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 Villalpando Johan

Nº de Cuenta: 422028657

GRUPO DE LABORATORIO: 04

GRUPO DE TEORÍA: 02

SEMESTRE 2025-2

FECHA DE ENTREGA LÍMITE: 01/03/25

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.

En esta práctica solamente se solicita un ejercicio, el cual es generar una pyraminx en 3D, utilizando piramides de 3 colores.

Para empezar, se debe rediseñar la pirámide triangular dada en los archivos de la práctica, debido a que es muy irregular en sus caras, por lo que se utilizarán nuevos vértices haciendo una figura más uniforme, es decir, un triángulo que mida lo mismo en sus 3 caras, o lo más aproximado.

```
// Pirámide triangular regular
void CrearPiramideTriangular()
{
    unsigned int indices_piramide_triangular[] = {
        0, 1, 2,  // Cara 1
        0, 2, 3,  // Cara 2
        0, 3, 1,  // Cara 3
        1, 2, 3   // Cara 4 (base)
    };
    GLfloat vertices_piramide_triangular[] = {
        0.0f,  0.4f,  0.0f,    // 0 Punta
        -0.5f, -0.2f,  0.3f,  // 1
        0.5f, -0.2f,  0.3f,   // 2
        0.0f, -0.2f, -0.6f    // 3
    };
    Mesh* obj1 = new Mesh();
    obj1->CreateMesh(vertices_piramide_triangular, indices_piramide_triangular, 12, 12);
    meshList.push_back(obj1);

}
```

Una vez teniendo una pirámide triangular más uniforme, esta se genera.

```
//INICIA ejercicio de práctica.

model = glm::mat4(1.0);
model = glm::translate(model, glm::vec3(0.0f, 0.0f, -3.0f));
model = glm::scale(model, glm::vec3(3.0f, 3.0f, 3.0f));
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, 0.0f, 0.0f);// color rojo
glUniform3fv(uniformColor, 1, glm::value_ptr(color));
meshList[1]->RenderMesh();
```

Ahora comienza el rellenado del pyraminx, con sus 4 colores, para esto se requiere una pirámide que contenga los 4 colores, uno en cada cara (rojo, verde, amarillo, azul). Esto no es posible con la figura de pirámide triangular contenida en la meshList, por ello se utilizará una pirámide realizada con meshColor, primero generando la lista meshColor.

```
vector<MeshColor*> meshColorList; //Se agrega vector mesh color
```

