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 09

NOMBRE COMPLETO: Aguilar Pérez José Ramón

Nº de Cuenta: 317515048

GRUPO DE LABORATORIO: 02

GRUPO DE TEORÍA: 04

SEMESTRE 2025-2

FECHA DE ENTREGA LÍMITE: 26/abril/2025

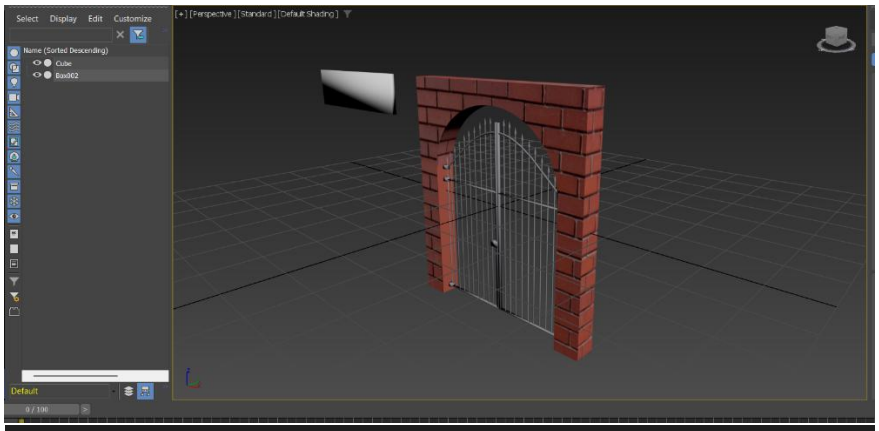
CALIFICACIÓN: _____

REPORTE DE PRÁCTICA:

1.- Ejercicios realizados.

- **1. Separar del arco la parte del letrero.**
- **2. Hacer que en el arco que crearon se muestre la palabra: PROYECTO CGEIHC Feria animado desplazándose las letras de izquierda a derecha como si fuera letrero LCD/LED de forma cíclica.**
- **3. Separar las cabezas (con todo y cuello) del Dragón y agregar las siguientes animaciones:**
  - **Movimiento del cuerpo ida y vuelta.**
  - **Aleteo**
  - **Cada cabeza se mueve de forma diferente de acuerdo a una función/algoritmo diferente (ejemplos: espiral de Arquímedes, movimiento senoidal, lemniscata, etc.…)**
  - **Cada cabeza debe de verse de un color diferente: roja, azul, verde, blanco, café.**

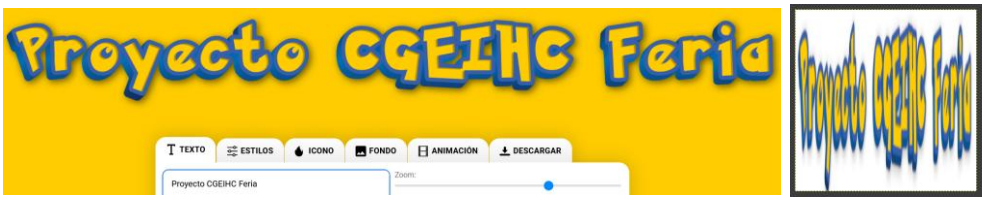
Para el primer ejercicio, se optimizó el arco generado en el previo, ya que se malinterpretó las instrucciones (se pensaba que la puerta debía tener forma de arco) por lo que se agregó el modelo del arco, el cual va separado del modelo del letrero. Dentro del programa, se instancian estos modelos por medio de jerarquía.




```
//Instancia del arco
model = glm::mat4(1.0);
model = glm::translate(model, glm::vec3(3.0f, 3.9f, -3.0f));
modelaux = model;
model = glm::scale(model, glm::vec3(0.2f, 0.2f, 0.2f));
glUniformMatrix4fv(uniformModel, 1, GL_FALSE, glm::value_ptr(model));
Arco_M.RenderModel();

//Instancia del letrero
model = modelaux;
model = glm::translate(model, glm::vec3(0.0f, 5.2f, 0.5f));
modelaux = model;
model = glm::scale(model, glm::vec3(0.2f, 0.2f, 0.2f));
glUniformMatrix4fv(uniformModel, 1, GL_FALSE, glm::value_ptr(model));
Letrero_M.RenderModel();
```

Después, en el segundo ejercicio se generó el texto que se iba a mostrar en el letrero del arco, esto con ayuda de una página web que permite escribir con la tipografía del universo elegido: Pokémon. Una vez se generó el texto, se optimizó la imagen a un formato válido para OpenGL así como se escaló a un tamaño de $2^n x 2^n$.



