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: MORENO SANTOYO MARIANA

Nº de Cuenta: 319170252

GRUPO DE LABORATORIO: 11

GRUPO DE TEORÍA: 04

SEMESTRE 2025-2

FECHA DE ENTREGA LÍMITE: 05 / 03 /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.

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

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

1. Bibliografía en formato APA

EJERCICIOS:

Generación de una Pyraminx con 9 Pirámides por Cara.

El objetivo de este proyecto es modelar y generar una pirámide de Rubik (Pyraminx) compuesta por un total de 9 pequeñas pirámides en cada una de sus caras. Para garantizar una correcta visualización, cada cara de la Pyraminx deberá estar representada con un color distinto, permitiendo así diferenciar cada lado de manera clara y precisa.

Asimismo, será necesario incorporar líneas de separación entre las pequeñas pirámides, simulando los cortes que permiten distinguir cada una de las subdivisiones dentro de la estructura general del rompecabezas. Estas líneas oscuras serán esenciales para la correcta percepción de la segmentación del modelo.

Requisitos de Entrega.

Para complementar la documentación del proyecto, se incluyen los siguientes elementos:

1. **Capturas de pantalla** que muestren claramente las cuatro caras de la Pyraminx desde distintos ángulos.
2. **Alternativamente, un video** en el que se visualicen las cuatro caras de la Pyraminx en movimiento, asegurando una perspectiva completa del modelo.

Este material servirá como evidencia del correcto desarrollo y representación visual del modelo tridimensional de la Pyraminx.

Desarrollo

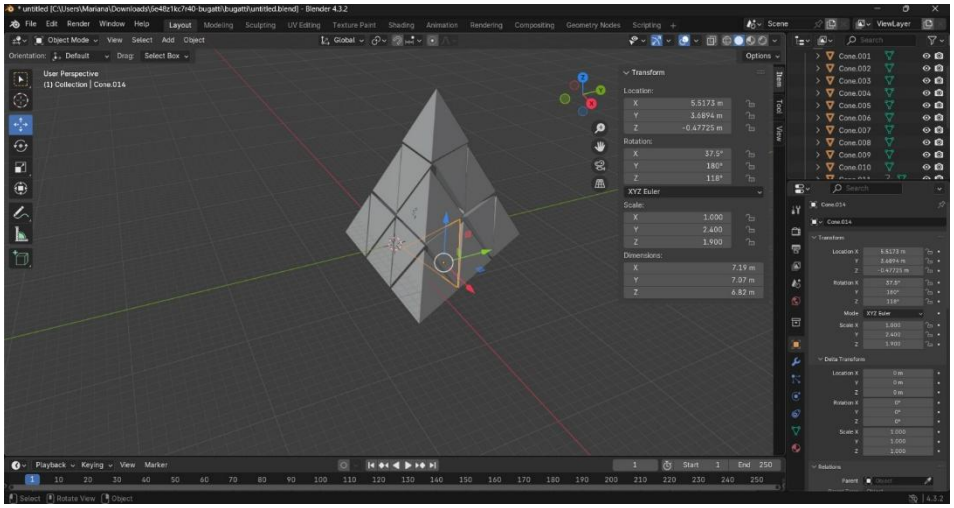
Para la realización de esta práctica, se tomó como base un enfoque similar al de la práctica anterior. En aquella ocasión, la estrategia utilizada para generar pirámides con colores distintos en cada una de sus caras consistió en dibujar triángulo por triángulo, ya que no se logró

implementar el proceso mediante el uso de índices. Dado que este método ya había sido explorado previamente y garantizaba cierto grado de control sobre la estructura de la pirámide, se optó por repetirlo en esta práctica, aunque con nuevos desafíos y un mayor nivel de precisión requerido.

Uno de los aspectos más complejos de esta implementación fue lograr que los triángulos de las caras centrales se alinearan correctamente. La necesidad de hacer rotaciones extremadamente precisas complicó significativamente el proceso, ya que cualquier mínima desviación generaba desajustes visibles en la geometría final. A pesar de los intentos por corregir este problema, no se logró alcanzar la alineación perfecta que se buscaba. Sin embargo, se priorizó la eficiencia en la construcción del modelo, utilizando el menor número de triángulos posible para optimizar recursos y minimizar el impacto en el rendimiento gráfico.

