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: Sanchez Calvillo Saida Mayela

Nº de Cuenta: 318164481

GRUPO DE LABORATORIO: 02

GRUPO DE TEORÍA: 04

SEMESTRE 2025-2

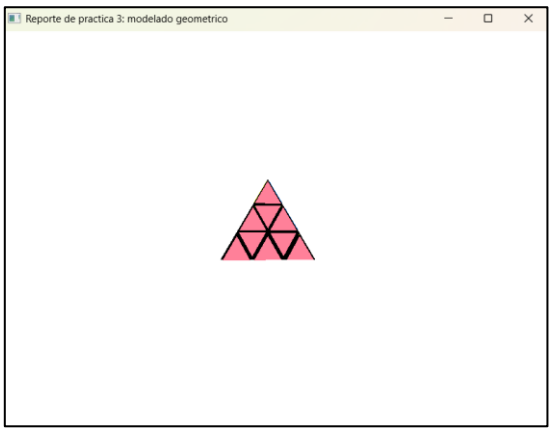
FECHA DE ENTREGA LÍMITE: 01 de marzo 2025

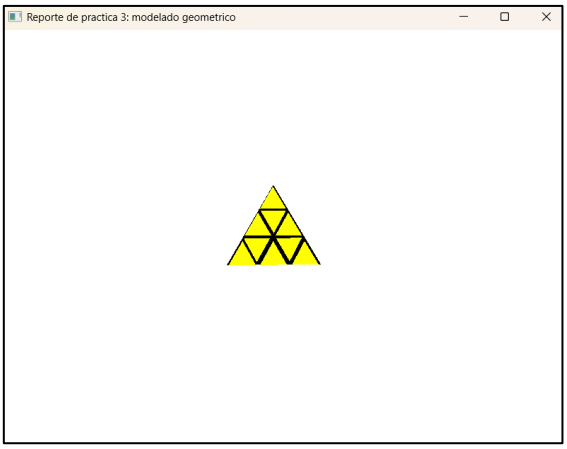
CALIFICACIÓN: _____

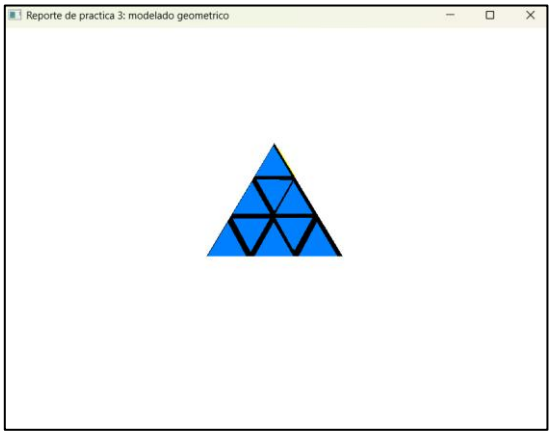
REPORTE DE PRÁCTICA:

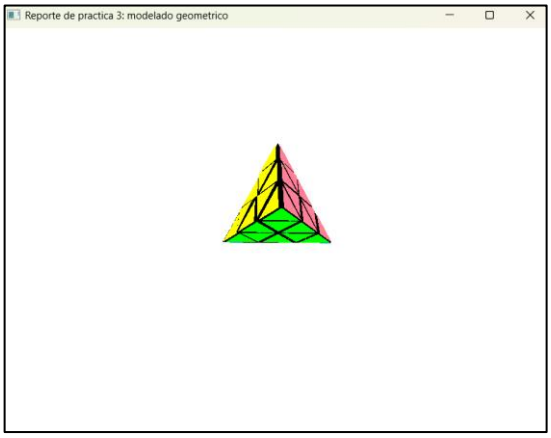
1.- Ejecución de los ejercicios que se dejaron, comentar cada uno y capturas de pantalla de bloques de código generados y de ejecución del programa.

- 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) Agregar en su documento escrito las capturas de pantalla necesarias para que se vean las 4 caras de toda la pyraminx o un video en el cual muestra las 4 caras









Primero modifique la función de piramide triangular para poder hacer una pirámide equilátera por todos sus lados, para ello se calculo la altura de la pirámide y así poder hacer que todos sus lados fueran iguales.

```cpp
//Piramide triangular equilatera
void CrearPiramideTriangularNegra()
{
    unsigned int indices_piramide_triangular_negra[] = {
        0, 1, 2,
        1, 3, 2,
        3, 0, 2,
        1, 0, 3
    };

    GLfloat vertices_piramide_triangular_negra[] = {
        // Base (Triángulo equilátero en el plano XZ)
       -0.5f,  -0.5f,   0.288f,  // Vértice 0
        0.5f,  -0.5f,   0.288f,  // Vértice 1
        0.0f,   0.366f,   0.0f,   // Vértice 2
        0.0f,  -0.5f,  -0.577f,  // Vértice 3
    };

    Mesh* e_negra = new Mesh();
    e_negra->CreateMesh(vertices_piramide_triangular_negra, indices_piramide_triangular_negra, 12, 12);
    meshList.push_back(e_negra);

}
```

Luego se creó otra función para poder crear una pirámide con las caras separadas y así poder dar el efecto de que son triángulos con separaciones (líneas) negras entre si a la hora de colocarlos en a pirámide.