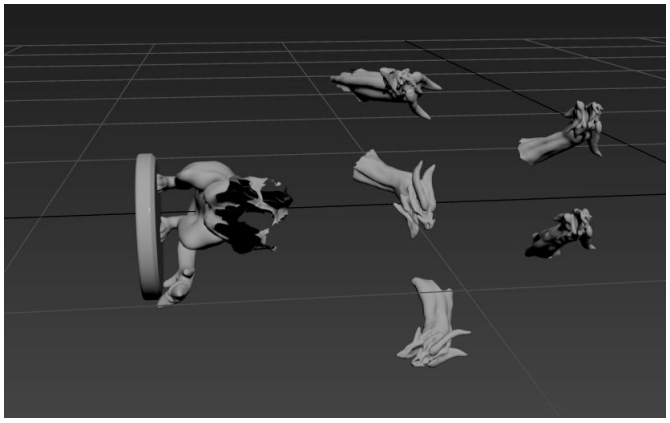
Dentro del main, se agrega la textura con movimiento del letrero. Por medio de *toffsetletrerou* se va desplazando la imagen a un ritmo de 0.001 unidades en el eje U de la textura, por lo que se hace un efecto de desplazamiento. Además, se añade la condicional para evitar el desbordamiento al llegar al límite de 1 en el eje U, reiniciando así la animación y permitiendo un efecto cíclico en el letrero.

```
//Letrero cpn movimiento
toffsetletrerou += 0.001;
toffsetletrerov = 0.000;
if (toffsetletrerou > 1.0)
    toffsetletrerou = 0.0;
toffset = glm::vec2(toffsetletrerou, toffsetletrerov);

model =modelaux;
model = glm::translate(model, glm::vec3(0.0f, 0.0f, 0.1f));
model = glm::rotate(model, 90 * toRadians, glm::vec3(1.0f, 0.0f, 0.0f));
model = glm::scale(model, glm::vec3(4.9f, 2.0f, 1.35f));
glUniform2fv(uniformTextureOffset, 1, glm::value_ptr(toffset));
glUniformMatrix4fv(uniformModel, 1, GL_FALSE, glm::value_ptr(model));

LetreroTexture.UseTexture();
meshList[4]->RenderMesh();
```

Para el último ejercicio, se separaron las cabezas del dragón con todo y cuello para poder animarlas por separado. Se establecieron los pivotes para cada cabeza en una posición (la base del cuello) que permita que el movimiento sea congruente.



Las cabezas, así como las alas, se instanciaron por medio de jerarquía partiendo de base el cuerpo principal de Tiamat. El color de cada cabeza se estableció por medio de *UniformColor* evitando así el uso de texturas extras.

Para las animaciones se utilizaron principalmente dos técnicas: para las dos cabezas superiores se utilizaron animación básica por medio de condicionales (similar a la animación implementada para el aleteo hecho en el ejercicio de clase), al llegar a cierto ángulo, se invierte el sentido de giro.

```
//Animación simple de las cabezas
rotCabeza += rotCabezaOffset * deltaTime;
// Cambia de dirección al alcanzar los límites
if (rotCabeza > 10.0f) {
    rotCabeza = 10.0f; //Limite para evitar atasco
    rotCabezaOffset *= -1.0f;
}
else if (rotCabeza < -10.0f) {
    rotCabeza = -10.0f; //Limite para evitar atasco
    rotCabezaOffset *= -1.0f;
}
```

```
//Cabeza roja
model = modelaux;
model = glm::translate(model, glm::vec3(-2.0f, 0.7f, 3.63f));
model = glm::rotate(model, rotCabeza * toRadians, glm::vec3(0.0f, 1.0f, 0.1));
Material_brillante.UseMaterial(uniformSpecularIntensity, uniformShininess);
glUniformMatrix4fv(uniformModel, 1, GL_FALSE, glm::value_ptr(model));
color = glm::vec3(1.0f, 0.0f, 0.0f);
glUniform3fv(uniformColor, 1, glm::value_ptr(color));
Cabeza1_M.RenderModel();

//Cabeza azul
model = modelaux;
model = glm::translate(model, glm::vec3(-2.02f, -0.305f, 3.6f));
model = glm::rotate(model, rotCabeza * toRadians, glm::vec3(0.1f, 0.0f, 0.0));
Material_brillante.UseMaterial(uniformSpecularIntensity, uniformShininess);
glUniformMatrix4fv(uniformModel, 1, GL_FALSE, glm::value_ptr(model));
color = glm::vec3(0.0f, 0.0f, 1.0f);
glUniform3fv(uniformColor, 1, glm::value_ptr(color));
Cabeza2_M.RenderModel();
```

