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: Sanchez Calvillo Saida Mayela

Nº de Cuenta: 318164481

GRUPO DE LABORATORIO: 02

GRUPO DE TEORÍA: 04

SEMESTRE 2025-2

FECHA DE ENTREGA LÍMITE: 09 de marzo 2025

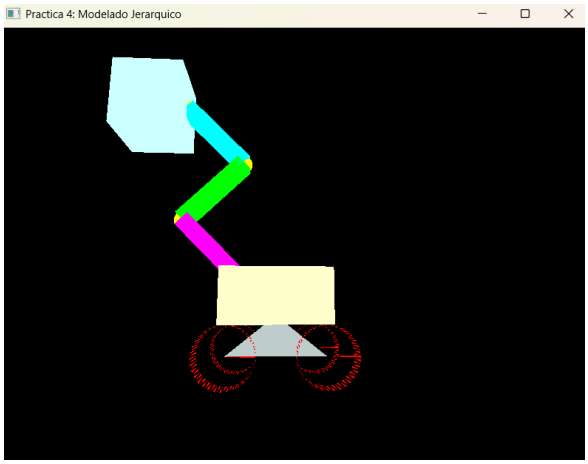
CALIFICACIÓN: _____

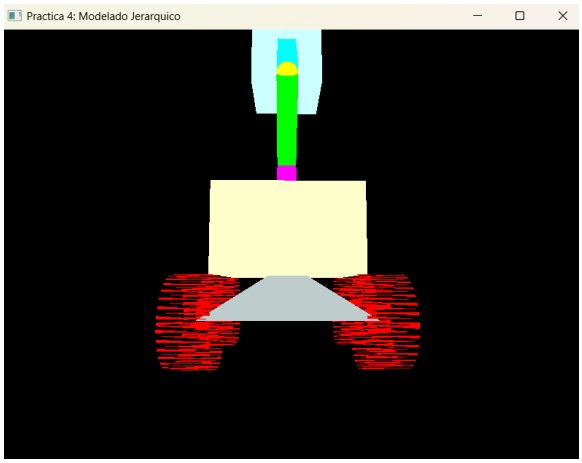
REPORTE DE PRÁCTICA:

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.- Terminar la Grúa con:

-Cuerpo (prisma rectangular)
-Base (pirámide cuadrangular)
-4 llantas (4 cilindros) con teclado se pueden girar las 4 llantas por separado





Para la grúa, lo primero que se instancio fue la cabina ya que será nuestro nodo padre. Vamos a construir nuestra base y nuestras llantas después de la cabina.

Para no reiniciar nuestra matriz, vamos a usar un model auxiliar justo donde trasladamos la cabina para poder revertir las traslaciones que se harán para la base al momento de hacer el brazo de la grúa.

```
//Cabina
modelaux3 = model;
model = glm::translate(model, glm::vec3(2.0f, 5.0f, -4.0f));
modelaux = model;
model = glm::scale(model, glm::vec3(6.0f, 3.0f, 5.0f));
glUniformMatrix4fv(uniformModel, 1, GL_FALSE, glm::value_ptr(model));
color = glm::vec3(1.0f, 1.0f, 0.8f);
glUniform3fv(uniformColor, 1, glm::value_ptr(color)); //para cambiar el color del objetos
meshList[0]->RenderMesh(); //dibuja cubo y pirámide triangular

// SE EMPIEZA EL DIBUJO DEL BRAZO
//articulación 1
//rotación alrededor de la articulación que une con la cabina
model = modelaux3;
model = glm::translate(model, glm::vec3(0.0f, 6.0f, -4.0f));
model = glm::rotate(model, glm::radians(mainWindow.getarticulacion1()), glm::vec3(0.0f, 0.0f, 1.0f));
```

Para construir la base, sabemos que la cabina se traslado 6 unidades hacia arriba, por lo que nosotros vamos a llamar una pirámide triangular y solo vamos a bajar 2 unidades en Y para que podamos visualizar la base

```
//Base
model = glm::translate(model, glm::vec3(0.0f, -2.0f, 0.0f));
modelaux = model;
model = glm::scale(model, glm::vec3(5.0f, 2.0f, 6.0f));
glUniformMatrix4fv(uniformModel, 1, GL_FALSE, glm::value_ptr(model));
color = glm::vec3(0.75f, 0.8f, 0.8f);
glUniform3fv(uniformColor, 1, glm::value_ptr(color)); //para cambiar el color del objetos
meshList[4]->RenderMesh(); //dibuja piramide cuadrangular
```

En el ejercicio de la práctica ya teníamos ocupados las articulaciones F, G, H y J para el brazo de la grúa, Por ello tenemos que crear más articulaciones parac poderlas asignar a las llantas.

