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 03

NOMBRE COMPLETO: Aguilar Pérez José Ramón

Nº de Cuenta: 317515048

GRUPO DE LABORATORIO: 02

GRUPO DE TEORÍA: 04

SEMESTRE 2025-2

FECHA DE ENTREGA LÍMITE: 01/marzo/2025

CALIFICACIÓN: _____

REPORTE DE PRÁCTICA:

1.- Ejercicios realizados.

- **Generar una pirámide rubik (pyraminx) de 9 pirámides por cara.**

Para la realización de este ejercicio, primero se creó una pirámide de color negro que servirá como base para posicionar las demás pirámides. Primero, en la función de *CrearPiramideTriangular()* se modificaron los vértices para que la pirámide sea equilátera; después, se agregaron unas operaciones que permiten posicionar el centro de la pirámide en el origen (se saca el promedio de los vértices iniciales y después se ajustan para que coincidan con el origen), esto con el fin de facilitar la localización de las distintas pirámides a utilizar.

```
GLfloat vertices_piramide_triangular[] = {
    // Base del triángulo equilátero
    -0.5f, -0.5f, 0.0f,  // Vértice 0
    0.5f, -0.5f, 0.0f,  // Vértice 1
    0.0f,  0.3f, -0.25f,  // Vértice 2
    0.0f,  -0.5f, -0.75f    // Vértice 3
};

// Cálculo el centro geométrico (promedio de las coordenadas)
float centro_x = 0.0f, centro_y = 0.0f, centro_z = 0.0f;
for (int i = 0; i < 4; i++) {
    centro_x += vertices_piramide_triangular[3 * i];
    centro_y += vertices_piramide_triangular[3 * i + 1];
    centro_z += vertices_piramide_triangular[3 * i + 2];
}
centro_x /= 4.0f;
centro_y /= 4.0f;
centro_z /= 4.0f;

// Ajuste de las coordenadas para que el centro esté en el origen
for (int i = 0; i < 4; i++) {
    vertices_piramide_triangular[3 * i] -= centro_x;
    vertices_piramide_triangular[3 * i + 1] -= centro_y;
    vertices_piramide_triangular[3 * i + 2] -= centro_z;
}
```

A partir de esto, se genera la pirámide de base con ayuda de *meshList* y se le aplica el color negro con ayuda de *UniformColor*. Además, se le aplica una escala para que su tamaño el suficiente para que las otras pirámides quepan.

```
shaderList[0].useShader();
uniformModel = shaderList[0].getModelLocation();
uniformProjection = shaderList[0].getProjectLocation();
uniformView = shaderList[0].getViewLocation();
uniformColor = shaderList[0].getColorLocation();

//Generando la piramide negra de base
model = glm::mat4(1.0);
//Traslación inicial para posicionar en -Z a los objetos
model = glm::translate(model, glm::vec3(0.0f, 0.0f, -2.0f));
color = glm::vec3(0.0f, 0.0f, 0.0f);
//otras transformaciones para el objeto
model = glm::scale(model, glm::vec3(2.1f, 2.1f, 2.1f));
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()));
glUniform3fv(uniformColor, 1, glm::value_ptr(color)); //para cambiar el color del objetos
meshList[1]->RenderMesh();
```

Para las pirámides con caras de distinto color, primero fue necesario agregar la lista *meshColorList* y modificar el shader *shaderColor.vert* para poder utilizar el segundo shader que permite que las caras puedan ser de distinto color.

```
vector<Mesh*> meshList;
vector<MeshColor*> meshColorList; //Vector MeshColor
vector<Shader>shaderList;
```

```
#version 330
layout (location =0) in vec3 pos;
layout (location =1) in vec3 color;
out vec4 vColor;
uniform mat4 model;
uniform mat4 projection;
uniform mat4 view;

void main()
{
    gl_Position=projection*view*model*vec4(pos,1.0f);
    vColor=vec4(color,1.0f);
}
```