Mientras que, para las animaciones de las cabezas inferiores, su movimiento depende de la función seno, la cual recibe el parámetro del rotamiento de la cabeza para así evitar que choquen entre sí. Cabe añadir que cada cabeza se mueve en torno a eje/ejes distintos, esto con el fin de que cada cabeza tenga un movimiento diferente entre sí.

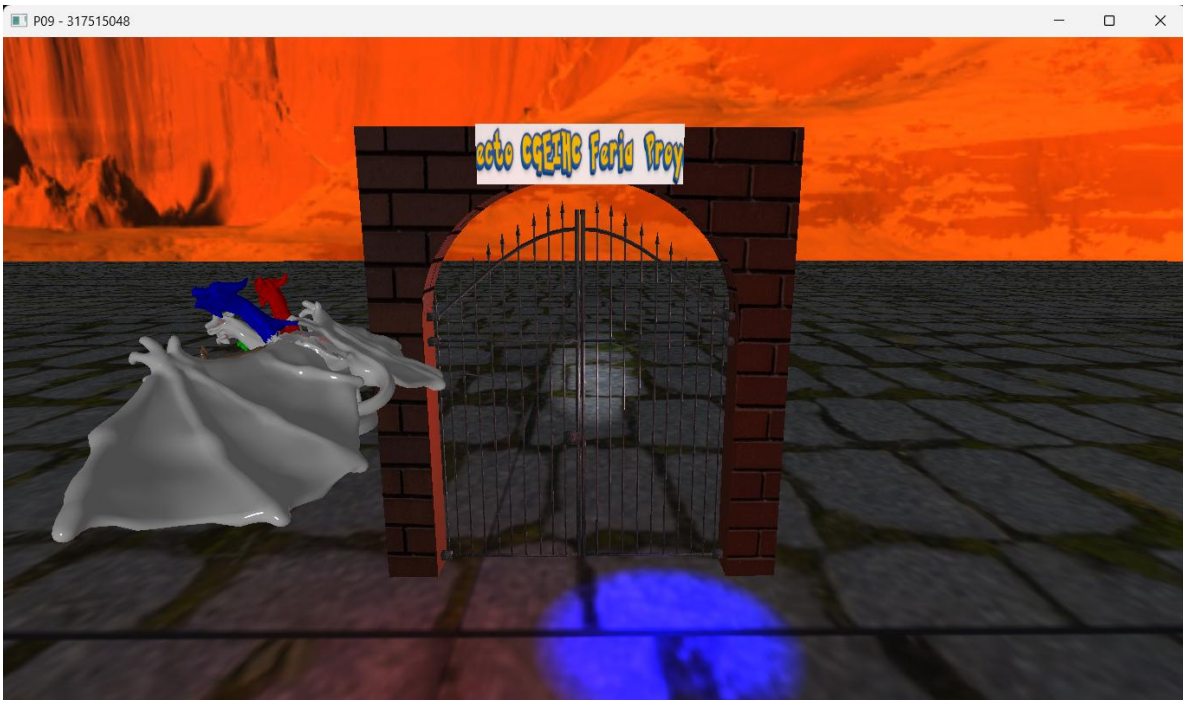
```
//Cabeza verde
model = modelaux;
model = glm::translate(model, glm::vec3(-2.12f, 0.85f, 2.61f));
model = glm::rotate(model, 2 * sin(glm::radians(rotCabeza)), glm::vec3(1.0f, 0.0f, 1.0));
Material_brillante.UseMaterial(uniformSpecularIntensity, uniformShininess);
glUniformMatrix4fv(uniformModel, 1, GL_FALSE, glm::value_ptr(model));
color = glm::vec3(0.0f, 1.0f, 0.0f);
glUniform3fv(uniformColor, 1, glm::value_ptr(color));
Cabeza3_M.RenderModel();

//Cabeza blanco
model = modelaux;
model = glm::translate(model, glm::vec3(-2.15f, 0.2f, 2.65f));
model = glm::rotate(model,2 * sin(glm::radians(rotCabeza)), glm::vec3(0.0f, 1.0f, 0.0));
Material_brillante.UseMaterial(uniformSpecularIntensity, uniformShininess);
glUniformMatrix4fv(uniformModel, 1, GL_FALSE, glm::value_ptr(model));
color = glm::vec3(1.0f, 1.0f, 1.0f);
glUniform3fv(uniformColor, 1, glm::value_ptr(color));
Cabeza4_M.RenderModel();

//Cabeza cafe
model = modelaux;
model = glm::translate(model, glm::vec3(-2.15f, -0.5f, 2.7f));
model = glm::rotate(model, 2 * sin(glm::radians(rotCabeza)), glm::vec3(0.0f, -1.0f, 1.0));
Material_brillante.UseMaterial(uniformSpecularIntensity, uniformShininess);
glUniformMatrix4fv(uniformModel, 1, GL_FALSE, glm::value_ptr(model));
color = glm::vec3(0.6f, 0.45f, 0.35f);
glUniform3fv(uniformColor, 1, glm::value_ptr(color));
Cabeza5_M.RenderModel();
```

