG.

Una vez teniendo nuestra textura optimizada, creamos nuestro prisma en Visual Studio.

Primero designamos los 23 vértices que se usarán, tres por cada cara.

```

251     void ... CrearDadoOcho()
252     {
253         unsigned int dado_indices[] = {
254             // Cara A
255             0, 1, 2,
256
257             // Cara B
258             3, 4, 5,
259
260             // Cara C
261             6, 7, 8,
262
263             // Cara D
264             9, 10, 11,
265

```

Luego, designamos los vértices de cada cara, poniendo la ubicación en coordenadas x,y,z seguido de las coordenadas de textura s,t y las normales nx,ny,nz .

```

281     GLfloat dado_vertices[] = {
282         // Cara A
283         //x   y   z   S   T   NX   NY   NZ
284         0.0f, 0.5f, 0.0f, 0.49f, 0.74f, 0.0f, -1.0f, -1.0f,
285         -0.5f, 0.0f, 0.5f, 0.27f, 0.5f, 0.0f, -1.0f, -1.0f,
286         0.5f, 0.0f, 0.5f, 0.71f, 0.5f, 0.0f, -1.0f, -1.0f,
287
288         // Cara B
289         //x   y   z   S   T   NX   NY   NZ
290         0.0f, 0.5f, 0.0f, 0.71f, 0.5f, -1.0f, -1.0f, 0.0f,
291         0.5f, 0.0f, 0.5f, 0.93f, 0.75f, -1.0f, -1.0f, 0.0f,
292         0.5f, 0.0f, -0.5f, 0.49f, 0.75f, -1.0f, -1.0f, 0.0f,
293
294         // Cara C
295         //x   y   z   S   T   NX   NY   NZ
296         0.0f, 0.5f, 0.0f, 0.71f, 0.99f, 0.0f, -1.0f, 1.0f,
297         0.5f, 0.0f, -0.5f, 0.49f, 0.74f, 0.0f, -1.0f, 1.0f,
298         -0.5f, 0.0f, -0.5f, 0.94f, 0.74f, 0.0f, -1.0f, 1.0f,

```

Para las coordenadas de x,y,z se tuvo como referencia el vértice 0 ubicado en el origen del plano XZ y con altura de 0.5 y -0.5 en el eje Y. Cada uno de estos dos vértices unen cuatro caras cada uno, divididas en las caras superiores del prisma y las caras inferiores.

Para las coordenadas de s,t se usó GIMP para obtener las coordenadas de cada punto dentro del panel de 1024x1024. tomando en cuenta que el origen está en la esquina superior izquierda, teniendo que restar a 1024 el valor de Y obtenido por el software. Luego, mediante regla de tres se obtiene su equivalente dentro del plano de dimensión 1x1 (s,t).

```

324     // Cara H
325     //x      y      z      S      T      NX     NY     NZ
326     0.5f,  0.0f, -0.5f,  0.49f, 0.25f,  0.0f,  1.0f,  1.0f,
327     -0.5f, 0.0f, -0.5f,  0.05f, 0.25f,  0.0f,  1.0f,  1.0f,
328     0.0f, -0.5f, 0.0f,  0.27f, 0.5f,   0.0f,  1.0f,  1.0f,
329
330 };
331
332 Mesh* dado8 = new Mesh();
333 dado8->CreateMesh(dado_vertices, dado_indices, 192, 24);
334 meshList.push_back(dado8);
335
336 }
337

```

Las normales se obtienen considerando un ángulo de 45°, por lo que se le asignó valor de 1 o -1 en los ejes correspondientes donde actuaría el vector normal.

Finalmente declaramos la cantidad de vértices y de índices usados para generar el prisma.

