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: Mino Guzmán Yara Amairani

Nº de Cuenta: 422017028

GRUPO DE LABORATORIO: 03

GRUPO DE TEORÍA: 04

SEMESTRE 2025-2

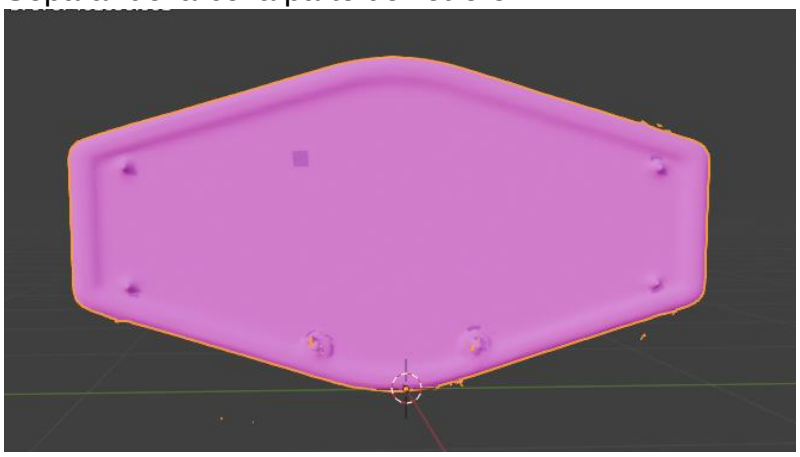
FECHA DE ENTREGA LÍMITE: 26 abril 2025

CALIFICACIÓN: _____

REPORTE DE PRÁCTICA:

1.- Actividades:

- Separar del arco la parte del letrero.



Se separó el letrero de mi puerta para poder agregarlo jerárquicamente en OpenGL.

```
Model Puerta_M;
Model Letrero_M;
```

Definí los nombres de mis modelos para la puerta y el letrero.

```
Puerta_M = Model();
Puerta_M.LoadModel("Models/puertarejas.obj");
Letrero_M = Model();
Letrero_M.LoadModel("Models/letrero.obj");
```

Cargué los archivos .obj para poder aplicar la jerarquización de los modelos.

```
//Agregar puerta letrero de manera jerarquica
model = glm::mat4(1.0f);
model = glm::translate(model, glm::vec3(8.0f, -1.9f, 2.0f));
modelaux = model;
model = glm::scale(model, glm::vec3(6.0f, 6.0f, 6.0f));
model = glm::rotate(model, glm::radians(270.0f), glm::vec3(0.0f, 1.0f, 0.0f));
glUniformMatrix4fv(uniformModel, 1, GL_FALSE, glm::value_ptr(model));
Puerta_M.RenderModel();

model = modelaux;
model = glm::translate(model, glm::vec3(0.0f, 10.0f, 0.8f));
model = glm::scale(model, glm::vec3(5.0f, 5.0f, 5.0f));
model = glm::rotate(model, glm::radians(270.0f), glm::vec3(0.0f, 1.0f, 0.0f));
glUniformMatrix4fv(uniformModel, 1, GL_FALSE, glm::value_ptr(model));
Letrero_M.RenderModel();
```

Rendericé mis modelos aplicando la jerarquización.

Ejecución:

- Hacer que en el arco que crearon se muestre la palabra: PROYECTO CGEIHC FERIA animado desplazándose las letras de derecha a izquierda como si fuera letrero LCD/LED de forma cíclica.

```
//Variables para la animación del letrero
float toffsetLetrau = 0.0f;
float toffsetLetrav = 0.0f;
//Arreglos para coordenadas U,V que nos guardan la posición de las letras a imprimir
//PROYECTO CGEIHC FERIA---> 19 caracteres (sin contar espacios)
float incrementoTipografia_U[] = {
    0.281f, 0.567f, 0.142f, 0.68f, 0.60f, 0.315f, 0.858f, 0.142f,       //PROYECTO
    0.315f, 0.85f, 0.60f, 0.143f, 0.04f, 0.315f,                        //CGEIHC
    0.7f, 0.572f, 0.567f, 0.143f, 0.02f                                 //FERIA
};

float incrementoTipografia_V[] = {
    0.5f, 0.5f, 0.5f, 0.75f, 0.0f, 0.0f, 0.5f, 0.5f,                    //PROYECTO
    0.0f, 0.0f, 0.0f, 0.25f, 0.25f, 0.0f,                               //CGEIHC
    0.0f, 0.0f, 0.5f, 0.25f, 0.0f                                       //FERIA
};
```