Para luego comenzar a crear la pirámide de colores, utilizando una nueva función CrearPiraminx()

```
void CrearPiraminx() {

    GLfloat vertices_piraminx[] = {

        /* VERTICES PIRAMIDE TRIANGULAR 3D
        0.0f,  0.4f,  0.0f,    // 0 Punta
        -0.5f, -0.2f,  0.3f,  // 1
        0.5f, -0.2f,  0.3f,    // 2
        0.0f, -0.2f, -0.6f     // 3 Base
        */
        // ROJO
         0.0f,  0.44f,  0.1f,    1.0f, 0.0f, 0.0f,
        -0.5f, -0.15f,  0.4f,    1.0f, 0.0f, 0.0f,
         0.5f, -0.15f,  0.4f,    1.0f, 0.0f, 0.0f,

         // AMARILLO
        -0.5f, -0.3f,  0.3f,    1.0f, 1.0f, 0.0f,
         0.0f, -0.3f, -0.6f,    1.0f, 1.0f, 0.0f,
         0.5f, -0.3f,  0.3f,    1.0f, 1.0f, 0.0f,

         // AZUL
         0.08f, -0.16f, -0.65f,    0.0f, 0.0f, 1.0f,
         0.08f,  0.45f, -0.05f,    0.0f, 0.0f, 1.0f,
         0.6f, -0.15f,  0.25f,    0.0f, 0.0f, 1.0f,

         // VERDE
         -0.6f, -0.15f,  0.26f,    0.0f, 1.0f, 0.0f,
         -0.08f,  0.45f, -0.04f,    0.0f, 1.0f, 0.0f,
         -0.08f, -0.16f, -0.64f,    0.0f, 1.0f, 0.0f,
    };

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

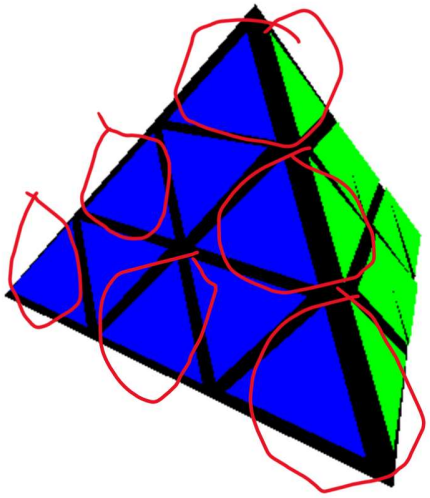
Para esta pirámide se tomaron como base los vértices de la pirámide uniforme generada anteriormente, sin embargo, para dar el efecto de que la pirámide tiene espacios entre cada “estampa” de color, la manera más efectiva que encontré fue comenzar a recortar las medidas de cada triangulo individual, de modo que cuando estén juntos den la apariencia de un espacio vacío entre ellos, cuidando siempre que se mantenga el ángulo, ya que un recorte mal hecho de alguna cara, da como resultado un triángulo cuyas caras no están en el mismo ángulo que el triangulo

base, lo que hace que al juntarlos, las caras del triángulo de colores se inserten dentro del triángulo base.

Estas utilizarán el shader color, el cual aún no está adaptado a la cámara móvil, por lo que se hace el pequeño cambio en el vert.

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

Comienzan las declaraciones de cada mini pirámide de colores, para las puntas fue sencillo, ya que únicamente se debe mover la posición de la punta de arriba, manteniendo una diagonal, esto mismo se repite para los triángulos de en medio y los triángulos de en medio-abajo.



La ventaja de realizar esto con pirámides de 3 colores, es que al generar la diagonal de cada cara, las demás caras se van armando también.

```
//piramides de colores
//SHADERS PARA LOS TRIANGULOS.
shaderList[1].useShader();
uniformModel = shaderList[1].getModelLocation();
uniformProjection = shaderList[1].getProjectLocation();
uniformView = shaderList[1].getViewLocation();
model = glm::mat4(1.0f);
model = glm::translate(model, glm::vec3(0.0f, 0.7f, -3.0f));
model = glm::scale(model, glm::vec3(0.8f, 0.8f, 0.8f));
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()));
meshColorList[0]->RenderMeshColor(); //piramide de colores arriba
```

```
//punta verde azul arriba
model = glm::mat4(1.0f);
model = glm::translate(model, glm::vec3(0.0f, -0.4f, -4.15f));
model = glm::scale(model, glm::vec3(0.8f, 0.8f,0.8f));
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()));
meshColorList[0]->RenderMeshColor(); //piramide de colores arriba

//punta verde roja abajo
model = glm::mat4(1.0f);
model = glm::translate(model, glm::vec3(-0.975f, -0.4f, -2.425f));
model = glm::scale(model, glm::vec3(0.8f, 0.8f, 0.8f));
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()));
meshColorList[0]->RenderMeshColor(); //piramide de colores arriba

//punta ultima
model = glm::mat4(1.0f);
model = glm::translate(model, glm::vec3(0.975f, -0.4f, -2.425f));
model = glm::scale(model, glm::vec3(0.8f, 0.8f, 0.8f));
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()));
meshColorList[0]->RenderMeshColor(); //piramide de colores arriba
```

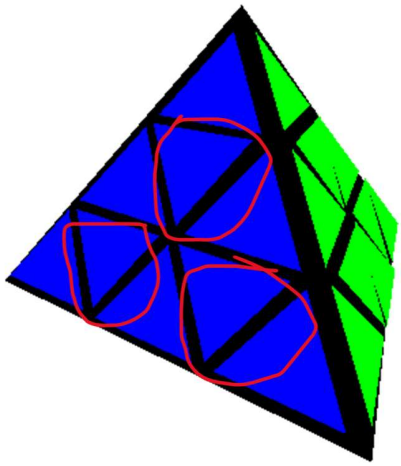
```
//medio azul verde
model = glm::mat4(1.0f);
model = glm::translate(model, glm::vec3(0.0f, 0.15f, -3.6f));
model = glm::scale(model, glm::vec3(0.8f, 0.8f, 0.8f));
glUniformMatrix4fv(uniformModel, 1, GL_FALSE, glm::value_ptr(model));
glUniformMatrix4fv(uniformProjection, 1, GL_FALSE, glm::value_ptr(projection));
glUniformMatrix4fv(uniformView, 1, GL_FALSE, glm::value_ptr(camera.calculateViewMatrix()));
meshColorList[0]->RenderMeshColor(); //piramide de colores arriba

