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: Hernández Vázquez Daniela

Nº de Cuenta: 318092867

GRUPO DE LABORATORIO: 01

GRUPO DE TEORÍA: 02

SEMESTRE 2024-2

FECHA DE ENTREGA LÍMITE: 2 de marzo de 2024

CALIFICACIÓN: _____

REPORTE DE PRÁCTICA:

Actividad 1.

La actividad consistía en terminar la Grúa con una base (pirámide cuadrangular) y 4 llantas (4 cilindros) considerando que con el teclado se girarían las 4 llantas por separado.

Tomamos como base el ejercicio en clase donde se instanció el brazo de la grúa, en primer lugar, corregimos el elemento de la cabina agregando su jerarquía correspondiente. Posteriormente instanciamos de acuerdo al diagrama proporcionado en clase el resto de los elementos de la grúa.

```
// Llanta 1
model = glm::mat4(1.0);

modelaux = model;
model = glm::translate(model, glm::vec3(2.0f, 2.5f, -7.0f));
model = glm::rotate(model, glm::radians(90.0f), glm::vec3(1.0f, 0.0f, 0.0f));
model = glm::rotate(model, glm::radians(mainWindow.getarticulacion5()), glm::vec3(0.0f, -1.0f, 0.0f));
model = glm::scale(model, glm::vec3(1.0f, 1.0f, 1.0f));
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(0.0f, 1.0f, 1.0f);
glUniform3fv(uniformColor, 1, glm::value_ptr(color)); //para cambiar el color del objetos
meshList[2]->RenderMeshGeometry();

model = modelaux;

// Llanta 2
modelaux = model;
model = glm::translate(model, glm::vec3(-2.0f, 2.5f, -7.0f));
model = glm::rotate(model, glm::radians(90.0f), glm::vec3(1.0f, 0.0f, 0.0f));
model = glm::rotate(model, glm::radians(mainWindow.getarticulacion6()), glm::vec3(0.0f, -1.0f, 0.0f));
model = glm::scale(model, glm::vec3(1.0f, 1.0f, 1.0f));
glUniformMatrix4fv(uniformModel, 1, GL_FALSE, glm::value_ptr(model));
color = glm::vec3(0.0f, 1.0f, 1.0f);
glUniform3fv(uniformColor, 1, glm::value_ptr(color)); //para cambiar el color del objetos
meshList[2]->RenderMeshGeometry();

model = modelaux;


// BASE Y CABINA

model = modelaux;

// Base
modelaux = model;
model = glm::translate(model, glm::vec3(0.0f, 4.5f, -4.0f));
model = glm::scale(model, glm::vec3(5.0f, 3.0f, 5.0f));
glUniformMatrix4fv(uniformModel, 1, GL_FALSE, glm::value_ptr(model));
color = glm::vec3(1.0f, 0.0f, 1.0f);
glUniform3fv(uniformColor, 1, glm::value_ptr(color)); //para cambiar el color del objetos
meshList[4]->RenderMeshGeometry();

model = modelaux;

// Cabina
model = glm::translate(model, glm::vec3(0.0f, 5.75f, -4.0f));
modelaux = model;
model = glm::scale(model, glm::vec3(5.0f, 3.0f, 5.0f));
glUniformMatrix4fv(uniformModel, 1, GL_FALSE, glm::value_ptr(model));
color = glm::vec3(0.0f, 1.0f, 0.0f);
glUniform3fv(uniformColor, 1, glm::value_ptr(color)); //para cambiar el color del objetos
meshList[0]->RenderMeshGeometry();

model = modelaux;
```

Figura 4.1 Cabina y llantas

Para instanciar las llantas la rotación se daba individualmente, por lo que para evitar que avanzara con cada llanta no se modifica la matriz *model*. Los elementos de las llantas asi como la base y la cabina los colocamos antes de la creación de los brazos de la grúa para que su movimiento no influya en los demás elementos, asi mismo, no se hereda su movimiento de rotación.