Cargamos la textura optimizada para nuestro dado.

```

360     dadoTexture = Texture("Textures/dado8.jpg");
361     dadoTexture.LoadTextureA();

```

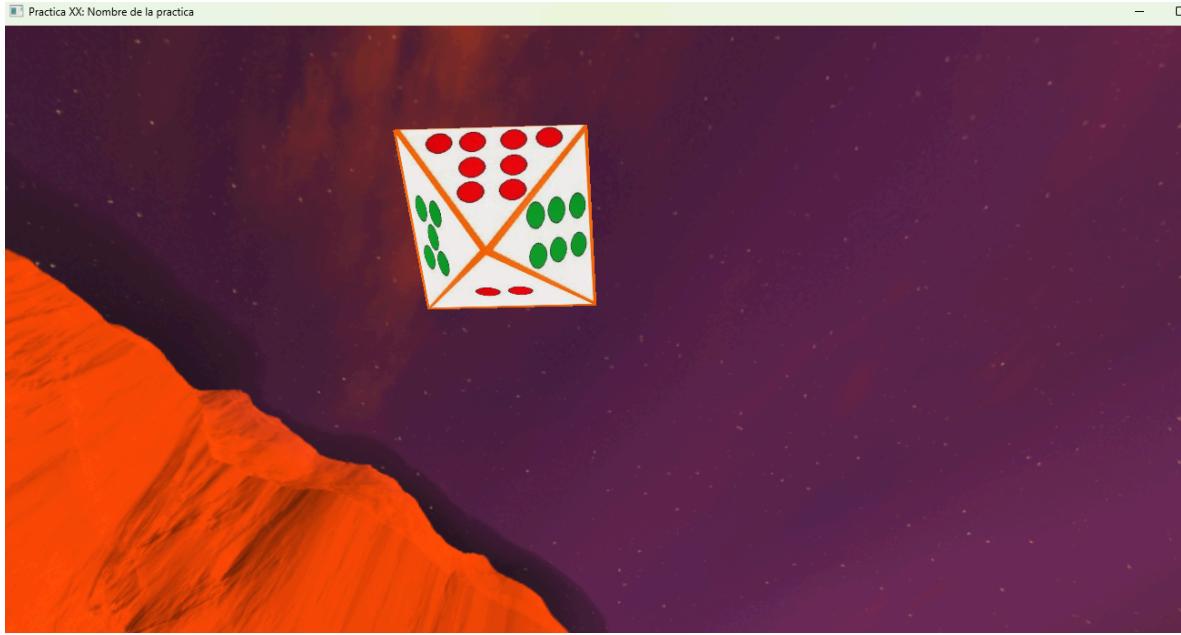
Llamamos la figura con su textura.

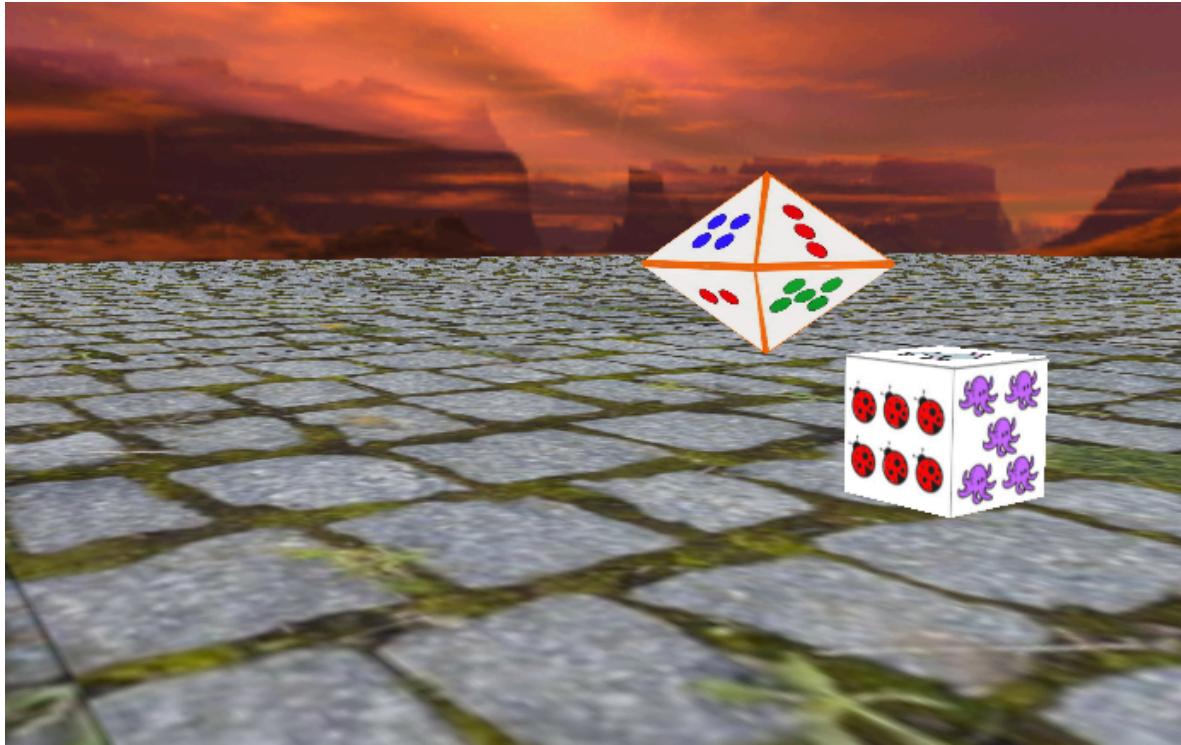
```

431 //Dado de ocho caras
432 model = glm::mat4(1.0);
433 model = glm::translate(model, glm::vec3(-1.5f, 4.5f, -2.0f));
434 glUniformMatrix4fv(uniformModel, 1, GL_FALSE, glm::value_ptr(model));
435 dadoTexture.UseTexture();
436 meshList[5]->RenderMesh();