//medio rojo verde
model = glm::mat4(1.0f);
model = glm::translate(model, glm::vec3(-0.487f, 0.15f, -2.715f));
model = glm::scale(model, glm::vec3(0.8f, 0.8f, 0.8f));
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()));
meshColorList[0]->RenderMeshColor(); //piramide de colores arriba

//medio rojo azul
model = glm::mat4(1.0f);
model = glm::translate(model, glm::vec3(0.4875f, 0.15f, -2.715f));
model = glm::scale(model, glm::vec3(0.8f, 0.8f, 0.8f));
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()));
meshColorList[0]->RenderMeshColor(); //piramide de colores arriba
```

```
//bajo verde
model = glm::mat4(1.0f);
model = glm::translate(model, glm::vec3(-0.5f, -0.4f, -3.3f));
model = glm::scale(model, glm::vec3(0.8f, 0.8f, 0.8f));
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()));
meshColorList[0]->RenderMeshColor(); //piramide de colores arriba

//bajo azul
model = glm::mat4(1.0f);
model = glm::translate(model, glm::vec3(0.5f, -0.4f, -3.3f));
model = glm::scale(model, glm::vec3(0.8f, 0.8f, 0.8f));
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()));
meshColorList[0]->RenderMeshColor(); //piramide de colores arriba
```

```
//bajo rojo
model = glm::mat4(1.0f);
model = glm::translate(model, glm::vec3(0.0f, -0.4f, -2.42f));
model = glm::scale(model, glm::vec3(0.8f, 0.8f, 0.8f));
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()));
meshColorList[0]->RenderMeshColor(); //piramide de colores arriba
```

El problema comienza al generar las partes donde las pirámides de colores están invertidas, ya que estas no solo implican desplazamiento, si no también una rotación para estar correctamente alineadas al ángulo del triángulo, para esto me apoyé de las transformaciones, particularmente la de escala, escalando Y en -1, de modo que la figura se invierte sin tener que crear otro objeto para la pirámide invertida, además también usando rotaciones, para una vez rotada la pirámide, ajustarla al ángulo necesario, para finalmente aplicar transformaciones de traslación para acomodarla en su posición correcta.

Adicionalmente para apoyar en esto se crearon 3 pirámides muy delgadas y ajustadas exactamente en cada eje XYZ, con el propósito de servir como guía para el acomodamiento.



```
//medio verde invertido
model = glm::mat4(1.0f);
model = glm::translate(model, glm::vec3(-0.23f, 0.2f, -3.15f));
model = glm::scale(model, glm::vec3(-0.8f, -0.8f, -0.8f));
model = glm::rotate(model, 180 * toRadians, glm::vec3(0.0f, 1.0f, 0.0f));
model = glm::rotate(model, 50 * toRadians, glm::vec3(-0.4f, 0.0f, 0.7f));
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()));
meshColorList[0]->RenderMeshColor(); //piramide de colores arriba

//izq verde invertido izq
model = glm::mat4(1.0f);
model = glm::translate(model, glm::vec3(-0.24f, -0.328f, -3.72f));
model = glm::scale(model, glm::vec3(-0.8f, -0.78f, -0.8f));
model = glm::rotate(model, 180 * toRadians, glm::vec3(0.0f, 1.0f, 0.0f));
model = glm::rotate(model, 50 * toRadians, glm::vec3(-0.4f, 0.0f, 0.7f));
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()));
meshColorList[0]->RenderMeshColor(); //piramide de colores arriba

