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: García Soto Jean Carlo

Nº de Cuenta: 319226304

GRUPO DE LABORATORIO: 03

GRUPO DE TEORÍA: 04

SEMESTRE 2025-2

FECHA DE ENTREGA LÍMITE: 26/04/2025

CALIFICACIÓN: _____

REPORTE DE PRÁCTICA:

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. Separar del arco la parte del letrero

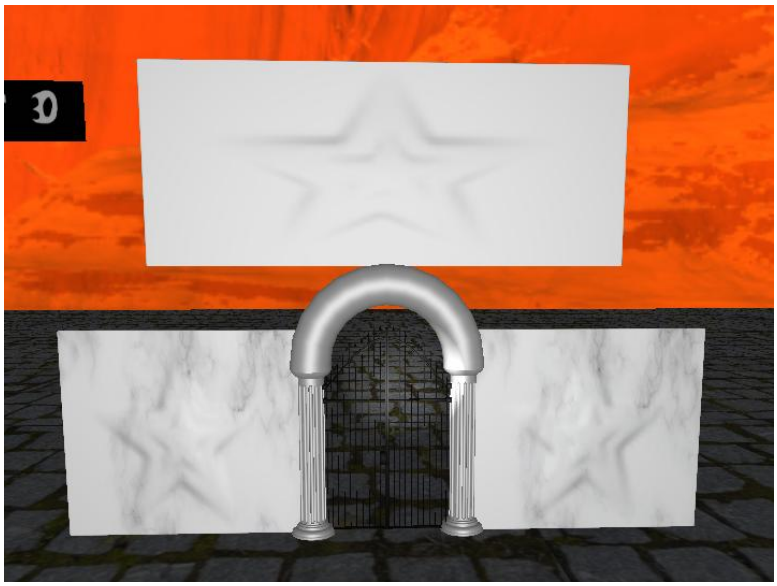
Para realizar esto, primero se tuvo que modificar el .obj en blender, con el fin de poder tener separados amos modelos y manipularlos a cada uno.
Una vez hecho esto, se importan ambos modelos en OpenGL y se cargan con las direcciones y texturas correspondientes.

```
113
114         Model ArcoRejas_M;
115         Model Letrero_M;
```

```
374     ArcoRejas_M = Model();
375     ArcoRejas_M.LoadModel("Models/ArcoYReja.obj");
376     Letrero_M = Model();
377     Letrero_M.LoadModel("Models/Letrero.obj");
378
```

Finalmente se jerarquiza el letrero donde se encuentra el arco con la reja.

```
809
810     //-------------------- Ejercicio 1 Practica 9 ------------------------------
811     //Modelo Arco con muros y reja
812     model = glm::mat4(1.0);
813     model = glm::translate(model, glm::vec3(20.0f, 0.0f, -20.0f));
814     Material_opaco.UseMaterial(uniformSpecularIntensity, uniformShininess);
815     modelaux = model;
816     glUniformMatrix4fv(uniformModel, 1, GL_FALSE, glm::value_ptr(model));
817     ArcoRejas_M.RenderModel();
818
819     //Modelo del letrero
820     model = modelaux;
821     model = glm::translate(model, glm::vec3(0.0f, 15.0f, 0.0f));
822     model = glm::scale(model, glm::vec3(2.0f, 2.0f, 1.0f));
823     glUniformMatrix4fv(uniformModel, 1, GL_FALSE, glm::value_ptr(model));
824     Letrero_M.RenderModel();
```



2. **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**

Primero a nuestra imagen de la tipografía se editó usando el software de GIMP, con el motivo de poder redimensionar la imagen, con una resolución de 512x512. Con el fin de poder texturizar con unas coordenadas más favorables y que se nos facilite. Ya en OpenGL primero se creó otro objeto en la función CreateObjects(), el cual por las coordenadas establecidas siempre que se mande a renderizar el nuevo objeto, por defecto renderizará la letra 'A' de la imagen de la tipografia.

```
269
270     unsigned int tipografiaDPIndices[] = {
271         0, 1, 2,
272         0, 2, 3,
273     };
274
275     GLfloat tipografiaDPVertices[] = {
276         -0.5f, 0.0f, 0.5f,      0.0f, 0.858f,       0.0f, -1.0f, 0.0f,
277         0.5f, 0.0f, 0.5f,       0.25f, 0.858f,      0.0f, -1.0f, 0.0f,
278         0.5f, 0.0f, -0.5f,      0.25f, 1.0f,        0.0f, -1.0f, 0.0f,
279         -0.5f, 0.0f, -0.5f,     0.0f, 1.0f,         0.0f, -1.0f, 0.0f,
280
281     };
282
313     Mesh* obj8 = new Mesh();    //Letra A de la imagen textura Tipografia_DP
314     obj8->CreateMesh(tipografiaDPVertices, tipografiaDPIndices, 32, 6);
315     meshList.push_back(obj8);
```