```

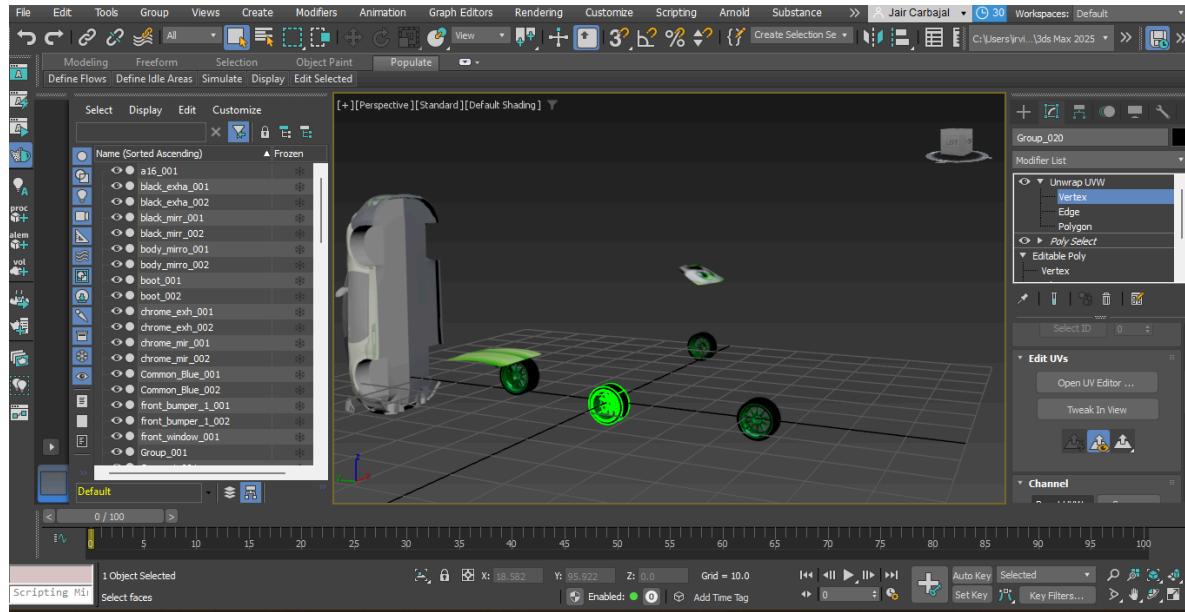
Ejecución





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

Para esta actividad fue necesario separar las partes del auto que se iban a texturizar y realizar este proceso a cada una de ellas.



Usamos nuestra imagen de texturizado para las llantas y colocamos cada una de las partes en la textura deseada.



