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: Pablo Arturo Gonzalez Cuellar

Nº de Cuenta: 319241013

GRUPO DE LABORATORIO: 02

GRUPO DE TEORÍA: 04

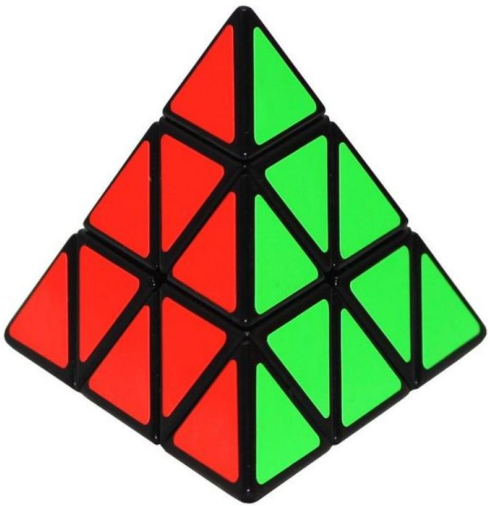
SEMESTRE 2026-1

FECHA DE ENTREGA LÍMITE: 08/09/2026

CALIFICACIÓN: _____

REPORTE DE PRÁCTICA:

1.- Generar una pirámide rubik (pyraminx) de 9 pirámides por cara. Cada cara de la pyraminx que se vea de un color diferente y que se vean las separaciones entre instancias (las líneas oscuras son las que permiten diferenciar cada pirámide pequeña)



Para esta práctica se solicitó construir un modelo 3D de un Pyraminx. El objetivo principal era componer una pirámide triangular grande a partir de instancias de una pirámide más pequeña. Específicamente, cada una de las cuatro caras del Pyraminx debía estar formada por 9 pirámides pequeñas y tener un color único (rojo, verde, azul y amarillo), además de mostrar unas líneas negras de separación entre cada pieza.

Como primer paso, aprovechamos el modelado de la pirámide que se nos proporcionó en el laboratorio para utilizarla como base fundamental para el desarrollo de la práctica.

```
// 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, 0.0f, -0.289f,   //0
        0.5f,0.0f,-0.289f,      //1
        0.0f,0.0f, 0.577f,      //2
        0.0f,0.816f,0.0f,       //3

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

Ahora bien, para crear el efecto de “separación”, se renderizó una única pirámide de color negro escalada a un 96% del tamaño total. Esto permitirá que lo bordes de la pirámide negra sobresalgan por detrás de las pirámides de colores, creando las líneas obscuras solicitadas.

```
glPolygonMode(GL_FRONT_AND_BACK, GL_FILL);  //relleno

model = glm::mat4(1.0f);
model = glm::scale(model, glm::vec3(0.96f));
glUniform3f(uniformColor, 0.0f, 0.0f, 0.0f);
glUniformMatrix4fv(uniformModel, 1, GL_FALSE, glm::value_ptr(model));
meshList[1]->RenderMesh();// Pirámide negra
```

Posterior a esto, se procedió a dibujar cada cara. Para la cara roja, se dibujaron 9 instancias de la pirámide base, y a cada instancia se le aplicó un escalamiento y traslación distinto para ajustarla dentro de la cara de la pirámide negra.

```
//Cara roja
glPolygonMode(GL_FRONT_AND_BACK, GL_FILL);
model = glm::mat4(1.0f);
color = glm::vec3(1.0f, 0.0f, 0.0f);
glUniform3fv(uniformColor, 1, glm::value_ptr(color));
model = glm::translate(model, glm::vec3(0.0f, 0.001f, -0.22f));
model = glm::scale(model, glm::vec3(0.3f, 0.3f, 0.3f));
glUniformMatrix4fv(uniformModel, 1, GL_FALSE, glm::value_ptr(model));
glUniformMatrix4fv(uniformProjection, 1, GL_FALSE, glm::value_ptr(projection));
meshList[1]->RenderMesh(); // Pirámide 1
```

```
model = glm::mat4(1.0f);
model = glm::translate(model, glm::vec3(0.333f, 0.001f, -0.22f));
model = glm::scale(model, glm::vec3(0.3f, 0.3f, 0.3f));
glUniformMatrix4fv(uniformModel, 1, GL_FALSE, glm::value_ptr(model));
glUniformMatrix4fv(uniformProjection, 1, GL_FALSE, glm::value_ptr(projection));
meshList[1]->RenderMesh(); //Pirámide 2
```

Ahora bien, para construir las caras verde y azul, en lugar de calcular nuevas coordenadas se reutilizó la misma lógica de la cara roja, pero aplicando una rotación inicial sobe el eje Y a cada una de las 9 transformaciones.
La cara verde se rotó -120° y la azul 120°. Esto posicionó las pirámides de forma correcta alrededor del eje central del Pyraminx. La cara amarilla se construyó bajo la misma lógica pero aplicando las translaciones sobre un plano diferente.