Para decirle que es lo que vamos a renderizar, también se agregó la imagen de nuestra tipografía como una textura.

```
104     Texture TipografiaDPTexture;

356
357     TipografiaDPTexture = Texture("Textures/Tipografia_DP.png");
358     TipografiaDPTexture.LoadTextureA();
359
```

Para poder crear o texturizar nuestras palabras que se nos pidió, tomando la lógica de texturizado de los números se hizo esto en un for(), pero dado que las letras deseadas no estaban una a lado de la otra se tenia que calcular el offset para las coordenadas U y V, para que se renderice correctamente la letra deseada. Es por ello que se crearon 2 arreglos, los cuales guardan el valor a sumar/restar a nuestras coordenadas U y V, siempre respecto a la letra 'A'. Es importante también declarar nuestras variables que vamos a usar para nuestra animación de estas mismas, las cuales son como los offsets, velocidad, desplazamiento y se crearon otras para poder controlar la animación como si fuera en una pantalla LCD.

```
60
61      //Variables de control para desplazamiento de las letras
62      int letrasVisibles = 1;                     // Cuántas letras se están mostrando
63      GLfloat tiempoUltimaLetra = glfwGetTime();  // Controla el tiempo entre letras
64      float intervaloLetra = 0.3f;                // Cada cuánto aparece una letra (1 segundo)
65      float desplazamientoTexto = 0.0f;           // Para mover todo el texto
66      float velocidadTexto = 0.01f;               // Ajusta la velocidad de desplazamiento
67
68
69      //Variables para texturizado del letrero
70      float toffsetTipografiaU = 0.0f;
71      float toffsetTipografiaV = 0.0f;
72
73
74      //Arreglos para coordenadas U,V que nos guardan la posición de las letras a imprimir
75      //PROYECTO CGEIHC FERIA---> 19 caracteres (sin contar espacios)
76      float incrementoTipografia_U[] = {
77          0.75f, 0.25f, 0.5f, 0.25f, 0.0f, 0.5f, 0.75f, 0.5f,     //PROYECTO
78          0.5f, 0.5f, 0.0f, 0.0f, 0.75f, 0.5f,                    //CGEIHC
79          0.25f, 0.0f, 0.25f, 0.0f, 0.0f                          //FERIA
80      };
81
82      float incrementoTipografia_V[] = {
83          0.429f, 0.572f, 0.429f, 0.858f, 0.143f, 0.0f, 0.572f, 0.429f,       //PROYECTO
84          0.0f, 0.143f, 0.143f, 0.286f, 0.143f, 0.0f,                         //CGEIHC
85          0.143f, 0.143f, 0.572f, 0.286f, 0.0f                                //FERIA
86      };
87
```

Dado que queremos que las letras vayan apareciendo de una a una, se hizo un pequeño algoritmo el cual nos permite decir cuantas letras se van a mostrar para que no se texturicen más de las deseadas, al igual que controlamos la velocidad a la que se muestran y a la que se desplazan. También de que una vez se muestra las palabras en cuestión, se reinicia la animación y se empiezas a mostrar desde el inicio.

```
469
470         // Desplaza todo el texto a la izquierda constantemente
471         desplazamientoTexto -= velocidadTexto * deltaTime;
472
473
474
475         // Cada segundo, agrega una letra más (máximo 19)
476         if (now - tiempoUltimaLetra >= intervaloLetra && letrasVisibles <= 20) {
477             letrasVisibles++;
478             tiempoUltimaLetra = now;
479
480         }
481         // Opcional: Reinicia el desplazamiento para que no se vaya muy lejos
482         if (letrasVisibles > 20 || desplazamientoTexto <= -5.0f) {  //mover desplazamientoTexto si se modifica la velocidadTexto
483             letrasVisibles = 1;
484             desplazamientoTexto = 0.0f;
485             tiempoUltimaLetra = now;
486         }
```

