UNIVERSIDAD NACIONAL AUTÓNOMA DE MÉXICO

FACULTAD DE INGENIERÍA

DIVISIÓN DE INGENIERÍA ELÉCTRICA

INGENIERÍA EN COMPUTACIÓN

LABORATORIO COMPUTACIÓN GRÁFICA e INTERACCIÓN HUMANOCOMPUTADORA

Reporte de Práctica 9-2: Animación Avanzada

NOMBRE COMPLETO: Gonzalez Villalba Bryan Jesus

Nº de Cuenta: 421530869

GRUPO DE LABORATORIO: 11

GRUPO DE TEORÍA: 4

SEMESTRE 2025-1

FECHA DE ENTREGA LÍMITE: 22 de Octubre del 2024

CALIFICACIÓN: _____

Introducción:

Importamos nuestro cartel y arco con animación, luego tomamos la tipografía de nuestro universo y ponemos la frase “PROYECTO CGEIHC MONOPOLY” en nuestro cartel y girando hacia la derecha, separamos las cabezas del dragón y le agregamos movimiento al dragón, las alas y a las cabezas.

Desarrollo:

Imagen 1.0, Separamos el modelo en 8 partes, una es el cuerpo, otras dos son las alas y 5 cabezas.

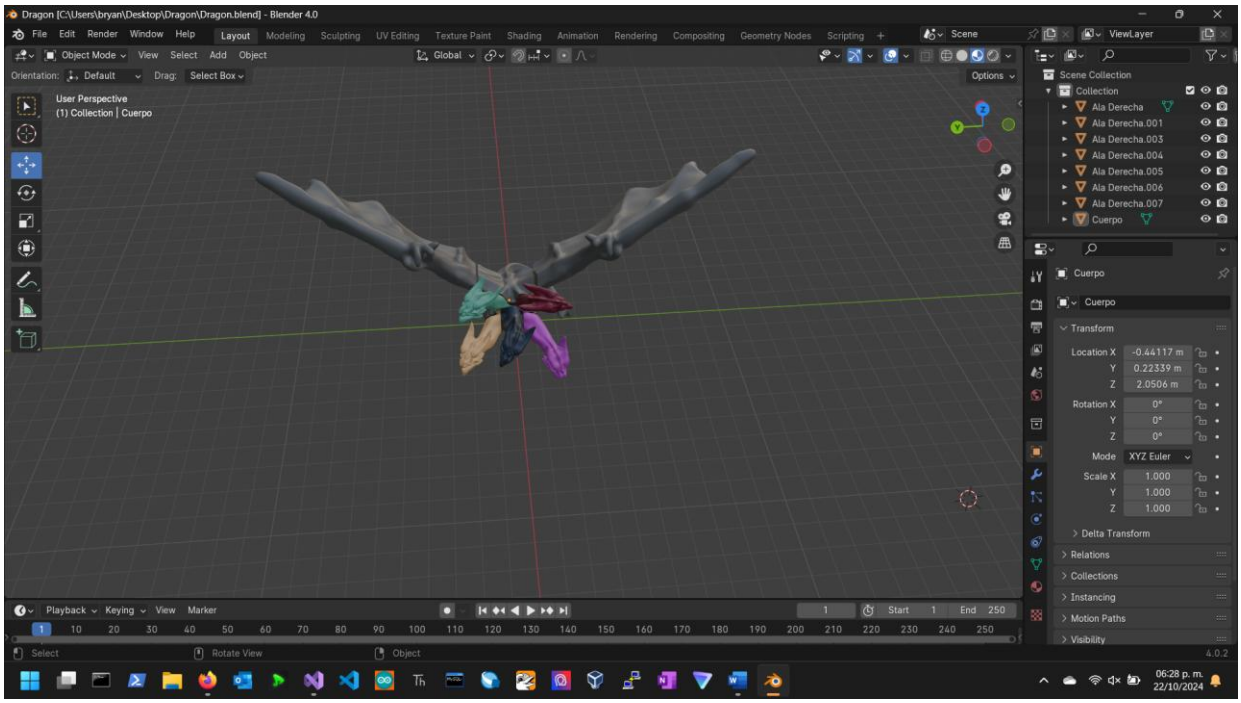

Imagen 1.0

Imagen 1.1 – 1.7, Declaramos las variables, de uso de textura, de uso de modelos, de importación de textura, de importación de modelos, banderas de control, banderas de tiepo, etc., que se utilizaron en el programa.

```
51      //variables para animación
52          float movCoche;
53          float movOffset;
54          float rotllanta;
55          float rotllantaOffset;
56          bool avanza;
57          float toffsetflechau = 0.0f;
58          float toffsetflechav = 0.0f;
59          float toffsetnumerou = 0.0f;
60          float toffsetnumerov = 0.0f;
61          float toffsetnumerocambiau = 0.0;
62          float angulovaria = 0.0f;
63          float Cartel_Fijo;
64          float Cartel_Rotacion;
65          float Tiempo_Pausa;
66          bool Inicio_Cartel;
67          bool Pausar;
68          bool Bajando;
69          bool Esperar_Arriba;
70          bool Esperar_Abajo;
```

Imagen 1.1

```
72          float toffsetTexturau;
73          float toffsetTexturav;
74          float Textura_Fija;
75          float Textura_Rotacion;
76          float rotacionDragon;
77          bool giroCompleto;
78
79          float posicionDragon;
80          float velocidadDragon;
81          bool moviendoDerecha;