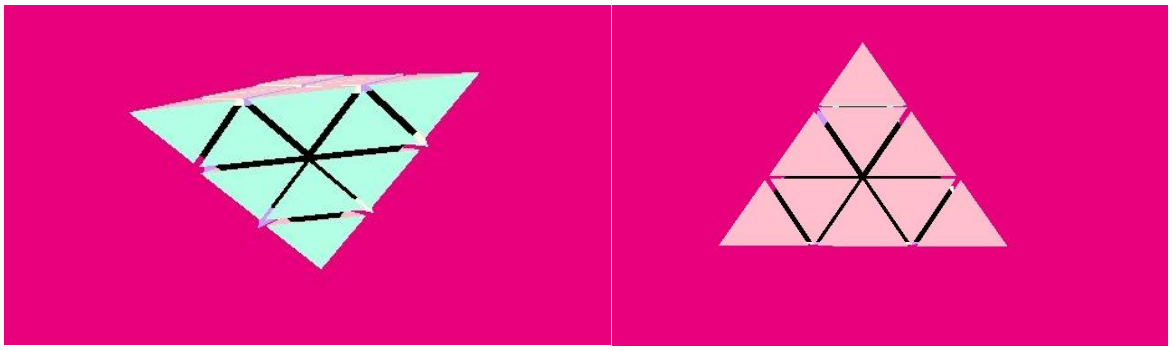
En cuanto a la aplicación del color, se hizo uso de un shader optimizado para asignar diferentes tonalidades a las caras de cada pirámide. Este shader fue modificado a partir del utilizado en la práctica anterior, lo que permitió un mejor desempeño y una apariencia visual más definida en el modelo. No obstante, al trasladar los shaders al código principal (**main**), surgieron problemas inesperados relacionados con la escala y la transformación de la figura. El uso del shader requería realizar un llamado a la cámara, lo que generó cierto grado de distorsión en las formas y afectó la precisión de los desplazamientos dentro del espacio tridimensional.

Para facilitar la disposición de los triángulos en el espacio, se utilizó la herramienta de modelado **Blender**. A través de su sistema de dibujo, se pudo obtener una representación visual más clara de la estructura de la pirámide y determinar la ubicación correcta de cada triángulo antes de su implementación en el código. Esta experiencia permitió adquirir conocimientos valiosos sobre el uso de Blender y sus capacidades para modelado 3D. Sin embargo, a pesar de su utilidad en la fase de diseño, no fue suficiente para solucionar el problema de rotación que se presentó durante la construcción de la pirámide.