Se creó una nueva función, donde a cada una de las caras de la pirámide se le asigna un color diferente (rojo, verde, azul y amarillo) además de que los vértices se les añadió una ligera separación para simular las líneas del Pyraminx. A esta función también se le agregaron las operaciones para que el centro de las pirámides sea en el origen, con el fin de que al momento de rotar/trasladar sea más práctico. Como esta función ahora utiliza *MeshColorList,* será el primer elemento dentro de dicha lista.

```
void CrearPiramideColor()
{
    GLfloat vertices_piramide_color[] = {
        // Rojo                 R   G   B
        -0.5f, -0.5f, 0.08f,    1.0f, 0.0f, 0.0f, // 0
        0.5f, -0.5f, 0.08f,     1.0f, 0.0f, 0.0f, // 1
        0.0f, 0.3f, -0.18f,     1.0f, 0.0f, 0.0f, // 2

        // Verde
        0.58f, -0.5f, -0.05f,   0.0f, 1.0f, 0.0f, // 1
        0.08f, -0.5f, -0.80f,   0.0f, 1.0f, 0.0f, // 3
        0.08f, 0.3f, -0.28f,    0.0f, 1.0f, 0.0f, // 2

        // Azul
        -0.08f, -0.5f, -0.80f,  0.0f, 0.0f, 1.0f, // 3
        -0.58f, -0.5f, -0.05f,  0.0f, 0.0f, 1.0f, // 0
        -0.08f, 0.3f, -0.28f,   0.0f, 0.0f, 1.0f, // 2

        // Amarillo
        0.5f, -0.65f, 0.0f,     1.0f, 1.0f, 0.0f, // 1
        -0.5f, -0.65f, 0.0f,    1.0f, 1.0f, 0.0f, // 0
        0.0f, -0.65f, -0.75f,   1.0f, 1.0f, 0.0f, // 3
    };
```

```
GLfloat vertices_unicos[4][3] = {
    {-0.5f, -0.5f, 0.0f}, // Vértice 0
    {0.5f, -0.5f, 0.0f},  // Vértice 1
    {0.0f, 0.3f, -0.25f}, // Vértice 2
    {0.0f, -0.5f, -0.75f}  // Vértice 3
};

//Calcular el centro geométrico
float centro_x = 0.0f, centro_y = 0.0f, centro_z = 0.0f;
for (int i = 0; i < 4; i++) {
    centro_x += vertices_unicos[i][0];
    centro_y += vertices_unicos[i][1];
    centro_z += vertices_unicos[i][2];
}
centro_x /= 4.0f;
centro_y /= 4.0f;
centro_z /= 4.0f;

// Ajustar las coordenadas de los vértices para centrar la pirámide en el origen
for (int i = 0; i < 12; i++) {
    vertices_piramide_color[6 * i] -= centro_x;     // Ajustar x
    vertices_piramide_color[6 * i + 1] -= centro_y; // Ajustar y
    vertices_piramide_color[6 * i + 2] -= centro_z; // Ajustar z
}

MeshColor* Pira = new MeshColor();
Pira->CreateMeshColor(vertices_piramide_color, 72);
meshColorList.push_back(Pira);
```

Ahora, para las pirámides de colores se utiliza el segundo shader ya que las figuras se renderizarán con ayuda de *RenderMeshColor*. A las pirámides se les aplica una escala para que sean de un tamaño pequeño para que las 9 pirámides por cara quepan adecuadamente. Si se trata de una arista o una esquina solo se le aplicará una traslación a la figura, en cambio, para los centros se le aplicará una o varias rotaciones para que encajen con su respectiva cara.

