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: Luis Daniel Salazar Islas

Nº de Cuenta: 320335163

GRUPO DE LABORATORIO: 02

GRUPO DE TEORÍA: 06

SEMESTRE 2026-1

FECHA DE ENTREGA LÍMITE: 20/09/2025

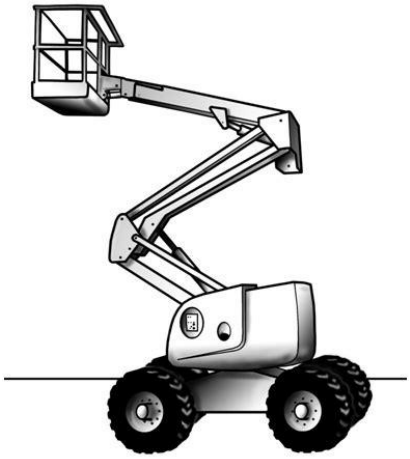
CALIFICACIÓN: _____

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.

Los ejercicios los definí en funciones separadas llamadas desde el main, de esta forma solo se comenta una línea.

El primer ejercicio consistió en terminar la grúa que previamente habíamos desarrollado en clase. Los elementos faltantes eran la base de la grúa y las llantas.



Antes de empezar con los ejercicios decidí guardar las figuras en un mapa, principalmente para no tener que recordar el índice de cada figura. De igual forma, para las llantas decidí hacerlas con cilindros con una resolución mayor. Por último, para los ejercicio iba a requerir rotar más elementos por lo que tuve que realizar modificaciones en window.cpp y window.h para agregar más variables de rotación.

```
std::map<std::string, Mesh*> fig;
```

```
    CrearCilindro(10, 1.0f);//índice 2 en MeshList
```

```
Mesh* cubo = new Mesh();
cubo->CreateMesh(cubo_vertices, cubo_indices, 24, 36);
fig["cubo"] = cubo;
meshList.push_back(cubo);
```

```
Mesh* obj1 = new Mesh();
obj1->CreateMesh(vertices_piramide_triangular, indices_piramide_triangular, 12, 12)
fig["piramideTriangular"] = obj1;
meshList.push_back(obj1);
```

```
Mesh* cilindro = new Mesh();
cilindro->CreateMeshGeometry(vertices, indices, vertices.size(), indices.size());
fig["cilindro"] = cilindro;
meshList.push_back(cilindro);
```

```
Mesh* piramide = new Mesh();
piramide->CreateMeshGeometry(piramidecuadrangular_vertices, piramidecuadrangular_i
fig["piramideCuadrangular"] = piramide;
meshList.push_back(piramide);
```

En window.h

```
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,
```

En window.cpp

```
if (key == GLFW_KEY_K)
{
    theWindow->articulacion5 += 10.0;
}
if (key == GLFW_KEY_L)
{
    theWindow->articulacion6 += 10.0;
}
if (key == GLFW_KEY_Z) {
    theWindow->articulacion7 += 10.0;
}
if (key == GLFW_KEY_X) {
    theWindow->articulacion8 += 10.0;
}
if (key == GLFW_KEY_C) {
    theWindow->articulacion9 += 10.0;
}
if (key == GLFW_KEY_V) {
    theWindow->articulacion10 += 10.0;
}
```

Procediendo con la grúa, para terminar de realizar el ejercicio utilicé el modelado con jerarquía. Dado que el nodo padre se escogió al cuerpo de la grúa, guardé sus propiedades.

```
glm::mat4 modelFather(1.0);
```

```
//CREANDO LA CABINA
model = glm::mat4(1.0f);
model = glm::translate(model, glm::vec3(0.0f, 5.0f, -4.0f));
modelFather = model;
```

Partiendo de modelFather construí las llantas y base. A cada llanta le coloque una variable que controlara la rotación a partir de una tecla.