82
83          GLfloat arrancar;
84          GLfloat Animacion_Cartel;
```

Imagen 1.2

```
87      // Variables para el Movimiento Cabeza Azul
88          float tiempoCabeza_Azul = 0.0f; // Tiempo acumulado para el movimiento senoidal
89          float velocidadMovimientoCabeza_Azul = 0.1f; // Controla la velocidad del movimiento senoidal
90          float amplitudCabeza_Azul = 0.1f; // Amplitud del movimiento senoidal
91          float frecuenciaCabeza_Azul = 1.0f; // Frecuencia del movimiento senoidal
92
93      // Variables para el Movimiento Cabeza Cafe
94          float aCabeza_Cafe = -0.5f;  // Posición inicial en X
95          float bCabeza_Cafe = 0.01f;  // Aumento del radio de la espiral
96          float cCabeza_Cafe = 0.20f;   // Posición inicial en Y
97          float tiempoCabeza_Cafe = 0.0f; // Tiempo acumulado
98          float velocidadFactor = 0.1f; // Factor para reducir la velocidad
99
100     // Variables para el Movimiento Cabeza Blanca
101         float velocidadCabeza_Blanca = 0.1f; // Cambia este valor para ajustar la velocidad
102         float tiempoCabeza_Blanca = 0.0f; // Tiempo acumulado
103         float aCabeza_Blanca = 0.3f; // Amplitud para el movimiento lemniscata
```

Imagen 1.3

```
105     // Variables para el Movimiento Cabeza Roja
106         float velocidadCabeza_Roja = 0.3f; // Cambia este valor para ajustar la velocidad
107         float tiempoCabeza_Roja = 0.0f; // Tiempo acumulado
108         float radioCabeza_Roja = 0.5f; // Radio del movimiento helicoidal
109         float alturaCabeza_Roja = 0.02f; // Altura de cada vuelta
110
111     // Variables para el Movimiento Cabeza Morada
112         float velocidadCabeza_Morada = 0.2f; // Cambia este valor para ajustar la velocidad
113         float tiempoCabeza_Morada = 0.0f; // Tiempo acumulado
114         // Puntos de control para la curva de Bézier
115         glm::vec3 punto1(-0.5f, 0.20f, 0.16f); // Inicio
116         glm::vec3 punto2(-0.3f, 0.50f, 0.20f); // Control 1
117         glm::vec3 punto3(-0.1f, 0.30f, 0.25f); // Control 2
118         glm::vec3 punto4(0.0f, 0.20f, 0.30f); // Fin
```

