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: Tapia Ledesma Angel Hazel

Nº de Cuenta: 320070358

GRUPO DE LABORATORIO: 02

GRUPO DE TEORÍA: 02

SEMESTRE 2026-1

FECHA DE ENTREGA LÍMITE: 07-09-25

CALIFICACIÓN: _____

Actividades realizadas

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)
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

Descripción de las actividades

Para el desarrollo de la actividad, lo primero que se hizo fue generar una nueva figura, ya que la pirámide dada, era una pirámide que base con un triángulo rectángulo, mientras que la necesaria para esta práctica era la isósceles, y todas las caras así, por lo que se generó esa nueva figura, aunque también la podríamos haber hecho con el cono y poner 3 caras solo.

Después lo que se hizo fue generar la pyraminx, para esto, se dividió en varios pasos, el primer paso fue generar la pyraminx grande de color negro, esta tiene la función de "simular" las divisiones entre las pirámides más pequeñas que están deeentro de la pyraminx. El segundo paro es generar las pirámides pequeñas, las cuales son 9, para este caso, lo que se hizo fue dividir por 3 niveles, de la punta a la base, para hacerlo más sencillo, aquí es donde se calculan las coordenadas, lo cual fue bastante desafiante, ya que el triángulo tiene varios cambios. Una vez generada una cara, lo que se hace es cambiar el color de las mini pirámides y seguir calculando las demás caras, para la primera no fue tan complejo, ya la base, descansaba paralela al eje x, por lo que solo había que modificar altura y profundidad en cada "nivel", pero para las laterales fue más difícil, ya que ocupábamos adaptar las coordenadas y además la rotación.

Finalmente, ya que hemos construido estas caras, pasamos a la inferior, la cual con las caras que ya teníamos construidas, fue sencillo, ya que la parte inferior de las mini pirámides de cada lado, coinciden con la posición de las mini pirámides en la base, por lo que solo se tuvo que calcular los triángulos internos o "invertidos" los cuales solo eran 3, y no fue tan difícil porque solo había que rotar 180 grados y calcular en x y z.

Código generado:

Figura generada para la pyraminx

```
void CrearTetaedroRegular()
{
    unsigned int indices_tetaedro_regular[] = {
            0,1,2,
            1,3,2,
            3,0,2,
            1,0,3

    };
    GLfloat vertices_tetaedro_regular[] = {
        -0.5f, -0.5f,0.25f, //0
        0.5f,-0.5f,0.25f,   //1
        0.0f,0.5f, 0.0f,    //2
        0.0f,-0.5f,-0.5f,   //3

    };
    Mesh* tetaedro = new Mesh();
    tetaedro->CreateMesh(vertices_tetaedro_regular, indices_tetaedro_regular, 12, 12);
    meshList.push_back(tetaedro);

}
```