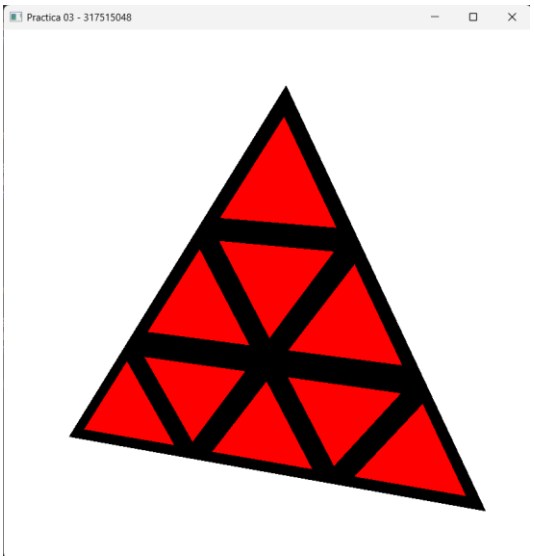
```
//Creando las esquinas y demás pirámides del Piraminx
shaderList[1].useShader();
uniformModel = shaderList[1].getModelLocation();
uniformProjection = shaderList[1].getProjectLocation();
uniformView = shaderList[1].getViewLocation();

//Esquina superior
model = glm::mat4(1.0f);
model = glm::translate(model, glm::vec3(0.0f, 0.85f, -2.0f));
model = glm::scale(model, glm::vec3(0.5f, 0.5f, 0.5f));
glUniformMatrix4fv(uniformModel, 1, GL_FALSE, glm::value_ptr(model)); //FALSE ES PARA QUE NO
glUniformMatrix4fv(uniformProjection, 1, GL_FALSE, glm::value_ptr(projection));
glUniformMatrix4fv(uniformView, 1, GL_FALSE, glm::value_ptr(camera.calculateViewMatrix()));
meshColorList[0]->RenderMeshColor();
```

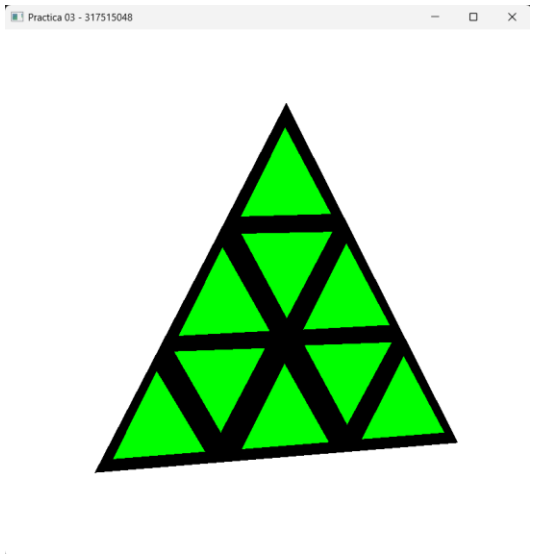
```
//Aristas
//Rojo-Azul
model = glm::mat4(1.0f);
model = glm::translate(model, glm::vec3(-0.34f, 0.31f, -1.83f));
model = glm::scale(model, glm::vec3(0.5f, 0.5f, 0.5f));
glUniformMatrix4fv(uniformModel, 1, GL_FALSE, glm::value_ptr(model));  //FALSE ES PARA QUE N
glUniformMatrix4fv(uniformProjection, 1, GL_FALSE, glm::value_ptr(projection));
glUniformMatrix4fv(uniformView, 1, GL_FALSE, glm::value_ptr(camera.calculateViewMatrix()));
meshColorList[0]->RenderMeshColor();

//Rojo-Verde
model = glm::mat4(1.0f);
model = glm::translate(model, glm::vec3(0.34f, 0.31f, -1.83f));
model = glm::scale(model, glm::vec3(0.5f, 0.5f, 0.5f));
glUniformMatrix4fv(uniformModel, 1, GL_FALSE, glm::value_ptr(model));   //FALSE ES PARA QUE
glUniformMatrix4fv(uniformProjection, 1, GL_FALSE, glm::value_ptr(projection));
glUniformMatrix4fv(uniformView, 1, GL_FALSE, glm::value_ptr(camera.calculateViewMatrix()));
meshColorList[0]->RenderMeshColor();
```

