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: Razo Villeda Fernando

Nº de Cuenta: 318299475

GRUPO DE LABORATORIO: 2

GRUPO DE TEORÍA: 4

SEMESTRE 2024-2

FECHA DE ENTREGA LÍMITE: 09/03/2024

CALIFICACIÓN: _____

Ejercicio 1:
Para la resolución de este ejercicio se implementó una grúa con una cabina base, cuatro llantas, un brazo con articulaciones móviles y una canasta.

Para lograrlo, se utilizaron clases y funciones para crear diferentes formas geométricas como cubos, pirámides, cilindros y conos, representados en el código por las funciones CrearCubo, CrearPiramideTriangular, CrearCilindro y CrearCono. También se utilizan matrices de transformación para aplicar rotaciones y traslaciones a las formas geométricas.

El código implementa un modelo jerárquico para dibujar una grúa utilizando OpenGL y GLFW. El modelo consiste en varios elementos conectados entre sí, como la cabina, los brazos y las articulaciones.

Para lograr que las articulaciones se muevan en base al modelado jerárquico, se utilizan matrices de transformación para cada articulación y se heredan las transformaciones a través de la jerarquía de la grúa. En el código, se pueden ver las transformaciones aplicadas a cada articulación mediante las funciones `glm::translate`, `glm::rotate` y `glm::scale`.

Para cada parte de la grúa se realizó lo siguiente:

- 1. Se reinicia la matriz de modelo a la matriz identidad.
- 2. Se traslada el primer brazo de la grúa a la posición adecuada.
- 3. Se aplica una rotación a la articulación 1 de la grúa.
- 4. Se traslada el primer brazo nuevamente para posicionarlo correctamente.
- 5. Se rota el primer brazo para darle la orientación adecuada.
- 6. Se escala el primer brazo para ajustarlo al tamaño deseado.
- 7. Se envía la matriz de modelo al shader para su renderización.

Este proceso se repite para cada articulación y elemento de la grúa, permitiendo que las transformaciones se acumulen y se hereden correctamente a lo largo de la jerarquía del modelo.