Imagen 1.4

```
142     // Letras
143         Texture Letras;
144         Texture Roja;
145         Texture Rayos;
146         Texture Gas;
147         Texture Hielo;
148         Texture Arena;
149
150     // Modelos
151         Model Cuerpo;
152         Model Ala_Derecha;
153         Model Ala_Izquierda;
154         Model Cabeza_Azul;
155         Model Cabeza_Cafe;
156         Model Cabeza_Gris;
157         Model Cabeza_Morada;
158         Model Cabeza_Roja;
159         Model Arco;
160         Model Cartel;
```

Imagen 1.5

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

// Dragon
    Cuerpo = Model();
    Cuerpo.LoadModel("Models/Dragon/Cuerpo.obj");
    Ala_Derecha = Model();
    Ala_Derecha.LoadModel("Models/Dragon/Ala_Derecha.obj");
    Ala_Izquierda = Model();
    Ala_Izquierda.LoadModel("Models/Dragon/Ala_Izquierda.obj");
    Cabeza_Azul = Model();
    Cabeza_Azul.LoadModel("Models/Dragon/Cabeza_Azul.obj");
    Cabeza_Cafe = Model();
    Cabeza_Cafe.LoadModel("Models/Dragon/Cabeza_Cafe.obj");
    Cabeza_Gris = Model();
    Cabeza_Gris.LoadModel("Models/Dragon/Cabeza_Gris.obj");
    Cabeza_Morada = Model();
    Cabeza_Morada.LoadModel("Models/Dragon/Cabeza_Morada.obj");
    Cabeza_Roja = Model();
    Cabeza_Roja.LoadModel("Models/Dragon/Cabeza_Roja.obj");
```

Imagen 1.6

```
// Cartel y Arco
    Arco = Model();
    Arco.LoadModel("Models/Arco/Arco.obj");
    Cartel = Model();
    Cartel.LoadModel("Models/Arco/Cartel.obj");
```

Imagen 1.7

Imagen 1.8 y 1.9, Creamos las funciones para el movimiento de cada cabeza de manera individual, con su tiempo de cada una, para no afectar el tiempo general y nos modifique tiempos de otras partes del código de las que no queremos modificar.

```
// Funcion de Movimiento Cabeza Azul
glm::vec3 Mov_Cabeza_Azul(float Tiempo_Cabeza_Azul) {
    float y = 0.48f + amplitudCabeza_Azul * sin(frecuenciaCabeza_Azul * Tiempo_Cabeza_Azul);
    return glm::vec3(-0.5f, y, -0.15f);
}

// Función de Movimiento Cabeza Cafe
glm::vec3 Mov_Cabeza_Cafe(float tiempo_Cafe) {
    float t = tiempo_Cafe; // Usa el tiempo acumulado
    float x = aCabeza_Cafe + bCabeza_Cafe * t * cos(t); // Movimiento en X
    float y = cCabeza_Cafe + bCabeza_Cafe * t * sin(t); // Movimiento en Y
    return glm::vec3(x, y, -0.15f); // Retorna la nueva posición, Z permanece constante
}

// Función de Movimiento Cabeza Blanca
glm::vec3 Mov_Cabeza_Blanca(float tiempo_Blanca) {
    float t = tiempo_Blanca; // Usa el tiempo acumulado
    float x = -0.5f + aCabeza_Blanca * sin(t); // Movimiento en X
    float y = 0.10f + aCabeza_Blanca * sin(t) * cos(t); // Movimiento en Y
    return glm::vec3(x, y, 0.0f); // Retorna la nueva posición, Z permanece constante
}
```

Imagen 1.8

```
// Función de Movimiento Cabeza Roja
glm::vec3 Mov_Cabeza_Roja(float tiempo_Roja) {
    float t = tiempo_Roja; // Usa el tiempo acumulado
    float x = -0.5f + radioCabeza_Roja * cos(t); // Movimiento en X (circular)
    float y = 0.48f + alturaCabeza_Roja * t; // Movimiento en Y (vertical)
    float z = 0.17f + radioCabeza_Roja * sin(t); // Movimiento en Z (circular)
    return glm::vec3(x, y, z); // Retorna la nueva posición
}