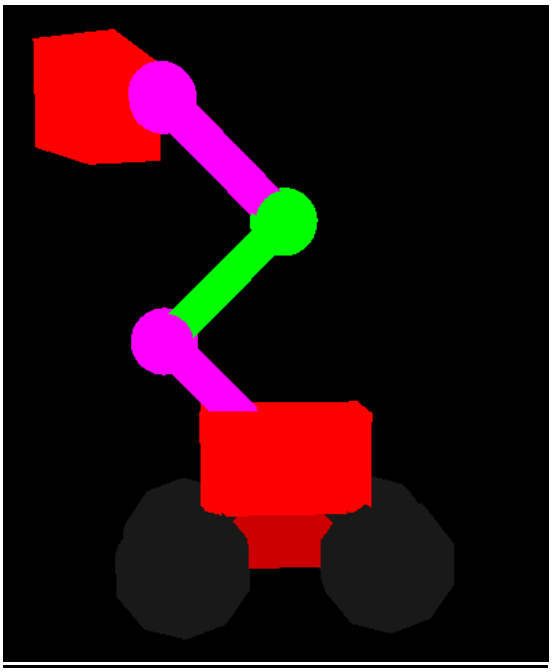
```
//llantas

model = modelFather;
model = glm::translate(model, glm::vec3(-3.0f, -3.0f, 1.0f));
model = glm::rotate(model, glm::radians(90.0f), glm::vec3(1.0f, 0.0f, 0.0f));
model = glm::scale(model, glm::vec3(2.0f, 0.75f, 2.0f));
model = glm::rotate(model, glm::radians(mainWindow.getarticulacion6()), glm::vec3(0.0f, -1.0f, 0.0f));
glUniformMatrix4fv(uniformModel, 1, GL_FALSE, glm::value_ptr(model));
color = glm::vec3(0.1f, 0.1f, 0.1f);
glUniform3fv(uniformColor, 1, glm::value_ptr(color));
glUniformMatrix4fv(uniformProjection, 1, GL_FALSE, glm::value_ptr(projection));
glUniformMatrix4fv(uniformView, 1, GL_FALSE, glm::value_ptr(camera.calculateViewMatrix()));
fig["cilindro"]->RenderMeshGeometry();

model = modelFather;
model = glm::translate(model, glm::vec3(3.0f, -3.0f, 1.0f));
model = glm::rotate(model, glm::radians(90.0f), glm::vec3(1.0f, 0.0f, 0.0f));
model = glm::scale(model, glm::vec3(2.0f, 0.75f, 2.0f));
model = glm::rotate(model, glm::radians(mainWindow.getarticulacion7()), glm::vec3(0.0f, -1.0f, 0.0f));
glUniformMatrix4fv(uniformModel, 1, GL_FALSE, glm::value_ptr(model));
color = glm::vec3(0.1f, 0.1f, 0.1f);
glUniform3fv(uniformColor, 1, glm::value_ptr(color));
glUniformMatrix4fv(uniformProjection, 1, GL_FALSE, glm::value_ptr(projection));
glUniformMatrix4fv(uniformView, 1, GL_FALSE, glm::value_ptr(camera.calculateViewMatrix()));
fig["cilindro"]->RenderMeshGeometry();
```

```
model = modelFather;
model = glm::translate(model, glm::vec3(-3.0f, -3.0f, -1.0f));
model = glm::rotate(model, glm::radians(90.0f), glm::vec3(1.0f, 0.0f, 0.0f));
model = glm::scale(model, glm::vec3(2.0f, 0.75f, 2.0f));
model = glm::rotate(model, glm::radians(mainWindow.getarticulacion8()), glm::vec3(0.0f, -1.0f, 0.0f));
glUniformMatrix4fv(uniformModel, 1, GL_FALSE, glm::value_ptr(model));
color = glm::vec3(0.1f, 0.1f, 0.1f);
glUniform3fv(uniformColor, 1, glm::value_ptr(color));
glUniformMatrix4fv(uniformProjection, 1, GL_FALSE, glm::value_ptr(projection));
glUniformMatrix4fv(uniformView, 1, GL_FALSE, glm::value_ptr(camera.calculateViewMatrix()));
fig["cilindro"]->RenderMeshGeometry();

model = modelFather;
model = glm::translate(model, glm::vec3(3.0f, -3.0f, -1.0f));
model = glm::rotate(model, glm::radians(90.0f), glm::vec3(1.0f, 0.0f, 0.0f));
model = glm::scale(model, glm::vec3(2.0f, 0.75f, 2.0f));
model = glm::rotate(model, glm::radians(mainWindow.getarticulacion9()), glm::vec3(0.0f, -1.0f, 0.0f));
glUniformMatrix4fv(uniformModel, 1, GL_FALSE, glm::value_ptr(model));
color = glm::vec3(0.1f, 0.1f, 0.1f);
glUniform3fv(uniformColor, 1, glm::value_ptr(color));
glUniformMatrix4fv(uniformProjection, 1, GL_FALSE, glm::value_ptr(projection));
glUniformMatrix4fv(uniformView, 1, GL_FALSE, glm::value_ptr(camera.calculateViewMatrix()));
fig["cilindro"]->RenderMeshGeometry();
```

```
//base

model = modelFather;
model = glm::translate(model, glm::vec3(0.0f, -1.8f, 0.0f));
model = glm::scale(model, glm::vec3(5.0f, 3.0f, 1.0f));
glUniformMatrix4fv(uniformModel, 1, GL_FALSE, glm::value_ptr(model));
color = glm::vec3(0.8f, 0.0f, 0.0f);
glUniform3fv(uniformColor, 1, glm::value_ptr(color));
glUniformMatrix4fv(uniformProjection, 1, GL_FALSE, glm::value_ptr(projection));
glUniformMatrix4fv(uniformView, 1, GL_FALSE, glm::value_ptr(camera.calculateViewMatrix()));
fig["piramideCuadrangular"]->RenderMesh();
```

Ejecución

No se muestran las rotaciones de llantas porque son difíciles de ver



Para el segundo problema se nos solicita dibujar un robot animal en 3D. Para dicha tarea me base en la siguiente imagen.

Para dibujar le modelo tome como nodo padre el cuerpo dado que tiene más conexiones.

Primero definí mis matrices para guardar propiedades geométricas de nodos ancestros.

```
glm::mat4 model(1.0);//Inicializar matriz de Modelo 4x4
glm::mat4 modelaux(1.0);//Inicializar matriz de Modelo 4x4
glm::mat4 modelCuello(1.0);
glm::mat4 modelFather(1.0);
```

Construí el torso, la matriz padre adquiere la ubicación del cuerpo.

```
//CREANDO TORSO
model = glm::mat4(1.0f);
model = glm::translate(model, glm::vec3(0.0f, 5.0f, -4.0f));
modelFather = model;
modelaux = model;
model = glm::scale(model, glm::vec3(8.0f, 3.0f, 4.0f));
glUniformMatrix4fv(uniformModel, 1, GL_FALSE, glm::value_ptr(model));
color = glm::vec3(1.0f, 1.0f, 1.0f);
glUniform3fv(uniformColor, 1, glm::value_ptr(color));
glUniformMatrix4fv(uniformProjection, 1, GL_FALSE, glm::value_ptr(projection));
glUniformMatrix4fv(uniformView, 1, GL_FALSE, glm::value_ptr(camera.calculateViewMatrix()));
fig["cubo"]->RenderMesh();
```