Dentro del bucle principal, se manipulan las matrices de modelo para posicionar y rotar las formas geométricas de acuerdo con las articulaciones de la grúa. También se configuran los uniformes de color, proyección y vista antes de renderizar cada forma geométrica.

```cpp
///*Práctica 4: Modelado Jerárquico.
//Se implementa el uso de matrices adicionales para almacenar información de transformaciones geométricas que se quiere
//heredar entre diversas instancias para que estén unidas
//Teclas de la F a la K para rotaciones de articulaciones

#include <stdio.h>
#include <string.h>
#include<cmath>
#include<vector>
#include <glew.h>
#include <glfw3.h>
//glm
#include<glm.hpp>
#include<gtc\matrix_transform.hpp>
#include<gtc\type_ptr.hpp>
#include <gtc\random.hpp>
//clases para dar orden y limpieza al código
#include"Mesh.h"
#include"Shader.h"
#include"Sphere.h"
#include"Window.h"
#include"Camera.h"
//tecla E: Rotar sobre el eje X
//tecla R: Rotar sobre el eje Y
//tecla T: Rotar sobre el eje Z
using std::vector;
//Dimensiones de la ventana
const float toRadians = 3.14159265f / 180.0; //grados a radianes
const float PI = 3.14159265f;
GLfloat deltaTime = 0.0f;
GLfloat lastTime = 0.0f;
static double limitFPS = 1.0 / 60.0;
Camera camera;
Window mainWindow;
vector<Mesh*> meshList;
vector<Shader>shaderList;
//Vertex Shader
static const char* vShader = "shaders/shader.vert";
static const char* fShader = "shaders/shader.frag";
Sphere sp = Sphere(1.0, 20, 20); //recibe radio, slices, stacks

void CrearCubo()
{
    unsigned int cubo_indices[] = {
        // front
        0, 1, 2,
        2, 3, 0,
        // right
        1, 5, 6,
        6, 2, 1,
        // back
        7, 6, 5,
        5, 4, 7,
        // left
        4, 0, 3,
        3, 7, 4,
        // bottom
        4, 5, 1,
        1, 0, 4,
        // top
        3, 2, 6,
        6, 7, 3
    };

    GLfloat cubo_vertices[] = {
        // front
        -0.5f, -0.5f,  0.5f,
        0.5f, -0.5f,  0.5f,
        0.5f,  0.5f,  0.5f,
        -0.5f,  0.5f,  0.5f,
        // back
        -0.5f, -0.5f, -0.5f,
        0.5f, -0.5f, -0.5f,
        0.5f,  0.5f, -0.5f,
        -0.5f,  0.5f, -0.5f
    };
    Mesh* cubo = new Mesh();
    cubo->CreateMesh(cubo_vertices, cubo_indices, 24, 36);
    meshList.push_back(cubo);
}

// Pirámide triangular regular
void CrearPiramideTriangular()
{
    unsigned int indices_piramide_triangular[] = {
            0,1,2,
            1,3,2,
            3,0,2,
            1,0,3

    };
    GLfloat vertices_piramide_triangular[] = {
        -0.5f, -0.5f,0.0f,  //0
        0.5f,-0.5f,0.0f,    //1
        0.0f,0.5f, -0.25f,  //2
        0.0f,-0.5f,-0.5f,   //3

    };
    Mesh* obj1 = new Mesh();
    obj1->CreateMesh(vertices_piramide_triangular, indices_piramide_triangular, 12, 12);
    meshList.push_back(obj1);

}
/*
```

```
Crear cilindro y cono con arreglos dinámicos vector creados en el Semestre 2023 - 1 : por Sánchez Pérez Om
*/
void CrearCilindro(int res, float R) {

    //constantes utilizadas en los ciclos for
    int n, i;
    //cálculo del paso interno en la circunferencia y variables que almacenarán cada coordenada de cada vé
    GLfloat dt = 2 * PI / res, x, z, y = -0.5f;

    vector<GLfloat> vertices;
    vector<unsigned int> indices;

    //ciclo for para crear los vértices de las paredes del cilindro
    for (n = 0; n <= (res); n++) {
        if (n != res) {
            x = R * cos((n)*dt);
            z = R * sin((n)*dt);
        }
        //caso para terminar el círculo
        else {
            x = R * cos((0) * dt);
            z = R * sin((0) * dt);
        }
        for (i = 0; i < 6; i++) {
            switch (i) {
            case 0:
                vertices.push_back(x);
                break;
            case 1:
                vertices.push_back(y);
                break;
            case 2:
                vertices.push_back(z);
                break;
            case 3:
                vertices.push_back(x);
                break;
            case 4:
                vertices.push_back(0.5);
                break;
            case 5:
                vertices.push_back(z);
                break;
            }
        }
    }

    //ciclo for para crear la circunferencia inferior
    for (n = 0; n <= (res); n++) {
        x = R * cos((n)*dt);
        z = R * sin((n)*dt);
        for (i = 0; i < 3; i++) {
            switch (i) {
            case 0:
                vertices.push_back(x);
                break;
            case 1:
                vertices.push_back(-0.5f);
                break;
            case 2:
                vertices.push_back(z);
                break;
            }
        }
    }

    //ciclo for para crear la circunferencia superior
    for (n = 0; n <= (res); n++) {
        x = R * cos((n)*dt);
        z = R * sin((n)*dt);
        for (i = 0; i < 3; i++) {
            switch (i) {
            case 0:
                vertices.push_back(x);
                break;
            case 1:
                vertices.push_back(0.5);
                break;
            case 2:
                vertices.push_back(z);
                break;
            }
        }
    }

    //Se generan los indices de los vértices
    for (i = 0; i < vertices.size(); i++) indices.push_back(i);

    //se genera el mesh del cilindro
    Mesh* cilindro = new Mesh();
    cilindro->CreateMeshGeometry(vertices, indices, vertices.size(), indices.size());
    meshList.push_back(cilindro);
}

//función para crear un cono
```

```cpp
void CrearCono(int res, float R) {

    //constantes utilizadas en los ciclos for
    int n, i;
    //cálculo del paso interno en la circunferencia y variables que almacenarán cada coordenada de cada vértice
    GLfloat dt = 2 * PI / res, x, z, y = -0.5f;

    vector<GLfloat> vertices;
    vector<unsigned int> indices;

    //caso inicial para crear el cono
    vertices.push_back(0.0);
    vertices.push_back(0.5);
    vertices.push_back(0.0);

    //ciclo for para crear los vértices de la circunferencia del cono
    for (n = 0; n <= (res); n++) {
        x = R * cos((n)*dt);
        z = R * sin((n)*dt);
        for (i = 0; i < 3; i++) {
            switch (i) {
            case 0:
                vertices.push_back(x);
                break;
            case 1:
                vertices.push_back(y);
                break;
            case 2:
                vertices.push_back(z);
                break;
            }
        }
    }
    vertices.push_back(R * cos(0) * dt);
    vertices.push_back(-0.5);
    vertices.push_back(R * sin(0) * dt);


    for (i = 0; i < res + 2; i++) indices.push_back(i);

    //se genera el mesh del cono
    Mesh* cono = new Mesh();
    cono->CreateMeshGeometry(vertices, indices, vertices.size(), res + 2);
    meshList.push_back(cono);
}

//función para crear pirámide cuadrangular unitaria
```

```cpp
void CrearPiramideCuadrangular()
{
    vector<unsigned int> piramidecuadrangular_indices = {
        0,3,4,
        3,2,4,
        2,1,4,
        1,0,4,
        0,1,2,
        0,2,4

    };
    vector<GLfloat> piramidecuadrangular_vertices = {
        0.5f,-0.5f,0.5f,
        0.5f,-0.5f,-0.5f,
        -0.5f,-0.5f,-0.5f,
        -0.5f,-0.5f,0.5f,
        0.0f,0.5f,0.0f,
    };
    Mesh* piramide = new Mesh();
    piramide->CreateMeshGeometry(piramidecuadrangular_vertices, piramidecuadrangular_indices, 15, 18);
    meshList.push_back(piramide);
}

void CreateShaders()
{
    Shader* shader1 = new Shader();
    shader1->CreateFromFiles(vShader, fShader);
    shaderList.push_back(*shader1);

}


int main()
{
    mainWindow = Window(1300, 1000);
    mainWindow.Initialise();
    //Cilindro y cono reciben resolución (slices, rebanadas) y Radio de circunferencia de la base y tap

    CrearCubo();//índice 0 en MeshList
    CrearPiramideTriangular();//índice 1 en MeshList
    CrearCilindro(5, 1.0f);//índice 2 en MeshList
    CrearCono(25, 2.0f);//índice 3 en MeshList
    CrearPiramideCuadrangular();//índice 4 en MeshList
    CreateShaders();



    /*Cámara se usa el comando: glm::lookAt(vector de posición, vector de orientación, vector up));
    En la clase Camera se reciben 5 datos:
    glm::vec3 vector de posición,
    glm::vec3 vector up,
    GlFloat yaw rotación para girar hacia la derecha e izquierda
    GlFloat pitch rotación para inclinar hacia arriba y abajo
```