Primero, en el archivo *Window.h* hemos declarado las articulaciones que necesitamos.

```
class Window
{
public:

    GLfloat getarticulacion1() { return articulacion1; }
    GLfloat getarticulacion2() { return articulacion2; }
    GLfloat getarticulacion3() { return articulacion3; }
    GLfloat getarticulacion4() { return articulacion4; }
    GLfloat getarticulacion5() { return articulacion5; }
    GLfloat getarticulacion6() { return articulacion6; }
    GLfloat getarticulacion7() { return articulacion7; }
    GLfloat getarticulacion8() { return articulacion8; }
    GLfloat getarticulacion9() { return articulacion9; }
    GLfloat getarticulacion10() { return articulacion10; }

GLfloat rotax,rotay,rotaz, articulacion1, articulacion2, articulacion3, articulacion4, articulacion5,
    articulacion6, articulacion7, articulacion8, articulacion9, articulacion10;
```

Después, en el archivo *Window.cpp* ya podemos inicializar nuestras variables de las articulaciones

```
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;
    articulacion7 = 0.0f;
    articulacion8 = 0.0f;
    articulacion9 = 0.0f;
    articulacion10 = 0.0f;
```

Y también se agregan a la función *ManejaTeclado* del mismo archivo.

```
if (key == GLFW_KEY_U)
{
    theWindow->articulacion7 += 10.0;
}
if (key == GLFW_KEY_I)
{
    theWindow->articulacion8 += 10.0;
}
if (key == GLFW_KEY_O)
{
    theWindow->articulacion9 += 10.0;
}
if (key == GLFW_KEY_P)
{
    theWindow->articulacion10 += 10.0;
}
```

Con esto ya podemos asignarles otras teclas a los movimientos de nuestras llantas.

Ahora, en nuestro archivo principal creamos las llantas heredando todo excepto la rotación que le damos a cada una. Para esto usamos cilindros con sus respectivas transformaciones geométricas y traslaciones para poder acomodarlas.

```
//LLanta uno
model = glm::translate(model, glm::vec3(-2.5f, -1.0f, 3.0f));
model = glm::scale(model, glm::vec3(1.5f, 1.5f, 1.5f));
model = glm::rotate(model, glm::radians(90.0f), glm::vec3(1.0f, 0.0f, 0.0f));   //rotate de 135° en eje z
modelaux = model;
model = glm::rotate(model, glm::radians(mainWindow.getarticulacion7()), glm::vec3(0.0f, 1.0f, 0.0f));  //Mueve con Tecla U
model = glm::rotate(model, glm::radians(mainWindow.getarticulacion6()), glm::vec3(0.0f, 1.0f, 0.0f));  //Mueve con Tecla L todas las llantas
glUniformMatrix4fv(uniformModel, 1, GL_FALSE, glm::value_ptr(model));
color = glm::vec3(1.0f, 0.0f, 0.0f);
glUniform3fv(uniformColor, 1, glm::value_ptr(color)); //para cambiar el color del objetos
meshList[2]->RenderMesh(); //dibuja cilindro


model = modelaux;
//Llanta dos
model = glm::translate(model, glm::vec3(0.0f, -4.0f, 0.0f));
modelaux = model;
model = glm::rotate(model, glm::radians(mainWindow.getarticulacion8()), glm::vec3(0.0f, 1.0f, 0.0f));  //Mueve con Tecla I
model = glm::rotate(model, glm::radians(mainWindow.getarticulacion6()), glm::vec3(0.0f, 1.0f, 0.0f));  //Mueve con Tecla L todas las llantas
glUniformMatrix4fv(uniformModel, 1, GL_FALSE, glm::value_ptr(model));
color = glm::vec3(1.0f, 0.0f, 0.0f);
glUniform3fv(uniformColor, 1, glm::value_ptr(color)); //para cambiar el color del objetos
meshList[2]->RenderMesh(); //dibuja cilindro
```

Como son del mismo tamaño, también se han heredado las escalas

Para los cilindros se debe usar *RenderMeshGeometry*, pero he usado *RenderMesh* porque le da un toque de llanta bonito y no se ve como un cilindro plano.