Luego proseguí a construir la patas, en este caso en concreto, dado que las patas muestran una simetría con respecto al cuerpo opté por construirlas por medio de

un ciclo. Itero sobre las orientaciones de los ejes, en este caso en base a los ejes “x” y “y”. Por supuesto en el caso de las articulaciones no me queda más opción que usar un switch para cada caso.

```
std::vector<std::tuple<float, float>> sim = { {1.0f, 1.0f}, {1.0f, -1.0f}, {-1.0f, 1.0f}, {-1.0f, -1.0f} };
int idx = 0;
for (auto [a, b] : sim) {
    //ARTICULACION PATA DELANTERA IZQUIERDA1
    model = modelFather;
    model = glm::translate(model, glm::vec3(2.7f*a, 0.0f, 2.0f*b));
    switch (idx) {
        case 0:
            model = glm::rotate(model, glm::radians(mainWindow.getarticulacion2()), glm::vec3(0.0f, 0.0f, -1.0f));
            break;
        case 1:
            model = glm::rotate(model, glm::radians(mainWindow.getarticulacion3()), glm::vec3(0.0f, 0.0f, -1.0f));
            break;
        case 2:
            model = glm::rotate(model, glm::radians(mainWindow.getarticulacion4()), glm::vec3(0.0f, 0.0f, -1.0f));
            break;
        case 3:
            model = glm::rotate(model, glm::radians(mainWindow.getarticulacion5()), glm::vec3(0.0f, 0.0f, -1.0f));
            break;

    }
```

```
modelaux = model;
model = glm::scale(model, glm::vec3(1.0f, 1.0f, 1.0f));
glUniformMatrix4fv(uniformModel, 1, GL_FALSE, glm::value_ptr(model));
color = glm::vec3(0.1f, 0.1f, 0.1f);
glUniform3fv(uniformColor, 1, glm::value_ptr(color)); //para cambiar el color del objetos
sp.render();

//CREANDO ANTERODILLA
model = modelaux;
model = glm::translate(model, glm::vec3(0.0f, -1.0f, b));
if (idx > 1) {
    model = glm::rotate(model, glm::radians(20.0f), glm::vec3(0.0f, 0.0f, 1.0f));
}
else {
    model = glm::rotate(model, glm::radians(-20.0f), glm::vec3(0.0f, 0.0f, 1.0f));
}
modelaux = model;
model = glm::scale(model, glm::vec3(2.0f, 2.5f, 0.5f));

glUniformMatrix4fv(uniformModel, 1, GL_FALSE, glm::value_ptr(model));
color = glm::vec3(0.9f, 0.9f, 0.9f);
glUniform3fv(uniformColor, 1, glm::value_ptr(color));
fig["cubo"]->RenderMesh();
```

```
//SEGUNDA ARTICULACION
model = modelaux;
model = glm::translate(model, glm::vec3(0.0f, -1.9f,0.0f));
if (idx > 1) {
    model = glm::rotate(model, glm::radians(-20.0f), glm::vec3(0.0f, 0.0f, 1.0f));
}
else {
    model = glm::rotate(model, glm::radians(20.0f), glm::vec3(0.0f, 0.0f, 1.0f));
}
switch (idx) {
case 0:
    model = glm::rotate(model, glm::radians(mainWindow.getarticulacion6()), glm::vec3(0.0f, 0.0f, -1.0f));
    break;
case 1:
    model = glm::rotate(model, glm::radians(mainWindow.getarticulacion7()), glm::vec3(0.0f, 0.0f, -1.0f));
    break;
case 2:
    model = glm::rotate(model, glm::radians(mainWindow.getarticulacion8()), glm::vec3(0.0f, 0.0f, -1.0f));
    break;
case 3:
    model = glm::rotate(model, glm::radians(mainWindow.getarticulacion9()), glm::vec3(0.0f, 0.0f, -1.0f));
    break;

}
```

```
modelaux = model;
model = glm::scale(model, glm::vec3(1.0f, 1.0f, 1.0f));

glUniformMatrix4fv(uniformModel, 1, GL_FALSE, glm::value_ptr(model));
color = glm::vec3(0.1f, 0.1f, 0.1f);
glUniform3fv(uniformColor, 1, glm::value_ptr(color)); //para cambiar el color del objetos
sp.render();
```

```
//RODILLA

model = modelaux;
model = glm::translate(model, glm::vec3(1.1f*a, -1.3f, 0.0f));
if (idx > 1) {
    model = glm::rotate(model, glm::radians(-30.0f), glm::vec3(0.0f, 0.0f, 1.0f));
}
else {
    model = glm::rotate(model, glm::radians(30.0f), glm::vec3(0.0f, 0.0f, 1.0f));
}
modelaux = model;
model = glm::scale(model, glm::vec3(2.0f, 5.0f, 0.5f));
glUniformMatrix4fv(uniformModel, 1, GL_FALSE, glm::value_ptr(model));
color = glm::vec3(0.9f, 0.9f, 0.9f);
glUniform3fv(uniformColor, 1, glm::value_ptr(color));
fig["cubo"]->RenderMesh();

//PATITA
model = modelaux;
model = glm::translate(model, glm::vec3(0.0f,-2.7f,0.0f));
model = glm::scale(model, glm::vec3(2.0f, 0.4f, 0.5f));
glUniformMatrix4fv(uniformModel, 1, GL_FALSE, glm::value_ptr(model));
color = glm::vec3(0.1f, 0.1f, 0.1f);
glUniform3fv(uniformColor, 1, glm::value_ptr(color));
fig["cilindro"]->RenderMeshGeometry();
idx++;
```

En mi caso, como extra agregue que la articulación de la cabeza pueda moverse.