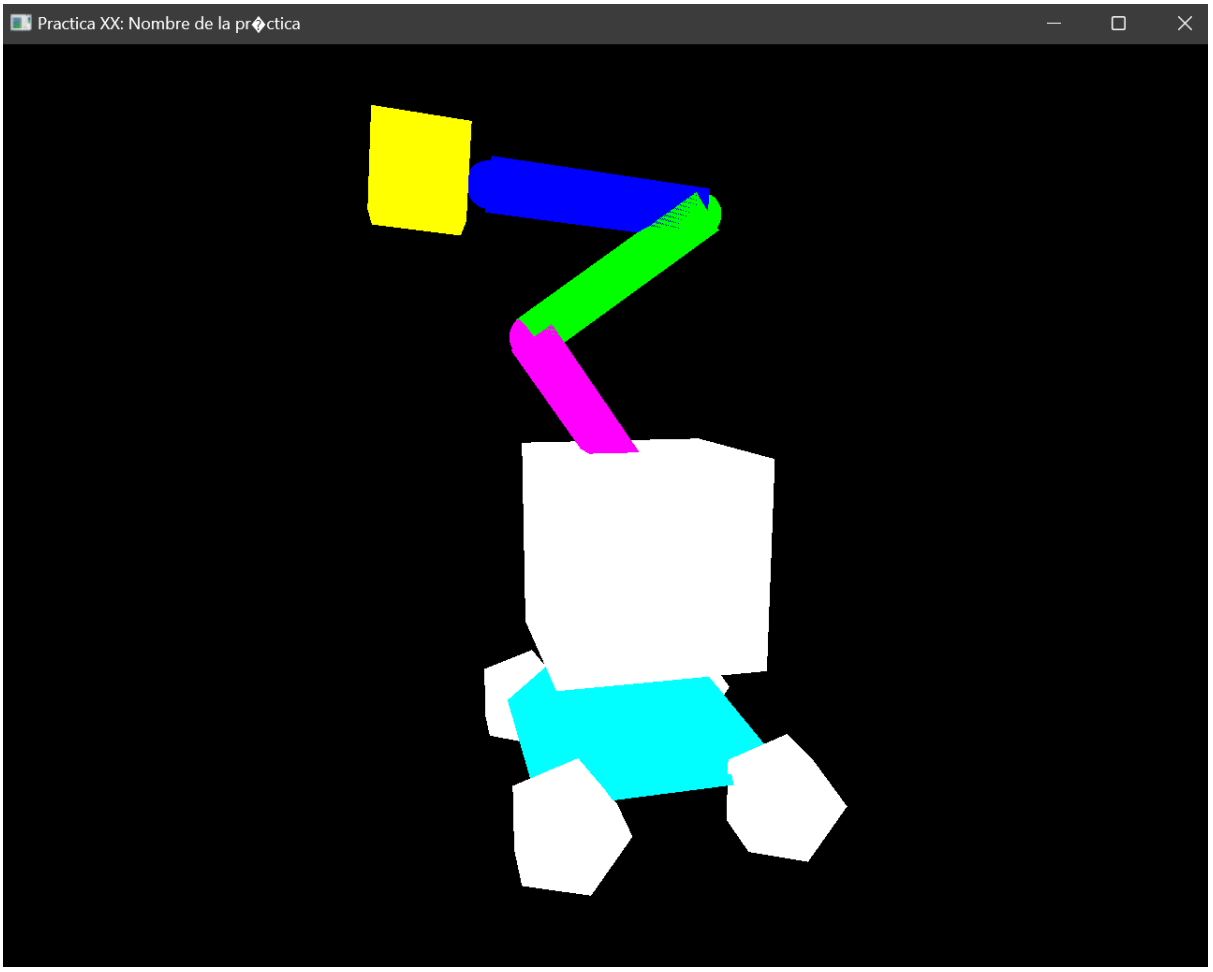
```
GlFloat velocidad de desplazamiento,
GlFloat velocidad de vuelta o de giro
Se usa el Mouse y las teclas WASD y su posición inicial está en 0,0,1 y ve hacia 0,0,-1.
*/
camera = Camera(glm::vec3(0.0f, 0.0f, 0.0f), glm::vec3(0.0f, 1.0f, 0.0f), -60.0f, 0.0f, 0.2f, 0.2f);
GLuint uniformProjection = 0;
GLuint uniformModel = 0;
GLuint uniformView = 0;
GLuint uniformColor = 0;
glm::mat4 projection = glm::perspective(glm::radians(60.0f), mainWindow.getBufferWidth() / mainWindow.getBufferHeight(),
//glm::mat4 projection = glm::ortho(-1, 1, -1, 1, 1, 10);

//Loop mientras no se cierra la ventana
sp.init(); //inicializar esfera
sp.load();//enviar la esfera al shader

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 externalModelAux(1.0);
glm::vec3 color = glm::vec3(0.0f, 0.0f, 0.0f); //inicializar Color para enviar a variable Uniform;

while (!mainWindow.getShouldClose())
{

    GLfloat now = glfwGetTime();
    deltaTime = now - lastTime;
    deltaTime += (now - lastTime) / limitFPS;
    lastTime = now;
    //Recibir eventos del usuario
    glfwPollEvents();
    //Cámara
    camera.keyControl(mainWindow.getsKeys(), deltaTime);
    camera.mouseControl(mainWindow.getXChange(), mainWindow.getYChange());

    //Limpiar la ventana
    glClearColor(0.0f, 0.0f, 0.0f, 1.0f);
    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();




    // Creando el brazo de una grúa
    //articulacion1 hasta articulación5 sólo son puntos de rotación o articulación, en este caso no dibujaremos esferas q

    //para reiniciar la matriz de modelo con valor de la matriz identidad
```

```
    model = glm::mat4(1.0);
    //AQUÍ SE DIBUJA LA CABINA, LA BASE, LAS 4 LLANTAS

    model = glm::translate(model, glm::vec3(2.5f, 5.0f, -4.0f));
    modelaux = model;
    model = glm::scale(model, glm::vec3(5.0f, 5.0f, 5.0f));
    glUniformMatrix4fv(uniformModel, 1, GL_FALSE, glm::value_ptr(model));
    //la linea de proyección solo se manda una vez a menos que en tiempo de ejecución
    //se programe cambio entre proyección ortogonal y perspectiva
    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);
    glUniform3fv(uniformColor, 1, glm::value_ptr(color)); //para cambiar el color del objetos
    meshList[0]->RenderMesh();

    model = modelaux;
    model = glm::translate(model, glm::vec3(0.0f, -3.0f, 0.0f));
    modelaux = model;
    model = glm::scale(model, glm::vec3(6.0f, 4.0f, 6.0f));
    glUniformMatrix4fv(uniformModel, 1, GL_FALSE, glm::value_ptr(model));
    //la línea de proyección solo se manda una vez a menos que en tiempo de ejecución
    //se programe cambio entre proyección ortogonal y perspectiva
    glUniformMatrix4fv(uniformProjection, 1, GL_FALSE, glm::value_ptr(projection));
    glUniformMatrix4fv(uniformView, 1, GL_FALSE, glm::value_ptr(camera.calculateViewMatrix()));
    color = glm::vec3(.0f, 1.0f, 1.0f);
    glUniform3fv(uniformColor, 1, glm::value_ptr(color)); //para cambiar el color del objetos
    meshList[4]->RenderMesh();

    model = modelaux;
    model = glm::translate(model, glm::vec3(-2.5f, -2.2f, 3.5f));
    model = glm::scale(model, glm::vec3(1.3f, 1.3f, 1.3f));
    model = glm::rotate(model, glm::radians(90.0f), glm::vec3(1.0f, 0.0f, .0f));
    model = glm::rotate(model, glm::radians(mainWindow.getarticulacion5()), glm::vec3(.0f, 1.0f, 0.0f));
    glUniformMatrix4fv(uniformModel, 1, GL_FALSE, glm::value_ptr(model));
    //la linea de proyección solo se manda una vez a menos que en tiempo de ejecución
    //se programe cambio entre proyección ortogonal y perspectiva
    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);
    glUniform3fv(uniformColor, 1, glm::value_ptr(color)); //para cambiar el color del objetos
    meshList[2]->RenderMeshGeometry();

    model = modelaux;
    model = glm::translate(model, glm::vec3(-2.5f, -2.2f, -3.5f));
    model = glm::scale(model, glm::vec3(1.3f, 1.3f, 1.3f));
