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 04

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: 09 marzo 2025

CALIFICACIÓN: _____

REPORTE DE PRÁCTICA:

1.- Actividades:

- Terminar la grúa con:
    a. Cuerpo (prisma rectangular).
    b. Base (pirámide cuadrangular).
    c. 4 llantas (4 cilindros) con teclado se pueden girar las 4 llantas por separado.

```
//Limpiar la ventana
//glClearColor(1.0f, 0.0f, 1.0f, 0.0f); //Color púrpura para el fondo de la grúa
glClearColor(1.0f, 0.5f, 0.0f, 0.0f); //Color naranja para el fondo del gato
glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT); //Se agrega limpiar el buffer de profundidad
shaderList[0].useShader();
uniformModel = shaderList[0].getModelLocation();
uniformProjection = shaderList[0].getProjectLocation();
uniformView = shaderList[0].getViewLocation();
uniformColor = shaderList[0].getColorLocation();
/*
```

Ocupé 2 clearColor, el primero para el fondo de la grúa y el segundo para el fondo del gato.

```
// Métodos getter para las llantas
GLfloat getllanta1() { return llanta1; }
GLfloat getllanta2() { return llanta2; }
GLfloat getllanta3() { return llanta3; }
GLfloat getllanta4() { return llanta4; }
```

En window.h, se agregaron los getter para las llantas y así poderles asignar un movimiento.

```
Window::Window(GLint windowWidth, GLint windowHeight)
{
    width = windowWidth;
    height = windowHeight;
    rotax = 0.0f;
    rotay = 0.0f;
    rotaz = 0.0f;

    articulacion1 = 0.0f;
    articulacion2 = 0.0f;
    articulacion3 = 0.0f;
    articulacion4 = 0.0f;
    articulacion5 = 0.0f;
    articulacion6 = 0.0f;

    llanta1 = 0.0f;
    llanta2 = 0.0f;
    llanta3 = 0.0f;
    llanta4 = 0.0f;
```

En window.cpp, se inicializaron las llantas en 0.0 en valor flotante.

```
if (key == GLFW_KEY_M)
{
    theWindow->llanta1 += 10;
}
if (key == GLFW_KEY_N)
{
    theWindow->llanta2 += 10;
}
if (key == GLFW_KEY_O)
{
    theWindow->llanta3 += 10;
}
if (key == GLFW_KEY_P)
{
    theWindow->llanta4 += 10;
}
```

En window.cpp, se le asignaron teclas a cada llanta, que son las que las estarán moviendo.

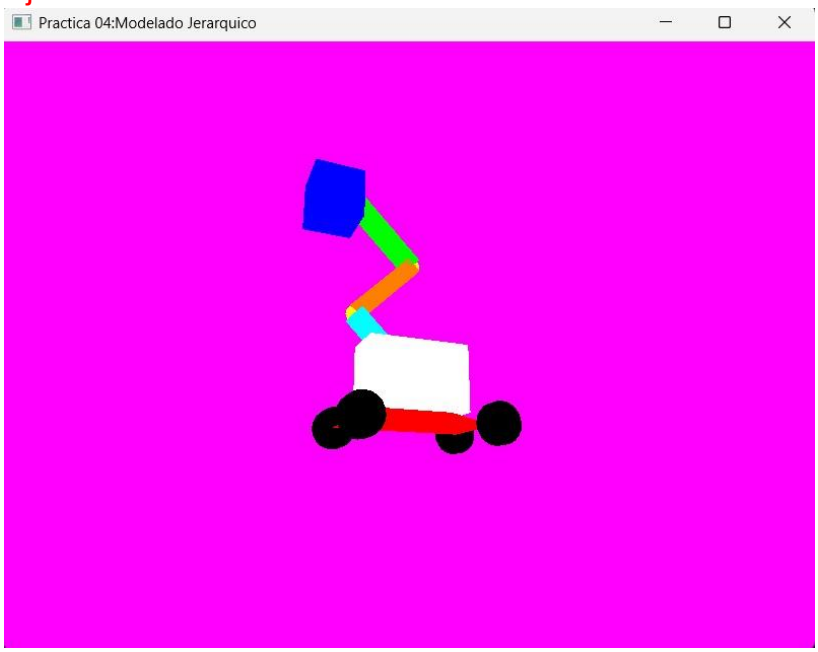
```
//AQUÍ SE DIBUJA LA CABINA, LA BASE, LAS 4 LLANTAS
//Cabina    (Cubo)
model = glm::mat4(1.0);
model = glm::translate(model, glm::vec3(2.0f, 4.0f, -4.0f));
modelaux = model;
modelaux2 = model;
model = glm::scale(model, glm::vec3(6.0f, 4.0f, 3.0f));
glUniformMatrix4fv(uniformModel, 1, GL_FALSE, glm::value_ptr(model));
glUniformMatrix4fv(uniformProjection, 1, GL_FALSE, glm::value_ptr(projection));
glUniformMatrix4fv(uniformView, 1, GL_FALSE, glm::value_ptr(camera.calculateViewMatrix()));
color = glm::vec3(1.0f, 1.0f, 1.0f);        //Color blanco
glUniform3fv(uniformColor, 1, glm::value_ptr(color)); //para cambiar el color del objetos
meshList[0]->RenderMesh(); //dibuja cubo y pirámide triangular
model = modelaux;

//Base de la cabina (Piramide cuadrangular)
model = glm::translate(model, glm::vec3(0.0f, -2.0f, 0.0f));
modelaux = model;
model = glm::scale(model, glm::vec3(8.0f, 1.5f, 4.5f));
glUniformMatrix4fv(uniformModel, 1, GL_FALSE, glm::value_ptr(model));
glUniformMatrix4fv(uniformProjection, 1, GL_FALSE, glm::value_ptr(projection));
glUniformMatrix4fv(uniformView, 1, GL_FALSE, glm::value_ptr(camera.calculateViewMatrix()));
color = glm::vec3(1.0f, 0.0f, 0.0f);    //Color rojo
glUniform3fv(uniformColor, 1, glm::value_ptr(color)); //para cambiar el color del objetos
meshList[4]->RenderMeshGeometry();  //dibuja piramide cuadrangular
model = modelaux;
```