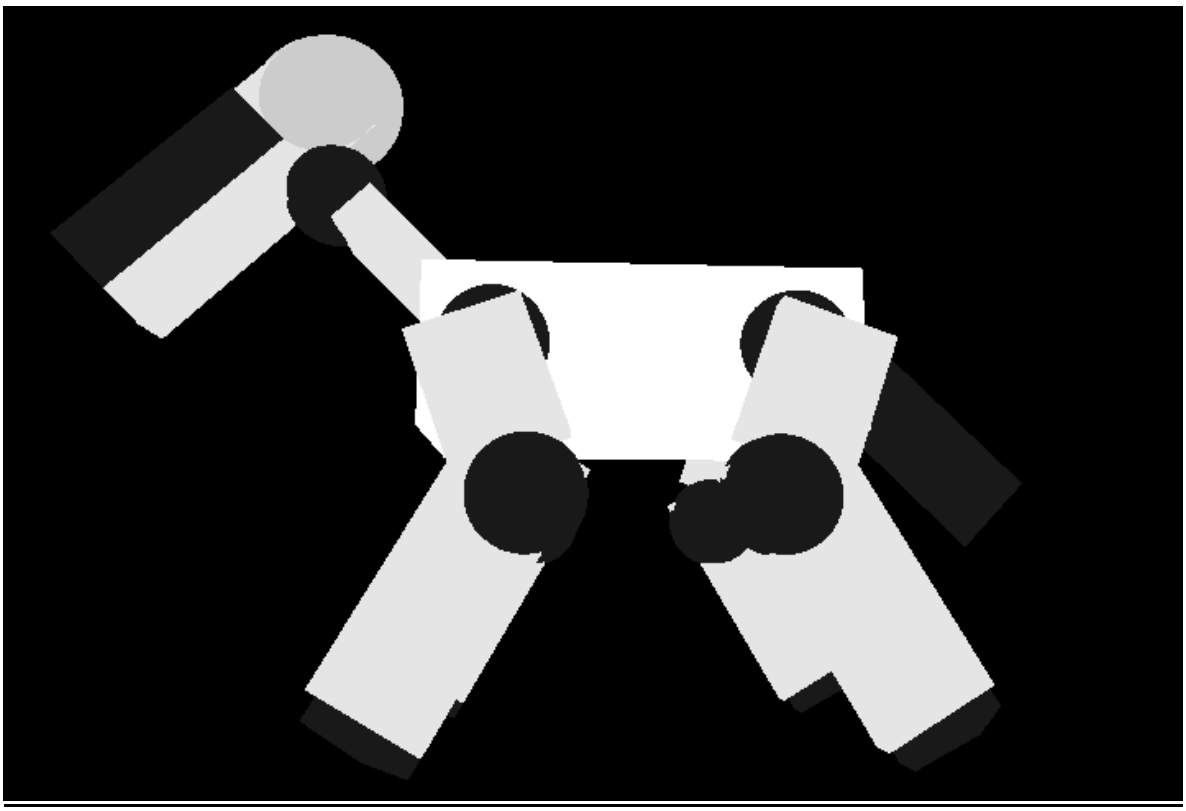
```
//CREANDO CUELLO
model = modelFather;
model = glm::translate(model, glm::vec3(-4.5f, 2.0f, 0.0f));
model = glm::rotate(model, glm::radians(45.0f), glm::vec3(0.0f, 0.0f, -1.0f));
modelCuello = model;
model = glm::scale(model, glm::vec3(3.0f, 1.0f, 2.0f));
glUniformMatrix4fv(uniformModel, 1, GL_FALSE, glm::value_ptr(model));
color = glm::vec3(0.9f, 0.9f, 0.9f);
glUniform3fv(uniformColor, 1, glm::value_ptr(color));
fig["cubo"]->RenderMeshGeometry();

//CREANDO ARTICLUACION CABEZA
model = modelCuello;
model = glm::translate(model, glm::vec3(-2.25f, 0.0f, 0.0f));
model = glm::rotate(model, glm::radians(mainWindow.getarticulacion10()), glm::vec3(0.0f, 0.0f, -1.0f));
modelaux = model;
glUniformMatrix4fv(uniformModel, 1, GL_FALSE, glm::value_ptr(model));
color = glm::vec3(0.1f, 0.1f, 0.1f);
glUniform3fv(uniformColor, 1, glm::value_ptr(color)); //para cambiar el color del objetos
sp.render();
```

```
//CREANDO CABEZA
model = modelaux;
model = glm::translate(model, glm::vec3(-1.6f, -1.5f, 0.0f));
modelaux = model;
model = glm::scale(model, glm::vec3(2.5f,6.0f,1.0f));
glUniformMatrix4fv(uniformModel, 1, GL_FALSE, glm::value_ptr(model));
color = glm::vec3(0.9f, 0.9f, 0.9f);
glUniform3fv(uniformColor, 1, glm::value_ptr(color)); //para cambiar el color del objetos
fig["cubo"]->RenderMesh();

//CREANDO FRENTE

model = modelaux;
model = glm::translate(model, glm::vec3(-0.52f, -0.76f, 0.0f));
model = glm::scale(model, glm::vec3(1.5f, 4.5f, 1.1f));
glUniformMatrix4fv(uniformModel, 1, GL_FALSE, glm::value_ptr(model));
color = glm::vec3(0.1f, 0.1f, 0.1f);
glUniform3fv(uniformColor, 1, glm::value_ptr(color)); //para cambiar el color del objetos
fig["cubo"]->RenderMesh();
```