Esta pirámide tiene como característica que cada lado tiene un color distinto, así que vamos a usar el shader donde se puede decidir que color tiene cada esquina y así poder darle un color dintinto a todas las caras incluyendo la base. Después se va a agregar a la lista de MeshColorList, no sin antes definir el vector en un principio

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

```cpp
void CrearPiramideTriangularEquilatera()
{
    float separacion = 0.06f; // Ajusta este valor para cambiar el espacio entre caras

    GLfloat vertices_piramide_triangular_equilatera[] = {
        //Cara 1 - Rosa (Separada en Z+)
       -0.5f,   -0.5f,     0.288f + separacion,   1.0f,  0.5f,  0.6f,
        0.5f,   -0.5f,     0.288f + separacion,   1.0f,  0.5f,  0.6f,
        0.0f,    0.366f,   0.0f + separacion,     1.0f,  0.5f,  0.6f,

        //Cara 2 - Azul (Separada en X+)
        0.5f + separacion,  -0.5f,     0.288,    0.0f,  0.5f,  1.0f,
        0.0f + separacion,  -0.5f,    -0.577f,   0.0f,  0.5f,  1.0f,
        0.0f + separacion,   0.366f,   0.0f,     0.0f,  0.5f,  1.0f,

        //Cara 3 - Amarillo (Separada en X- y Z-)
         0.0f - separacion,  -0.5f,   -0.577f - separacion,   1.0f,  1.0f,  0.0f,
        -0.5f - separacion,  -0.5f,    0.288f - separacion,   1.0f,  1.0f,  0.0f,
         0.0f - separacion,   0.366f,  0.0f - separacion,     1.0f,  1.0f,  0.0f,

        //Base - Verde
         0.5f,   -0.5f,   0.288f,   0.0f,  1.0f,  0.0f,
        -0.5f,   -0.5f,   0.288f,   0.0f,  1.0f,  0.0f,
         0.0f,   -0.5f,  -0.577f,   0.0f,  1.0f,  0.0f,

    };
    MeshColor* equilatera = new MeshColor();
    equilatera->CreateMeshColor(vertices_piramide_triangular_equilatera, 72);
    meshColorList.push_back(equilatera);
}
```

Mande a llamar las funciones que recién creamos y recordamos en que posición se encuentran ahora

```
CrearPiramideCuadrangular();//índice 4 en MeshList
CrearPiramideTriangularNegra(); //indice 5 en meshlist
CrearPiramideTriangularEquilatera(); //Indice 0 en meshColorList
CreateShaders();
```

Para la creación de la pirámide primero haremos la pirámide negra donde se incrustarán el resto de las piamides de colores

Aquí le damos color, traslación y todo lo que se necesita para poder hacer uso de todos los recursos vistos hasta ahora

```
//Piramide negra
shaderList[0].useShader();
uniformModel = shaderList[0].getModelLocation();
uniformProjection = shaderList[0].getProjectLocation();
uniformView = shaderList[0].getViewLocation();
uniformColor = shaderList[0].getColorLocation();
model = glm::mat4(1.0);
//Traslación inicial para posicionar en -Z a los objetos
model = glm::translate(model, glm::vec3(0.0f, 0.0f, -4.0f));
//otras transformaciones para el objeto
//model = glm::scale(model, glm::vec3(0.5f,0.5f,0.5f));
model = glm::rotate(model, glm::radians(mainWindow.getrotax()), glm::vec3(1.0f, 0.0f, 0.0f));
model = glm::rotate(model, glm::radians(mainWindow.getrotay()), glm::vec3(0.0f, 1.0f, 0.0f));
model = glm::rotate(model, glm::radians(mainWindow.getrotaz()), glm::vec3(0.0f, 0.0f, 1.0f));
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(0.0f, 0.0f, 0.0f);
glUniform3fv(uniformColor, 1, glm::value_ptr(color)); //para cambiar el color del objetos
meshList[5]->RenderMesh(); //dibuja cubo y pirámide triangular
```