Se realizó el cuerpo y la base antes de realizar las llantas, para que se pudieran ir ajustando, utilizando modelaux = model y model = modelaux.

```
//Llantas (4 cilindros)
// Llanta frontal izquierda
model = modelaux;
model = glm::translate(model, glm::vec3(-3.5f, -1.5f, 2.5f));
model = glm::rotate(model, glm::radians(90.0f), glm::vec3(0.0f, 0.0f, 1.0f)); // Orientación horizontal
model = glm::rotate(model, glm::radians(90.0f), glm::vec3(1.0f, 0.0f, 0.0f));
model = glm::rotate(model, glm::radians(mainWindow.getllanta1()), glm::vec3(0.0f, 1.0f, 0.0f)); // Rotación de la llanta 1
model = glm::scale(model, glm::vec3(1.5f, 0.8f, 1.5f));
glUniformMatrix4fv(uniformModel, 1, GL_FALSE, glm::value_ptr(model));
color = glm::vec3(0.1f, 0.1f, 0.1f); // Negro para las llantas
glUniform3fv(uniformColor, 1, glm::value_ptr(color));
meshList[2]->RenderMeshGeometry(); // Cilindro para la llanta

// Llanta frontal derecha
model = modelaux;
model = glm::translate(model, glm::vec3(3.5f, -1.5f, 2.5f));
model = glm::rotate(model, glm::radians(90.0f), glm::vec3(0.0f, 0.0f, 1.0f)); // Orientación horizontal
model = glm::rotate(model, glm::radians(90.0f), glm::vec3(1.0f, 0.0f, 0.0f));
model = glm::rotate(model, glm::radians(mainWindow.getllanta2()), glm::vec3(0.0f, 1.0f, 0.0f)); // Rotación de la llanta 2
model = glm::scale(model, glm::vec3(1.5f, 0.8f, 1.5f));
glUniformMatrix4fv(uniformModel, 1, GL_FALSE, glm::value_ptr(model));
glUniform3fv(uniformColor, 1, glm::value_ptr(color));
meshList[2]->RenderMeshGeometry(); // Cilindro para la llanta

// Llanta trasera izquierda
model = modelaux;
model = glm::translate(model, glm::vec3(-3.5f, -1.5f, -2.5f));
model = glm::rotate(model, glm::radians(90.0f), glm::vec3(0.0f, 0.0f, 1.0f)); // Orientación horizontal
model = glm::rotate(model, glm::radians(90.0f), glm::vec3(1.0f, 0.0f, 0.0f));
model = glm::rotate(model, glm::radians(mainWindow.getllanta3()), glm::vec3(0.0f, 1.0f, 0.0f)); // Rotación de la llanta 3
model = glm::scale(model, glm::vec3(1.5f, 0.8f, 1.5f));
glUniformMatrix4fv(uniformModel, 1, GL_FALSE, glm::value_ptr(model));
glUniform3fv(uniformColor, 1, glm::value_ptr(color));
meshList[2]->RenderMeshGeometry(); // Cilindro para la llanta
```

Para las llantas se utilizó un rotate en orientación vertical y horaizontal para poderlas a justar en su respectivo lado, se utilizaron cilindros para poder realizar las llantas.

Ejecución:



- Crear un animal robot 3D:
    a. Instanciando cubos, pirámides, cilindros, conos y esferas.
    b. 4 patas articuladas en 2 partes (con teclado se puede mover las dos articulaciones de cada pata).
    c. Cola articulada o 2 orejas articuladas (con teclado se puede mover la cola o cada oreja independiente).

```
//Método getter para las articulaciones de la cola
GLfloat getArticulacionCola() { return ArticulacionCola; }
GLfloat getArticulacionCola1() { return ArticulacionCola1; }
GLfloat getArticulacionCola2() { return ArticulacionCola2; }
//Método getter para las articulaciones de las patas
GLfloat getArticulacionPataDI() { return articulacionPataDI; }
GLfloat getArticulacionPataDI_inf() { return articulacionPataDI_inf;
GLfloat getArticulacionPataDD() { return articulacionPataDD; }
GLfloat getArticulacionPataDD_inf() { return articulacionPataDD_inf;
GLfloat getArticulacionPataPI() { return articulacionPataPI; }
GLfloat getArticulacionPataPI_inf() { return articulacionPataPI_inf;
GLfloat getArticulacionPataPD() { return articulacionPataPD; }
GLfloat getArticulacionPataPD_inf() { return articulacionPataPD_inf;
```

En window.h se pusieron los getter para poder mover las articulaciones de la pata y cola.

```
if (key == GLFW_KEY_1)
{
    theWindow->ArticulacionCola += 10;
}

if (key == GLFW_KEY_2)
{
    theWindow->ArticulacionCola1 += 10;
}

if (key == GLFW_KEY_3)
{
    theWindow->ArticulacionCola2 += 10;
}

if (key == GLFW_KEY_4)
{
    theWindow->articulacionPataDI += 10;
}

if (key == GLFW_KEY_5)
{
    theWindow->articulacionPataDI_inf += 10;
}

if (key == GLFW_KEY_6)
{
    theWindow->articulacionPataDD += 10;
}

if (key == GLFW_KEY_7)
{
    theWindow->articulacionPataDD_inf += 10;
}
```