    model = glm::rotate(model, glm::radians(90.0f), glm::vec3(1.0f, 0.0f, .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));
    //la linea de proyección solo se manda una vez a menos que en tiempo de ejecución
    //se programe cambio entre proyección ortogonal y perspectiva
    glUniformMatrix4fv(uniformProjection, 1, GL_FALSE, glm::value_ptr(projection));
    glUniformMatrix4fv(uniformView, 1, GL_FALSE, glm::value_ptr(camera.calculateViewMatrix()));
```

```
402         color = glm::vec3(1.0f, 1.0f, 1.0f);
403         glUniform3fv(uniformColor, 1, glm::value_ptr(color)); //para cambiar el color del objetos
404         meshList[2]->RenderMeshGeometry();
405
406         model = modelaux;
407         model = glm::translate(model, glm::vec3(2.5f, -2.2f, 3.5f));
408         model = glm::scale(model, glm::vec3(1.3f, 1.3f, 1.3f));
409         model = glm::rotate(model, glm::radians(90.0f), glm::vec3(1.0f, 0.0f, .0f));
410         model = glm::rotate(model, glm::radians(mainWindow.getarticulacion7()), glm::vec3(0.0f, 1.0f, 0.0f));
411         glUniformMatrix4fv(uniformModel, 1, GL_FALSE, glm::value_ptr(model));
412         //la línea de proyección solo se manda una vez a menos que en tiempo de ejecución
413         //se programe cambio entre proyección ortogonal y perspectiva
414         glUniformMatrix4fv(uniformProjection, 1, GL_FALSE, glm::value_ptr(projection));
415         glUniformMatrix4fv(uniformView, 1, GL_FALSE, glm::value_ptr(camera.calculateViewMatrix()));
416         color = glm::vec3(1.0f, 1.0f, 1.0f);
417         glUniform3fv(uniformColor, 1, glm::value_ptr(color)); //para cambiar el color del objetos
418         meshList[2]->RenderMeshGeometry();
419
420         model = modelaux;
421         model = glm::translate(model, glm::vec3(2.5f, -2.2f, -3.5f));
422         model = glm::scale(model, glm::vec3(1.3f, 1.3f, 1.3f));
423         model = glm::rotate(model, glm::radians(90.0f), glm::vec3(1.0f, 0.0f, .0f));
424         model = glm::rotate(model, glm::radians(mainWindow.getarticulacion8()), glm::vec3(0.0f, 1.0f, 0.0f));
425         glUniformMatrix4fv(uniformModel, 1, GL_FALSE, glm::value_ptr(model));
426         //la línea de proyección solo se manda una vez a menos que en tiempo de ejecución
427         //se programe cambio entre proyección ortogonal y perspectiva
428         glUniformMatrix4fv(uniformProjection, 1, GL_FALSE, glm::value_ptr(projection));
429         glUniformMatrix4fv(uniformView, 1, GL_FALSE, glm::value_ptr(camera.calculateViewMatrix()));
430         color = glm::vec3(1.0f, 1.0f, 1.0f);
431         glUniform3fv(uniformColor, 1, glm::value_ptr(color)); //para cambiar el color del objetos
432         meshList[2]->RenderMeshGeometry();
433
434         ////////////////////////////////////////////////////////////
435         model = glm::mat4(1.0);
436         model = glm::translate(model, glm::vec3(2.0f, 6.0f, -4.0f));
437         //rotación alrededor de la articulación que une con la cabina
438         model = glm::rotate(model, glm::radians(mainWindow.getarticulacion1()), glm::vec3(0.0f, 0.0f, 1.0f));
439
440         //primer brazo que conecta con la cabina
441         // //Traslación inicial para posicionar en -Z a los objetos
442         //model = glm::translate(model, glm::vec3(0.0f, 0.0f, -4.0f));
443         //otras transformaciones para el objeto
444         model = glm::translate(model, glm::vec3(-1.0f, 2.0f, 0.0f));
445         model = glm::rotate(model, glm::radians(135.0f), glm::vec3(0.0f, 0.0f, 1.0f));
446         modelaux = model;
447         model = glm::scale(model, glm::vec3(5.0f, 1.0f, 1.0f));
448
```

```
449         glUniformMatrix4fv(uniformModel, 1, GL_FALSE, glm::value_ptr(model));
450         //la línea de proyección solo se manda una vez a menos que en tiempo de ejecución
451         //se programe cambio entre proyección ortogonal y perspectiva
452         glUniformMatrix4fv(uniformProjection, 1, GL_FALSE, glm::value_ptr(projection));