//izq verde invertido der
model = glm::mat4(1.0f);
model = glm::translate(model, glm::vec3(-0.7f, -0.328f, -2.85f));
model = glm::scale(model, glm::vec3(-0.8f, -0.78f, -0.8f));
model = glm::rotate(model, 180 * toRadians, glm::vec3(0.0f, 1.0f, 0.0f));
model = glm::rotate(model, 50 * toRadians, glm::vec3(-0.4f, 0.0f, 0.7f));
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()));
meshColorList[0]->RenderMeshColor(); //piramide de colores arriba
```

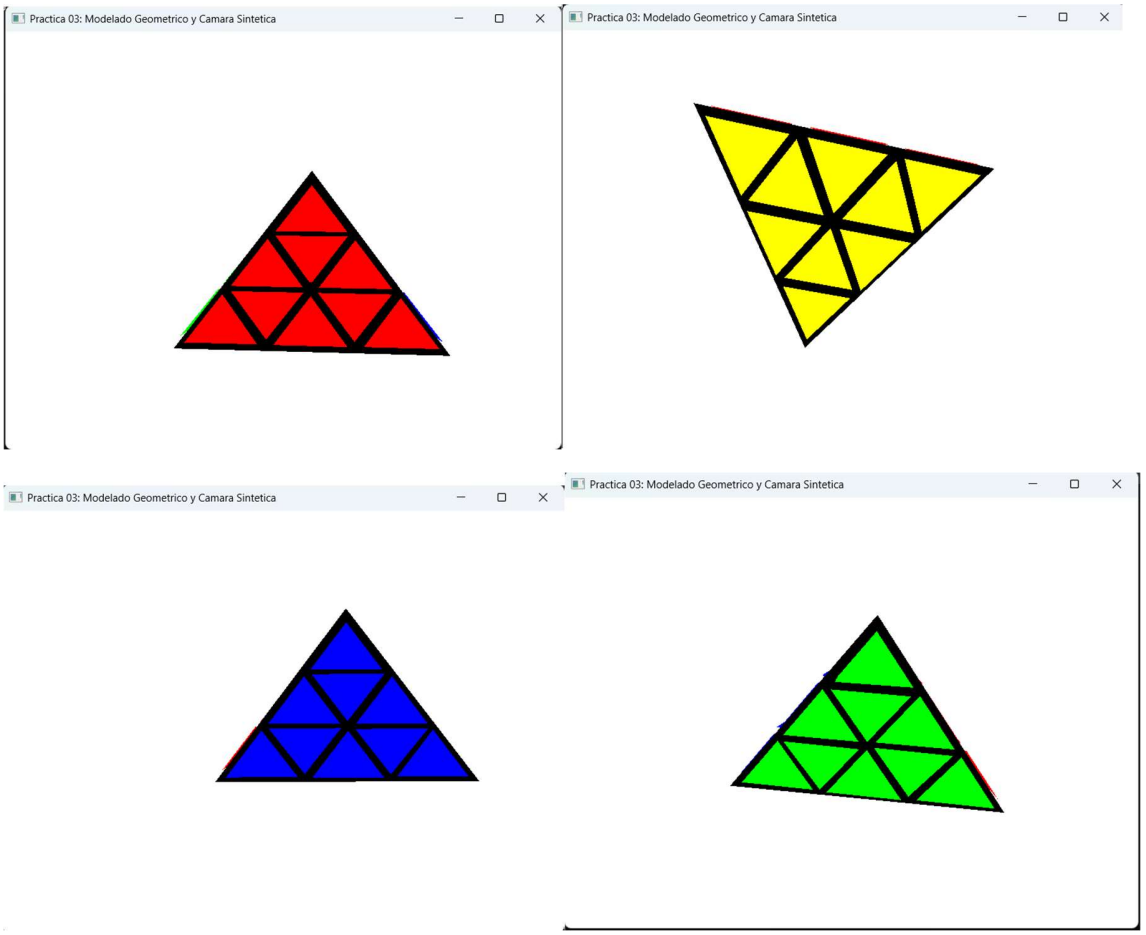
```
//medio azul invertido arriba
model = glm::mat4(1.0f);
model = glm::translate(model, glm::vec3(0.225f, 0.225f, -3.15f));
model = glm::scale(model, glm::vec3(0.8f, -0.8f, 0.8f));
model = glm::rotate(model, -50 * toRadians, glm::vec3(1.1325f, 0.1f,2.0f));
glUniformMatrix4fv(uniformModel, 1, GL_FALSE, glm::value_ptr(model));
glUniformMatrix4fv(uniformProjection, 1, GL_FALSE, glm::value_ptr(projection));
glUniformMatrix4fv(uniformView, 1, GL_FALSE, glm::value_ptr(camera.calculateViewMatrix()));
meshColorList[0]->RenderMeshColor(); //piramide de colores arriba