// Función de Movimiento Cabeza Morada
glm::vec3 Mov_Cabeza_Morada(float tiempo_Morada) {
    // Normaliza el tiempo a [0, 1] para el movimiento
    float t = fmod(tiempo_Morada * velocidadCabeza_Morada, 1.0f); // Rango [0, 1
    // Cálculo de la curva de Bézier de 4 puntos
    glm::vec3 puntoA = glm::mix(punto1, punto2, t);
    glm::vec3 puntoB = glm::mix(punto2, punto3, t);
    glm::vec3 puntoC = glm::mix(punto3, punto4, t);
    glm::vec3 puntoAB = glm::mix(puntoA, puntoB, t);
    glm::vec3 puntoBC = glm::mix(puntoB, puntoC, t);
    return glm::mix(puntoAB, puntoBC, t); // Retorna la posición en la curva
}
```

Imagen 1.9

Imagen 2.0 y 2.1, Le damos valores a las variables que andamos a llamar.

```
// Variables
    movCoche = 0.0f;
    movOffset = 0.01f;
    rotllanta = 0.0f;
    rotllantaOffset = 10.0f;

    Textura_Fija = 0.0f;
    Textura_Rotacion = 0.19f;
    Cartel_Fijo = 0.0f;
    Cartel_Rotacion = 0.1f;
    Inicio_Cartel = true;
    Tiempo_Pausa = 0.0f;
    Pausar = false;
    Bajando = true;
    Esperar_Arriba = false;
    Esperar_Abajo = false;
    glfwGetTime();
```

Imagen 2.0

```
// Variables de control para el movimiento del dragón
    posicionDragon = 0.0f;  // Inicia en -20 en el eje X
    velocidadDragon = 0.04f; // Velocidad del dragon
    moviendoDerecha = true;     // El dragón comienza moviéndose a la derech
    rotacionDragon = 0.0f; // Rotación en grados
    giroCompleto = false; // Indica si el dragón ha realizado un giro completo
```

Imagen 2.1

Imagen 2.2 – 2.6, Creamos las animaciones del cartel y el arco, y creamos la animación del dragón y las

alas.

```
// Animacion del Cartel
            Animacion_Cartel = glfwGetTime();

            if (Animacion_Cartel > 2)
            {
                if (!Pausar)
                {
                    if (Bajando)
                    {
                        // Ambas condiciones deben cumplirse para seguir bajando
                        if (Cartel_Fijo < 3.3f) // Ajusta los límites según sea necesario
                        {
                            Cartel_Fijo += Cartel_Rotacion * deltaTime;
                            Textura_Fija += Textura_Rotacion * deltaTime;
                        }
                        else
                        {
                            Pausar = true;
                            Tiempo_Pausa = Animacion_Cartel;
                            Esperar_Arriba = true;
                        }
                    }
```

Imagen 2.2

```
                    else
                    {
                        // Ambas condiciones deben cumplirse para seguir subiendo
                        if (Cartel_Fijo > -2.5f) // Ajusta los límites según sea necesario
                        {
                            Cartel_Fijo -= Cartel_Rotacion * deltaTime;
                            Textura_Fija -= Textura_Rotacion * deltaTime;
                        }
                        else
                        {
                            Pausar = true;
                            Tiempo_Pausa = Animacion_Cartel;
                            Esperar_Abajo = true;
                        }
                    }
                }
                else
                {
                    float tiempoTranscurrido = Animacion_Cartel - Tiempo_Pausa;

                    if (Esperar_Arriba && tiempoTranscurrido >= 2.0f)
                    {
                        Pausar = false;
                        Esperar_Arriba = false;
                        Bajando = false;
                    }
```

Imagen 2.3

```
                    else if (Esperar_Abajo && tiempoTranscurrido >= 2.0f)
                    {
                        Pausar = false;
                        Esperar_Abajo = false;
                        Bajando = true;
                    }
                }
            }
```

Imagen 2.4