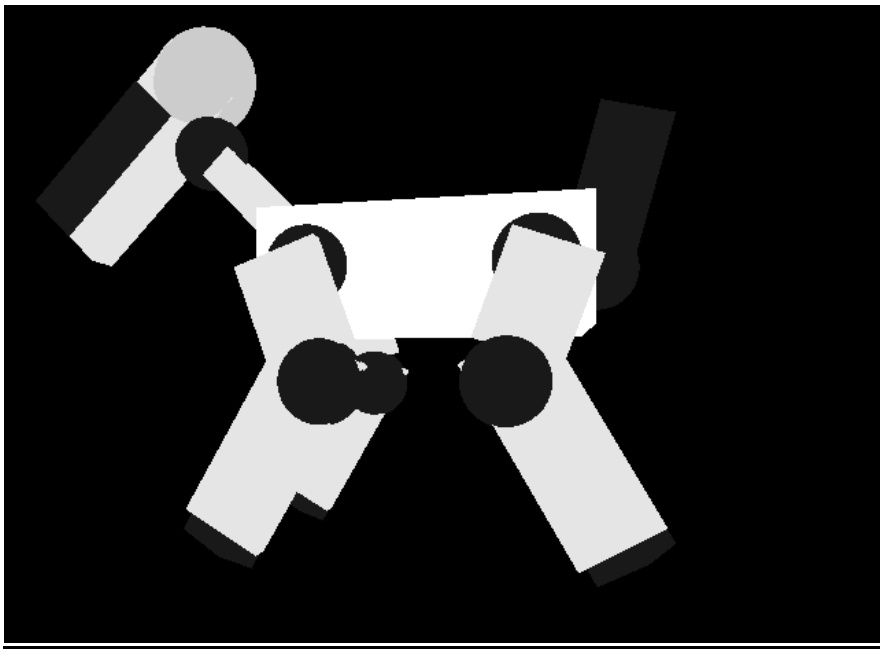
```
//CREANDO NUCA
model = modelaux;
model = glm::translate(model, glm::vec3(0.0f,2.7f,0.0f));
model = glm::scale(model, glm::vec3(1.55f, 1.55f, 0.85f));
glUniformMatrix4fv(uniformModel, 1, GL_FALSE, glm::value_ptr(model));
color = glm::vec3(0.8f, 0.8f, 0.8f);
glUniform3fv(uniformColor, 1, glm::value_ptr(color)); //para cambiar el color del objetos
sp.render();
//CREANDO ARTICULACION COLA
model = modelFather;
model = glm::translate(model, glm::vec3(4.2f, 0.0f, 0.0f));
model = glm::rotate(model, glm::radians(mainWindow.getarticulacion1()), glm::vec3(0.0f, 0.0f, -1.0f));
modelaux = model;
glUniformMatrix4fv(uniformModel, 1, GL_FALSE, glm::value_ptr(model));
color = glm::vec3(0.1f, 0.1f, 0.1f);
glUniform3fv(uniformColor, 1, glm::value_ptr(color)); //para cambiar el color del objetos
sp.render();
```

```
//CREANDO COLA
model = modelaux;
model = glm::translate(model, glm::vec3(1.5f, -1.5f, 0.0f));
model = glm::rotate(model,glm::radians(45.0f), glm::vec3(0.0f,0.0f,1.0f));
model = glm::scale(model, glm::vec3(1.0f, 4.0f, 0.0f));
glUniformMatrix4fv(uniformModel, 1, GL_FALSE, glm::value_ptr(model));
color = glm::vec3(0.1f, 0.1f, 0.1f);
glUniform3fv(uniformColor, 1, glm::value_ptr(color)); //para cambiar el color del objetos
fig["cilindro"]->RenderMeshGeometry();
```