453         glUniformMatrix4fv(uniformView, 1, GL_FALSE, glm::value_ptr(camera.calculateViewMatrix()));
454         color = glm::vec3(1.0f, 0.0f, 1.0f);
455         glUniform3fv(uniformColor, 1, glm::value_ptr(color)); //para cambiar el color del objetos
456         meshList[0]->RenderMesh(); //dibuja cubo, pirámide triangular, pirámide base cuadrangular
457         //meshList[2]->RenderMeshGeometry(); //dibuja las figuras geométricas cilindro y cono
458
459         //para descartar la escala que no quiero heredar se carga la información de la matrix auxiliar
460         model = modelaux;
461         //articulación 2
462         model = glm::translate(model, glm::vec3(2.5f, 0.0f, 0.0f));
463         model = glm::rotate(model, glm::radians(mainWindow.getarticulacion2()), glm::vec3(0.0f, 0.0f, 1.0f));
464         modelaux = model;
465         //dibujar una pequeña esfera
466         model = glm::scale(model, glm::vec3(0.5f, 0.5f, 0.5f));
467         glUniformMatrix4fv(uniformModel, 1, GL_FALSE, glm::value_ptr(model));
468         sp.render();
469
470         model = modelaux;
471         //segundo brazo
472         model = glm::translate(model, glm::vec3(0.0f, -2.5f, 0.0f));
473
474         modelaux = model;
475         model = glm::scale(model, glm::vec3(1.0f, 5.0f, 1.0f));
476         glUniformMatrix4fv(uniformModel, 1, GL_FALSE, glm::value_ptr(model));
477         color = glm::vec3(0.0f, 1.0f, 0.0
478         glUniform3fv(uniformColor, 1, glm
479         meshList[0]->RenderMesh(); //dibuja
480
481         model = modelaux;
482
483         //articulación 3 extremo derecho del segundo brazo
484         model = glm::translate(model, glm::vec3(0.0f, -2.5f, 0.0f));
485         model = glm::rotate(model, glm::radians(mainWindow.getarticulacion3()), glm::vec3(0.0f, 0.0f, 1.0f));
486         modelaux = model;
487
488         //dibujar una pequeña esfera
489         model = glm::scale(model, glm::vec3(0.5f, 0.5f, 0.5f));
490         glUniformMatrix4fv(uniformModel, 1, GL_FALSE, glm::value_ptr(model));
491         sp.render();
492
493         ////////////////tercer brazo////////////////
494         model = modelaux;
495         model = glm::rotate(model, glm::radians(50.0f), glm::vec3(0.0f, 0.0f, 1.0f));
496         model = glm::translate(model, glm::vec3(2.5f, .0f, 0.0f));
497
498         modelaux = model;
499         model = glm::scale(model, glm::vec3(5.0f, 1.0f, 1.0f));
500         glUniformMatrix4fv(uniformModel, 1, GL_FALSE, glm::value_ptr(model));
```

```
501         color = glm::vec3(0.0f, .0f, 1.0f);
502         glUniform3fv(uniformColor, 1, glm::value_ptr(color)); //para cambiar el color del objetos
503         meshList[0]->RenderMesh(); //dibuja cubo y pirámide triangular
504         model = modelaux;
505
506         //articulación 3 extremo derecho del segundo brazo
507         model = glm::translate(model, glm::vec3(2.5f, .0f, 0.0f));
508         //model = glm::rotate(model, glm::radians(mainWindow.getarticulacion3()), glm::vec3(0.0f, 0.0f, 1.0f));
509         modelaux = model;
510
511         //dibujar una pequeña esfera
512
513         model = glm::scale(model, glm::vec3(0.5f, 0.5f, 0.5f));
514         glUniformMatrix4fv(uniformModel, 1, GL_FALSE, glm::value_ptr(model));
515         sp.render();
516
517         model = modelaux;
518         model = glm::translate(model, glm::vec3(1.5f, .0f, 0.0f));
519         model = glm::scale(model, glm::vec3(2.0f, 2.0f, 2.0f));
520         model = glm::rotate(model, glm::radians(mainWindow.getarticulacion4()), glm::vec3(0.0f, 1.0f, .0f));
521         glUniformMatrix4fv(uniformModel, 1, GL_FALSE, glm::value_ptr(model));
522         color = glm::vec3(1.0f, 1.0f, .0f);
523         glUniform3fv(uniformColor, 1, glm::value_ptr(color)); //para cambiar el color del objetos
524         meshList[0]->RenderMesh(); //dibuja cubo y pirámide triangular
525         model = modelaux;
526
527
528
529
```