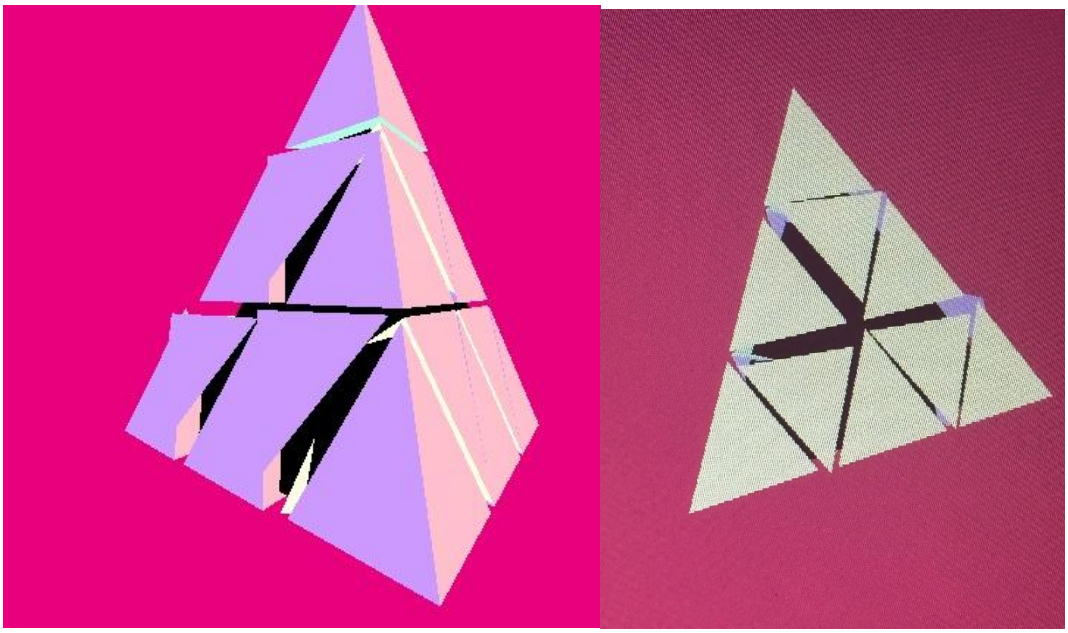


Modelo en *Blender*.

Pirámide definitiva.



Caras buenas.



Caras malas.

```
//Limpiar la ventana
glClearColor(0.91f, 0.0f, 0.49f, 0.0f);
glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT); //Se agrega limpiar el buffer de profundidad
shaderList[1].useShader();
uniformModel = shaderList[1].getModelLocation();
uniformProjection = shaderList[1].getProjectLocation();
uniformView = shaderList[1].getViewLocation();
uniformColor = shaderList[1].getColorLocation();
```

Shader agregado para pirámides de colores.

```
void CrearPiramideTriangularRECTANGULAR()
{

    GLfloat vertices_piramide_triangularRECTANGULAR[] = {


        -3.5967f,-1.7941f,-0.98135f,    1.0f, 0.75f, 0.8f, // vertice izquierdo piramide rectangular cara frontal
        3.5967f, -1.7941f,-0.98135f,    1.0f, 0.75f, 0.8f, // vertice derecho piramide rectangular cara frontal
        0.0f,1.7941f, 0.0f,             1.0f, 0.75f, 0.8f, // vertice superior piramide rectangular cara frontal

        3.5967f, -1.7941f,-0.98135f,    0.8f, 0.6f, 1.0f, // vertice derecho piramide rectangular cara frontal
        0.0f,-1.7941f,    1.9627f,          0.8f, 0.6f, 1.0f, // vertice superior piramide rectangular cara frontal
        0.0f,1.7941f, 0.0f,         0.8f, 0.6f, 1.0f, // vertice inferior piramide rectangular cara frontal

        -3.5967f,-1.7941f,-0.98135f,           10.7f, 1.0f, 0.9f, // vertice inferior piramide rectangular cara frontal
        0.0f,1.7941f, 0.0f,      10.7f, 1.0f, 0.9f, // vertice izquierdo piramide rectangular cara frontal
        0.0f,-1.7941f,1.9627f,             10.7f, 1.0f, 0.9f, // vertice superior piramide rectangular cara frontal

        -3.5967f,-1.7941f,-0.98135f,       0.7f, 1.0f, 0.9f, // vertice inferior piramide rectangular cara frontal
        3.5967f, -1.7941f,-0.98135f,    0.7f, 1.0f, 0.9f, // vertice izquierdo piramide rectangular cara frontal
        0.0f,-1.7941f,1.9627f,  0.7f, 1.0f, 0.9f, // vertice derecho piramide rectangular cara frontal


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

Función para crea pirámides con cada uno de sus lados de diferente color.

```
// Pirámide triangular regular
void CrearPiramideTriangularINVERSA()
{
    unsigned int indices_piramide_triangularINVERSA[] = {
            0,1,2,
            1,3,2,
            3,0,2,
            1,0,3

    };
    GLfloat vertices_piramide_triangularINVERSA[] = {
        3.6463,3.6609f,6.7681f, //0
        7.0234f, 3.6609f, 0.41671f,     //1
        7.3495f,-3.1816, 4.6639f,   //2////PICOS
        0.38538f,-0.6404f, 0.96875f,    //3



    };
    Mesh* obj2 = new Mesh();
    obj2->CreateMesh(vertices_piramide_triangularINVERSA, indices_piramide_triangularINVERSA, 12, 12);
    meshList.push_back(obj2);

}
```

Creación de una pirámide rectangular para la pirámide negra del fondo, y las minipirámides de color sólido.