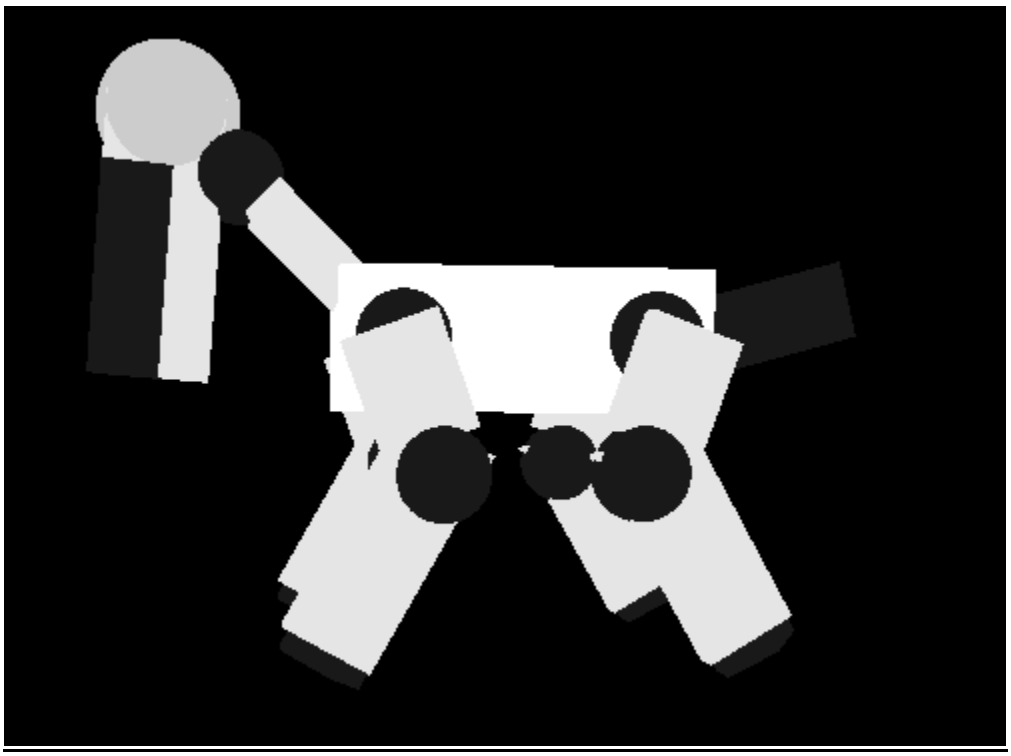
Ejecución postura neutra



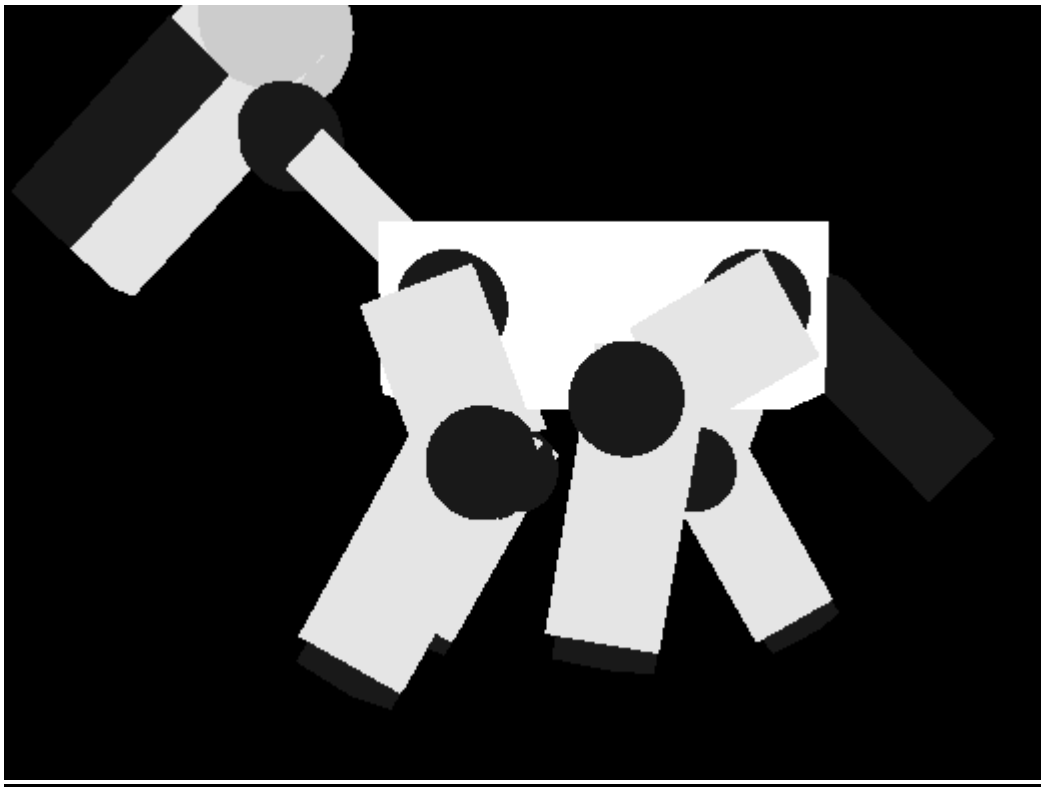
Rotando cola



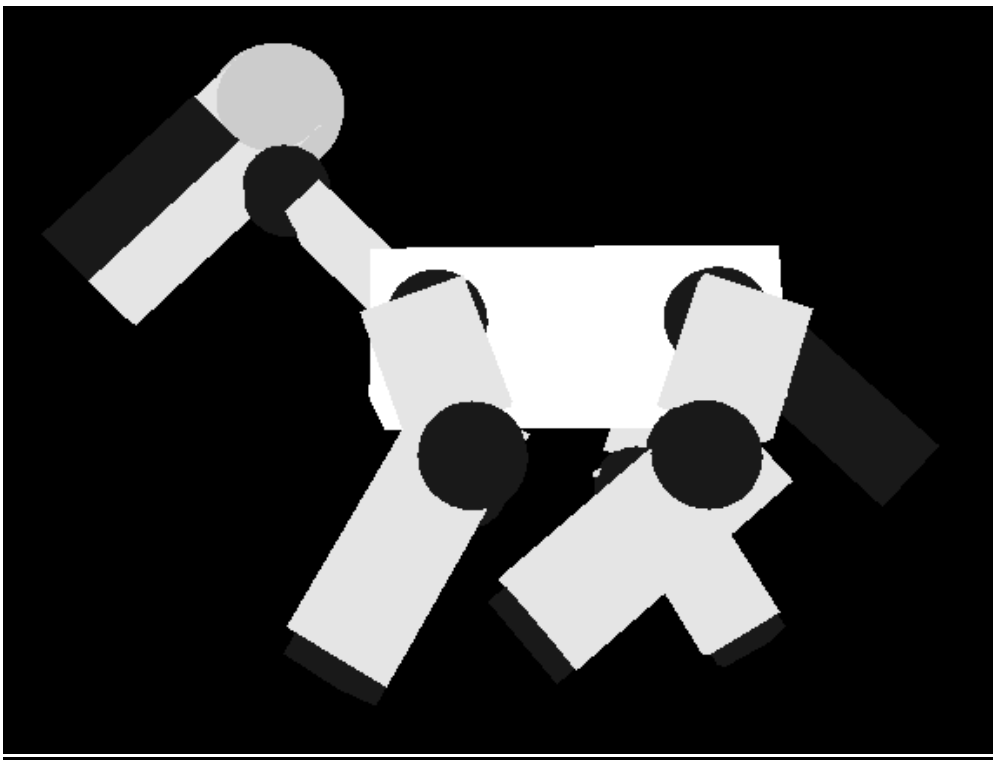
Rotando cabeza



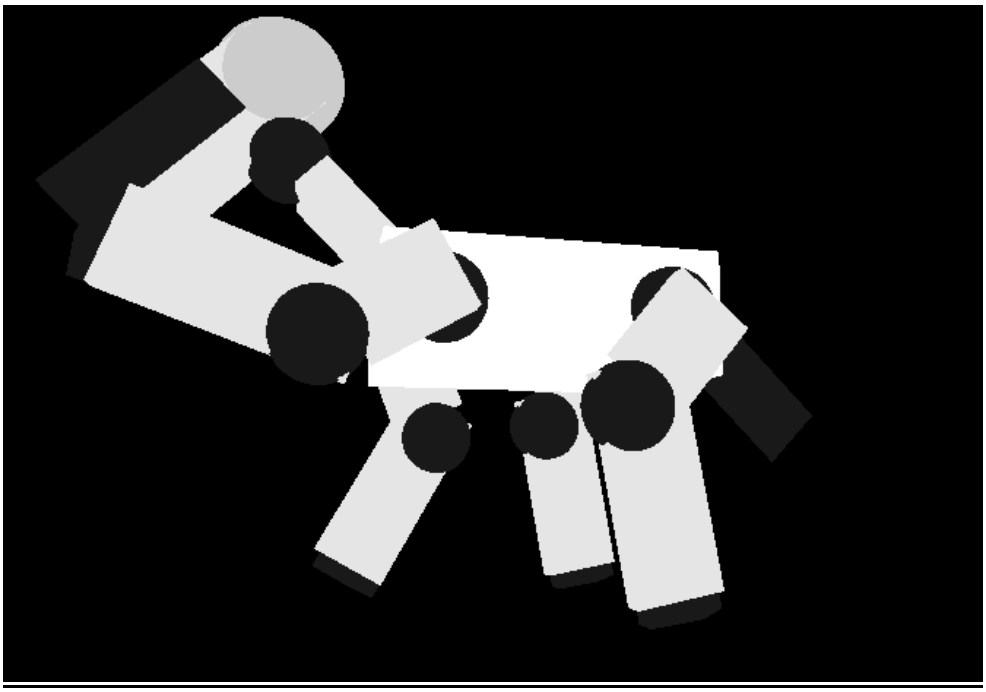
Rotando pata trasera izquierda articulación 1