Main

```
// 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);
model = glm::translate(model, glm::vec3(0.0f, 0.0f, -4.0f));
model = glm::scale(model, glm::vec3(1.0f, 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));
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();

// Cara roja

// Primer nivel
// Piramide punta
shaderList[0].useShader();
uniformModel = shaderList[0].getModelLocation();
uniformProjection = shaderList[0].getProjectLocation();
uniformView = shaderList[0].getViewLocation();
uniformColor = shaderList[0].getColorLocation();
model = glm::mat4(1.0);
model = glm::translate(model, glm::vec3(0.0f, 0.3, -3.98f));
model = glm::scale(model, glm::vec3(0.25f, 0.25f, 0.25));
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));
glUniformMatrix4fv(uniformView, 1, GL_FALSE, glm::value_ptr(camera.calculateViewMatrix()));
color = glm::vec3(1.0f, 0.0f, 0.0f);
glUniform3fv(uniformColor, 1, glm::value_ptr(color)); //para cambiar el color del objetos
meshList[5]->RenderMesh();

// segundo nivel
// Piramide izquierda
shaderList[0].useShader();
uniformModel = shaderList[0].getModelLocation();
uniformProjection = shaderList[0].getProjectLocation();
uniformView = shaderList[0].getViewLocation();
uniformColor = shaderList[0].getColorLocation();
model = glm::mat4(1.0);
model = glm::translate(model, glm::vec3(-0.15f, 0.0f, -3.9f));
model = glm::scale(model, glm::vec3(0.25f, 0.25f, 0.25f));
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));
glUniformMatrix4fv(uniformView, 1, GL_FALSE, glm::value_ptr(camera.calculateViewMatrix()));
color = glm::vec3(1.0f, 0.0f, 0.0f);
glUniform3fv(uniformColor, 1, glm::value_ptr(color)); //para cambiar el color del objetos
meshList[5]->RenderMesh();
```

```
// Piramide  invertida medio
shaderList[0].useShader();
uniformModel = shaderList[0].getModelLocation();
uniformProjection = shaderList[0].getProjectLocation();
uniformView = shaderList[0].getViewLocation();
uniformColor = shaderList[0].getColorLocation();
model = glm::mat4(1.0);
model = glm::translate(model, glm::vec3(0.0f, 0.00, -3.9f));
model = glm::rotate(model, -30 * toRadians, glm::vec3(1.0f, 0.0f, 0.0f)); // x
model = glm::rotate(model, 0 * toRadians, glm::vec3(0.0f, 1.0f, 0.0f));  // y
model = glm::rotate(model, 180 * toRadians, glm::vec3(0.0f, 0.0f, 1.0f)); // Z
model = glm::scale(model, glm::vec3(0.25f, 0.25f, 0.25f));
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));
glUniformMatrix4fv(uniformView, 1, GL_FALSE, glm::value_ptr(camera.calculateViewMatrix()));
color = glm::vec3(1.0f, 0.0f, 0.0f);
glUniform3fv(uniformColor, 1, glm::value_ptr(color)); //para cambiar el color del objetos
meshList[5]->RenderMesh();

// Piramide derecha
shaderList[0].useShader();
uniformModel = shaderList[0].getModelLocation();
uniformProjection = shaderList[0].getProjectLocation();
uniformView = shaderList[0].getViewLocation();
uniformColor = shaderList[0].getColorLocation();
model = glm::mat4(1.0);
model = glm::translate(model, glm::vec3(0.15f, 0.0f, -3.9f));
model = glm::scale(model, glm::vec3(0.25f, 0.25f, 0.25f));
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));
glUniformMatrix4fv(uniformView, 1, GL_FALSE, glm::value_ptr(camera.calculateViewMatrix()));
color = glm::vec3(1.0f, 0.0f, 0.0f);
glUniform3fv(uniformColor, 1, glm::value_ptr(color)); //para cambiar el color del objetos
meshList[5]->RenderMesh();

// Tercer nivel
// Piramide izquierda
shaderList[0].useShader();
uniformModel = shaderList[0].getModelLocation();
uniformProjection = shaderList[0].getProjectLocation();
uniformView = shaderList[0].getViewLocation();
uniformColor = shaderList[0].getColorLocation();