```
// Aniamcion del Dragon
        float anguloAla = 40.0f * sin(glfwGetTime() * 2.0f);

        if (moviendoDerecha)
        {
            // El dragón se mueve hacia +20 en el eje X
            posicionDragon += deltaTime * velocidadDragon;  // Ajusta 'velocidadDragon' según la velocidad deseada
            if (posicionDragon >= 20.0f)  // Si llega a +20, cambia de dirección
            {
                // Gira 180 grados en Y
                rotacionDragon += 180.0f;
                moviendoDerecha = false;
                giroCompleto = true; // Marca que el dragón ha girado
            }
        }
        else
        {
            // El dragón se mueve hacia -20 en el eje X
            posicionDragon -= deltaTime * velocidadDragon;
            if (posicionDragon <= -20.0f)  // Si llega a -20, cambia de dirección
            {
                // Gira de regreso a la posición inicial
                if (giroCompleto)
                {
                    rotacionDragon -= 180.0f; // Regresa a la posición original
```

Imagen 2.5

```
                    giroCompleto = false; // Resetea el estado de giro
                }
                moviendoDerecha = true;
            }
        }
```

Imagen 2.6

Imagen 2.7 – 3.4, Mandamos a dibujar todas las partes del modelo que ocupamos para la animación del cartel, la animación del dragón y el uso de funciones de las cabezas.

```
// Arco
        model = glm::mat4(1.0);
        model = glm::translate(model, glm::vec3(-10.1, 5.2f, -3.0f));
        modelaux = model;
        model = glm::scale(model, glm::vec3(0.5f, 0.5f, 0.5f));
        model = glm::rotate(model, -90 * toRadians, glm::vec3(0.0f, 1.0f, 0.0f));
        glUniformMatrix4fv(uniformModel, 1, GL_FALSE, glm::value_ptr(model));
        Arco.RenderModel();

// Cartel
        model = modelaux;
        model = glm::translate(model, glm::vec3(0.0, Cartel_Fijo, 0.0f));
        model = glm::rotate(model, -90 * toRadians, glm::vec3(0.0f, 1.0f, 0.0f));
        model = glm::translate(model, glm::vec3(0.0f, Cartel_Fijo, 0.0f));  // Mover el letrero a su posición
        model = glm::scale(model, glm::vec3(0.4f, 0.4f, 0.4f));
        glUniformMatrix4fv(uniformModel, 1, GL_FALSE, glm::value_ptr(model));
        Cartel.RenderModel();
```

Imagen 2.7

```
// Textura de letras

        // Inicializa variables para el temporizador
        steady_clock::time_point lastTime_Textura_Letras = steady_clock::now();
        float updateInterval_Textura_Letras = 0.1f; // Intervalo de actualización en segundos
        // Calcula el tiempo transcurrido
        steady_clock::time_point currentTime_Textura_Letras = steady_clock::now();
        float deltaTime_Textura_Letras = duration_cast<duration<float>>(currentTime_Textura_Letras - lastTime_Textura_Letras

        // Actualiza toffsetnumerocambiau solo si ha pasado el intervalo
        if (deltaTime >= updateInterval_Textura_Letras) {
            toffsetTexturau += 0.3f; // Incrementa el número
            if (toffsetTexturau > 1.0f)
                toffsetTexturau = 0.0f; // Reinicia si excede 1.0
            lastTime_Textura_Letras = currentTime_Textura_Letras; // Resetea el tiempo
        }