En window.cpp se crearon los if para poder asignar las teclas de movimiento para cada articulación.

```
//-----------------GATO-----------------
//CUERPO PRINCIPAL
model = glm::mat4(1.0);
model = glm::translate(model, glm::vec3(0.0f, 0.0f, 0.0f));
modelaux = model;
modelaux2 = model;
model = glm::scale(model, glm::vec3(4.0f, 2.0f, 2.0f));
glUniformMatrix4fv(uniformModel, 1, GL_FALSE, glm::value_ptr(model));
glUniformMatrix4fv(uniformProjection, 1, GL_FALSE, glm::value_ptr(projection));
glUniformMatrix4fv(uniformView, 1, GL_FALSE, glm::value_ptr(camera.calculateViewMatrix()));
color = glm::vec3(0.5f, 0.5f, 0.5f); //Color gris
glUniform3fv(uniformColor, 1, glm::value_ptr(color)); //para cambiar el color del objetos
meshList[0]->RenderMesh(); //dibuja cubo y pirámide triangular

// CUELLO
model = modelaux;
model = glm::translate(model, glm::vec3(-2.0f, 0.5f, 0.0f)); // Posición entre el cuerpo y la cabeza
model = glm::rotate(model, glm::radians(35.0f), glm::vec3(0.0f, 0.0f, 1.0f)); // Rotación para orientarlo hacia la cabeza
modelaux = model;
model = glm::scale(model, glm::vec3(0.35f, 2.0f, 0.35f)); // Cuello más largo (aumentado de 1.2f a 2.0f en X)
glUniformMatrix4fv(uniformModel, 1, GL_FALSE, glm::value_ptr(model));
color = glm::vec3(0.0f, 0.0f, 0.0f); // Color negro
glUniform3fv(uniformColor, 1, glm::value_ptr(color));
meshList[2]->RenderMeshGeometry();
```

Para el cuerpo principal, utilicé un cubo, el cual uní a un cilindro que era el que representaba el cuello del gato.

```
// CABEZA (ESFERA)
model = modelaux;
model = glm::translate(model, glm::vec3(0.0f, 2.0f, 0.0f));
modelaux = model; // Guardar matriz para orejas y visor
modelaux3 = modelaux;
model = glm::scale(model, glm::vec3(1.5f, 1.5f, 1.5f));
glUniformMatrix4fv(uniformModel, 1, GL_FALSE, glm::value_ptr(model));
color = glm::vec3(0.5f, 0.5f, 0.5f); // Color gris
glUniform3fv(uniformColor, 1, glm::value_ptr(color));
sp.render();

//OREJA IZQUIERDA (CONO)
model = glm::rotate(model, glm::radians(-35.0f), glm::vec3(0.0f, 0.0f, 1.0f));
model = glm::translate(model, glm::vec3(0.0f, 1.1f, 0.8f));
modelaux = model;
model = glm::rotate(model, glm::radians(35.0f), glm::vec3(1.0f, 0.0f, 0.0f));
model = glm::scale(model, glm::vec3(0.05f, 1.0f, 0.15f));
glUniformMatrix4fv(uniformModel, 1, GL_FALSE, glm::value_ptr(model));
color = glm::vec3(0.0f, 0.0f, 0.0f); //Color negro
glUniform3fv(uniformColor, 1, glm::value_ptr(color)); //para cambiar el color del objetos
meshList[3]->RenderMeshGeometry(); //dibuja Cono
model = modelaux;

//OREJA DERECHA (CONO)
model = glm::translate(model, glm::vec3(0.0f, 0.0f, -1.6f));
modelaux = model;
model = glm::rotate(model, glm::radians(-35.0f), glm::vec3(1.0f, 0.0f, 0.0f));
model = glm::scale(model, glm::vec3(0.05f, 1.0f, 0.15f));
glUniformMatrix4fv(uniformModel, 1, GL_FALSE, glm::value_ptr(model));
color = glm::vec3(0.0f, 0.0f, 0.0f); //Color negro
glUniform3fv(uniformColor, 1, glm::value_ptr(color)); //para cambiar el color del objetos
meshList[3]->RenderMeshGeometry(); //dibuja Cono
model = modelaux2;
```