model = glm::mat4(1.0);
model = glm::translate(model, glm::vec3(-0.3f, -0.3f, -3.82f));
model = glm::scale(model, glm::vec3(0.25f, 0.25f, 0.25f));
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));
glUniformMatrix4fv(uniformView, 1, GL_FALSE, glm::value_ptr(camera.calculateViewMatrix()));
color = glm::vec3(1.0f, 0.0f, 0.0f);
glUniform3fv(uniformColor, 1, glm::value_ptr(color)); //para cambiar el color del objetos
meshList[5]->RenderMesh();
```

```
// Piramide invertida izquierda
shaderList[0].useShader();
uniformModel = shaderList[0].getModelLocation();
uniformProjection = shaderList[0].getProjectLocation();
uniformView = shaderList[0].getViewLocation();
uniformColor = shaderList[0].getColorLocation();
model = glm::mat4(1.0);
model = glm::translate(model, glm::vec3(-0.15f, -0.282, -3.82f));
model = glm::rotate(model, -30 * toRadians, glm::vec3(1.0f, 0.0f, 0.0f)); // x
model = glm::rotate(model, 0 * toRadians, glm::vec3(0.0f, 1.0f, 0.0f));  // y
model = glm::rotate(model, 180 * toRadians, glm::vec3(0.0f, 0.0f, 1.0f)); // Z
model = glm::scale(model, glm::vec3(0.25f, 0.25f, 0.25f));
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));
glUniformMatrix4fv(uniformView, 1, GL_FALSE, glm::value_ptr(camera.calculateViewMatrix()));
color = glm::vec3(1.0f, 0.0f, 0.0f);
glUniform3fv(uniformColor, 1, glm::value_ptr(color)); //para cambiar el color del objetos
meshList[5]->RenderMesh();

// Piramide medio
shaderList[0].useShader();
uniformModel = shaderList[0].getModelLocation();
uniformProjection = shaderList[0].getProjectLocation();
uniformView = shaderList[0].getViewLocation();
uniformColor = shaderList[0].getColorLocation();
model = glm::mat4(1.0);
model = glm::translate(model, glm::vec3(0.0f, -0.3f, -3.82f));
model = glm::scale(model, glm::vec3(0.25f, 0.25f, 0.25f));
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));
glUniformMatrix4fv(uniformView, 1, GL_FALSE, glm::value_ptr(camera.calculateViewMatrix()));
color = glm::vec3(1.0f, 0.0f, 0.0f);
glUniform3fv(uniformColor, 1, glm::value_ptr(color)); //para cambiar el color del objetos
meshList[5]->RenderMesh();

// Piramide invertida derecha
shaderList[0].useShader();
uniformModel = shaderList[0].getModelLocation();
uniformProjection = shaderList[0].getProjectLocation();
uniformView = shaderList[0].getViewLocation();
uniformColor = shaderList[0].getColorLocation();
model = glm::mat4(1.0);
model = glm::translate(model, glm::vec3(0.15f, -0.3, -3.82f));
model = glm::rotate(model, -30 * toRadians, glm::vec3(1.0f, 0.0f, 0.0f)); // x
model = glm::rotate(model, 0 * toRadians, glm::vec3(0.0f, 1.0f, 0.0f));  // y
model = glm::rotate(model, 180 * toRadians, glm::vec3(0.0f, 0.0f, 1.0f)); // Z
model = glm::scale(model, glm::vec3(0.25f, 0.25f, 0.25f));
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));
glUniformMatrix4fv(uniformView, 1, GL_FALSE, glm::value_ptr(camera.calculateViewMatrix()));
color = glm::vec3(1.0f, 0.0f, 0.0f);
glUniform3fv(uniformColor, 1, glm::value_ptr(color)); //para cambiar el color del objetos
meshList[5]->RenderMesh();
```

```
// Piramide derecha
shaderList[0].useShader();
uniformModel = shaderList[0].getModelLocation();
uniformProjection = shaderList[0].getProjectLocation();
uniformView = shaderList[0].getViewLocation();
uniformColor = shaderList[0].getColorLocation();
model = glm::mat4(1.0);
model = glm::translate(model, glm::vec3(0.3f, -0.3f, -3.82f));
model = glm::scale(model, glm::vec3(0.25f, 0.25f, 0.25f));
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));
glUniformMatrix4fv(uniformView, 1, GL_FALSE, glm::value_ptr(camera.calculateViewMatrix()));
color = glm::vec3(1.0f, 0.0f, 0.0f);