Definí mis variables para las letras y los arreglos para poder guardar la posición de cada letra.

```
Texture LetrasTexture;
```

Definí el nombre de la textura de mis letras.

```
GLfloat letraVertices[] = {
-0.5f, 0.0f, 0.5f,     0.0f, 0.75f,      0.0f, -1.0f, 0.0f,
 0.5f, 0.0f, 0.5f,     0.143f, 0.75f,    0.0f, -1.0f, 0.0f,
 0.5f, 0.0f, -0.5f,    0.143f,1.0f,      0.0f, -1.0f, 0.0f,
-0.5f, 0.0f, -0.5f,    0.0f, 1.0f,       0.0f, -1.0f, 0.0f,
};
```

También en la función de void CreateObjects(), cree letraVertices, para no confundirme con el de numeroVertices.

```
LetrasTexture = Texture("Textures/letras.png");
LetrasTexture.LoadTextureA();
```

Cargué la textura de mi imagen, la imagen utilizada es la siguiente:


```
//Dibujo de las letras animadas tipo letrero LED/LCD
static int palabraActual = 0;  // 0 = PROYECTO, 1 = CGEIHC, 2 = FERIA
static float tiempoLetras = 0.0f;
float intervaloLetras = 10.0f;  // Tiempo entre aparición de letras

static int letrasMostradas = 1;  // Letras visibles por palabra
static float desplazamientoLetras = 0.0f;
float velocidadDesplazamiento = 0.00001f * deltaTime;
desplazamientoLetras -= velocidadDesplazamiento;

// Cambio de letra por tiempo
tiempoLetras += deltaTime;
if (tiempoLetras >= intervaloLetras) {
    letrasMostradas++;
    int totalLetrasPalabra = (palabraActual == 0) ? 8 : (palabraActual == 1) ? 6 : 5;

    if (letrasMostradas > totalLetrasPalabra) {
        letrasMostradas = 1;  // Reinicia letras visibles para la siguiente palabra
        palabraActual = (palabraActual + 1) % 3;  // Cambia de palabra
        desplazamientoLetras = 0.0f;  // Reinicia desplazamiento
    }

    tiempoLetras = 0.0f;
}
```

```
// Configuración de posición y escala
float centroLetreroX = 8.0f;
float alturaBaseLetrero = 11.5f;
float profundidadLetrero = 3.0f;
float espacioEntreLetras = 0.8f;
float escalaLetra = 1.0f;

toffsetLetrau = 0.0f;
toffsetLetrav = 0.0f;

int inicioLetra = 0;
float alturaLetrero = alturaBaseLetrero;
int totalLetras = 0;

if (palabraActual == 0) {  // PROYECTO
    inicioLetra = 1;
    alturaLetrero = alturaBaseLetrero;
    totalLetras = 8;
}
else if (palabraActual == 1) {  // CGEIHC
    inicioLetra = 9;
    alturaLetrero = alturaBaseLetrero - 1.1f;
    totalLetras = 6;
}
```

```
else {  // FERIA
    inicioLetra = 15;
    alturaLetrero = alturaBaseLetrero - 2.1f;
    totalLetras = 5;
}

int primeraLetra = inicioLetra;
int ultimaLetra = inicioLetra + letrasMostradas - 1;

// Centrado horizontal
float anchoLinea = (totalLetras - 1) * espacioEntreLetras;
float desplazamientoInicialLinea = centroLetreroX - (anchoLinea / 2.0f);

//  Recorrer de derecha a izquierda dentro de cada palabra
for (int k = ultimaLetra; k >= primeraLetra; k--) {
    int posicionEnLinea = k - inicioLetra;
    float xOffset = desplazamientoInicialLinea + (posicionEnLinea * espacioEntreLetras) + desplazamientoLetras;

    toffsetLetrau += incrementoTipografia_U[k - 1];
    toffsetLetrav -= incrementoTipografia_V[k - 1];
```

```
    toffset = glm::vec2(toffsetLetrau, toffsetLetrav);
    model = glm::mat4(1.0);
    model = glm::translate(model, glm::vec3(xOffset, alturaLetrero, profundidadLetrero));
    model = glm::rotate(model, 90 * toRadians, glm::vec3(1.0f, 0.0f, 0.0f));
    model = glm::scale(model, glm::vec3(escalaLetra, escalaLetra, escalaLetra));
    glUniform2fv(uniformTextureOffset, 1, glm::value_ptr(toffset));
    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));
    LetrasTexture.UseTexture();
    Material_brillante.UseMaterial(uniformSpecularIntensity, uniformShininess);
    meshList[7]->RenderMesh();

    // Reinicia UV
    toffsetLetrau -= incrementoTipografia_U[k - 1];
    toffsetLetrav += incrementoTipografia_V[k - 1];
}
```

Dibujé las letras requeridas para el letrero para que se mostraran de izquierda a derecha y pareciera un letrero LCD/LED.