Inicializamos nuestras variables y agregamos las condiciones para que la animación se muestre de acuerdo con lo planeado. Dado que el arreglo tiene las letras en orden de la palabra que queremos mostrar, podemos controlar las coordenadas de acuerdo con las letras visibles que se van a amostrar en ese momento. Y para darle un poco mejor de visibilidad se agregaron “saltos de línea” entre las palabras para que el letrero se vea de una mejor forma y se vea más bonito.

```
834         toffsetTipografiaU = 0.0f;
835         toffsetTipografiaV = 0.0f;
836
837         if (desplazamientoTexto <= -20.0f) {
838             desplazamientoTexto = 0.0f;
839         }
840
```

```
840
841     for (int k = 1; k < letrasVisibles; k++)
842     {
843         //LETRAS PROYECTO CGEIHC FERIA
844         //Incrementa la variable al rango donde debe de texturizar
845         toffsetTipografiaU += incrementoTipografia_U[k-1];
846         toffsetTipografiaV -= incrementoTipografia_V[k-1];
847
848         toffset = glm::vec2(toffsetTipografiaU, toffsetTipografiaV);
849         model = glm::mat4(1.0);
850
851         if (k >= 1 && k <= 8) {
852             model = glm::translate(model, glm::vec3(11.0f + (k * 3.0) + desplazamientoTexto, 23.0f, -19.0f));
853         }
854         else if (k > 8 && k <= 14) {
855             model = glm::translate(model, glm::vec3(-10.0f + (k * 3.0) + desplazamientoTexto, 20.0f, -19.0f));
856         }
857         else if (k > 14) {
858             model = glm::translate(model, glm::vec3(-25.0f + (k * 3.0) + desplazamientoTexto, 17.0f, -19.0f));
859         }
860
861         model = glm::rotate(model, 90 * toRadians, glm::vec3(1.0f, 0.0f, 0.0f));
862         model = glm::scale(model, glm::vec3(3.0f, 3.0f, 3.0f));
863         glUniform2fv(uniformTextureOffset, 1, glm::value_ptr(toffset));
864         glUniformMatrix4fv(uniformModel, 1, GL_FALSE, glm::value_ptr(model));
865         color = glm::vec3(1.0f, 1.0f, 1.0f);
866         glUniform3fv(uniformColor, 1, glm::value_ptr(color));
867         TipografiaDPTexture.UseTexture();
868         Material_brillante.UseMaterial(uniformSpecularIntensity, uniformShininess);
869         meshList[7]->RenderMesh();
870
871         //Regresar los offset al punto de referencia
872         toffsetTipografiaU -= incrementoTipografia_U[k - 1];
873         toffsetTipografiaV += incrementoTipografia_V[k - 1];
874     }
```

3. **Separar las cabezas (con todo y cuello) del Dragón y agregar las siguientes animaciones:**
   - **Movimiento del cuerpo ida y vuelta.**
     Una vez separado todas las cabezas se importaron cada una de ellas, así como las alas y el cuerpo principal de este y se acomodaron de forma jerárquica. El algoritmo creado para el movimiento no fue mucho el cual explicar dado que solo afecta a la traslación del dragón, pero de igual modo se le agregó una rotación la cual la da al momento de ir al lado contrario, es decir que el dragón siempre mirará al lugar donde vaya, y no irá de reversa como es en el caso del coche que se había enseñado en el laboratorio.

```
46      float movDragon;
47      float rotCuerpo;
48      bool avanzaDragon;
```

```
447     movDragon = 0.0f;
448     rotCuerpo = 0.0f;
449     avanzaDragon = true;            //Dragon avanza por defecto
```