Para la cabeza, se utilizó una esfera, y para las orejas se utilizaron conos, el modelaux me ayudó para que se fueran guardando los translate y poder guiarme y así ir realizando las orejas.
El modelaux3 me ayudó posteriormente para poder realizar ojos, nariz y boca.
El modelaux2 al final de la oreja derecha me ayudó a poder regresar al cuerpo principal y de ahí poder realizar la cola.

```
//COLA ARTICULADA

//Primera articulacion (base de la cola)
//Esfera de la primera articulacion
model = glm::translate(model, glm::vec3(2.0f, 0.0f, 0.0f));
model = glm::rotate(model, glm::radians(mainWindow.getArticulacionCola()), glm::vec3(0.0f, 0.0f, 1.0f)); // Usar el valor variable
modelaux = model;
model = glm::scale(model, glm::vec3(0.4f, 0.4f, 0.4f));
glUniformMatrix4fv(uniformModel, 1, GL_FALSE, glm::value_ptr(model));
color = glm::vec3(0.0f, 0.0f, 0.0f); //Color negro
glUniform3fv(uniformColor, 1, glm::value_ptr(color));
sp.render();
model = modelaux;

//Primer segmento de la cola
model = glm::translate(model, glm::vec3(1.5f, 0.0f, 0.0f));
modelaux = model;
model = glm::scale(model, glm::vec3(1.2f, 0.18f, 0.18f));
glUniformMatrix4fv(uniformModel, 1, GL_FALSE, glm::value_ptr(model));
color = glm::vec3(0.5f, 0.5f, 0.5f); // Color gris
glUniform3fv(uniformColor, 1, glm::value_ptr(color));
meshList[2]->RenderMeshGeometry(); // Cilindro
model = modelaux;

//Esfera de la segunda articulacion
model = glm::translate(model, glm::vec3(1.4f, 0.0f, 0.0f));
model = glm::rotate(model, glm::radians(mainWindow.getArticulacionCola1()), glm::vec3(0.0f, 0.0f, 1.0f)); // Usar el valor variable
modelaux = model;
model = glm::scale(model, glm::vec3(0.25f, 0.25f, 0.25f));
glUniformMatrix4fv(uniformModel, 1, GL_FALSE, glm::value_ptr(model));
color = glm::vec3(0.0f, 0.0f, 0.0f); //Color negro
glUniform3fv(uniformColor, 1, glm::value_ptr(color));
sp.render();
model = modelaux;
```

```
//Segundo segmento de la cola
model = glm::translate(model, glm::vec3(0.8f, 0.0f, 0.0f));
modelaux = model;
model = glm::scale(model, glm::vec3(1.0f, 0.15f, 0.15f));
glUniformMatrix4fv(uniformModel, 1, GL_FALSE, glm::value_ptr(model));
color = glm::vec3(0.5f, 0.5f, 0.5f); // Color gris
glUniform3fv(uniformColor, 1, glm::value_ptr(color));
meshList[2]->RenderMeshGeometry(); // Cilindro
model = modelaux;

//Esfera de la tercer articulacion
model = glm::translate(model, glm::vec3(1.1f, 0.0f, 0.0f));
model = glm::rotate(model, glm::radians(mainWindow.getArticulacionCola2()), glm::vec3(0.0f, 0.0f, 1.0f)); // Rotación controlada por tecla
modelaux = model;
model = glm::scale(model, glm::vec3(0.15f, 0.15f, 0.15f));
glUniformMatrix4fv(uniformModel, 1, GL_FALSE, glm::value_ptr(model));
color = glm::vec3(0.0f, 0.0f, 0.0f); //Color negro
glUniform3fv(uniformColor, 1, glm::value_ptr(color));
sp.render(); // Dibuja esfera
model = modelaux;

//Tercer segmento de la cola
model = glm::translate(model, glm::vec3(0.6f, 0.0f, 0.0f));
modelaux = model;
model = glm::scale(model, glm::vec3(0.5f, 0.08f, 0.08f));
glUniformMatrix4fv(uniformModel, 1, GL_FALSE, glm::value_ptr(model));
color = glm::vec3(0.5f, 0.5f, 0.5f); // Color gris
glUniform3fv(uniformColor, 1, glm::value_ptr(color));
meshList[2]->RenderMeshGeometry(); // Cilindro
model = modelaux3;
```