Para mover cada elemento por separado dentro de la ventana Window.h y Window.cpp es necesario agregar más funciones de teclas para mover cada elemento de articulación.

```
bool* getsKeys() { return keys; }
GLfloat getXChange();
GLfloat getYChange();
void swapBuffers() { return glfwSwapBuffers(mainWindow); }
GLfloat getrotay() { return rotay; }
GLfloat getrotax() { return rotax; }
GLfloat getrotaz() { return rotaz; }
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; }
```

```
if (key == GLFW_KEY_J)
{
    theWindow->articulacion4 += 10.0;
}
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;
}
```

Figura 4.2 Declarar articulaciones Window.h y Window.cpp

Luego de instanciar obtenemos los siguientes resultados: los brazos se mueven con ‘F’,’G’,’H’ y ‘J’ mientras que las llantas se mueven con ‘K’, ‘L’,’Z’ y ‘X’

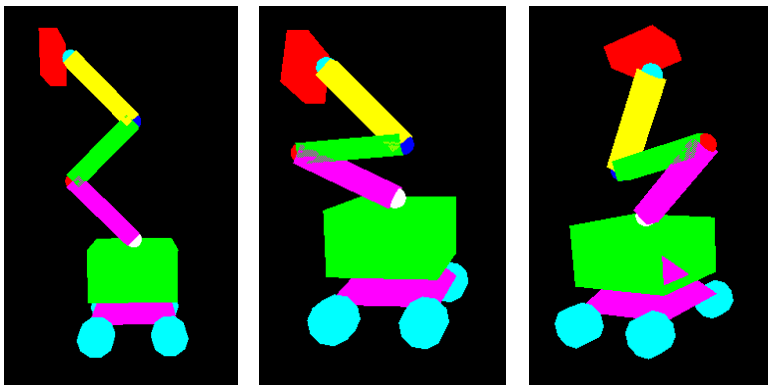

Figura 4.3 Resultado de ejecución

Actividad 2.

Como segunda actividad, instanciando cubos, pirámides, cilindros, conos, esferas, se crea un animal robot con: 4 patas articuladas en 2 partes (con teclado se puede mover las dos articulaciones de cada pata) y cola articulada o 2 orejas articuladas. (con teclado se puede mover la cola o cada oreja independiente.

Para esta actividad decidí crear un conejo robot. Tomé como base el ejercicio de la grúa peor en este caso comencé a instanciar con la matriz modelaux los elementos fijos del modelo (cola cuerpo y cabeza) y en esta misma instancia la primera oreja del conejo (se mueve con ‘F’), posteriormente comienzo a separar los elementos por matrices agregando su jerarquía correspondiente.

Solamente se crean la patas delantera y trasera izquierda junto con la oreja, las demás se reflejan con respecto a Z. Para estas se utilizan esferas para todas las articulaciones y también se utilizan esferas escaladas diferencialmente en lugar de cilindros para que el conejo quedara redondito. La nariz y las plantas de las patas se hacen con cilindros divididos en 3 partes para que sean triangulares.

```
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 modelaux1(1.0);//Inicializar matriz de Modelo 4x4 auxiliar para la jerarquía //Oreja Izq
glm::mat4 modelaux2(1.0);//Inicializar matriz de Modelo 4x4 auxiliar para la jerarquía //Oreja Der
glm::mat4 modelaux3(1.0);//Inicializar matriz de Modelo 4x4 auxiliar para la jerarquía //Pata 1
glm::mat4 modelaux4(1.0);//Inicializar matriz de Modelo 4x4 auxiliar para la jerarquía //Pata 2
glm::mat4 modelaux5(1.0);//Inicializar matriz de Modelo 4x4 auxiliar para la jerarquía //Pata 3
glm::mat4 modelaux6(1.0);//Inicializar matriz de Modelo 4x4 auxiliar para la jerarquía //Pata 4

glm::vec3 color = glm::vec3(0.0f,0.0f,0.0f); //inicializar Color para enviar a variable Uniform;
```

Figura 4.4 Matrices auxiliares utilizadas

Para instanciar las patas la rotación se daba individualmente, por lo que para evitar que girara cada elemento se utilizan matrices auxiliares diferentes, aquí en los elementos si se considera que heredan el elemento de posición y rotación. Se siguen utilizando las articulaciones declaradas anteriormente en Window.h y Window.cpp