Se limitó las coordenadas donde puede avanzar con el fin de que se pueda ver con mayor rapidez cuando avanza y gira para ir hacia el otro lado. Y las banderas nos ayudan para saber en que proceso está el dragón es decir, que si tiene que ir al eje -x o al +x.

```
511
512         //Control que determina si avanza o retrocede el dragon
513         if (avanzaDragon) {
514             if (movDragon > -10) {
515                 movDragon -= movOffset * deltaTime;
516             }
517             else {
518                 rotCuerpo += 180.0f;
519                 avanzaDragon = false;
520             }
521         }
522         else if (movDragon < 10) {
523             movDragon += movOffset * deltaTime;
524         }
525         else {
526             rotCuerpo -= 180.0f;
527             avanzaDragon = true;
528         }
```

Una vez hecho esto se aplican los offset tanto al translate y al rotate y dado que está todo hecho jerárquicamente el cuerpo del dragón y sus extremidades, todo se mueve junto.

```
876         //---------------------- Ejercicio 3 Practica 9 -------------------------------
877         //------------------Modelo del dragón separado en partes--------------
878         //Cuerpo Dragón sin cabeza
879         model = glm::mat4(1.0);
880         model = glm::translate(model, glm::vec3(movDragon - 0.0f, 5.0f + 2*sin(glm::radians(3*angulovaria)), 6.0));
881         model = glm::scale(model, glm::vec3(0.3f, 0.3f, 0.3f));
882         model = glm::rotate(model, rotCuerpo * toRadians, glm::vec3(0.0f, 1.0f, 0.0f));
883         modelaux = model;
884         Material_brillante.UseMaterial(uniformSpecularIntensity, uniformShininess);
885         glUniformMatrix4fv(uniformModel, 1, GL_FALSE, glm::value_ptr(model));
886         DragonSinCabeza_M.RenderModel();
887
888         //Modelo del Ala derecha
889         model = modelaux;
890         model = glm::translate(model, glm::vec3(-0.71f, 3.5f, -0.59f));
891         model = glm::rotate(model, glm::radians(anguloAla), glm::vec3(1.0f, 0.0f, 0.0f));
892         Material_brillante.UseMaterial(uniformSpecularIntensity, uniformShininess);
893         glUniformMatrix4fv(uniformModel, 1, GL_FALSE, glm::value_ptr(model));
894         AlaDer_M.RenderModel();
895
896         //Modelo del Ala Izquierda
897         model = modelaux;
898         model = glm::translate(model, glm::vec3(-0.75f, 3.6f, 0.33f));
899         model = glm::rotate(model, glm::radians(-anguloAla), glm::vec3(1.0f, 0.0f, 0.0f));
900         Material_brillante.UseMaterial(uniformSpecularIntensity, uniformShininess);
901         glUniformMatrix4fv(uniformModel, 1, GL_FALSE, glm::value_ptr(model));
902         AlaIzq_M.RenderModel();
903         /**/
```

```
//Modelo cabeza roja dragon
//Movimiento Senoidal
model = modelaux;
model = glm::translate(model, glm::vec3(-2.32f, 2.63f, 0.37));
model = glm::rotate(model, sin(glm::radians(3*angulovaria)), glm::vec3(0.0f, 0.0f, 1.0f));
Material_opaco.UseMaterial(uniformSpecularIntensity, uniformShininess);
glUniformMatrix4fv(uniformModel, 1, GL_FALSE, glm::value_ptr(model));
CabezaRoja_M.RenderModel();

//Modelo cabeza blanca dragon
// Movimiento rotación espiral de arquimides
//model = glm::translate(model, glm::vec3(0.0f, 5.0f+sin(glm::radians(angulovaria)), 6.0));
model = modelaux;
model = glm::translate(model, glm::vec3(-2.13f, 3.79f, 0.37));
model = glm::rotate(model, 0.1f* angulovaria, glm::vec3(1.0f, 0.0f, 0.0f));
Material_opaco.UseMaterial(uniformSpecularIntensity, uniformShininess);
glUniformMatrix4fv(uniformModel, 1, GL_FALSE, glm::value_ptr(model));
CabezaBlanca_M.RenderModel();
```

```
//Modelo cabeza verde dragon
//Movimiento rotación Lemniscata
model = modelaux;
model = glm::translate(model, glm::vec3(-2.65f, 3.01f, -0.26));
model = glm::rotate(model, (20.0f * cos(glm::radians(angulovaria))) /
    (10.0f + pow(sin(glm::radians(angulovaria)), 2.0f)), glm::vec3(1.0f, 0.0f, 0.0f));
Material_opaco.UseMaterial(uniformSpecularIntensity, uniformShininess);
glUniformMatrix4fv(uniformModel, 1, GL_FALSE, glm::value_ptr(model));
CabezaVerde_M.RenderModel();
```

```
//Modelo cabeza café dragon
//Movimiento rotación Cicloide
model = modelaux;
model = glm::translate(model, glm::vec3(-2.15f, 3.79f, -0.85));
model = glm::rotate(model, 10.0f * (1.0f - cos(glm::radians(angulovaria))), glm::vec3(1.0f, 0.0f, 0.0f));
Material_opaco.UseMaterial(uniformSpecularIntensity, uniformShininess);
glUniformMatrix4fv(uniformModel, 1, GL_FALSE, glm::value_ptr(model));
CabezaCafe_M.RenderModel();

//Modelo cabeza azul dragon
//Movimiento rotación Eliptica
model = modelaux;
model = glm::translate(model, glm::vec3(-2.49f, 2.61f, -0.99));
model = glm::rotate(model, 10.0f * sin(glm::radians(angulovaria)) * cos(glm::radians(angulovaria)), glm::vec3(0.0f, 0.0f, 1.0f));
Material_opaco.UseMaterial(uniformSpecularIntensity, uniformShininess);
glUniformMatrix4fv(uniformModel, 1, GL_FALSE, glm::value_ptr(model));
CabezaAzul_M.RenderModel();
```