//medio azul invertido izq
model = glm::mat4(1.0f);
model = glm::translate(model, glm::vec3(0.7f, -0.328f, -2.85f));
model = glm::scale(model, glm::vec3(0.8f, -0.78f, 0.8f));
model = glm::rotate(model, -50 * toRadians, glm::vec3(1.1325f, 0.1f, 2.0f));
//model = glm::rotate(model, -50 * toRadians, glm::vec3(0.0f, 0.1f, 0.0f));
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()));
meshColorList[0]->RenderMeshColor(); //piramide de colores arriba

//medio azul invertido der
model = glm::mat4(1.0f);
model = glm::translate(model, glm::vec3(0.225f, -0.328f, -3.725f));
model = glm::scale(model, glm::vec3(0.8f, -0.78f, 0.8f));
model = glm::rotate(model, -50 * toRadians, glm::vec3(1.1325f, 0.1f, 2.0f));
//model = glm::rotate(model, -50 * toRadians, glm::vec3(0.0f, 0.1f, 0.0f));
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()));
meshColorList[0]->RenderMeshColor(); //piramide de colores arriba
```

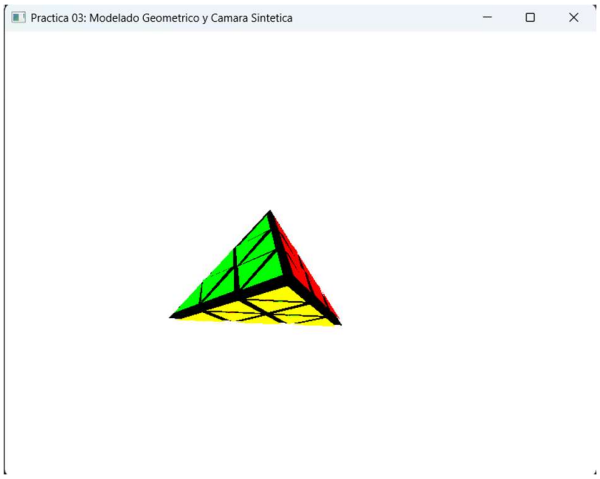
```
//Piramide roja invertida arriba
model = glm::mat4(1.0f);
model = glm::translate(model, glm::vec3(0.0f, 0.22f, -2.75f));
model = glm::scale(model, glm::vec3(0.8f, -0.8f, 0.8f));
model = glm::rotate(model, 50 * toRadians, glm::vec3(1.0f, 0.0f, 0.0f));
glUniformMatrix4fv(uniformModel, 1, GL_FALSE, glm::value_ptr(model));
glUniformMatrix4fv(uniformProjection, 1, GL_FALSE, glm::value_ptr(projection));
glUniformMatrix4fv(uniformView, 1, GL_FALSE, glm::value_ptr(camera.calculateViewMatrix()));
meshColorList[0]->RenderMeshColor(); //piramide de colores arriba

//Piramide roja invertida izq
model = glm::mat4(1.0f);
model = glm::translate(model, glm::vec3(-0.48f, -0.328f, -2.47f));
model = glm::scale(model, glm::vec3(0.8f, -0.78f, 0.8f));
model = glm::rotate(model, 50 * toRadians, glm::vec3(1.0f, 0.0f, 0.0f));
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()));
meshColorList[0]->RenderMeshColor(); //piramide de colores arriba

//Piramide roja invertida der
model = glm::mat4(1.0f);
model = glm::translate(model, glm::vec3(0.48f, -0.328f, -2.47f));
model = glm::scale(model, glm::vec3(0.8f, -0.78f, 0.8f));
model = glm::rotate(model, 50 * toRadians, glm::vec3(1.0f, 0.0f, 0.0f));
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()));
meshColorList[0]->RenderMeshColor(); //piramide de colores arriba
```