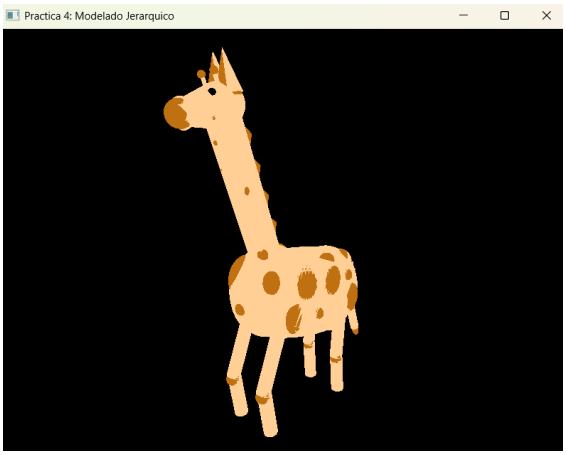
```
//LLanta tres
model = glm::translate(model, glm::vec3(3.3f, 4.0f, 0.0f));
//model = glm::rotate(model, glm::radians(90.0f), glm::vec3(1.0f, 0.0f, 0.0f));   //rotate de 135° en eje zmodelaux = model;
modelaux = model;
model = glm::rotate(model, glm::radians(mainWindow.getarticulacion9()), glm::vec3(0.0f, 1.0f, 0.0f));   //Mueve con Tecla O
model = glm::rotate(model, glm::radians(mainWindow.getarticulacion6()), glm::vec3(0.0f, 1.0f, 0.0f));  //Mueve con Tecla L todas las llantas
glUniformMatrix4fv(uniformModel, 1, GL_FALSE, glm::value_ptr(model));
color = glm::vec3(1.0f, 0.0f, 0.0f);
glUniform3fv(uniformColor, 1, glm::value_ptr(color)); //para cambiar el color del objetos
meshList[2]->RenderMesh(); //dibuja cilindro

model = modelaux;
//LLanta cuatro
model = glm::translate(model, glm::vec3(0.0f, -4.0f, 0.0f));
model = glm::rotate(model, glm::radians(mainWindow.getarticulacion10()), glm::vec3(0.0f, 1.0f, 0.0f));  //Mueve con Tecla P
model = glm::rotate(model, glm::radians(mainWindow.getarticulacion6()), glm::vec3(0.0f, 1.0f, 0.0f));  //Mueve con Tecla L todas las llantas
glUniformMatrix4fv(uniformModel, 1, GL_FALSE, glm::value_ptr(model));
color = glm::vec3(1.0f, 0.0f, 0.0f);
glUniform3fv(uniformColor, 1, glm::value_ptr(color)); //para cambiar el color del objetos
meshList[2]->RenderMesh(); //dibuja cilindro
```

A cada llanta se le ha asignado una articulación distinta para así poder moverlas de manera independiente. También les agregamos una misma articulación a todas para que se pudieran mover al mismo tiempo.

2.- Crear un animal robot 3d

-Instanciando cubos, pirámides, cilindros, conos, esferas:
-4 patas articuladas en 2 partes (con teclado se puede mover las dos articulaciones de cada pata)
-cola articulada o 2 orejas articuladas. (con teclado se puede mover la cola o cada oreja independiente



Para la jirafa, nuestro nodo padre será el cuerpo.

De ahí iremos heredando configuraciones a las demás partes del cuerpo y en este orden: cuerpo, cuello, cabeza, nariz, oreja 1, oreja 2, cuerno 1, cuerno 2, pelo y cola.

Para cada parte hemos creado varias figuras y así poderles dar una forma correcta.