- **Aleteo**

  Para el aleteo, se agregó algo parecido, pero enfocado en la rotación, el cual va a controlar el ángulo del giro de las alas y si estas deben de ir arriba o abajo.

```
59  float anguloAla = 0.0f;    //variable que controla el angulo de rotacion de las alas
```

```
456     bool banderaAla = true;
457
```

```
488
489     //Control de rotacion de las alas
490     if (banderaAla) {
491         if (anguloAla >= -60.0f)
492             anguloAla -= 1 * deltaTime;
493         else
494             banderaAla = false;
495
496     }
497     else if (banderaAla == false) {
498         if (anguloAla <= 30.0f)
499             anguloAla += 1 * deltaTime;
500         else
501             banderaAla = true;
502
503     }
```

Ya nada más, este offset para las alas, se le agrega a una rotación a cada modelo de las alas.

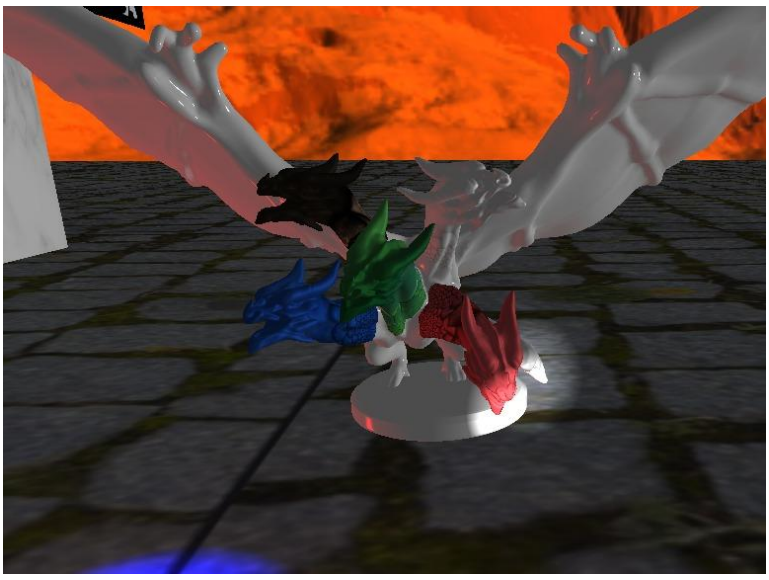
```
887
888         //Modelo del Ala derecha
889         model = modelaux;
890         model = glm::translate(model, glm::vec3(-0.71f, 3.5f, -0.59f));
891         model = glm::rotate(model, glm::radians(anguloAla), glm::vec3(1.0f, 0.0f, 0.0f));
892         Material_brillante.UseMaterial(uniformSpecularIntensity, uniformShininess);
893         glUniformMatrix4fv(uniformModel, 1, GL_FALSE, glm::value_ptr(model));
894         AlaDer_M.RenderModel();
895
896         //Modelo del Ala Izquierda
897         model = modelaux;
898         model = glm::translate(model, glm::vec3(-0.75f, 3.6f, 0.33f));
899         model = glm::rotate(model, glm::radians(-anguloAla), glm::vec3(1.0f, 0.0f, 0.0f));
900         Material_brillante.UseMaterial(uniformSpecularIntensity, uniformShininess);
901         glUniformMatrix4fv(uniformModel, 1, GL_FALSE, glm::value_ptr(model));
902         AlaIzq_M.RenderModel();
```