Después se ha creado una pirámide a escala de la negra, pero con la pirámide de colores con caras separadas que creamos después y así se vean parejas las incrustaciones. Primero se pusieron las pirámides de las esquinas usando transfomaciones geométricas y así ver como se irían acomodando el resto de las pirámides. No olvidar en que lista de mesh estaba guardada.

```
//******Piramides de colores*******
//Cara rosa
    //1
shaderList[1].useShader();
uniformModel = shaderList[1].getModelLocation();
uniformProjection = shaderList[1].getProjectLocation();
uniformView = shaderList[1].getViewLocation();
uniformColor = shaderList[1].getColorLocation();
model = glm::mat4(1.0);
//Traslación inicial para posicionar en -Z a los objetos
model = glm::translate(model, glm::vec3(0.0f, 0.24f, -4.0f));
model = glm::scale(model, glm::vec3(0.285f, 0.285f, 0.285f));
//Rotar la figura
//model = glm::rotate(model, glm::radians(180.0f), glm::vec3(0.0f, 0.0f, 1.0f));     //invierte la piramide
//model = glm::rotate(model, glm::radians(34.0f), glm::vec3(1.0f, 0.0f, 0.0f));     //inclina la piramide hacia atras
model = glm::rotate(model, glm::radians(mainWindow.getrotax()), glm::vec3(1.0f, 0.0f, 0.0f));
model = glm::rotate(model, glm::radians(mainWindow.getrotay()), glm::vec3(0.0f, 1.0f, 0.0f));
model = glm::rotate(model, glm::radians(mainWindow.getrotaz()), glm::vec3(0.0f, 0.0f, 1.0f));
glUniformMatrix4fv(uniformProjection, 1, GL_FALSE, glm::value_ptr(projection));
glUniformMatrix4fv(uniformModel, 1, GL_FALSE, glm::value_ptr(model));
glUniformMatrix4fv(uniformView, 1, GL_FALSE, glm::value_ptr(camera.calculateViewMatrix()));
//glUniform3fv(uniformColor, 1, glm::value_ptr(color)); //para cambiar el color del objetos
meshColorList[0]->RenderMeshColor();
```

Las líneas comentadas nos ayudaron con el resto de las pirámides, sobre todo las que estaban invertidas las que estaban en las otras caras.

Para esas inclinaciones ocupamos rotaciones por X, Y y Z para poder colocarlas en las posiciones que necesitábamos y así poder hacer efecto de que eran triángulos de diferentes colores y no pirámides.

```
    //3
model = glm::mat4(1.0);
//Traslación inicial para posicionar en -Z a los objetos
model = glm::translate(model, glm::vec3(-0.095f, -0.099f, -4.065f));
model = glm::scale(model, glm::vec3(0.28f, 0.28f, 0.28f));
//Rotar la figura
model = glm::rotate(model, glm::radians(115.0f), glm::vec3(0.0f, 1.0f, 0.0f));      //inclina la piramide hacia Y
model = glm::rotate(model, glm::radians(165.0f), glm::vec3(0.0f, 0.0f, 1.0f));     //invierte la piramide
model = glm::rotate(model, glm::radians(45.0f), glm::vec3(0.0f, 0.0f, 1.0f));     //invierte la piramide
model = glm::rotate(model, glm::radians(-20.0f), glm::vec3(1.0f, 0.0f, 0.0f));      //inclina la piramide hacia X
model = glm::rotate(model, glm::radians(mainWindow.getrotax()), glm::vec3(1.0f, 0.0f, 0.0f));
model = glm::rotate(model, glm::radians(mainWindow.getrotay()), glm::vec3(0.0f, 1.0f, 0.0f));  //al presionar la te
model = glm::rotate(model, glm::radians(mainWindow.getrotaz()), glm::vec3(0.0f, 0.0f, 1.0f));
glUniformMatrix4fv(uniformProjection, 1, GL_FALSE, glm::value_ptr(projection));
glUniformMatrix4fv(uniformModel, 1, GL_FALSE, glm::value_ptr(model));
glUniformMatrix4fv(uniformView, 1, GL_FALSE, glm::value_ptr(camera.calculateViewMatrix()));
//glUniform3fv(uniformColor, 1, glm::value_ptr(color)); //para cambiar el color del objetos
meshColorList[0]->RenderMeshColor();
```

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

Uno de los problemas mas grandes fue decidir como le daríamos el efecto visual de que nuestra pirámide tenía divisiones negras entre cada triangulo. Eso se resolvió separando las caras de las pirámides, como si desarmáramos la pirámide por partes y solo dejáramos los triángulos flotando listo para ser colocados y formar la pirámide.

Otro problema fue encontrar la inclinación correcta para las pirámides invertidas ya que si movías en cualquier eje, el resultado no era el esperado. Esto se resolvió investigando como eran los ejes correctamente y al final era más atinarle al ángulo exacto que hasta tuve que usar 3 o 4 decimales.

3.- Conclusión:

a. Los ejercicios del reporte: Complejidad, Explicación.

La creación de esta pirámide implica varios retos técnicos: se requiere calcular con exactitud los vértices de cada subpirámide para garantizar que todas sean equiláteras y correctamente espaciadas, la necesidad de aplicar translaciones y escalados adecuados para colocar cada pirámide en la cuadrícula correcta dentro de la cara mayor, agregar espacios entre las subpirámides mediante una variable de separación para mejorar la visualización de las divisiones, etc.

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

Faltó explicar a detalle la generación de subpirámides, Aun no entiendo del todo como se usa la cámara y como poder mover toda la pirámide en conjunto cuando se usan los movimientos de los ejes y no cada objeto se mueva en su propio centro

c. Conclusión

La creación de la pirámide Pyraminx con subdivisiones implicó un desafío en la correcta definición de sus vértices, su distribución espacial y su renderizado con separaciones visibles. A pesar de los retos, se logró una representación visual clara con divisiones adecuadas y colores diferenciados en cada cara. La práctica permitió afianzar conocimientos sobre modelado geométrico, transformaciones espaciales y optimización de instancias en OpenGL.