```
model = glm::mat4(1.0f);
//color = glm::vec3(0.0f, 0.0f, 0.0f);

model = glm::translate(model, glm::vec3(0.0f, 0.0f, -11.0f));
model = glm::rotate(model, glm::radians(180.0f), glm::vec3(0.0f, 1.0f, 0.0f));
model = glm::translate(model, glm::vec3(0.0f, -1.4f, 0.0f)); //traslación para posicionar en -Z a los objetos
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));
model = glm::scale(model, glm::vec3(0.7f, 1.0f, 1.0f));
glUniformMatrix4fv(uniformModel, 1, GL_FALSE, glm::value_ptr(model));//FALSE ES PARA QUE NO SEA TRANSPUESTA y se envían al shader como variables de tipo uniform
glUniformMatrix4fv(uniformProjection, 1, GL_FALSE, glm::value_ptr(projection));
meshColorList[0]->RenderMeshColor();


// izquierda

model = glm::mat4(1.0f);
model = glm::translate(model, glm::vec3(0.0f, 0.0f, -11.0f));
model = glm::rotate(model, glm::radians(180.0f), glm::vec3(0.0f, 1.0f, 0.0f));
model = glm::translate(model, glm::vec3(-5.5f, -1.4f, 0.0f)); //traslación para posicionar en -Z a los objetos
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));
model = glm::scale(model, glm::vec3(0.7f, 1.0f, 1.0f));
glUniformMatrix4fv(uniformModel, 1, GL_FALSE, glm::value_ptr(model));//FALSE ES PARA QUE NO SEA TRANSPUESTA y se envían al shader como variables de tipo uniform
glUniformMatrix4fv(uniformProjection, 1, GL_FALSE, glm::value_ptr(projection));
meshColorList[0]->RenderMeshColor();

// derecha
model = glm::mat4(1.0f);
model = glm::translate(model, glm::vec3(0.0f, 0.0f, -11.0f));
model = glm::rotate(model, glm::radians(180.0f), glm::vec3(0.0f, 1.0f, 0.0f));
model = glm::translate(model, glm::vec3(5.5f, -1.4f, 0.0f)); //traslación para posicionar en -Z a los objetos
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));
model = glm::scale(model, glm::vec3(0.7f, 1.0f, 1.0f));
glUniformMatrix4fv(uniformModel, 1, GL_FALSE, glm::value_ptr(model));//FALSE ES PARA QUE NO SEA TRANSPUESTA y se envían al shader como variables de tipo uniform
glUniformMatrix4fv(uniformProjection, 1, GL_FALSE, glm::value_ptr(projection));
meshColorList[0]->RenderMeshColor();


// derecha atras

model = glm::mat4(1.0f);
model = glm::translate(model, glm::vec3(0.0f, 0.0f, -11.0f));
model = glm::rotate(model, glm::radians(180.0f), glm::vec3(0.0f, 1.0f, 0.0f));
model = glm::translate(model, glm::vec3(-2.75f, -1.4f, 3.5f)); //traslación para posicionar en -Z a los objetos
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));
model = glm::scale(model, glm::vec3(0.7f, 1.0f, 1.0f));
glUniformMatrix4fv(uniformModel, 1, GL_FALSE, glm::value_ptr(model));//FALSE ES PARA QUE NO SEA TRANSPUESTA y se envían al shader como variables de tipo uniform
glUniformMatrix4fv(uniformProjection, 1, GL_FALSE, glm::value_ptr(projection));
meshColorList[0]->RenderMeshColor();