Para la cola se utilizaron 3 articulaciones y 3 segmentos, las articulaciones fueron las esferas y mis segmentos fueron cilindros.
El modelaux3 al final del tercer segmento de la cola me ayudó a poder regresar a mi cabeza y poder realizar ojos, nariz y boca.

```
//OJOS
// OJO IZQUIERDO
model = glm::translate(model, glm::vec3(-0.7f, 0.7f, 0.6f)); // Hacia el lado izquierdo de la cara
model = glm::scale(model, glm::vec3(0.4f, 0.4f, 0.4f)); // Tamaño proporcional
glUniformMatrix4fv(uniformModel, 1, GL_FALSE, glm::value_ptr(model));
color = glm::vec3(0.0f, 0.0f, 0.0f); //Color negro
glUniform3fv(uniformColor, 1, glm::value_ptr(color));
sp.render(); // Dibuja esfera para el ojo


// PUPILA IZQUIERDA
model = glm::translate(model, glm::vec3(-0.5f, 0.6f, 0.2f));
model = glm::scale(model, glm::vec3(0.3f, 0.3f, 0.3f));
glUniformMatrix4fv(uniformModel, 1, GL_FALSE, glm::value_ptr(model));
color = glm::vec3(1.0f, 1.0f, 1.0f); //Color blanco
glUniform3fv(uniformColor, 1, glm::value_ptr(color));
sp.render(); // Dibuja esfera para la pupila

// OJO DERECHO
model = modelaux3;
model = glm::translate(model, glm::vec3(-0.7f, 0.7f, -0.6f));
model = glm::scale(model, glm::vec3(0.4f, 0.4f, 0.4f));
glUniformMatrix4fv(uniformModel, 1, GL_FALSE, glm::value_ptr(model));
color = glm::vec3(0.0f, 0.0f, 0.0f); //Color negro
glUniform3fv(uniformColor, 1, glm::value_ptr(color));
sp.render(); // Dibuja esfera para el ojo

// PUPILA DERECHA
model = glm::translate(model, glm::vec3(-0.5f, 0.6f, -0.2f));
model = glm::scale(model, glm::vec3(0.3f, 0.3f, 0.3f));
glUniformMatrix4fv(uniformModel, 1, GL_FALSE, glm::value_ptr(model));
color = glm::vec3(1.0f, 1.0f, 1.0f); //Color blanco
glUniform3fv(uniformColor, 1, glm::value_ptr(color));
sp.render();
```

```
// NARIZ
model = modelaux3;
model = glm::translate(model, glm::vec3(-1.35f, 0.5f, 0.0f));
model = glm::scale(model, glm::vec3(0.2f, 0.2f, 0.3f));
glUniformMatrix4fv(uniformModel, 1, GL_FALSE, glm::value_ptr(model));
color = glm::vec3(0.0f, 0.0f, 0.0f); //Color negro
glUniform3fv(uniformColor, 1, glm::value_ptr(color));
sp.render(); // Usar esfera para una nariz redonda

// BOCA
model = modelaux3;
model = glm::translate(model, glm::vec3(-1.4f, 0.0f, 0.0f));
model = glm::scale(model, glm::vec3(0.2f, 0.1f, 0.8f));
glUniformMatrix4fv(uniformModel, 1, GL_FALSE, glm::value_ptr(model));
color = glm::vec3(0.9f, 0.2f, 0.3f); // Rosa intenso
glUniform3fv(uniformColor, 1, glm::value_ptr(color));
meshList[0]->RenderMesh(); // Usar cubo para la boca
```

```
// SONRISA LINEAL
model = modelaux3;
model = glm::translate(model, glm::vec3(-1.42f, -0.05f, 0.0f)); // Ligeramente más adelante que la boca
model = glm::scale(model, glm::vec3(0.05f, 0.03f, 0.6f)); // Línea delgada pero ancha
glUniformMatrix4fv(uniformModel, 1, GL_FALSE, glm::value_ptr(model));
color = glm::vec3(0.0f, 0.0f, 0.0f); // Color negro para la línea
glUniform3fv(uniformColor, 1, glm::value_ptr(color));
meshList[0]->RenderMesh(); // Usar cubo para hacer una línea recta

// ESQUINA IZQUIERDA CURVADA
model = modelaux3;
model = glm::translate(model, glm::vec3(-1.42f, 0.0f, 0.35f)); // Extremo izquierdo
model = glm::scale(model, glm::vec3(0.05f, 0.05f, 0.1f)); // Pequeño punto para la curvatura
glUniformMatrix4fv(uniformModel, 1, GL_FALSE, glm::value_ptr(model));
color = glm::vec3(0.0f, 0.0f, 0.0f); // Color negro
glUniform3fv(uniformColor, 1, glm::value_ptr(color));
sp.render(); // Usar esfera para punto redondeado

// ESQUINA DERECHA CURVADA
model = modelaux3;
model = glm::translate(model, glm::vec3(-1.42f, 0.0f, -0.35f)); // Extremo derecho
model = glm::scale(model, glm::vec3(0.05f, 0.05f, 0.1f)); // Pequeño punto para la curvatura
glUniformMatrix4fv(uniformModel, 1, GL_FALSE, glm::value_ptr(model));
color = glm::vec3(0.0f, 0.0f, 0.0f); // Color negro
glUniform3fv(uniformColor, 1, glm::value_ptr(color));
sp.render(); // Usar esfera para punto redondeado
model = modelaux2;
```

Para la creación de ojos, nariz y boca, se regresó a la esfera utilizando modelaux3.