Para este ejercicio se a escogido la cola para que tuviera movimiento y no las orejas, entonces, una vez hecho todo lo anterior a la cola, vamos a crear nuestra primera articulación para que una la cola con el cuerpo.

```
//Colita
    //Articulacion de cola
model = glm::rotate(model, glm::radians(91.0f), glm::vec3(0.0f, 0.0f, 1.0f));   //rotate de 135° en eje z
model = glm::translate(model, glm::vec3(3.8f, -8.7f, 0.0f));
model = glm::rotate(model, glm::radians(mainWindow.getarticulacion1()), glm::vec3(1.0f, 0.0f, 0.0f));
//modelaux = model;
//dibujar una pequeña esfera
//model = glm::scale(model, glm::vec3(2.0f, 2.5f, 2.0f));
glUniformMatrix4fv(uniformModel, 1, GL_FALSE, glm::value_ptr(model));
color = glm::vec3(1.0f, 1.0f, 0.0f);    //color Amarillo
glUniform3fv(uniformColor, 1, glm::value_ptr(color)); //para cambiar el color del objetos
sp.render();
```

A esta articulación ya se le ha puesto su respectiva articulación, que coincide con la tecla F, entonces se agregamos la cola que será un cilindro extendido en Y seguido de pelito al terminar la cola, a la que se le a asignado otra articulación y así se pueda mover de forma independiente a la cola

```
    //Cilindro
model = glm::translate(model, glm::vec3(0.0f, -1.8f, 0.0f));
modelaux = model;
model = glm::scale(model, glm::vec3(0.3f, 4.0f, 0.3f));
glUniformMatrix4fv(uniformModel, 1, GL_FALSE, glm::value_ptr(model));
color = glm::vec3(1.0f, 0.812f, 0.584f);
//color = glm::vec3(0.749f, 0.443f, 0.071f);
glUniform3fv(uniformColor, 1, glm::value_ptr(color));
meshList[2]->RenderMeshGeometry(); //dibuja cilindro

model = modelaux;

    //pelito
model = glm::translate(model, glm::vec3(-0.0f, -2.2f, 0.0f));
model = glm::rotate(model, glm::radians(mainWindow.getarticulacion2()), glm::vec3(0.0f, 0.5f, 0.2f));
modelaux = model;
//dibujar una pequeña esfera
model = glm::scale(model, glm::vec3(2.0f, 3.5f, 2.0f));
glUniformMatrix4fv(uniformModel, 1, GL_FALSE, glm::value_ptr(model));
color = glm::vec3(0.749f, 0.443f, 0.071f);
glUniform3fv(uniformColor, 1, glm::value_ptr(color)); //para cambiar el color del objetos
sp.render();
```

Para las patas hice lo mismo que con la grúa al separar el brazo de la base. Puse un modelaux3 para poder reiniciar la traslación y así poder separar las patas de las articulaciones de la cola.

```
//Cuerpo
modelaux3 = model;

model = glm::translate(model, glm::vec3(2.0f, 6.0f, -4.0f));
modelaux = model;
model = glm::scale(model, glm::vec3(3.5f, 3.0f, 2.5f));
glUniformMatrix4fv(uniformModel, 1, GL_FALSE, glm::value_ptr(model));
```

```
//Base (más cuerpo para unir las patas)
model = modelaux3;
model = glm::translate(model, glm::vec3(2.0f, 6.0f, -4.0f));
modelaux = model;
```