// izquierda atras
model = glm::mat4(1.0f);
model = glm::translate(model, glm::vec3(0.0f, 0.0f, -11.0f));
model = glm::rotate(model, glm::radians(180.0f), glm::vec3(0.0f, 1.0f, 0.0f));
model = glm::translate(model, glm::vec3(2.75f, -1.4f, 3.5f)); //traslación para posicionar en -Z a los objetos
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));
model = glm::scale(model, glm::vec3(0.7f, 1.0f, 1.0f));
glUniformMatrix4fv(uniformModel, 1, GL_FALSE, glm::value_ptr(model));//FALSE ES PARA QUE NO SEA TRANSPUESTA y se envían al shader como variables de tipo uniform
glUniformMatrix4fv(uniformProjection, 1, GL_FALSE, glm::value_ptr(projection));
meshColorList[0]->RenderMeshColor();
```

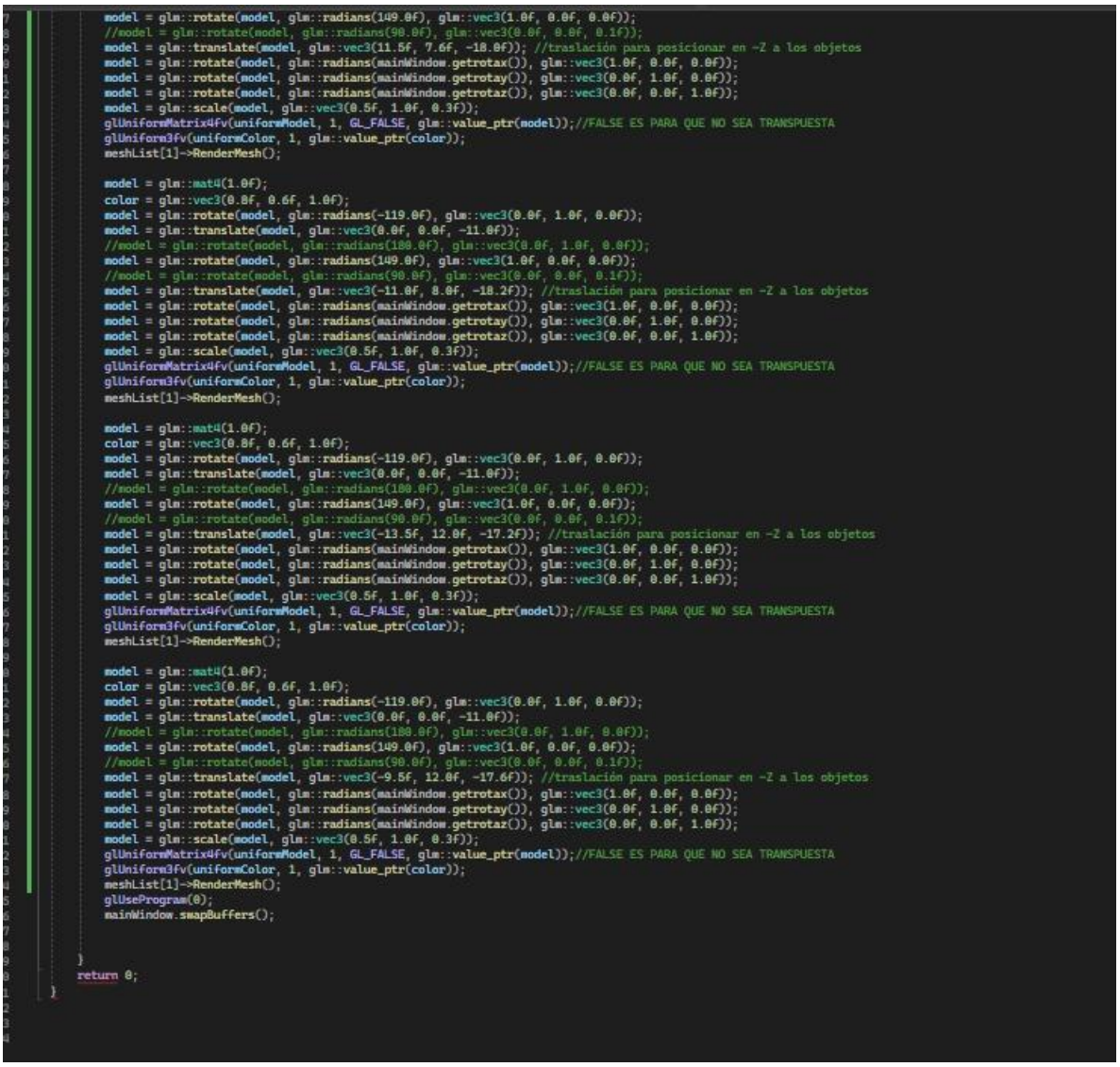
Instanciación de las pirámides de colores acomodadas (Funcionó con éxito).