La ejecución del ejercicio se mostrará en un video en clasroom.

- Separar las cabezas (con todo y cuello) del Dragón y agregar las siguientes animaciones:

```
Model Rojo_M;
Model Azul_M;
Model Verde_M;
Model Blanco_M;
Model Cafe_M;
```

Debido a que en el ejercicio se pidió que se separaran las alas del dragón, lo único que cargué para la práctica fue cada cabeza por separado.

```
Rojo_M = Model();
Rojo_M.LoadModel("Models/CabezaRoja.obj");
Azul_M = Model();
Azul_M.LoadModel("Models/CabezaAzul.obj");
Verde_M = Model();
Verde_M.LoadModel("Models/CabezaVerde.obj");
Blanco_M = Model();
Blanco_M.LoadModel("Models/CabezaBlanca.obj");
Cafe_M = Model();
Cafe_M.LoadModel("Models/CabezaCafe.obj");
```

Cargué los modelos .obj para jerarquizar mis alas de acuerdo con el cuerpo del dragón.

- Movimiento del cuerpo ida y vuelta.

```
float movDragon;        //Control del movimiento para que avance y regrese
bool avanzaDragon;      //Bandera para avanzar o retroceder Dragon
```

Definí las variables para hacer que el Dragón avance o retroceda.

```
movOffset = 0.05f;
```

Ajusté la velocidad para que no se viera tan lento el avance y retroceso del dragón.

```
//Variables para el movimiento del dragon
movDragon = 0.0f;
avanzaDragon = true;
```

Declaré mis variables para el movimiento del dragón utilizando un booleano para el avance del dragón.

```
model = glm::mat4(1.0);
model = glm::translate(model, glm::vec3(movDragon + 0.0f, 5.0f + sin(glm::radians(angulovaria)), 6.0f));
modelaux = model;
model = glm::scale(model, glm::vec3(0.5f, 0.5f, 0.5f));
Material_brillante.UseMaterial(uniformSpecularIntensity, uniformShininess);
glUniformMatrix4fv(uniformModel, 1, GL_FALSE, glm::value_ptr(model));
Dragon_M.RenderModel();
```

Agregué el cuerpo del dragón, agregando movDragon en X y el translate con sin en y se quedó igual.
La ejecución se mostrará en un video en Classroom.

- Aleteo.

El aleteo se realizó en el ejercicio.

```
float wingAngle = 0.0f;
bool wingUp = true;
```

Definí las variables para el aleteo de las alas del dragón.

```
//Aleteo
if (wingUp) {
    wingAngle += 5.0f * deltaTime;
    if (wingAngle >= 30.0f) wingUp = false;
}
else {
    wingAngle -= 5.0f * deltaTime;
    if (wingAngle <= -30.0f) wingUp = true;
}

if (avanzaDragon) {
    if (movDragon > -100.0f) {
        movDragon -= movOffset * deltaTime;
    }
    else {
        avanzaDragon = false;
    }
}
else if (movDragon < 100.0f) {
    movDragon += movOffset * deltaTime;
}
else {
    avanza = true;
}
```

Para el aleteo de mis alas, cree una condición if para que las alas subieran y bajaran, haciendo el movimiento de aleteo de un dragón.

```
model = modelaux;
model = glm::translate(model, glm::vec3(-0.38f, 1.8f, 0.1f));
model = glm::rotate(model, glm::radians(wingAngle), glm::vec3(1.0f, 0.0f, 0.0f));
model = glm::scale(model, glm::vec3(0.5f, 0.5f, 0.5f));
glUniformMatrix4fv(uniformModel, 1, GL_FALSE, glm::value_ptr(model));
AlaIzq_M.RenderModel();

model = modelaux;
model = glm::translate(model, glm::vec3(-0.4f, 1.6f, -0.1f));
model = glm::rotate(model, glm::radians(-wingAngle), glm::vec3(1.0f, 0.0f, 0.0f));
model = glm::scale(model, glm::vec3(0.5f, 0.5f, 0.5f));
glUniformMatrix4fv(uniformModel, 1, GL_FALSE, glm::value_ptr(model));
AlaDer_M.RenderModel();
```

Jerarquicé mis alas y agregué una rotación con wingAngle en X, para que se hiciera el movimiento de aleteo.