El modelaux2 al final de la realización de mi boca, me ayudó a regresar al cuerpo principal y poder realizar las patas del gato.

```
//PATA DELANTERA IZQUIERDA
//Articulacion superior
model = glm::translate(model, glm::vec3(-1.7f, -1.0f, 0.8f));
model = glm::rotate(model, glm::radians(mainWindow.getArticulacionPataDI()), glm::vec3(0.0f, 0.0f, 1.0f));
modelaux = model; // Guardar para la articulación inferior
model = glm::scale(model, glm::vec3(0.3f, 0.3f, 0.3f));
glUniformMatrix4fv(uniformModel, 1, GL_FALSE, glm::value_ptr(model));
color = glm::vec3(0.0f, 0.0f, 0.0f); //Color negro
glUniform3fv(uniformColor, 1, glm::value_ptr(color));
sp.render(); // Articulación esférica
model = modelaux;

//Segmento superior de la pata
model = glm::translate(model, glm::vec3(0.0f, -0.8f, 0.0f));
modelaux = model;
model = glm::scale(model, glm::vec3(0.25f, 1.5f, 0.25f));
glUniformMatrix4fv(uniformModel, 1, GL_FALSE, glm::value_ptr(model));
color = glm::vec3(0.5f, 0.5f, 0.5f); // Color gris
glUniform3fv(uniformColor, 1, glm::value_ptr(color));
meshList[2]->RenderMeshGeometry(); // Cilindro para la pata
model = modelaux;

//Articulacion inferior
model = glm::translate(model, glm::vec3(0.0f, -0.75f, 0.0f));
model = glm::rotate(model, glm::radians(mainWindow.getArticulacionPataDI_inf()), glm::vec3(0.0f, 0.0f, 1.0f));
modelaux = model;
model = glm::scale(model, glm::vec3(0.3f, 0.3f, 0.3f));
glUniformMatrix4fv(uniformModel, 1, GL_FALSE, glm::value_ptr(model));
color = glm::vec3(0.0f, 0.0f, 0.0f); //Color negro
glUniform3fv(uniformColor, 1, glm::value_ptr(color));
sp.render(); // Articulación esférica
model = modelaux;
```

```
//Segmento inferior de la pata
model = glm::translate(model, glm::vec3(0.0f, -0.6f, 0.0f));
modelaux = model;
model = glm::scale(model, glm::vec3(0.25f, 1.5f, 0.25f));
glUniformMatrix4fv(uniformModel, 1, GL_FALSE, glm::value_ptr(model));
color = glm::vec3(0.5f, 0.5f, 0.5f); // Color gris
glUniform3fv(uniformColor, 1, glm::value_ptr(color));
meshList[2]->RenderMeshGeometry(); // Cilindro para la pata
model = modelaux;

//Esfera
model = glm::translate(model, glm::vec3(0.0f, -0.6f, 0.0f));
modelaux = model;
model = glm::scale(model, glm::vec3(0.3f, 0.3f, 0.3f));
glUniformMatrix4fv(uniformModel, 1, GL_FALSE, glm::value_ptr(model));
color = glm::vec3(0.0f, 0.0f, 0.0f); //Color negro
glUniform3fv(uniformColor, 1, glm::value_ptr(color));
sp.render(); // Articulación esférica
model = modelaux;

// PATA TRASERA IZQUIERDA
model = modelaux2;
model = glm::translate(model, glm::vec3(1.7f, -1.0f, 0.8f));
model = glm::rotate(model, glm::radians(mainWindow.getArticulacionPataPI_inf()), glm::vec3(0.0f, 0.0f, 1.0f));
modelaux = model;
model = glm::scale(model, glm::vec3(0.3f, 0.3f, 0.3f));
glUniformMatrix4fv(uniformModel, 1, GL_FALSE, glm::value_ptr(model));
color = glm::vec3(0.0f, 0.0f, 0.0f); //Color negro
glUniform3fv(uniformColor, 1, glm::value_ptr(color));
sp.render(); // Articulación esférica
model = modelaux;
```

```
// Segmento superior de la pata
model = glm::translate(model, glm::vec3(0.0f, -0.8f, 0.0f)); // Hacia abajo
modelaux = model; // Guardar para la articulación inferior
model = glm::scale(model, glm::vec3(0.2f, 1.5f, 0.25f));
glUniformMatrix4fv(uniformModel, 1, GL_FALSE, glm::value_ptr(model));
color = glm::vec3(0.5f, 0.5f, 0.5f); // Color gris
glUniform3fv(uniformColor, 1, glm::value_ptr(color));
meshList[2]->RenderMeshGeometry(); // Cilindro para la pata
model = modelaux;

// Articulación inferior (rodilla)
model = glm::translate(model, glm::vec3(0.0f, -0.75f, 0.0f)); // Fin del segmento superior
model = glm::rotate(model, glm::radians(mainWindow.getArticulacionPataPI()), glm::vec3(0.0f, 0.0f, 1.0f)); // Rotación controlada
modelaux = model;
model = glm::scale(model, glm::vec3(0.3f, 0.3f, 0.3f));
glUniformMatrix4fv(uniformModel, 1, GL_FALSE, glm::value_ptr(model));
color = glm::vec3(0.0f, 0.0f, 0.0f); //Color negro
glUniform3fv(uniformColor, 1, glm::value_ptr(color));
sp.render(); // Articulación esférica
model = modelaux;

// Segmento inferior de la pata
model = glm::translate(model, glm::vec3(0.0f, -0.6f, 0.0f)); // Hacia abajo
modelaux = model;
model = glm::scale(model, glm::vec3(0.2f, 1.5f, 0.2f));
glUniformMatrix4fv(uniformModel, 1, GL_FALSE, glm::value_ptr(model));
color = glm::vec3(0.5f, 0.5f, 0.5f); // Color gris
glUniform3fv(uniformColor, 1, glm::value_ptr(color));
meshList[2]->RenderMeshGeometry(); // Cilindro para la pata
model = modelaux;
```

```
//Esfera
model = glm::translate(model, glm::vec3(0.0f, -0.6f, 0.0f)); // Posición en la esquina del cuerpo
modelaux = model; // Guardar para la articulación inferior
model = glm::scale(model, glm::vec3(0.3f, 0.3f, 0.3f));
glUniformMatrix4fv(uniformModel, 1, GL_FALSE, glm::value_ptr(model));
color = glm::vec3(0.0f, 0.0f, 0.0f); //Color negro
glUniform3fv(uniformColor, 1, glm::value_ptr(color));
sp.render(); // Articulación esférica
model = modelaux2;

//PATA DELANTERA DERECHA
model = glm::translate(model, glm::vec3(-1.7f, -1.0f, -0.8f)); // Posición en la esquina del cuerpo
model = glm::rotate(model, glm::radians(mainWindow.getArticulacionPataDD()), glm::vec3(0.0f, 0.0f, 1.0f)); // Rotación controlada
modelaux = model; // Guardar para la articulación inferior
model = glm::scale(model, glm::vec3(0.3f, 0.3f, 0.3f));
glUniformMatrix4fv(uniformModel, 1, GL_FALSE, glm::value_ptr(model));
color = glm::vec3(0.0f, 0.0f, 0.0f); //Color negro
glUniform3fv(uniformColor, 1, glm::value_ptr(color));
sp.render(); // Articulación esférica
model = modelaux;

// Segmento superior de la pata
model = glm::translate(model, glm::vec3(0.0f, -0.8f, 0.0f)); // Hacia abajo
modelaux = model; // Guardar para la articulación inferior
model = glm::scale(model, glm::vec3(0.25f, 1.5f, 0.25f));
glUniformMatrix4fv(uniformModel, 1, GL_FALSE, glm::value_ptr(model));
color = glm::vec3(0.5f, 0.5f, 0.5f); // Color gris
glUniform3fv(uniformColor, 1, glm::value_ptr(color));
meshList[2]->RenderMeshGeometry(); // Cilindro para la pata
model = modelaux;
```