```
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));  //al presionar la tecla Y se rota sobre el eje y
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(1.0f, 0.0f, 1.0f);
glUniform3fv(uniformColor, 1, glm::value_ptr(color)); //para cambiar el color del objetos
```

Intento de cambio del shader. Posible error

```
// izquierda

model = glm::mat4(1.0f);
color = glm::vec3(0.0f, 0.0f, 0.0f);
model = glm::translate(model, glm::vec3(0.0f, 0.0f, -11.0f));
model = glm::rotate(model, glm::radians(180.0f), glm::vec3(0.0f, 1.0f, 0.0f));
model = glm::translate(model, glm::vec3(0.0f, 1.6f, 2.0f)); //traslación para posicionar en -Z a los objetos
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));
model = glm::scale(model, glm::vec3(2.0f, 2.6f, 2.6f));
glUniformMatrix4fv(uniformModel, 1, GL_FALSE, glm::value_ptr(model));//FALSE ES PARA QUE NO SEA TRANSPUESTA
glUniform3fv(uniformColor, 1, glm::value_ptr(color));
meshList[1]->RenderMesh();



model = glm::mat4(1.0f);
color = glm::vec3(10.7f, 1.0f, 0.9f);
model = glm::rotate(model, glm::radians(119.0f), glm::vec3(0.0f, 1.0f, 0.0f));
model = glm::translate(model, glm::vec3(0.0f, 0.0f, -11.0f));
//model = glm::rotate(model, glm::radians(180.0f), glm::vec3(0.0f, 1.0f, 0.0f));
model = glm::rotate(model, glm::radians(149.0f), glm::vec3(1.0f, 0.0f, 0.0f));
//model = glm::rotate(model, glm::radians(90.0f), glm::vec3(0.0f, 0.0f, 0.1f));
model = glm::translate(model, glm::vec3(13.3f, 12.0f, -17.0f)); //traslación para posicionar en -Z a los objetos
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));
model = glm::scale(model, glm::vec3(0.5f, 1.0f, 0.3f));
glUniformMatrix4fv(uniformModel, 1, GL_FALSE, glm::value_ptr(model));//FALSE ES PARA QUE NO SEA TRANSPUESTA
glUniform3fv(uniformColor, 1, glm::value_ptr(color));
meshList[1]->RenderMesh();


model = glm::mat4(1.0f);
color = glm::vec3(10.7f, 1.0f, 0.9f);
model = glm::rotate(model, glm::radians(119.0f), glm::vec3(0.0f, 1.0f, 0.0f));
model = glm::translate(model, glm::vec3(0.0f, 0.0f, -11.0f));
//model = glm::rotate(model, glm::radians(180.0f), glm::vec3(0.0f, 1.0f, 0.0f));
model = glm::rotate(model, glm::radians(149.0f), glm::vec3(1.0f, 0.0f, 0.0f));
//model = glm::rotate(model, glm::radians(90.0f), glm::vec3(0.0f, 0.0f, 0.1f));
model = glm::translate(model, glm::vec3(9.0f, 12.5f, -17.6f)); //traslación para posicionar en -Z a los objetos
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));
model = glm::scale(model, glm::vec3(0.5f, 1.0f, 0.3f));
glUniformMatrix4fv(uniformModel, 1, GL_FALSE, glm::value_ptr(model));//FALSE ES PARA QUE NO SEA TRANSPUESTA
glUniform3fv(uniformColor, 1, glm::value_ptr(color));
meshList[1]->RenderMesh();