- **Cada cabeza se mueve de forma diferente de acuerdo a una función/algoritmo diferente (ejemplos: espiral de Arquímedes, movimiento senoidal, lemniscata, etc..)**
  A cada cabeza se le tuvo que agregar una función/algoritmo, para que esta se mueva de forma diferente, es con esto que se le agregó a una rotación independiente a cada cabeza. Es por ello que cada una se movería de forma diferente y velocidad también.

```
904         //Modelo cabeza roja dragon
905         //Movimiento Senoidal
906         model = modelaux;
907         model = glm::translate(model, glm::vec3(-2.32f, 2.63f, 0.37));
908         model = glm::rotate(model, sin(glm::radians(3*angulovaria)), glm::vec3(0.0f, 0.0f, 1.0f));
909         Material_opaco.UseMaterial(uniformSpecularIntensity, uniformShininess);
910         glUniformMatrix4fv(uniformModel, 1, GL_FALSE, glm::value_ptr(model));
911         CabezaRoja_M.RenderModel();
913         //Modelo cabeza blanca dragon
914         // Movimiento rotación espiral de arquimides
915         //model = glm::translate(model, glm::vec3(0.0f, 5.0f+sin(glm::radians(angulovaria)), 6.0));
916         model = modelaux;
917         model = glm::translate(model, glm::vec3(-2.13f, 3.79f, 0.37));
918         model = glm::rotate(model, 0.1f* angulovaria, glm::vec3(1.0f, 0.0f, 0.0f));
919         Material_opaco.UseMaterial(uniformSpecularIntensity, uniformShininess);
920         glUniformMatrix4fv(uniformModel, 1, GL_FALSE, glm::value_ptr(model));
921         CabezaBlanca_M.RenderModel();
923         //Modelo cabeza verde dragon
924         //Movimiento rotación Lemniscata
925         model = modelaux;
926         model = glm::translate(model, glm::vec3(-2.65f, 3.01f, -0.26));
927         model = glm::rotate(model, (20.0f * cos(glm::radians(angulovaria))) /
928             (10.0f + pow(sin(glm::radians(angulovaria)), 2.0f)), glm::vec3(1.0f, 0.0f, 0.0f));
929         Material_opaco.UseMaterial(uniformSpecularIntensity, uniformShininess);
930         glUniformMatrix4fv(uniformModel, 1, GL_FALSE, glm::value_ptr(model));
931         CabezaVerde_M.RenderModel();
932
933         //Modelo cabeza café dragon
934         //Movimiento rotación Cicloide
935         model = modelaux;
936         model = glm::translate(model, glm::vec3(-2.15f, 3.79f, -0.85));
937         model = glm::rotate(model, 10.0f * (1.0f - cos(glm::radians(angulovaria))),
938             glm::vec3(1.0f, 0.0f, 0.0f));
939         Material_opaco.UseMaterial(uniformSpecularIntensity, uniformShininess);
940         glUniformMatrix4fv(uniformModel, 1, GL_FALSE, glm::value_ptr(model));
941         CabezaCafe_M.RenderModel();
```

```
943         //Modelo cabeza azul dragon
944         //Movimiento rotación Elíptica
945         model = modelaux;
946         model = glm::translate(model, glm::vec3(-2.49f, 2.61f, -0.99));
947         model = glm::rotate(model, 10.0f * sin(glm::radians(angulovaria)) *
948             cos(glm::radians(angulovaria)), glm::vec3(0.0f, 0.0f, 1.0f));
949         Material_opaco.UseMaterial(uniformSpecularIntensity, uniformShininess);
950         glUniformMatrix4fv(uniformModel, 1, GL_FALSE, glm::value_ptr(model));
951         CabezaAzul_M.RenderModel();
952
```