glUniform3fv(uniformColor, 1, glm::value_ptr(color)); //para cambiar el color del objetos
meshList[5]->RenderMesh();

// Cara Azul
// Piramide punta
shaderList[0].useShader();
uniformModel = shaderList[0].getModelLocation();
uniformProjection = shaderList[0].getProjectLocation();
uniformView = shaderList[0].getViewLocation();
uniformColor = shaderList[0].getColorLocation();
model = glm::mat4(1.0);
model = glm::translate(model, glm::vec3(-0.03f, 0.3, -4.02f));
model = glm::scale(model, glm::vec3(0.25f, 0.25f, 0.25));
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));
glUniformMatrix4fv(uniformView, 1, GL_FALSE, glm::value_ptr(camera.calculateViewMatrix()));
color = glm::vec3(0.658f, 0.837f, 0.95f);
glUniform3fv(uniformColor, 1, glm::value_ptr(color)); //para cambiar el color del objetos
meshList[5]->RenderMesh();

// segundo nivel
// Piramide izquierda
shaderList[0].useShader();
uniformModel = shaderList[0].getModelLocation();
uniformProjection = shaderList[0].getProjectLocation();
uniformView = shaderList[0].getViewLocation();
uniformColor = shaderList[0].getColorLocation();
model = glm::mat4(1.0);
model = glm::translate(model, glm::vec3(-0.03f, 0.0f, -4.16f));
model = glm::scale(model, glm::vec3(0.25f, 0.25f, 0.25f));
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));
glUniformMatrix4fv(uniformView, 1, GL_FALSE, glm::value_ptr(camera.calculateViewMatrix()));
color = glm::vec3(0.658f, 0.837f, 0.95f);
glUniform3fv(uniformColor, 1, glm::value_ptr(color)); //para cambiar el color del objetos
meshList[5]->RenderMesh();
```

```
// Piramide invertida medio
shaderList[0].useShader();
uniformModel = shaderList[0].getModelLocation();
uniformProjection = shaderList[0].getProjectLocation();
uniformView = shaderList[0].getViewLocation();
uniformColor = shaderList[0].getColorLocation();
model = glm::mat4(1.0);
model = glm::translate(model, glm::vec3(-0.11f, 0.0f, -4.05f));
model = glm::rotate(model, 15 * toRadians, glm::vec3(1.0f, 0.0f, 0.0f)); // x
model = glm::rotate(model, 0 * toRadians, glm::vec3(0.0f, 1.0f, 0.0f));  // y
model = glm::rotate(model, 151 * toRadians, glm::vec3(0.0f, 0.0f, 1.0f)); // Z;
model = glm::scale(model, glm::vec3(0.25f, 0.25f, 0.25f));
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));
glUniformMatrix4fv(uniformView, 1, GL_FALSE, glm::value_ptr(camera.calculateViewMatrix()));
color = glm::vec3(0.658f, 0.837f, 0.95f);
glUniform3fv(uniformColor, 1, glm::value_ptr(color)); //para cambiar el color del objetos
meshList[5]->RenderMesh();

// Piramide derecha
shaderList[0].useShader();
uniformModel = shaderList[0].getModelLocation();
uniformProjection = shaderList[0].getProjectLocation();
uniformView = shaderList[0].getViewLocation();
uniformColor = shaderList[0].getColorLocation();
model = glm::mat4(1.0);
model = glm::translate(model, glm::vec3(-0.18f, 0.0f, -3.95f));
model = glm::scale(model, glm::vec3(0.25f, 0.25f, 0.25f));
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));
glUniformMatrix4fv(uniformView, 1, GL_FALSE, glm::value_ptr(camera.calculateViewMatrix()));
color = glm::vec3(0.658f, 0.837f, 0.95f);
glUniform3fv(uniformColor, 1, glm::value_ptr(color)); //para cambiar el color del objetos
meshList[5]->RenderMesh();

// Tercer nivel
// Piramide izquierda
shaderList[0].useShader();
uniformModel = shaderList[0].getModelLocation();
uniformProjection = shaderList[0].getProjectLocation();
uniformView = shaderList[0].getViewLocation();
uniformColor = shaderList[0].getColorLocation();
model = glm::mat4(1.0);
model = glm::translate(model, glm::vec3(-0.04f, -0.3f, -4.32f));
model = glm::scale(model, glm::vec3(0.25f, 0.25f, 0.25f));