model = glm::mat4(1.0f);
color = glm::vec3(10.7f, 1.0f, 0.9f);
model = glm::rotate(model, glm::radians(119.0f), glm::vec3(0.0f, 1.0f, 0.0f));
model = glm::translate(model, glm::vec3(0.0f, 0.0f, -11.0f));
//model = glm::rotate(model, glm::radians(180.0f), glm::vec3(0.0f, 1.0f, 0.0f));
model = glm::rotate(model, glm::radians(149.0f), glm::vec3(1.0f, 0.0f, 0.0f));
//model = glm::rotate(model, glm::radians(90.0f), glm::vec3(0.0f, 0.0f, 0.1f));
model = glm::translate(model, glm::vec3(11.5f, 7.6f, -18.0f)); //traslación para posicionar en -Z a los objetos
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));
model = glm::scale(model, glm::vec3(0.5f, 1.0f, 0.3f));
glUniformMatrix4fv(uniformModel, 1, GL_FALSE, glm::value_ptr(model));//FALSE ES PARA QUE NO SEA TRANSPUESTA
glUniform3fv(uniformColor, 1, glm::value_ptr(color));
meshList[1]->RenderMesh();
```

Instanciación de las pirámides invertidas.



```
        model = glm::rotate(model, glm::radians(149.0f), glm::vec3(1.0f, 0.0f, 0.0f));
        //model = glm::rotate(model, glm::radians(90.0f), glm::vec3(0.0f, 0.0f, 0.1f));
        model = glm::translate(model, glm::vec3(11.5f, 7.6f, -18.0f)); //traslación para posicionar en -Z a los objetos
        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));
        model = glm::scale(model, glm::vec3(0.5f, 1.0f, 0.3f));
        glUniformMatrix4fv(uniformModel, 1, GL_FALSE, glm::value_ptr(model));//FALSE ES PARA QUE NO SEA TRANSPUESTA
        glUniform3fv(uniformColor, 1, glm::value_ptr(color));
        meshList[1]->RenderMesh();

        model = glm::mat4(1.0f);
        color = glm::vec3(0.8f, 0.6f, 1.0f);
        model = glm::rotate(model, glm::radians(-119.0f), glm::vec3(0.0f, 1.0f, 0.0f));
        model = glm::translate(model, glm::vec3(0.0f, 0.0f, -11.0f));
        //model = glm::rotate(model, glm::radians(180.0f), glm::vec3(0.0f, 1.0f, 0.0f));
        model = glm::rotate(model, glm::radians(149.0f), glm::vec3(1.0f, 0.0f, 0.0f));
        //model = glm::rotate(model, glm::radians(90.0f), glm::vec3(0.0f, 0.0f, 0.1f));
        model = glm::translate(model, glm::vec3(-11.0f, 8.0f, -18.2f)); //traslación para posicionar en -Z a los objetos
        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));
        model = glm::scale(model, glm::vec3(0.5f, 1.0f, 0.3f));
        glUniformMatrix4fv(uniformModel, 1, GL_FALSE, glm::value_ptr(model));//FALSE ES PARA QUE NO SEA TRANSPUESTA
        glUniform3fv(uniformColor, 1, glm::value_ptr(color));
        meshList[1]->RenderMesh();

        model = glm::mat4(1.0f);
        color = glm::vec3(0.8f, 0.6f, 1.0f);
        model = glm::rotate(model, glm::radians(-119.0f), glm::vec3(0.0f, 1.0f, 0.0f));
        model = glm::translate(model, glm::vec3(0.0f, 0.0f, -11.0f));
        //model = glm::rotate(model, glm::radians(180.0f), glm::vec3(0.0f, 1.0f, 0.0f));
        model = glm::rotate(model, glm::radians(149.0f), glm::vec3(1.0f, 0.0f, 0.0f));
        //model = glm::rotate(model, glm::radians(90.0f), glm::vec3(0.0f, 0.0f, 0.1f));
        model = glm::translate(model, glm::vec3(-13.5f, 12.0f, -17.2f)); //traslación para posicionar en -Z a los objetos
        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));
        model = glm::scale(model, glm::vec3(0.5f, 1.0f, 0.3f));
        glUniformMatrix4fv(uniformModel, 1, GL_FALSE, glm::value_ptr(model));//FALSE ES PARA QUE NO SEA TRANSPUESTA
        glUniform3fv(uniformColor, 1, glm::value_ptr(color));
        meshList[1]->RenderMesh();