- **Cada cabeza debe de verse de un color diferente: roja, azul, verde, blanco, café.**
  Para este apartado, el texturizado se hizo desde blender, por lo que se tuvieron que descargar imagen para cada una de las cabezas y texturizarlo desde ahí, se escogió una textura lo más parecido a las escamas, para simular a la piel de dragón, y cada una con el respectivo color que se mencionó en la actividad.



Link del video de la ejecución:
https://drive.google.com/file/d/1vcKFjqm3W5LDkirhV01MrfmpxVKJdFJT/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

Principalmente fue el ejercicio con el texturizado del letrero, dado que en un inicio no sabía el cómo realizar el texturizado, por lo que siento que se creó de una forma rebuscada el texturizado de las letras, al igual que pensé que el código se iba haciendo más complejo a la hora de tratar de controlar el desplazamiento como la velocidad y el tiempo en el que se debe de ver cada palabra. Por esto mismo considero yo que fue lo más difícil y tardado de esta práctica.

3.- Conclusión:

a. Los ejercicios del reporte: Complejidad, Explicación.

Si bien a nivel teórico siento que los ejercicios estuvieron al nivel, siento yo que a nivel implementación subió bastante el nivel, dado que no conocíamos el como manejar la animación por texturizado de una forma más elaborada o exacta, por lo cual si sentí que ese ejercicio estuvo un poco más arriba de complejidad que los otros.

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

Considero que no faltó explicar nada, pero si a lo mejor ver otros ejemplos más para poder familiarizarnos más con el manejo de las animaciones.

c. Conclusión

Gracias a esta práctica pude entender el como son implementadas las animaciones, los requisitos que estas deben de tener para ser considerada como tal, al igual de las formas en las que estas se pueden modificar para obtener un resultado o varios, los cuales nos ayudan a darle vida a los modelos que tenemos.

1. Bibliografía en formato APA

- JoeyDeVries. (s.f.). LearnOpenGL - Transformations. Recuperado el 23 de abril de 2025, de https://learnopengl.com/Getting-started/Transformations
- JoeyDeVries. (s.f.). LearnOpenGL - Textures. Recuperado el 23 de abril de 2025, de https://learnopengl.com/Getting-started/Textures
- JoeyDeVries. (s.f.). LearnOpenGL - Time-based movement. Recuperado el 23 de abril de 2025, de https://learnopengl.com/Getting-started/Camera#Time-based-movement
- Microsoft. (s.f.). std::chrono::duration - cppreference.com. Recuperado el 23 de abril de 2025, de https://en.cppreference.com/w/cpp/chrono/duration
- glfw.org. (s.f.). GLFW - Time Input. Recuperado el 23 de abril de 2025, de https://www.glfw.org/docs/latest/group__time.html
- Ogldev.org. (s.f.). OpenGL Tutorial 6: Textures. Recuperado el 23 de abril de 2025, de http://ogldev.org/www/tutorial06/tutorial06.html
- Depositphotos. (s.f.). Escamas rojas [Imagen]. Depositphotos. Recuperado el 23 de abril de 2025, de https://depositphotos.com/es/vectors/escamas-rojas.html
- Depositphotos. (s.f.). Escamas azules [Imagen]. Depositphotos. Recuperado el 23 de abril de 2025, de https://depositphotos.com/es/photos/escamas-azules.html

- Dreamstime. (s.f.). Piel verde de serpiente o dragón con escamas, textura fantasía [Imagen]. Dreamstime. Recuperado el 23 de abril de 2025, de https://es.dreamstime.com/piel-verde-de-serpiente-o-drag%C3%B3n-con-escamas-textura-fantas%C3%ADa-fondo-d-representado-image209541742
- Shutterstock. (s.f.). Seamless texture of dragon scales, reptile skin [Imagen]. Shutterstock. Recuperado el 23 de abril de 2025, de https://www.shutterstock.com/es/image-illustration/seamless-texture-dragon-scales-reptile-skin-681952291