Finalmente, para el movimiento general del dragón se estableció una bandera boolena llamada *avanza*, la cual varía entre verdadero y falso cada que llega al límite del escenario, permitiendo así que Tiamat cambie de sentido y no se salga del espacio delimitado por el piso. Este movimiento es registrado en la variable *movDragon*, al llegar a -280 empieza a moverse en sentido contrario y al llegar a 290 sucede lo mismo. Este movimiento traslacional es sinusoidal.

```
//Animacion basica para Tiamat
if (avanza) {
    if (movDragon > -280.0f) {
        movDragon -= movOffset * deltaTime;
    }
    else {
        avanza = false;
    }
}
else {
    if (movDragon < 290.0f) {
        movDragon += movOffset * deltaTime;
    }
    else {
        avanza = true;
    }
}
```



Da clic en la imagen para ver el funcionamiento del programa :D.

2.- Problemas presentados

No se presentaron problemas a la hora de realizar los ejercicios solicitados.

3.- Conclusión:

a. Los ejercicios del reporte: Complejidad, Explicación.

La elaboración de esta práctica fue muy interesante de realizar, ya que permitió comprender de mejor manera como se pueden aplicar los distintos tipos de animación presentes en OpenGL, como puede ser la básica, por medio de texturas y por medio de funciones.

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

La explicación de cómo funcionan los distintos tipos de animaciones en el ambiente de OpenGL fueron claras y precisas.

c. Conclusión

Esta práctica me permitió comprender mejor cómo funcionan las animaciones dentro de OpenGL, así como los distintos tipos de estas. Por un lado, se tienen las animaciones básicas/simples, las cuales trabajan por medio de condicionales y banderas que permiten cambiar la animación en un momento dado. Después se encuentran las animaciones por textura, en estas por medio de un offset se va recorriendo la textura en sus ejes UV permitiendo así el efecto animado. Finalmente, se trabajo con animaciones que trabajan con funciones, como lo fue el seno, el cual permite un movimiento más fluido. Es importante añadir que un elemento importante para que las animaciones funcionen es la variable de tipo *deltaTime*, la cual permite sincronizar los tiempos de reloj sin importar en el equipo en el que se trabaje.

1. Bibliografía en formato APA

- 3ds Max Quick Start Guide. (s/f). Autodesk.com. Recuperado el 20 de marzo de 2025, de https://www.autodesk.com/learn/ondemand/curated/3ds-max-quick-start-guide

- Archway - Download Free 3D model by brysondurham18. (2024, marzo 19). https://sketchfab.com/3d-models/archway-c605b0f2414e487787f3462c1d363545

- ¿Por qué necesitamos utilizar Delta Time? (2017, octubre 25). Parallelcube.com. https://www.parallelcube.com/es/2017/10/25/por-que-necesitamos-utilizar-delta-time/

- Texto 3D de Pokémon. (s/f). Textstudio.com. Recuperado el 20 de abril de 2025, de https://es.textstudio.com/logo/texto-3d-de-pokemon-318