        //Importantes porque la variable uniform no podemos modificarla directamente
        //toffsetTexturau -= 0.0001; // Desplazamiento en X y -X
        toffsetTexturav = 0.000; // Desplazamiento en Y y -Y
        //para que no se desborde la variable
        if (toffsetTexturau > 1.0)
            toffsetTexturau = 0.0;

        toffset = glm::vec2(toffsetTexturau, toffsetTexturav);
```

Imagen 2.8

```
785                 model = modelaux;
786                 model = glm::translate(model, glm::vec3(0.0f, Textura_Fija, 0.2f));
787                 model = glm::rotate(model, 90 * toRadians, glm::vec3(1.0f, 0.0f, 0.0f));
788                 model = glm::scale(model, glm::vec3(9.0f, 3.0f, 3.0f));
789                 glUniform2fv(uniformTextureOffset, 1, glm::value_ptr(toffset));
790                 glUniformMatrix4fv(uniformModel, 1, GL_FALSE, glm::value_ptr(model));
791                 Letras.UseTexture();
792                 Material_brillante.UseMaterial(uniformSpecularIntensity, uniformShininess);
793                 meshList[4]->RenderMesh();
794                 modelaux = model;
```

Imagen 2.9

```
796 // Cuerpo del Dragón
797             model = glm::mat4(1.0f);
798             model = glm::translate(model, glm::vec3(posicionDragon, 5.0f + sin(glm::radians(angulovaria * 8)), 6.0f));
799
800             // Aplicar la rotación según la dirección del movimiento
801             if (moviendoDerecha) {
802                 rotacionDragon = 180.0f; // Mirando hacia la derecha
803             }
804             else {
805                 rotacionDragon = 0.0f; // Mirando hacia la izquierda
806             }
807
808             model = glm::rotate(model, glm::radians(rotacionDragon), glm::vec3(0.0f, 1.0f, 0.0f)); // Aplica la rotación
809             modelaux = model;  // Guardamos la posición actual del cuerpo para usarla en las alas
810             model = glm::scale(model, glm::vec3(0.3f, 0.3f, 0.3f));
811             Material_brillante.UseMaterial(uniformSpecularIntensity, uniformShininess);
812             glUniformMatrix4fv(uniformModel, 1, GL_FALSE, glm::value_ptr(model));
813             Cuerpo.RenderModel();
```

Imagen 3.0

```
815 // Ala Derecha
816             model = modelaux;
817             model = glm::translate(model, glm::vec3(-0.08f, 0.5f, 0.38f));  // Posición relativa del ala derecha
818             model = glm::rotate(model, glm::radians(anguloAla), glm::vec3(1.0f, 0.0f, 0.0f));  // Rotación en Y
819             model = glm::scale(model, glm::vec3(0.3f, 0.3f, 0.3f));
820             glUniformMatrix4fv(uniformModel, 1, GL_FALSE, glm::value_ptr(model));
821             Ala_Derecha.RenderModel();
822
823 // Ala Izquierda
824             model = modelaux;
825             model = glm::translate(model, glm::vec3(-0.1f, 0.55f, -0.48f));  // Posición relativa del ala izquierda
826             model = glm::rotate(model, glm::radians(-anguloAla), glm::vec3(1.0f, 0.0f, 0.0f));  // Rotación en Y opuesta
827             model = glm::scale(model, glm::vec3(0.3f, 0.3f, 0.3f));
828             glUniformMatrix4fv(uniformModel, 1, GL_FALSE, glm::value_ptr(model));
829             Ala_Izquierda.RenderModel();
```

Imagen 3.1

```
831 // Cabeza_Azul Senoidal se mueve como una ola, de arriaba a abajo
832             model = glm::mat4(1.0f);
833             tiempoCabeza_Azul += deltaTime; // Asegúrate de que deltaTime sea adecuado
834             glm::vec3 posicionCabeza = Mov_Cabeza_Azul(tiempoCabeza_Azul);
835             model = modelaux; // Reinicia la matriz del modelo
836             model = glm::translate(model, posicionCabeza); // Usa la nueva posición
837             model = glm::scale(model, glm::vec3(0.3f, 0.3f, 0.3f));
838             glUniformMatrix4fv(uniformModel, 1, GL_FALSE, glm::value_ptr(model));
839             Cabeza_Azul.RenderModel(); // Renderiza la cabeza azul
840
841 // Cabeza_Cafe Espiral de Alquimiques, como si fuera una caracola
842             model = glm::mat4(1.0f);
843             tiempoCabeza_Cafe += deltaTime * velocidadFactor; // Aumenta el tiempo acumulado con reducción de velocidad
844             glm::vec3 posicionCabezaCafe = Mov_Cabeza_Cafe(tiempoCabeza_Cafe); // Llama a la función de movimiento
845             model = modelaux; // Reinicia la matriz del modelo
846             model = glm::translate(model, posicionCabezaCafe); // Usa la nueva posición
847             model = glm::scale(model, glm::vec3(0.3f, 0.3f, 0.3f));
848             glUniformMatrix4fv(uniformModel, 1, GL_FALSE, glm::value_ptr(model));
849             Cabeza_Cafe.RenderModel(); // Renderiza la cabeza cafe
```

Imagen 3.2