        model = glm::mat4(1.0f);
        color = glm::vec3(0.8f, 0.6f, 1.0f);
        model = glm::rotate(model, glm::radians(-119.0f), glm::vec3(0.0f, 1.0f, 0.0f));
        model = glm::translate(model, glm::vec3(0.0f, 0.0f, -11.0f));
        //model = glm::rotate(model, glm::radians(180.0f), glm::vec3(0.0f, 1.0f, 0.0f));
        model = glm::rotate(model, glm::radians(149.0f), glm::vec3(1.0f, 0.0f, 0.0f));
        //model = glm::rotate(model, glm::radians(90.0f), glm::vec3(0.0f, 0.0f, 0.1f));
        model = glm::translate(model, glm::vec3(-9.5f, 12.0f, -17.6f)); //traslación para posicionar en -Z a los objetos
        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));
        model = glm::scale(model, glm::vec3(0.5f, 1.0f, 0.3f));
        glUniformMatrix4fv(uniformModel, 1, GL_FALSE, glm::value_ptr(model));//FALSE ES PARA QUE NO SEA TRANSPUESTA
        glUniform3fv(uniformColor, 1, glm::value_ptr(color));
        meshList[1]->RenderMesh();
        glUseProgram(0);
        mainWindow.swapBuffers();


    }
    return 0;
}
```

Instanciación de pirámides invertidas (Fallido).

Conclusiones

Esta práctica resultó ser un desafío significativo debido a la complejidad de las incrustaciones necesarias en la estructura de la Pyraminx. Uno de los mayores retos fue la correcta alineación de los triángulos que conforman cada una de las pequeñas pirámides, ya que cualquier desajuste afectaba la simetría y la coherencia visual del modelo. Lograr una distribución precisa y armoniosa requirió un alto nivel de atención al detalle y un proceso iterativo de ajustes y correcciones.

Además, este proyecto permitió reforzar conocimientos clave en el uso de shaders para la aplicación de distintos colores en cada cara de la pirámide. Experimentar con los shaders no solo mejoró la apariencia visual del modelo, sino que también aportó una comprensión más

profunda sobre cómo manipular materiales y efectos gráficos dentro de un entorno tridimensional. Sin embargo, en la integración de los shaders dentro del código principal surgieron ciertos inconvenientes, como la distorsión en la escala de la figura, lo que representó otro reto dentro del desarrollo del proyecto.

En general, la práctica representó un reto significativo, tanto en términos de alineación geométrica como en la correcta integración de los shaders y la optimización de la estructura. A pesar de las dificultades encontradas, se logró obtener un modelo funcional y se adquirieron conocimientos valiosos sobre manipulación de gráficos en 3D, rotaciones y uso de herramientas de modelado como **Blender**. Este proceso deja abiertas oportunidades para futuras mejoras y optimizaciones que permitan una representación más precisa y eficiente de la Pyraminx.

No obstante, debo reconocer que quedé insatisfecha con mi desempeño en esta práctica, ya que esperaba lograr una mejor alineación en las pirámides. A pesar del esfuerzo y la dedicación invertidos, el problema principal radicó en los triángulos invertidos, los cuales afectaron considerablemente la precisión del modelo final. Esto demuestra la importancia de una planificación más detallada y una revisión más minuciosa de los elementos geométricos antes de su implementación. Sin embargo, este desafío también representa una oportunidad de aprendizaje que permitirá mejorar en futuros proyectos y enfrentar problemas similares con mayor preparación y experiencia