Problemas presentados

No se presentaron problemas en este ejercicio.

Ejercicio 2:

Para este ejercicio se implementó un modelo jerárquico de una araña utilizando OpenGL y la biblioteca glm en C++. La araña está compuesta por varias partes articuladas, incluyendo cuatro patas articuladas en dos partes y dos orejas articuladas.

El programa comienza inicializando una ventana y cargando shaders. Luego, se definen varias funciones para crear los componentes geométricos de la araña, como cubos, pirámides, cilindros y conos.

El ciclo principal del programa renderiza la araña en función de las posiciones y rotaciones de sus partes articuladas. Se utilizan matrices de modelo para transformar y posicionar cada parte de la araña en relación con las otras. Por ejemplo, se utiliza una matriz de modelo para la pata, que se escala, rota y traslada para formar la estructura articulada.

Los controles del teclado se utilizan para manipular las articulaciones de la araña, lo que permite al usuario mover las patas y las orejas para cambiar la pose de la araña en tiempo real.

Para añadir articulaciones a la araña robot 3D en el ejercicio, se tuvo que modificar los archivos “Window.h” y “Window.cpp” para incorporar la funcionalidad de manejar las articulaciones mediante el teclado. Esto implica la adición de métodos y variables relacionadas con el control de las articulaciones, como por ejemplo, métodos para obtener el estado de las teclas asociadas a las articulaciones y variables para almacenar los ángulos de rotación de las articulaciones.

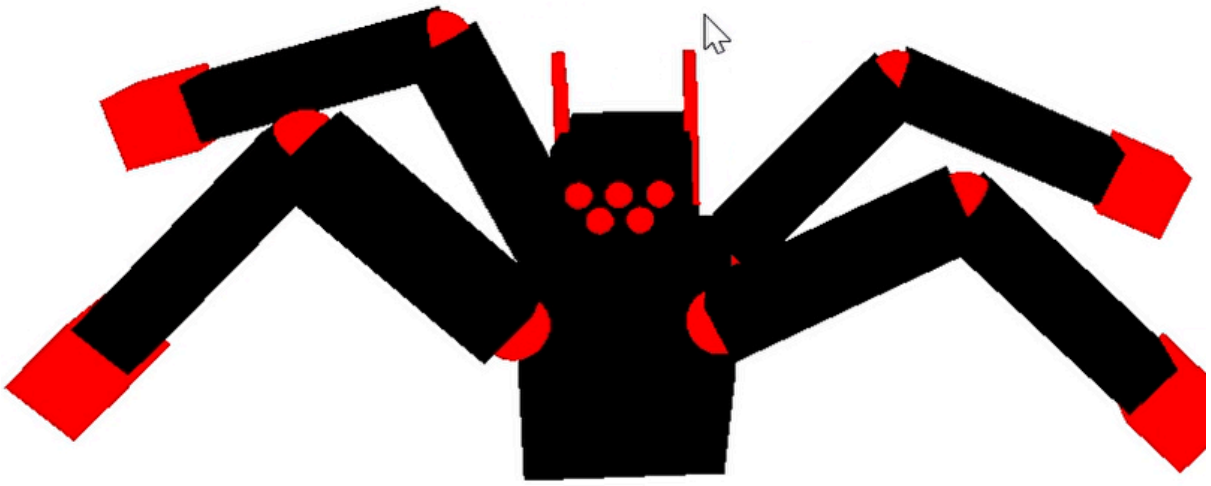
```
////Segundo ejercicio - ARAÑA
GLfloat now = glfwGetTime();
deltaTime = now - lastTime;
deltaTime += (now - lastTime) / limitFPS;
lastTime = now;
//Recibir eventos del usuario
glfwPollEvents();
//Cámara
camera.keyControl(mainWindow.getsKeys(), deltaTime);
camera.mouseControl(mainWindow.getXChange(), mainWindow.getYChange());

//Limpiar la ventana
glClearColor(1.0f, 1.0f, 1.0f, 1.0f);
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();




// Creando el brazo de una grúa
//articulacion1 hasta articulación5 sólo son puntos de rotación o articulación, en este caso no dibujaremo

//para reiniciar la matriz de modelo con valor de la matriz identidad
model = glm::mat4(1.0);
//AQUÍ SE DIBUJA LA CABINA, LA BASE, LAS 4 LLANTAS

glUniformMatrix4fv(uniformProjection, 1, GL_FALSE, glm::value_ptr(projection));
//Cuerpo
model = glm::translate(model, glm::vec3(0.0f, 0.0f, 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));
glUniformMatrix4fv(uniformView, 1, GL_FALSE, glm::value_ptr(camera.calculateViewMatrix()));
color = glm::vec3(0.0f, 0.0f, 0.0f);
glUniform3fv(uniformColor, 1, glm::value_ptr(color)); //para cambiar el color del objetos
meshList[0]->RenderMesh();
```

```
//Cabeza
model = modelaux;
model = glm::translate(model, glm::vec3(0.0f, 2.5f, 0.0f));
modelaux = model;
externalModelAux = model;
model = glm::scale(model, glm::vec3(2.0f, 2.0f, 2.0f));
glUniformMatrix4fv(uniformModel, 1, GL_FALSE, glm::value_ptr(model));
color = glm::vec3(0.0f, 0.0f, 0.0f);
glUniform3fv(uniformColor, 1, glm::value_ptr(color)); //para cambiar el color del objetos
meshList[0]->RenderMesh();

//Patas

float vec[] = { 1.5f,-1.5f };
for (float x : vec)
{
    for (float z : vec) {
        model = externalModelAux;
        model = glm::translate(model, glm::vec3(x, -1.5f, z));
        glUniformMatrix4fv(uniformModel, 1, GL_FALSE, glm::value_ptr(model));
        if (x > 0 and z > 0) {
            model = glm::rotate(model, glm::radians(mainWindow.getarticulacion1()), glm::vec3(0.0f, 0.0f, 1.0f));
        }
        else if (x > 0 and z < 0) {
            model = glm::rotate(model, glm::radians(mainWindow.getarticulacion3()), glm::vec3(0.0f, 0.0f, 1.0f));
        }
        else if (x < 0 and z < 0) {
            model = glm::rotate(model, glm::radians(mainWindow.getarticulacion5()), glm::vec3(0.0f, 0.0f, 1.0f));
        }
        else {
            model = glm::r                                    m::radians(mainWindow.getarticulacion7()), glm::vec3(0.0f, 0.0f, 1.0f));
        }

        color = glm::vec3(1.0f, 0.0f, 0.0f);
        glUniform3fv(uniformColor, 1, glm::value_ptr(color)); //para cambiar el color del objetos
        modelaux = model;
        model = glm::scale(model, glm::vec3(.5f, .5f, .5f));
        glUniformMatrix4fv(uniformModel, 1, GL_FALSE, glm::value_ptr(model));
        sp.render();

        model = modelaux;
        if (x > 0) {
            model = glm::rotate(model, glm::radians(45.0f), glm::vec3(0.0f, 0.0f, 1.0f));
        }
        else {
            model = glm::rotate(model, glm::radians(135.0f), glm::vec3(0.0f, 0.0f, 1.0f));
        }
        model = glm::translate(model, glm::vec3(2.f, 0.f, 0.0f));
```