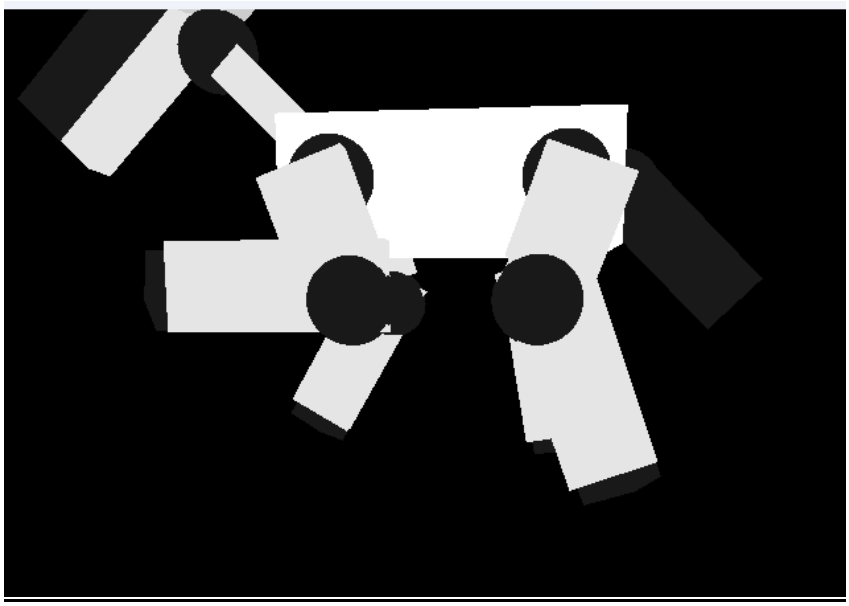
Rotando pata trasera izquierda articulación 2



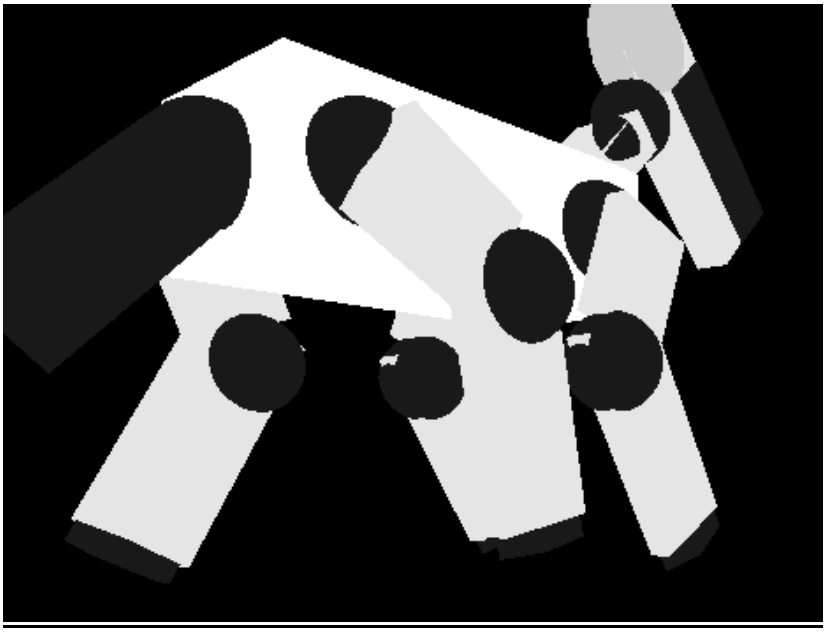
Rotando pata frontal izquierda articulación 1



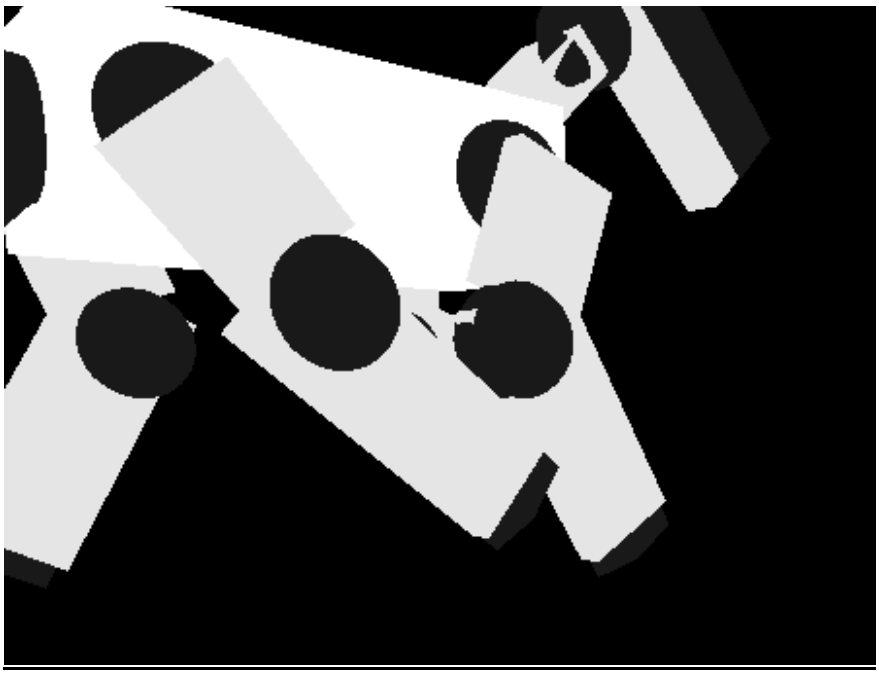
Rotando pata delantera izquierda articulación 2