Una vez que se hayan exportado cada una de nuestras llantas, creamos los modelos dentro de Visual Studio.

```
50     Model Llanta1_M;
51     Model Llanta2_M;
52     Model Llanta3_M;
53     Model Llanta4_M;
```

Y hacemos referencia a los modelos exportados para llamarlos.

```
380     Llanta1_M = Model();
381     Llanta1_M.LoadModel("Models/llanta1_tex.obj");
382     Llanta2_M = Model();
383     Llanta2_M.LoadModel("Models/llanta2_tex.obj");
384     Llanta3_M = Model();
385     Llanta3_M.LoadModel("Models/llanta3_tex.obj");
386     Llanta4_M = Model();
387     Llanta4_M.LoadModel("Models/llanta4_tex.obj");
```

Además tenemos que tener nuestra base del auto previamente exportada de igual manera.

```
372     CarroBase_M = Model();
373     CarroBase_M.LoadModel("Models/carro_base.obj");
374 }
```

Después colocamos los modelos texturizados dentro de nuestro *main*.

```
477 //Base auto
478 //color = glm::vec3(0.0f, 0.0f, 0.0f); //modelo del auto de color negro
479
480     model = glm::mat4(1.0);
481     model = glm::translate(model, glm::vec3(15.0f, -12.0f, -1.5f));
482     modelaux = model;
483     glUniform3fv(uniformColor, 1, glm::value_ptr(color));
484     glUniformMatrix4fv(uniformModel, 1, GL_FALSE, glm::value_ptr(model));
485     CarroBase_M.RenderModel(); //modificar por el modelo sin las 4 llantas y sin cofre
486
```

Al igual que cada una de las llantas

```
497 //Llanta1
498 model = glm::translate(model, glm::vec3(10.0f, -6.0f, 4.5f));
499     modelaux = model;
500     glUniformMatrix4fv(uniformModel, 1, GL_FALSE, glm::value_ptr(model));
501     Llanta1_M.RenderModel();
502
503     model = modelaux;
504     //Llanta2
505     model = glm::translate(model, glm::vec3(-20.0f, 0.0f, 0.0f));
506     modelaux = model;
507     glUniformMatrix4fv(uniformModel, 1, GL_FALSE, glm::value_ptr(model));
508     Llanta2_M.RenderModel();
509
510     model = modelaux;
511     //Llanta3
512     model = glm::translate(model, glm::vec3(0.0f, 0.0f, -30.0f));
513     modelaux = model;
514     glUniformMatrix4fv(uniformModel, 1, GL_FALSE, glm::value_ptr(model));
515     Llanta3_M.RenderModel();
516
517     model = modelaux;
518     //Llanta4
519     model = glm::translate(model, glm::vec3(20.0f, 0.0f, 0.0f));
520     modelaux = model;
521     glUniformMatrix4fv(uniformModel, 1, GL_FALSE, glm::value_ptr(model));
522     Llanta4_M.RenderModel();
523
```

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

Primero texturizamos nuestros objetos previamente separados en 3d max como el procedimiento anterior usando la siguiente imagen:



Luego, exportamos los objetos (cofre, toldo y parabrisas) para después llamarlos dentro de Visual Studio.

```
46     Model CarroBase_M;
47     Model Cofre_M;
48     Model Toldo_M;
49     Model Parabrisas_M;
50     Model Llanta1_M;
51     Model Llanta2_M;
52     Model Llanta3_M;
53     Model Llanta4_M;
54     Model Cofre_Tex;
55     Model Toldo_Tex;
56     Model Parabrisas_Tex;
57     Model Llanta1_Tex;

372     CarroBase_M = Model();
373     CarroBase_M.LoadModel("Models/carro_base.obj");
374     Cofre_M = Model();
375     Cofre_M.LoadModel("Models/cofre_tex.obj");
376     Toldo_M = Model();
377     Toldo_M.LoadModel("Models/toldo_tex.obj");
378     Parabrisas_M = Model();
379     Parabrisas_M.LoadModel("Models/parabrisas_tex.obj");
380     Llanta1_M = Model();
381     Llanta1_M.LoadModel("Models/llanta1_tex.obj");
```