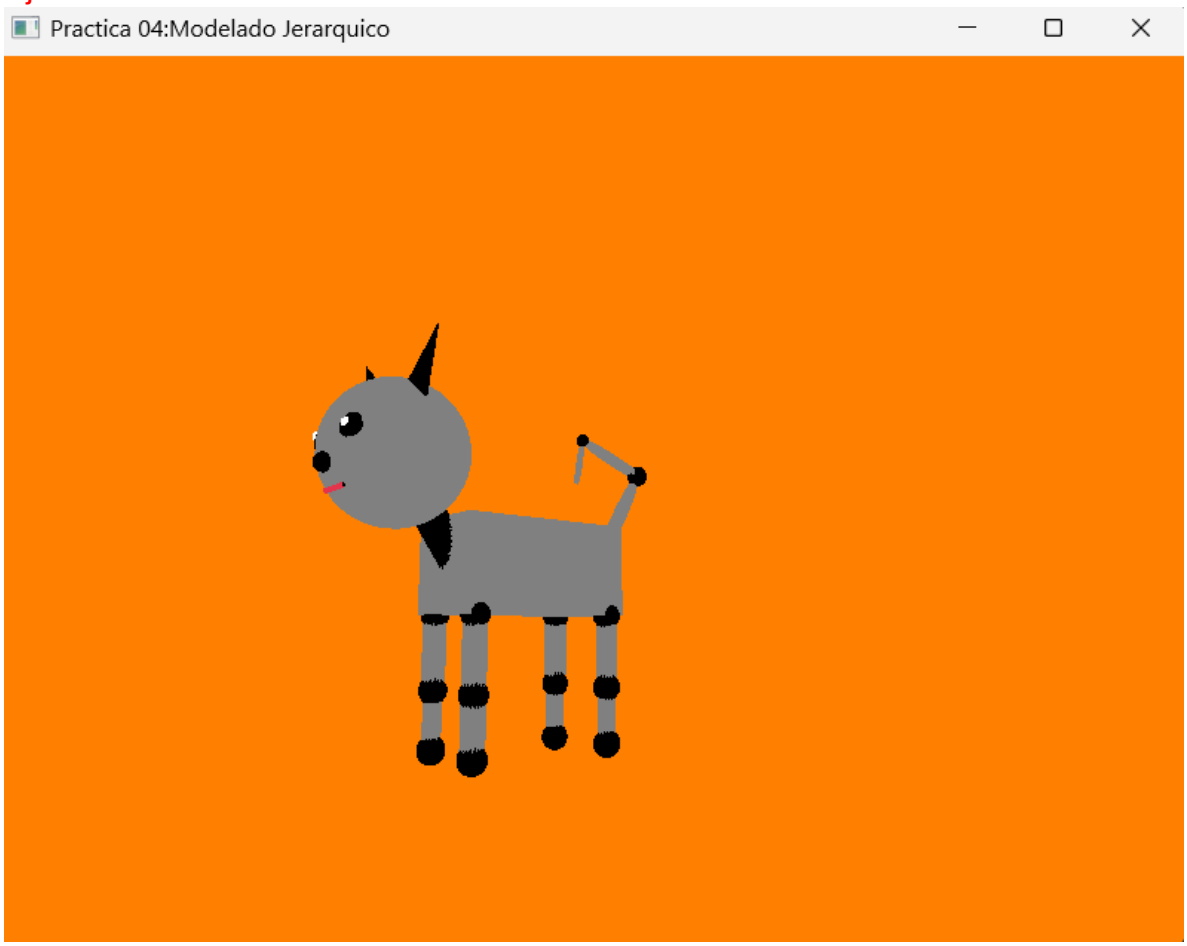
```
// Articulación inferior (rodilla)
model = glm::translate(model, glm::vec3(0.0f, -0.75f, 0.0f)); // Fin del segmento superior
model = glm::rotate(model, glm::radians(mainWindow.getArticulacionPataDD_inf()), glm::vec3(0.0f, 0.0f, 1.0f)); // Rotación controlada
modelaux = model;
model = glm::scale(model, glm::vec3(0.3f, 0.3f, 0.3f));
glUniformMatrix4fv(uniformModel, 1, GL_FALSE, glm::value_ptr(model));
color = glm::vec3(0.0f, 0.0f, 0.0f); //Color negro
glUniform3fv(uniformColor, 1, glm::value_ptr(color));
sp.render(); // Articulación esférica
model = modelaux;

// Segmento inferior de la pata
model = glm::translate(model, glm::vec3(0.0f, -0.6f, 0.0f)); // Hacia abajo
modelaux = model;
model = glm::scale(model, glm::vec3(0.2f, 1.5f, 0.2f));
glUniformMatrix4fv(uniformModel, 1, GL_FALSE, glm::value_ptr(model));
color = glm::vec3(0.5f, 0.5f, 0.5f); // Color gris
glUniform3fv(uniformColor, 1, glm::value_ptr(color));
meshList[2]->RenderMeshGeometry(); // Cilindro para la pata
model = modelaux;

//Esfera
model = glm::translate(model, glm::vec3(0.0f, -0.6f, 0.0f)); // Posición en la esquina del cuerpo
modelaux = model; // Guardar para la articulación inferior
model = glm::scale(model, glm::vec3(0.3f, 0.3f, 0.3f));
glUniformMatrix4fv(uniformModel, 1, GL_FALSE, glm::value_ptr(model));
color = glm::vec3(0.0f, 0.0f, 0.0f); //Color negro
glUniform3fv(uniformColor, 1, glm::value_ptr(color));
sp.render(); // Articulación esférica
```

```
//PATA TRASERA DERECHA
model = modelaux2;
model = glm::translate(model, glm::vec3(1.5f, -1.0f, -0.8f)); // Posición en la esquina trasera derecha del cuerpo
model = glm::rotate(model, glm::radians(mainWindow.getArticulacionPataPD()), glm::vec3(0.0f, 0.0f, 1.0f)); // Rotación controlada
modelaux = model; // Guardar para la articulación inferior
model = glm::scale(model, glm::vec3(0.3f, 0.3f, 0.3f));
glUniformMatrix4fv(uniformModel, 1, GL_FALSE, glm::value_ptr(model));
color = glm::vec3(0.0f, 0.0f, 0.0f); //Color negro
glUniform3fv(uniformColor, 1, glm::value_ptr(color));
sp.render(); // Articulación esférica
model = modelaux;

// Segmento superior de la pata
model = glm::translate(model, glm::vec3(0.0f, -0.8f, 0.0f)); // Hacia abajo
modelaux = model; // Guardar para la articulación inferior
model = glm::scale(model, glm::vec3(0.25f, 1.5f, 0.25f));
glUniformMatrix4fv(uniformModel, 1, GL_FALSE, glm::value_ptr(model));
color = glm::vec3(0.5f, 0.5f, 0.5f); // Color gris
glUniform3fv(uniformColor, 1, glm::value_ptr(color));
meshList[2]->RenderMeshGeometry(); // Cilindro para la pata
model = modelaux;

// Articulación inferior (rodilla)
model = glm::translate(model, glm::vec3(0.0f, -0.75f, 0.0f)); // Fin del segmento superior
model = glm::rotate(model, glm::radians(mainWindow.getArticulacionPataPD_inf()), glm::vec3(0.0f, 0.0f, 1.0f)); // Rotación controlada
modelaux = model;
model = glm::scale(model, glm::vec3(0.3f, 0.3f, 0.3f));
glUniformMatrix4fv(uniformModel, 1, GL_FALSE, glm::value_ptr(model));
color = glm::vec3(0.0f, 0.0f, 0.0f); //Color negro
glUniform3fv(uniformColor, 1, glm::value_ptr(color));
sp.render(); // Articulación esférica
model = modelaux;
```

```
// Segmento inferior de la pata
model = glm::translate(model, glm::vec3(0.0f, -0.6f, 0.0f)); // Hacia abajo
modelaux = model;
model = glm::scale(model, glm::vec3(0.2f, 1.5f, 0.2f));
glUniformMatrix4fv(uniformModel, 1, GL_FALSE, glm::value_ptr(model));
color = glm::vec3(0.5f, 0.5f, 0.5f); // Color gris
glUniform3fv(uniformColor, 1, glm::value_ptr(color));
meshList[2]->RenderMeshGeometry(); // Cilindro para la pata
model = modelaux;

//Esfera
model = glm::translate(model, glm::vec3(0.0f, -0.6f, 0.0f)); // Posición en la esquina del cuerpo
modelaux = model; // Guardar para la articulación inferior
model = glm::scale(model, glm::vec3(0.3f, 0.3f, 0.3f));
glUniformMatrix4fv(uniformModel, 1, GL_FALSE, glm::value_ptr(model));
color = glm::vec3(0.0f, 0.0f, 0.0f); //Color negro
glUniform3fv(uniformColor, 1, glm::value_ptr(color));
sp.render(); // Articulación esférica
```