```
// Cabeza_Blanca Lemiscata, como si fuera un infinito
            model = glm::mat4(1.0f);
            tiempoCabeza_Blanca += deltaTime * velocidadCabeza_Blanca; // Ajusta la velocidad aquí
            glm::vec3 posicionCabezaBlanca = Mov_Cabeza_Blanca(tiempoCabeza_Blanca);
            model = modelaux; // Reinicia la matriz del modelo
            model = glm::translate(model, posicionCabezaBlanca); // Usa la nueva posición
            model = glm::scale(model, glm::vec3(0.3f, 0.3f, 0.3f)); // Escalar el modelo
            color = glm::vec3(1.0f, 1.0f, 1.0f); // Color blanco
            glUniform3fv(uniformColor, 1, glm::value_ptr(color));
            glUniformMatrix4fv(uniformModel, 1, GL_FALSE, glm::value_ptr(model));
            Cabeza_Gris.RenderModel(); // Renderiza la cabeza blanca

// Cabeza_Roja
            model = glm::mat4(1.0f);
            tiempoCabeza_Roja += 0.01f * velocidadCabeza_Roja; // Ajusta la velocidad aquí
            glm::vec3 posicionCabeza_Roja = Mov_Cabeza_Roja(tiempoCabeza_Roja);
            model = modelaux; // Reinicia la matriz del modelo
            model = glm::translate(model, posicionCabeza_Roja); // Usa la nueva posición
            model = glm::scale(model, glm::vec3(0.3f, 0.3f, 0.3f)); // Escalar el modelo
            glUniformMatrix4fv(uniformModel, 1, GL_FALSE, glm::value_ptr(model));
            Cabeza_Roja.RenderModel(); // Renderiza la cabeza roja
```

Imagen 3.3

```
// Cabeza_Morada
            model = glm::mat4(1.0f);
            tiempoCabeza_Morada += 0.01f; // Ajusta el incremento de tiempo para controlar la velocidad
            glm::vec3 posicionCabeza_Morada = Mov_Cabeza_Morada(tiempoCabeza_Morada);
            model = modelaux; // Reinicia la matriz del modelo
            model = glm::translate(model, posicionCabeza_Morada); // Usa la nueva posición
            model = glm::scale(model, glm::vec3(0.3f, 0.3f, 0.3f)); // Escalar el modelo
            glUniformMatrix4fv(uniformModel, 1, GL_FALSE, glm::value_ptr(model));
            Cabeza_Morada.RenderModel(); // Renderiza la cabeza morada
```

Imagen 3.4

Conclusión:
421530869:

Fue una práctica difícil ya que para poder mover la textura me daba problemas, lo que no pude realizar de esa parte fue, hacer que la dirección del movimiento fuera de Izquierda a Derecha, además no pude dibujar la textura con todo el abecedario e ir seleccionando por partes la letra, para la separación del dragón fue muy sencilla, para el dibujado de las alas y la animación también fue sencilla ya que utilice información de mis practicas anteriores, el uso de funciones fue algo difícil ya que no podía dibujarlas, hasta que ocupe funciones en el código, y mandaba a dibujarlas como objeto normales.

Referencias:

Fuente Adventure Time - Descargar. (2024). Fontmeme.com. https://fontmeme.com/fuentes/fuente-adventure-time/

Modelo:
Proporcionado por el profesor.