Para las patas también se han creado las articulaciones en un punto y ahí se han unido los cilindros que figuraran las patas.

```
//Patas delanteras
    //Articulacion 1 de pata 1
model = glm::translate(model, glm::vec3(-2.0f, -1.0f, 1.0f));
//model = glm::rotate(model, glm::radians(91.0f), glm::vec3(0.0f, 0.0f, 1.0f));   //rotate de 135° en eje z
//model = glm::translate(model, glm::vec3(3.8f, -8.7f, 0.0f));
modelaux2 = model;
model = glm::rotate(model, glm::radians(mainWindow.getarticulacion3()), glm::vec3(0.0f, 0.0f, 1.0f)); //Mueve con H
modelaux = model;
//dibujar una pequeña esfera
model = glm::scale(model, glm::vec3(2.0f, 2.0f, 2.0f));
glUniformMatrix4fv(uniformModel, 1, GL_FALSE, glm::value_ptr(model));
color = glm::vec3(1.0f, 1.0f, 0.0f);    //color Amarillo
glUniform3fv(uniformColor, 1, glm::value_ptr(color)); //para cambiar el color del objetos
sp.render();

model = modelaux;

model = glm::translate(model, glm::vec3(-0.8f, -1.4f, 0.0f));
model = glm::rotate(model, glm::radians(-20.0f), glm::vec3(0.0f, 0.0f, 1.0f));   //rotate de 135° en eje z
modelaux = model;
model = glm::scale(model, glm::vec3(0.4f, 3.0f, 0.4f));
glUniformMatrix4fv(uniformModel, 1, GL_FALSE, glm::value_ptr(model));
color = glm::vec3(1.0f, 0.812f, 0.584f);
//color = glm::vec3(0.749f, 0.443f, 0.071f);
glUniform3fv(uniformColor, 1, glm::value_ptr(color));
meshList[2]->RenderMeshGeometry(); //dibuja cilindro
```

Se uso *modelaux* para no heredar las escalas y *modelaux2* para no heredar las rotaciones de articulaciones.

Se usaron las articulaciones creadas en el ejercicio de la grúa para que podamos mover todas las articulaciones:

F - Mueve cola

G - Mueve pelitos de la cola

H - Mueve articulación 1 de la pata 1

J - Mueve articulación 2 de la pata 1

K - Mueve articulación 1 de la pata 2

L - Mueve articulación 2 de la pata 2

U - Mueve articulación 1 de la pata 3

I - Mueve articulación 2 de la pata 3

O - Mueve articulación 1 de la pata 4

P - Mueve articulación 2 de la pata 4

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

El problema más grande fue no querer reiniciar la matriz ya que no sabíamos cómo acomodar la jerarquía, pero con la ayuda del profesor pudimos resolverlos colocando varios model auxiliares, no sin antes declararlos, y así no comenzar de cero, sino como iniciar de nuevo desde el punto que nosotros escogiéramos o sin heredar las cosas que no queríamos.

```
glm::mat4 model(1.0);//Inicializar matriz de Modelo 4x4
glm::mat4 modelaux(1.0);//Inicializar matriz de Modelo 4x4 auxiliar para la jerarquía
glm::mat4 modelaux2;
glm::mat4 modelaux3;
```

También se tuvo el problema de acomodar las figuras para darle la forma que quería a la jirafa, pero con paciencia también se pudieron acomodar de una forma que al menos se entendiera lo que se quería hacer.

3.- Conclusión:

a. Los ejercicios del reporte: Complejidad, Explicación.

El principal desafío en este proceso fue gestionar correctamente las transformaciones en cascada, asegurando que cada articulación rotara de manera independiente, pero manteniendo su conexión con el resto del modelo. Esto requirió el uso de matrices auxiliares (*modelaux, modelaux2, modelaux3*), que facilitaron la aplicación de transformaciones de manera estructurada, evitando que los cambios en una parte afectaran inesperadamente a otra

b. Conclusión

En esta práctica, se implementó el modelado jerárquico para representar estructuras articuladas en una escena 3D. Para ello, se utilizó una jirafa como modelo, incorporando dos articulaciones en Z en cada pata y dos articulaciones en X en la cola, lo que permitió simular movimientos realistas en estos segmentos del cuerpo.

Desde una perspectiva técnica, se observó que el orden de las transformaciones es clave para un comportamiento adecuado de las articulaciones. En este caso, cada pata y la cola de la jirafa dependían de una jerarquía en la que primero se realizaban las traslaciones para posicionar los segmentos correctamente y luego se aplicaban las rotaciones sobre los ejes correspondientes.

El modelado jerárquico es una técnica esencial para la representación de estructuras articuladas en gráficos 3D. A través de su implementación, logramos construir un animal robótico con un sistema de articulaciones que permite movimientos en las patas y la cola. Esta práctica reforzó conocimientos sobre transformaciones en OpenGL, jerarquía de matrices y optimización del código, elementos clave en el desarrollo de animaciones y simulaciones tridimensionales.