(GLclampf)(1.0F)
Buscar en línea

```
619             modelaux = model;
620             model = glm::scale(model, glm::vec3(4.0f, 1.0f, 1.0f));
621             glUniformMatrix4fv(uniformModel, 1, GL_FALSE, glm::value_ptr(model));
622             color = glm::vec3(0.0f, 0.0f, 0.0f);
623             glUniform3fv(uniformColor, 1, glm::value_ptr(color)); //para cambiar el color del objetos
624             meshList[0]->RenderMesh();
625 
626             model = modelaux;
627             model = glm::translate(model, glm::vec3(2.f, .0f, .0f));
628             glUniformMatrix4fv(uniformModel, 1, GL_FALSE, glm::value_ptr(model));
629             if (x > 0 and z > 0) {
630                 model = glm::rotate(model, glm::radians(mainWindow.getarticulacion2()), glm::vec3(0.0f, 0.0f, 1.0f));
631             }
632             else if (x > 0 and z < 0) {
633                 model = glm::rotate(model, glm::radians(mainWindow.getarticulacion4()), glm::vec3(0.0f, 0.0f, 1.0f));
634             }
635             else if (x < 0 and z < 0) {
636                 model = glm::rotate(model, glm::radians(mainWindow.getarticulacion6()), glm::vec3(0.0f, 0.0f, 1.0f));
637             }
638             else {
639                 model = glm::rotate(model, glm::radians(mainWindow.getarticulacion8()), glm::vec3(0.0f, 0.0f, 1.0f));
640             }
641 
642             color = glm::vec3(1.0f, 0.0f, 0.0f);
643             glUniform3fv(uniformColor, 1, glm::value_ptr(color)); //para cambiar el color del objetos
644             modelaux = model;
645             model = glm::scale(model, glm::vec3(.5f, .5f, .5f));
646             glUniformMatrix4fv(uniformModel, 1, GL_FALSE, glm::value_ptr(model));
647             sp.render();
648 
649             model = modelaux;
650             if (x > 0) {
651                 model = glm::rotate(model, glm::radians(-90.0f), glm::vec3(0.0f, 0.0f, 1.0f));
652             }
653             else {
654                 model = glm::rotate(model, glm::radians(90.0f), glm::vec3(0.0f, 0.0f, 1.0f));
655             }
656             model = glm::translate(model, glm::vec3(2.f, 0.f, 0.0f));
657             modelaux = model;
658             model = glm::scale(model, glm::vec3(4.0f, 1.0f, 1.0f));
659             glUniformMatrix4fv(uniformModel, 1, GL_FALSE, glm::value_ptr(model));
660             color = glm::vec3(0.0f, 0.0f, 0.0f);
661             glUniform3fv(uniformColor, 1, glm::value_ptr(color)); //para cambiar el color del objetos
662             meshList[0]->RenderMesh();
663 
664             model = modelaux;
665             model = glm::translate(model, glm::vec3(2.6f, 0.f, 0.0f));
666             modelaux = model;
667             model = glm::scale(model, glm::vec3(1.2f, 1.2f, 1.2f));
668             glUniformMatrix4fv(uniformModel, 1, GL_FALSE, glm::value_ptr(model));
669             color = glm::vec3(1.0f, 0.0f, 0.0f);
670             glUniform3fv(uniformColor, 1, glm::value_ptr(color)); //para cambiar el color del objetos
671             meshList[0]->RenderMesh();
672         }
673 
674     }
675     //Antenas
676     model = externalModelAux;
677     model = glm::translate(model, glm::vec3(1.f, 1.f, 0.0f));
678     modelaux = model;
679     model = glm::scale(model, glm::vec3(0.2f, 2.0f, 0.2f));
680     color = glm::vec3(1.0f, 0.0f, 0.0f);
681     model = glm::rotate(model, glm::radians(mainWindow.getarticulacion9()), glm::vec3(1.0f, 0.0f, .0f));
682     glUniform3fv(uniformColor, 1, glm::value_ptr(color)); //para cambiar el color del objetos
683     glUniformMatrix4fv(uniformModel, 1, GL_FALSE, glm::value_ptr(model));
684     meshList[0]->RenderMesh();
685 
686     model = modelaux;
687     model = glm::translate(model, glm::vec3(-2.0f, 0.0f, 0.0f));
688     modelaux = model;
689     model = glm::scale(model, glm::vec3(0.2f, 2.0f, 0.2f));
690 
691     color = glm::vec3(1.0f, 0.0f, 0.0f);
692     model = glm::rotate(model, glm::radians(mainWindow.getarticulacion10()), glm::vec3(1.0f, 0.0f, .0f));
693     glUniform3fv(uniformColor, 1, glm::value_ptr(color)); //para cambiar el color del objetos
694     glUniformMatrix4fv(uniformModel, 1, GL_FALSE, glm::value_ptr(model));
695     meshList[0]->RenderMesh();
696 
697     //Cara
698     model = externalModelAux;
699     model = glm::translate(model, glm::vec3(0.0f, 0.3f, 1.f));
700     modelaux = model;
701     model = glm::scale(model, glm::vec3(0.2f, 0.2f, 0.2f));
702     color = glm::vec3(1.0f, .0f, 0.f);
703     glUniform3fv(uniformColor, 1, glm::value_ptr(color)); //para cambiar el color del objetos
704     glUniformMatrix4fv(uniformModel, 1, GL_FALSE, glm::value_ptr(model));
705     /*meshList[0]->RenderMesh();*/
706     sp.render();
707 
708     model = externalModelAux;
709     model = glm::translate(model, glm::vec3(0.6f, 0.3f, 1.f));
710     modelaux = model;
711     model = glm::scale(model, glm::vec3(0.2f, 0.2f, 0.2f));
712     color = glm::vec3(1.0f, .0f, 0.f);
713     glUniform3fv(uniformColor, 1, glm::value_ptr(color)); //para cambiar el color del objetos
714     glUniformMatrix4fv(uniformModel, 1, GL_FALSE, glm::value_ptr(model));
```

```
715             /*meshList[0]->RenderMesh();*/
716             sp.render();
717 
718             model = externalModelAux;
719             model = glm::translate(model, glm::vec3(-.6f, 0.3f, 1.f));
720             modelaux = model;
721             model = glm::scale(model, glm::vec3(0.2f, 0.2f, 0.2f));
722             color = glm::vec3(1.0f, .0f, 0.f);
723             glUniform3fv(uniformColor, 1, glm::value_ptr(color)); //para cambiar el color del objetos
724             glUniformMatrix4fv(uniformModel, 1, GL_FALSE, glm::value_ptr(model));
725             /*meshList[0]->RenderMesh();*/
726             sp.render();
727 
728             model = externalModelAux;
729             model = glm::translate(model, glm::vec3(0.3f, -0.1f, 1.f));
730             modelaux = model;
731             model = glm::scale(model, glm::vec3(0.2f, 0.2f, 0.2f));
732             color = glm::vec3(1.0f, .0f, 0.f);
733             glUniform3fv(uniformColor, 1, glm::value_ptr(color)); //para cambiar el color del objetos
734             glUniformMatrix4fv(uniformModel, 1, GL_FALSE, glm::value_ptr(model));
735             /*meshList[0]->RenderMesh();*/
736             sp.render();
737 
738             model = externalModelAux;
739             model = glm::translate(model, glm::vec3(-0.3f, -0.1f, 1.0f));
740             modelaux = model;
741             model = glm::scale(model, glm::vec3(0.2f, 0.2f, 0.2f));
742             color = glm::vec3(1.0f, 0.0f, 0.0f);
743             glUniform3fv(uniformColor, 1, glm::value_ptr(color)); //para cambiar el color del objetos
744             glUniformMatrix4fv(uniformModel, 1, GL_FALSE, glm::value_ptr(model));
745             /*meshList[0]->RenderMesh();*/
746             sp.render();
747 
748             glUseProgram(0);
749             mainWindow.swapBuffers();
750         }
751         return 0;
752     }
```