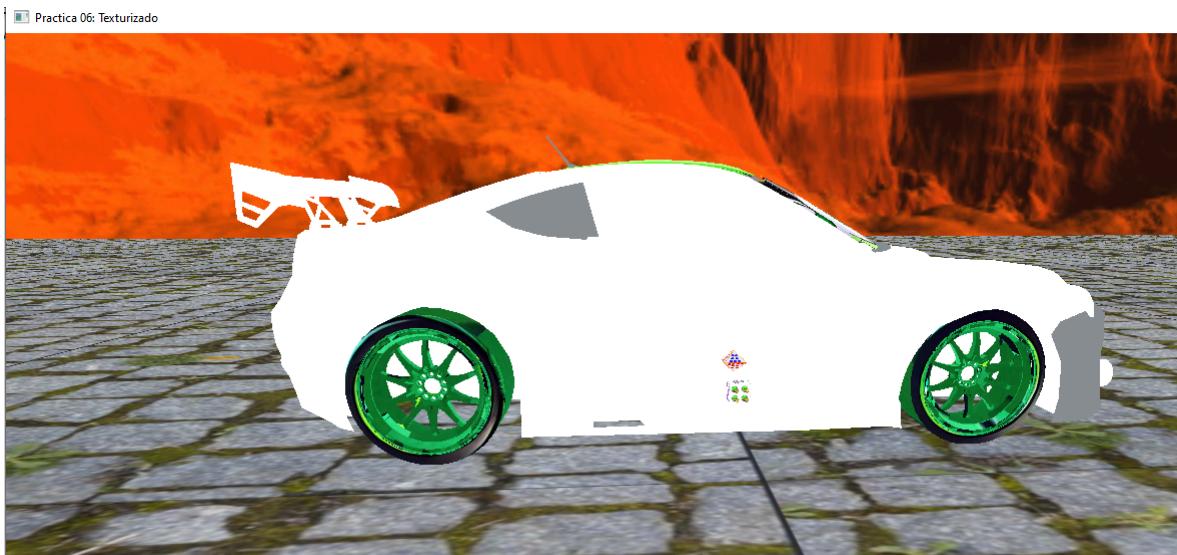
Luego, colocamos nuestros modelos dentro del *main* siguiendo la jerarquía.

```
489     model = modelaux;
490     //Cofre
491     model = glm::translate(model, glm::vec3(0.0f, 20.0f, 10.0f));
492     modelaux = model;
493     glUniformMatrix4fv(uniformModel, 1, GL_FALSE, glm::value_ptr(model));
494     Cofre_M.RenderModel();
```

```
523     model = modelaux;
524     //Parabrisas
525     model = glm::translate(model, glm::vec3(-9.0f, 1.0f, 17.0f));
526     modelaux = model;
527     glUniformMatrix4fv(uniformModel, 1, GL_FALSE, glm::value_ptr(model));
528     Parabrisas_M.RenderModel();
529
530     model = modelaux;
531     //Toldo
532     model = glm::translate(model, glm::vec3(0.0f, 0.0f, 0.0f));
533     modelaux = model;
534     glUniformMatrix4fv(uniformModel, 1, GL_FALSE, glm::value_ptr(model));
535     Toldo_M.RenderModel();
536
```

La ejecución queda de la siguiente manera:





Problemáticas

La primera problemática que surgió es que dentro de 3d max al volver a aplicar una textura a un objeto, la escala de este se volvía muy pequeña, la solución fue refrescar la escala en UV Editor y el objeto volvía a un tamaño normal.

La segunda problemática que surgió fue que al ejecutar el programa en otra computadora los modelos texturizados exportados aparecían pero sin textura, esto fue porque al momento de colocar los archivos necesarios para ejecutar, no se encontraban las texturas usadas, por lo que la solución fue agregar las imágenes de textura para que los modelos tuvieran visible su texturizado.

Conclusiones

El desarrollo de esta práctica permitió una introducción muy completa del texturizado tanto por código como por modelos, dándonos las herramientas necesarias para modificar las imágenes con las que se implementará el procedimiento como para exportar los modelos con las texturas. La complejidad se me hizo adecuada, con sus retos y problemáticas, pero solucionables por la cantidad de tiempo brindada para cada una de las actividades.

Bibliografía

- OpenGL Tutorial. (n.d.). Tutorial 5: A textured cube. OpenGL Tutorial.
<http://www.opengl-tutorial.org/es/beginners-tutorials/tutorial-5-a-textured-cube/>