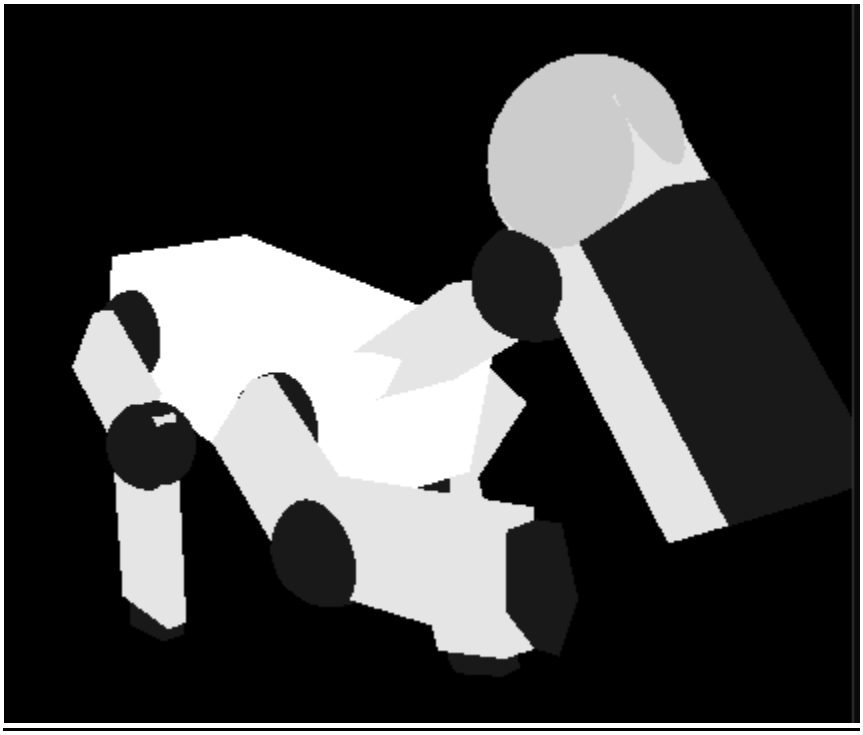
Rotando pata trasera derecha articulación 1



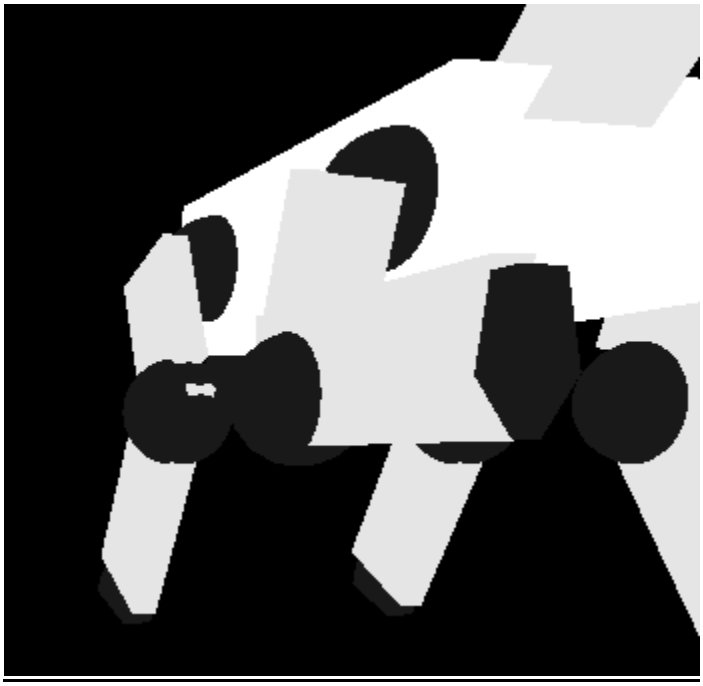
Rotando pata trasera derecha articulación 2



Rotando pata delantera derecha articulación 1



Rotando pata delantera derecha articulación 2



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

Para la elaboración del carrito no hubo mayor complicación, únicamente fue deducir un poco las medidas que debían de tener las ruedas y la base y colocarlas en posición.

La elaboración del robot fue más compleja dado que se tenía que cuidar las proporciones del cuerpo. En principio pensaba agregar más detalles, pero me resulto difícil ir agregando más elementos que encajaran con la estructura que ya tenía, por lo que al final decidí dejarlo tal cual se ve en las imágenes.

3.- Conclusión:

a. Los ejercicios del reporte

Los ejercicios, si bien se me dificultaron al inicio para escalar las partes del cuerpo me ayudaron a mejorar mi visualización espacial y aprendí una nueva forma para dibujar figuras aprovechando su simetría. De hecho, el modelo del robot se me hizo tan entretenido que quise seguir añadiendo más cosas, aunque no lo logré.

b. Comentarios generales

En general me agrado la explicación del modelado jerárquico, al principio no entendía muy bien porque queríamos algunas veces guardar las rotaciones y porque otras no, porque necesitamos más de una matriz del modelado. No fue hasta que desarrollé el ejercicio de la grúa que lo logré entender concretamente la explicación y para el ejercicio del robot fue aplicar lo aprendido por lo que me parece que la explicación cumplió.

c. Conclusión

En conclusión, aprendí a realizar modelos utilizando una jerarquía de elementos dado que construí una figura 3D desde 0. Desarrollé una nueva forma de dibujar elementos aprovechando su simetría, esto se puede observar en el código para dibujar las patas de mi robot y también aprendí como generar rotaciones por medio de teclado a través de articulaciones.

4.- Bibliografía

- Pendimarfuad Adv. (2019). Dog robot [Modelo 3D]. Free3D. https://free3d.com/es/modelo-3d/dog-robot-8103.html