La ejecución se mostrará en un video en Classroom.

- Cada cabeza se mueve de forma diferente de acuerdo con una función/algoritmo diferente (ejemplos: espiral de Arquímedes, movimiento senoidal, lemniscata, etc.)

```
//Variables para animacion espiral de la cabeza roja
float tCabezaRoja = 0.0f;  // Tiempo acumulado para la cabeza roja
float radioBaseRoja = 0.0f;  // Radio base de la espiral
float incrementoRadioRoja = 0.0f;  // Cuánto se incrementa el radio por vuelta
float velocidadAngularRoja = 0.0f;  // Velocidad de rotación de la espiral
//Variables para animacíon senoidal de la cabeza azul
float tCabezaAzul = 0.0f;
float velocidadAngularAzul = 0.0f;  // Más lento que la roja
//Variables para animacion eliptica de la cabeza verde
float tCabezaVerde = 0.0f;
float velocidadAngularVerde = 0.0f;
//Variables para animacion lemnistica de la cabeza blanca
float tCabezaBlanca = 0.0f;
float velocidadAngularBlanca = 0.0f;
//Variables para animacion con movimiento aleatorio de la cabeza cafe
float tCabezaCafe = 0.0f;
float velocidadAngularCafe = 0.0f;
float objetivoRotacionCafe = 0.0f;
float rotacionActualCafe = 0.0f;
```

Definí las variables para los movimientos de mis cabezas.

```
//Cabeza roja
// Parámetros locales ajustados directamente (más lentos)
velocidadAngularRoja = 0.1f;
radioBaseRoja = 0.05f;
incrementoRadioRoja = 0.005f;
// Actualizar tiempo para la espiral de Arquímedes
tCabezaRoja += velocidadAngularRoja * deltaTime;
// Calcular ángulo y radio para la espiral
float anguloRoja = tCabezaRoja;
float radioRoja = radioBaseRoja + incrementoRadioRoja * tCabezaRoja;
// Movimiento lento de la cabeza roja: rotación sobre eje Y
float rotacionCabezaRoja = sin(anguloRoja) * 10.0f;  // Suave

//Cabeza azul
// Parámetro local ajustado directamente (más lento)
velocidadAngularAzul = 0.2f;  // Controla qué tan rápido se balancea
// Animación senoidal para la cabeza azul
tCabezaAzul += velocidadAngularAzul * deltaTime;
float rotacionCabezaAzul = sin(tCabezaAzul) * 15.0f;  // Movimiento lateral tipo péndulo

//Cabeza verde
// Parámetro local ajustado directamente (más lento que la azul)
velocidadAngularVerde = 0.15f;
// Animación elíptica para la cabeza verde (rota en X y Z simultáneamente)
tCabezaVerde += velocidadAngularVerde * deltaTime;
float rotacionCabezaVerdeX = sin(tCabezaVerde) * 8.0f;  // Oscilación en X
float rotacionCabezaVerdeZ = cos(tCabezaVerde) * 8.0f;  // Oscilación en Z
```

```
//Cabeza blanca
// Parámetro local ajustado directamente (más lento)
velocidadAngularBlanca = 0.05f;
// Animación tipo lemniscata para la cabeza blanca ( figura de 8)
tCabezaBlanca += velocidadAngularBlanca * deltaTime;
// Movimiento en lemniscata: (x = a * sin(t)) (z = a * sin(t) * cos(t))
float a = 0.1f;  // Escala de la figura
float rotacionCabezaBlancaY = sin(tCabezaBlanca) * 10.0f;  // Rotación suave en Y

//Cabeza cafe
// Parámetros locales ajustados directamente
velocidadAngularCafe = 0.1f;
tCabezaCafe += deltaTime;
// Cada cierto tiempo, cambiar el objetivo aleatorio
if (fmod(tCabezaCafe, 5.0f) < deltaTime) {  // Cada 5 segundos aprox.
    objetivoRotacionCafe = ((rand() % 20) - 10);  // Entre -10 y 10 grados
}
// Suavizar la rotación actual hacia el objetivo
rotacionActualCafe += (objetivoRotacionCafe - rotacionActualCafe) * 0.02f;  // Suavizado lento
```