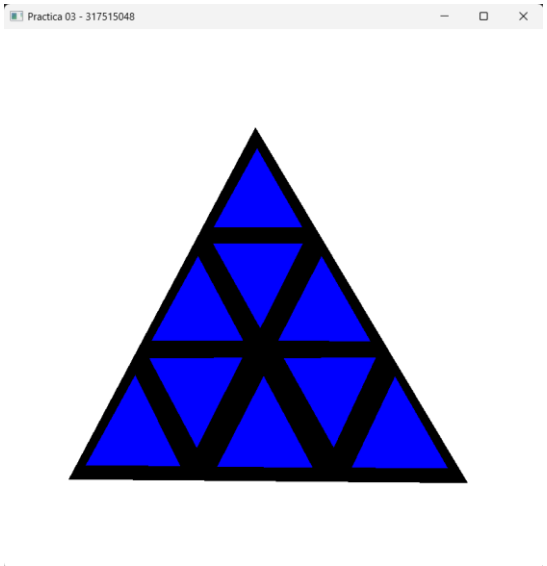
```
//Centros
//Rojo-Superior
model = glm::mat4(1.0f);
model = glm::translate(model, glm::vec3(0.0f, 0.48f, -1.88f));
model = glm::rotate(model, 180 * toRadians, glm::vec3(0.0f, 0.0f, 1.0f));
model = glm::rotate(model, 35 * toRadians, glm::vec3(1.0f, 0.0f, 0.0f));
model = glm::scale(model, glm::vec3(0.5f, 0.5f, 0.5f));
glUniformMatrix4fv(uniformModel, 1, GL_FALSE, glm::value_ptr(model));   //FALSE ES PARA QUE
glUniformMatrix4fv(uniformProjection, 1, GL_FALSE, glm::value_ptr(projection));
glUniformMatrix4fv(uniformView, 1, GL_FALSE, glm::value_ptr(camera.calculateViewMatrix()));
meshColorList[0]->RenderMeshColor();
```

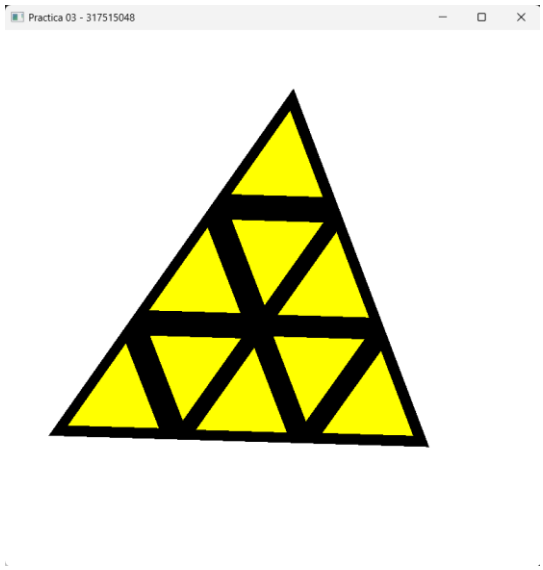
```
//Verde-Superior
model = glm::mat4(1.0f);
model = glm::translate(model, glm::vec3(0.14f, 0.48f, -2.05f));
model = glm::rotate(model, -180 * toRadians, glm::vec3(0.0f, 0.0f, 1.0f));
model = glm::rotate(model, 115 * toRadians, glm::vec3(0.0f, 1.0f, 0.0f));
model = glm::rotate(model, -38 * toRadians, glm::vec3(1.0f, 0.0f, 1.5f));
model = glm::scale(model, glm::vec3(0.5f, 0.5f, 0.5f));
glUniformMatrix4fv(uniformModel, 1, GL_FALSE, glm::value_ptr(model));   //FALSE ES PARA QUE NO
glUniformMatrix4fv(uniformProjection, 1, GL_FALSE, glm::value_ptr(projection));
glUniformMatrix4fv(uniformView, 1, GL_FALSE, glm::value_ptr(camera.calculateViewMatrix()));
meshColorList[0]->RenderMeshColor();
```