Problemas presentados

No se presentaron problemas para este ejercicio.

Conclusión

Esta experiencia me ha permitido comprender mejor cómo se puede ampliar la funcionalidad de un programa existente mediante la modificación de sus archivos fuente. Además, he aprendido sobre el uso de bibliotecas gráficas y cómo interactuar con objetos 3D en un entorno de programación.

Este práctica requirió adquirir conocimiento en la manipulación de matrices para lograr la jerarquía y articulaciones del modelo 3D del animal robot. Además, la interacción con el teclado para controlar las articulaciones añade un nivel de interactividad interesante para la visualización y manipulación del modelo.

Referencias:

- **OpenGL Programming Guide: The Official Guide to Learning OpenGL, Version 4.3: URL OpenGL Programming Guide** (9th ed.). (2014). Addison-Wesley Professional.
- **3D Game Engine Design: URL 3D Game Engine Design** (2nd ed.). (2018). CRC Press.
- **Cómo crear una grúa con OpenGL: URL Cómo crear una grúa con OpenGL**. (2023, February 14). Grafiati.
- **Animación de rotación de objetos con OpenGL: URL Animación de rotación de objetos con OpenGL**. (2022, December 1). Khronos Group.
- **OpenGL SuperBible: Comprehensive Tutorial and Reference: URL OpenGL SuperBible** (8th ed.). (2020). Sams Publishing.
- **Real-Time Rendering: URL Real-Time Rendering** (4th ed.). (2018). A K Peters/CRC Press.