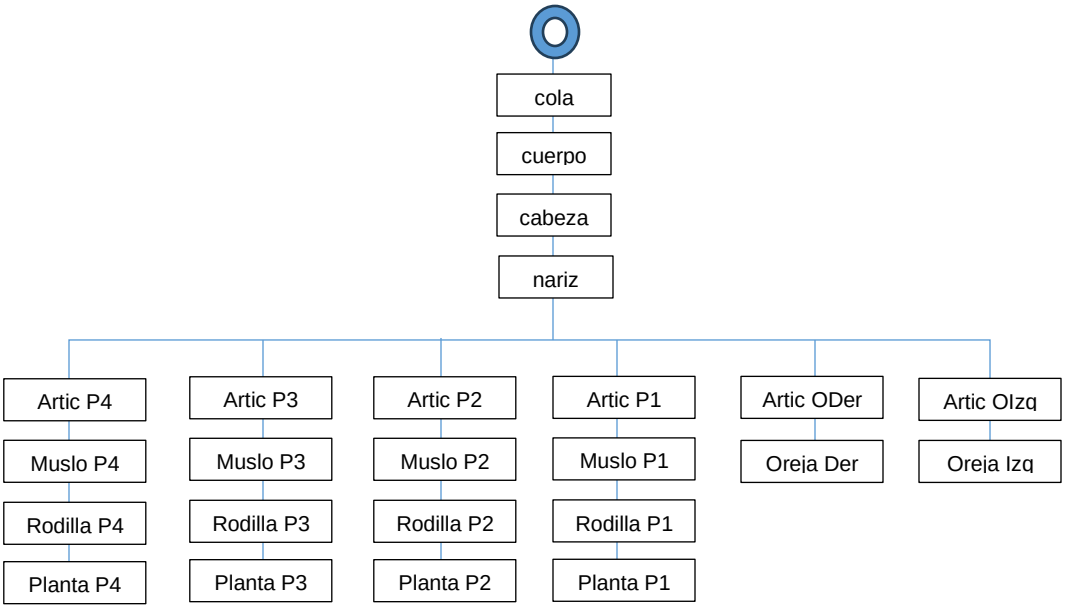

Figura 4.5 Jerarquía

```
//COLA
model = glm::translate(model, glm::vec3(0.0f, 0.0f, 0.0f));
modelaux = model;
model = glm::scale(model, glm::vec3(1.5f, 1.5f, 1.5f));
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(0.9f, 0.9f, 0.9f);
glUniform3fv(uniformColor, 1, glm::value_ptr(color)); //para cambiar el color del objetos
sp.render();

model = modelaux;

//CUERPO
model = glm::translate(model, glm::vec3(-5.0f, 2.0f, 0.0f));
modelaux = model;
model = glm::scale(model, glm::vec3(5.0f, 5.0f, 5.0f));
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)); //para cambiar el color del objetos
sp.render();

model = modelaux;

//CABEZA
model = glm::translate(model, glm::vec3(-3.0f, 6.0f, 0.0f));
modelaux = model;
model = glm::scale(model, glm::vec3(3.0f, 3.0f, 3.0f));
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
```

```
//ARTICULACIÓN OREJA IZQ
model = glm::translate(model, glm::vec3(2.5f, 3.5f, -1.75f));
model = glm::rotate(model, glm::radians(mainWindow.getarticulacion1()), glm::vec3(0.0f, 0.0f, -1.0f));
modelaux1 = model;
model = glm::scale(model, glm::vec3(0.5f, 0.5f, 0.5f));
glUniformMatrix4fv(uniformModel, 1, GL_FALSE, glm::value_ptr(model));
color = glm::vec3(0.5f, 0.5f, 0.5f);
glUniform3fv(uniformColor, 1, glm::value_ptr(color)); //para cambiar el color del objetos
sp.render();

model = modelaux1;

//OREJA IZQ
model = glm::translate(model, glm::vec3(0.0f, 3.0f, 0.0f));
modelaux1 = model;
model = glm::scale(model, glm::vec3(1.0f, 3.0f, 0.75f));
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();

model = modelaux;

//ARTICULACIÓN OREJA DER
model = glm::translate(model, glm::vec3(2.5f, 3.5f, 1.75f));
model = glm::rotate(model, glm::radians(mainWindow.getarticulacion2()), glm::vec3(0.0f, 0.0f, -1.0f));
modelaux2 = model;
model = glm::scale(model, glm::vec3(0.5f, 0.5f, 0.5f));
glUniformMatrix4fv(uniformModel, 1, GL_FALSE, glm::value_ptr(model));
color = glm::vec3(0.5f, 0.5f, 0.5f);
glUniform3fv(uniformColor, 1, glm::value_ptr(color)); //para cambiar el color del objetos
```