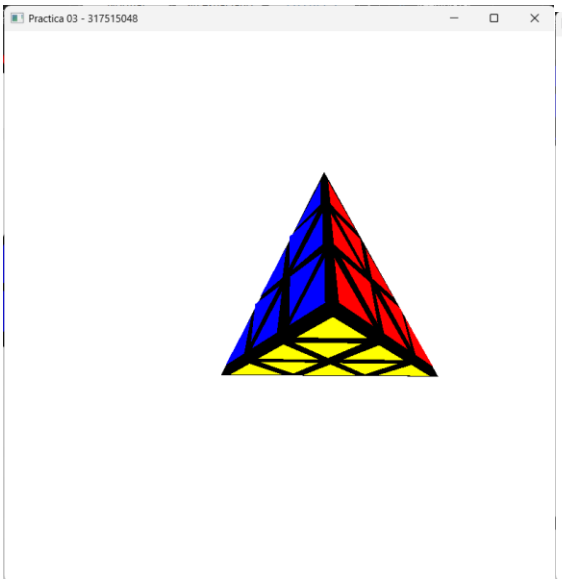
La mayoría de estos vértices (sobre todo los de los centros) se calcularon a base de prueba y error hasta encontrar la posición más adecuada.

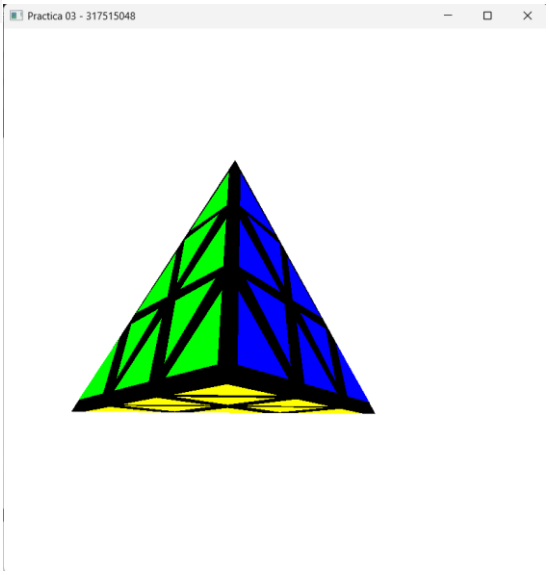

Practica 03 - 317515048


Practica 03 - 317515048


Practica 03 - 317515048


Practica 03 - 317515048


Practica 03 - 317515048


Practica 03 - 317515048

2.- Problemas presentados

Fue necesario saber la distribución de los ejes X, Y, Z en el plano, ya que a la hora de hacer las rotaciones, la figura no se comportaba de la manera que uno se esperaba, por lo que se dibujaron en el programa unas líneas de guía para comprender mejor el movimiento de las figuras.

3.- Conclusión:

a. Los ejercicios del reporte: Complejidad, Explicación.

El ejercicio de esta práctica fue muy interesante de realizar, ya que la complejidad de este era más elevada de lo que estaba acostumbrado. Gracias a la explicación del funcionamiento de los distintos shader y de la creación de distintas figuras, se pudo llevar a cabo esta práctica sin contratiempos.

b. Comentarios generales: Faltó explicar a detalle, ir más lento en alguna explicación, otros comentarios y sugerencias para mejorar desarrollo de la práctica

Podría ser de utilidad explicar más a fondo como lograr que la figuras giren como una sola y que no hagan traslaciones raras durante este movimiento.

c. Conclusión

Esta práctica me permitió comprender mejor cómo funcionan los shaders para colorear una misma figura con distintos colores, así como la utilización de los distintos Mesh para la creación de distintas figuras. Fue importante entender el movimiento de las figuras en el plano 3D, ya que con un pequeño cambio la rotación podría no ser la adecuada y esto arruinaría/retrasaría el trabajo hecho.

1. Bibliografía en formato APA

- OpenGL Programming/Basics/Color. (s/f). Wikibooks.org. Recuperado el 14 de febrero de 2025, de https://en.wikibooks.org/wiki/OpenGL_Programming/Basics/Color
- Tutorial 3: Matrices. (s/f). Opengl-tutorial.org. Recuperado el 1 de marzo de 2025, de http://www.opengl-tutorial.org/es/beginners-tutorials/tutorial-3-matrices/