```
model = glm::mat4(1.0f);
model = glm::rotate(model, glm::radians(-120.0f), glm::vec3(0.0f, 1.0f, 0.0f));
model = glm::rotate(model, glm::radians(180.0f), glm::vec3(0.0f, 0.0f, 1.0f));
model = glm::rotate(model, glm::radians(-35.0f), glm::vec3(1.0f, 0.0f, 0.0f));
model = glm::translate(model, glm::vec3(-0.165f, -0.08f, -0.25f));
model = glm::scale(model, glm::vec3(0.3f, 0.3f, 0.3f));
glUniformMatrix4fv(uniformModel, 1, GL_FALSE, glm::value_ptr(model));
glUniformMatrix4fv(uniformProjection, 1, GL_FALSE, glm::value_ptr(projection));
meshList[1]->RenderMesh(); // 4
```

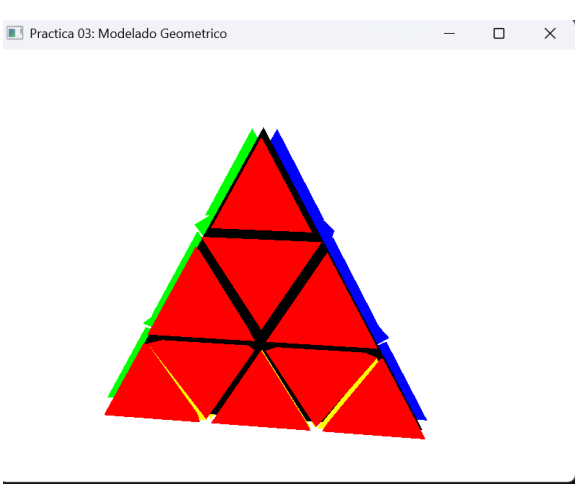
PROBLEMAS SURGIDOS:

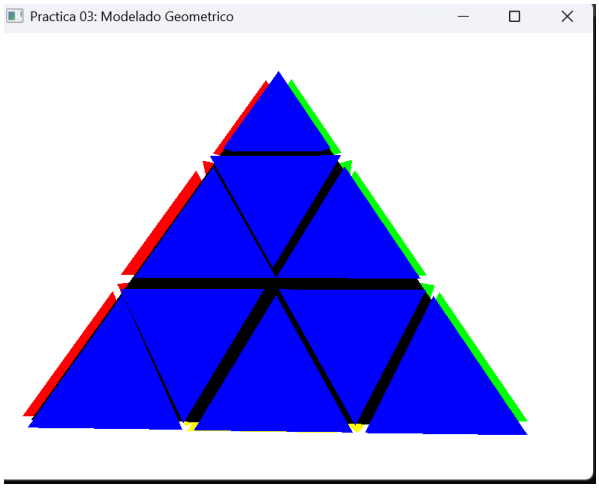
El principal desafío fuel el posicionamiento preciso de cada una de las 9 pirámides en cada cara. Las transformaciones de traslación y rotación requerían de mucha prueba y error para que las piezas encajaran visualmente sin superponerse ni dejar

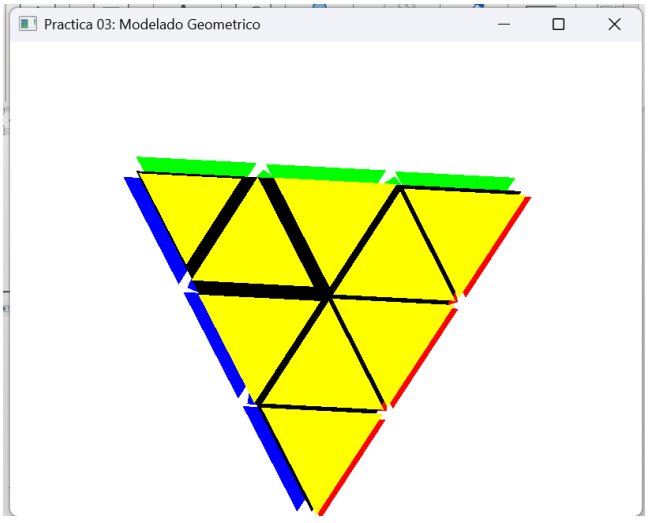
huecos. El encontrar los valores para las coordenadas de traslación y los ángulos de rotación fue un proceso laborioso.

Otro problema, aunque en menor medida, fue el parpadeo entre la pirámide negra y las mini pirámides que, aunque el problema se resolvió de manera más rápida debido a que ya es un problema recurrente en este tipo de prácticas y ejercicios, presentó repetidos problemas debido a la constante modificación en las posiciones de las pirámides, este problema se resolvió aplicando un desplazamiento mínimo a las pirámides para asegurar que siempre se dibujaran ligeramente por delante de la pirámide negra.

CAPTURAS DE EJECUCIÓN







CONCLUSIÓN

En esta práctica se siguió perfeccionando el poder de la graficación por instancias y las transformaciones para posicionar figuras. En lugar de definir la geometría de cada una de las pirámides por separado, se partió de la geometría de una y se reutilizó aplicando transformaciones de `scale`, `rotate` y `translate` para posicionar cada una en el lugar correcto.

Además, la implementación de la pirámide negra para lograr las separaciones del “borde negro” me enseño a resolver problemas visuales de forma “lógica” sin recurrir a funciones o técnicas complejas.