Figura 4.6 Patas y orejas

Luego de instanciar obtenemos los siguientes resultados: las orejas se mueven con ‘F’ y ’G’, mientras que las patas se mueven con 2 teclas cada una ’H-J’ para pata 1, ’K-L’ para pata 2, ’Z-X’ para pata 3, y ’C-V’ para pata 4.

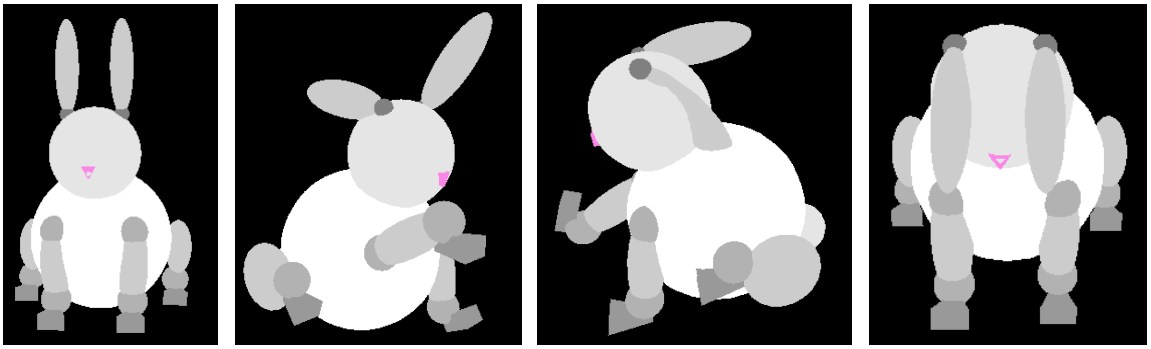

Figura 4.7 Resultado de ejecución

Problemas:

En esta práctica se tuvieron más problemas que con las prácticas anteriores, principalmente para heredar atributos a algunos elementos y que lso demás no se vieran afectados.

En un principio al no entender bien la jerarquía no podía separa los movimientos e los elementos, pero al utilizar diferentes matrices esto se arreglaba, más poque los nuevos modelos comenzaban a instanciarse desde el origen por lo que no tenía que moverlos más de lo necesario.

Aunque es posible trabajar con pocos elementos como en la grúa con un mismo modelo si necesitamos heredar atributos a otros pegados a ello sin que afecte a lso demás si es mejor utilizar diferentes matrices auxiliares.

Conclusión:

Siento que esta actividad no fue tan compleja en comparación con las otras. Fue algo difícil manejar la rotación y traslación asi como el decidir qué elementos heredar y cuales no a cada elemento independientemente, y más aún porque no es muy cómodo que se mueva la figura al colocar el mouse o apretar las teclas. Al no establecer límites se veía extraño el movimiento pero aun asi se podía trabajar bien con el modelo.

En cuanto a la explicación de clase me pareció buena para poder tener los elementos para modificar el programa, y el video nuevamente fue de ayuda para poder identificar a que elemento pertenecía que jerarquía.

Referencias:

Kessenich, J. M., Sellers, G., & Shreiner, D. (2016). *OpenGL Programming Guide: The Official Guide to Learning OpenGL, Version 4.5 with SPIR-V*. OpenGL.

Méndez Servín, M. (2022). *NOTAS PARA EL CURSO DE GRAFICACIÓN POR COMPUTADORA* [Libro electrónico]. https://prometeo.matem.unam.mx/recursos/VariosNiveles/iCartesiLibri/recursos/Notas_Graficacion_por_Computadora/index.html

RGB 0-1 Color Picker. (s. f.). https://rgbcolorpicker.com/0-1