Cree los parámetros para que los movimientos de las cabezas de mi dragón se vieran como lo pedido para el reporte de esta práctica.

```
model = modelaux;
model = glm::translate(model, glm::vec3(-1.15f, 1.32f, 0.19f));  // Posición base
model = glm::rotate(model, glm::radians(rotacionCabezaRoja), glm::vec3(0.0f, 1.0f, 0.0f));
model = glm::scale(model, glm::vec3(0.5f, 0.5f, 0.5f));
Material_brillante.UseMaterial(uniformSpecularIntensity, uniformShininess);
glUniformMatrix4fv(uniformModel, 1, GL_FALSE, glm::value_ptr(model));
Rojo_M.RenderModel();

model = modelaux;
model = glm::translate(model, glm::vec3(-1.32f, 1.51f, -0.13f));  // Posición base
model = glm::rotate(model, glm::radians(rotacionCabezaVerdeX), glm::vec3(1.0f, 0.0f, 0.0f));  // Rotación en X
model = glm::rotate(model, glm::radians(rotacionCabezaVerdeZ), glm::vec3(0.0f, 0.0f, 1.0f));  // Rotación en Z
model = glm::scale(model, glm::vec3(0.5f, 0.5f, 0.5f));
Material_brillante.UseMaterial(uniformSpecularIntensity, uniformShininess);
glUniformMatrix4fv(uniformModel, 1, GL_FALSE, glm::value_ptr(model));
Verde_M.RenderModel();

model = modelaux;
model = glm::translate(model, glm::vec3(-1.23f, 1.31f, -0.49f));  // Posición base
model = glm::rotate(model, glm::radians(rotacionCabezaAzul), glm::vec3(0.0f, 0.0f, 1.0f));  // Balanceo sobre Z
model = glm::scale(model, glm::vec3(0.5f, 0.5f, 0.5f));
Material_brillante.UseMaterial(uniformSpecularIntensity, uniformShininess);
glUniformMatrix4fv(uniformModel, 1, GL_FALSE, glm::value_ptr(model));
Azul_M.RenderModel();
```

```
model = modelaux;
model = glm::translate(model, glm::vec3(-1.08f, 1.9f, -0.42f));  // Posición base
model = glm::rotate(model, glm::radians(rotacionActualCafe), glm::vec3(0.0f, 1.0f, 0.0f));  // Rotación aleatoria lateral
model = glm::scale(model, glm::vec3(0.5f, 0.5f, 0.5f));
Material_brillante.UseMaterial(uniformSpecularIntensity, uniformShininess);
glUniformMatrix4fv(uniformModel, 1, GL_FALSE, glm::value_ptr(model));
Cafe_M.RenderModel();

model = modelaux;
model = glm::translate(model, glm::vec3(-1.05f, 1.88f, 0.18f));  // Posición FIJA
model = glm::rotate(model, glm::radians(rotacionCabezaBlancaY), glm::vec3(0.0f, 1.0f, 0.0f));  // Solo gira, sin moverse
model = glm::scale(model, glm::vec3(0.5f, 0.5f, 0.5f));
Material_brillante.UseMaterial(uniformSpecularIntensity, uniformShininess);
glUniformMatrix4fv(uniformModel, 1, GL_FALSE, glm::value_ptr(model));
Blanco_M.RenderModel();
```

Agregué las rotaciones necesarias a cada cabeza de mi dragón para que realizaran el movimiento que les asigné.

La ejecución se mostrará en un video en classroom.

- Cada cabeza debe de verse de un color diferente: roja, azul, verde, blanco y café.

  Al separar cada cabeza del cuerpo de mi dragón, cree un .obj para cada una de ellas, posteriormente, realicé el texturizado de acuerdo con los colores solicitados.

2.- Problemas a la hora de hacer las actividades solicitadas:

- Me costó trabajo realizar la animación de mi letrero, ya que como mi imagen no estaba 100% pareja en filas ni columnas, tuve que realizar una medición entre 0 y 1 y darle medida a cada letra, para así poderlas acomodar.

3.- Conclusión:

a. Los ejercicios del reporte fueron los adecuados para poder comprender el uso de la animación en mis objetos.
b. Las explicaciones fueron claras y, gracias a eso, se realizó con éxito la práctica.
c. En conclusión, la práctica me ayudó a explorar nuevas cosas de la computación gráfica, en especial de la animación de mis objetos que, sin duda es importante para realizar el proyecto.