```
//amarillo invertido
model = glm::mat4(1.0f);
model = glm::translate(model, glm::vec3(0.0f, -0.4f, -3.55f));
model = glm::scale(model, glm::vec3(-0.8f, 0.8f, -0.8f));
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()));
meshColorList[0]->RenderMeshColor(); //piramide de colores arriba

//amarillo invertido lat 1
model = glm::mat4(1.0f);
model = glm::translate(model, glm::vec3(0.5f, -0.4f, -2.7f));
model = glm::scale(model, glm::vec3(-0.8f, 0.8f, -0.8f));
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()));
meshColorList[0]->RenderMeshColor(); //piramide de colores arriba

//amarillo invertido lat 2
model = glm::mat4(1.0f);
model = glm::translate(model, glm::vec3(-0.5f, -0.4f, -2.7f));
model = glm::scale(model, glm::vec3(-0.8f, 0.8f, -0.8f));
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()));
meshColorList[0]->RenderMeshColor(); //piramide de colores arriba
```

En el caso de la cara amarilla, al ser la base, una vez invertida la pirámide en los ejes X,Z en lugar de Y, la posición se ajusta para simplemente desplazarla sin aplicar rotaciones, lo que hace esa cara más sencilla para trabajar.

Finalmente se obtiene el resultado esperado, el cual es la pyraminx con sus 4 caras de colores, con sus respectivas “estampas”.





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.

Esta práctica resultó bastante compleja y con varios problemas, el primero fue al tratar de generar la pirámide de colores con espacios transparentes, ya que al realizar mal un recorte de cada, se pierde el ángulo de la pirámide con respecto a la pirámide negra de base, lo que imposibilita acomodarlos correctamente, para resolver esto los recortes se hicieron prácticamente décima a décima, de manera de poder ir revisando en cada ejecución que las caras se están recortando manteniendo el ángulo deseado, además de realizar bocetos a mano con los ejes X,Z para ir revisando el desplazamiento que se debía aplicar en cada eje.

Otro de los problemas fue al generar los triángulos invertidos que requerían rotación, al generar la pirámide de colores con punta en el eje Y, las rotaciones se aplican desde el origen, por lo que facilita su acomodo, sin embargo aún así las rotaciones resultan bastante complicadas a la hora de encontrar el ángulo correcto, ya que para ir revisando si cada aumento o disminución del ángulo en cada eje se acercaba o se alejaba más a lo deseado, se tenía que realizar una modificación (aumento/disminución) de cada valor y luego ejecutar para ver si esa rotación era la adecuada y a qué eje aumentarle o disminuirle la rotación.

Estos 2 problemas fueron muy tardados, ya que consistieron en mover cuidadosamente cada aspecto de la figura, incluso llegando a hacer movimientos de 0.025 puntos por eje para tener la mayor exactitud.

3.- Conclusión:

a. Los ejercicios del reporte: Complejidad, Explicación.

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

Como conclusión puedo decir que esta práctica aumentó la complejidad con respecto a las anteriores, ya que, al añadir transformaciones de rotación, requiere mover en ángulos específicos, ayuda bastante el volver a crear la pirámide original, ajustándola para tener su centro en el origen y las 3 caras de tamaño similar, ya que hace las transformaciones más sencillas.

Pienso que faltó explicar un poco más sobre la creación de la pirámide de 3 colores con espacios vacíos en entre caras, y sobre las restricciones, como que no podemos solamente generar las “estampas”.

Como conclusión, esta práctica fue más retadora, ya que se aplican los conocimientos de las 2 prácticas anteriores, y se añaden los conocimientos de esta práctica, fue entretenida e interesante aunque algo tediosa y tardada, por el acomodo pieza a pieza de los elementos.

Bibliografía en formato APA

- *LearnOpenGL - shaders*. (s/f). Learnopengl.com. Recuperado el 19 de febrero de 2025, de https://learnopengl.com/Getting-started/Shaders

- *Tutorial 3 : Matrices*. (s/f). Opengl-tutorial.org. Recuperado el 26 de febrero de 2025, de https://www.opengl-tutorial.org/es/beginners-tutorials/tutorial-3-matrices/