Para las patas, se utilizaron 2 articulaciones que fueron esferas, 2 segmentos que fueron cilindros y 1 esfera al final para que pudiera simular su pata.

Ejecución:



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

- El problema presentado fue que, al momento de invertir mis triángulos en mi cara verde o roja, no quedaban parejos, si yo intentaba pegarlos, el triángulo se cortaba y ya no se veía la punta del triángulo, por lo que, por eso mismo, dichos triángulos quedaron un poco salidos.

3.- Conclusión:

a. Los ejercicios del reporte fueron los adecuados para poder comprender el uso de jerarquías para crear figuras.
b. Siento que hace falta más explicación al momento de que el profesor nos introduce a la práctica, ya que suelo revolverme un poco.
c. En conclusión, la práctica me ayudó a explorar nuevas cosas de la computación gráfica y a realizar figuras sin perder el uso de la jerarquía.

Bibliografía

- Khronos Group. (2023). OpenGL 4.6 API Core Profile Specification. Recuperado el 1 de marzo de 2025, de https://www.khronos.org/registry/OpenGL/specs/gl/glspec46.core.pdf
- Bailey, M. (2024). Learn OpenGL: Extensive tutorial resource for learning Modern OpenGL. Recuperado el 1 de marzo de 2025, de https://learnopengl.com
- de Vries, J. (2024). Modern OpenGL Tutorials: Transformations and coordinate systems. Recuperado el 1 de marzo de 2025, de https://learnopengl.com/Getting-started/Transformations
- Abi-Chahla, F. (2023). Creating Hierarchical 3D Models with Modern OpenGL and GLM. Recuperado el 1 de marzo de 2025, de https://www.opengl-tutorial.org/intermediate-tutorials/tutorial-17-quaternions/
- Dalmau, D. S. C. (2024). Modelado geométrico y transformaciones jerárquicas con OpenGL y GLM. Revista Digital de Computación Gráfica, 18(2), 45-67. Recuperado el 1 de marzo de 2025, de https://www.revistas.unam.mx/index.php/computacion-grafica/article/view/58942