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));
glUniformMatrix4fv(uniformView, 1, GL_FALSE, glm::value_ptr(camera.calculateViewMatrix()));
color = glm::vec3(0.658f, 0.837f, 0.95f);
glUniform3fv(uniformColor, 1, glm::value_ptr(color)); //para cambiar el color del objetos
meshList[5]->RenderMesh();
```

```
// Piramide invertida izquierda
shaderList[0].useShader();
uniformModel = shaderList[0].getModelLocation();
uniformProjection = shaderList[0].getProjectLocation();
uniformView = shaderList[0].getViewLocation();
uniformColor = shaderList[0].getColorLocation();
model = glm::mat4(1.0);
model = glm::translate(model, glm::vec3(-0.12f, -0.3f, -4.21f));
model = glm::rotate(model, 15 * toRadians, glm::vec3(1.0f, 0.0f, 0.0f)); // x
model = glm::rotate(model, 0 * toRadians, glm::vec3(0.0f, 1.0f, 0.0f));  // y
model = glm::rotate(model, 151 * toRadians, glm::vec3(0.0f, 0.0f, 1.0f)); // Z;
model = glm::scale(model, glm::vec3(0.25f, 0.25f, 0.25f));
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));
glUniformMatrix4fv(uniformView, 1, GL_FALSE, glm::value_ptr(camera.calculateViewMatrix()));
color = glm::vec3(0.658f, 0.837f, 0.95f);
glUniform3fv(uniformColor, 1, glm::value_ptr(color)); //para cambiar el color del objetos
meshList[5]->RenderMesh();

// Piramide medio
shaderList[0].useShader();
uniformModel = shaderList[0].getModelLocation();
uniformProjection = shaderList[0].getProjectLocation();
uniformView = shaderList[0].getViewLocation();
uniformColor = shaderList[0].getColorLocation();
model = glm::mat4(1.0);
model = glm::translate(model, glm::vec3(-0.2f, -0.3f, -4.1f));
model = glm::scale(model, glm::vec3(0.25f, 0.25f, 0.25f));
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));
glUniformMatrix4fv(uniformView, 1, GL_FALSE, glm::value_ptr(camera.calculateViewMatrix()));
color = glm::vec3(0.658f, 0.837f, 0.95f);
glUniform3fv(uniformColor, 1, glm::value_ptr(color)); //para cambiar el color del objetos
meshList[5]->RenderMesh();

// Piramide invertida derecha
shaderList[0].useShader();
uniformModel = shaderList[0].getModelLocation();
uniformProjection = shaderList[0].getProjectLocation();
uniformView = shaderList[0].getViewLocation();
uniformColor = shaderList[0].getColorLocation();
model = glm::mat4(1.0);
model = glm::translate(model, glm::vec3(-0.26f, -0.3f, -3.98f));
model = glm::rotate(model, 15 * toRadians, glm::vec3(1.0f, 0.0f, 0.0f)); // x
model = glm::rotate(model, 0 * toRadians, glm::vec3(0.0f, 1.0f, 0.0f));  // y
model = glm::rotate(model, 151 * toRadians, glm::vec3(0.0f, 0.0f, 1.0f)); // Z;
model = glm::scale(model, glm::vec3(0.25f, 0.25f, 0.25f));
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));
glUniformMatrix4fv(uniformView, 1, GL_FALSE, glm::value_ptr(camera.calculateViewMatrix()));
color = glm::vec3(0.658f, 0.837f, 0.95f);
glUniform3fv(uniformColor, 1, glm::value_ptr(color)); //para cambiar el color del objetos
meshList[5]->RenderMesh();
```

```
// Piramide derecha
shaderList[0].useShader();
uniformModel = shaderList[0].getModelLocation();
uniformProjection = shaderList[0].getProjectLocation();
uniformView = shaderList[0].getViewLocation();
uniformColor = shaderList[0].getColorLocation();
model = glm::mat4(1.0);
model = glm::translate(model, glm::vec3(-0.35f, -0.3f, -3.88f));
model = glm::scale(model, glm::vec3(0.25f, 0.25f, 0.25f));
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));
glUniformMatrix4fv(uniformView, 1, GL_FALSE, glm::value_ptr(camera.calculateViewMatrix()));
color = glm::vec3(0.658f, 0.837f, 0.95f);
glUniform3fv(uniformColor, 1, glm::value_ptr(color)); //para cambiar el color del objetos
meshList[5]->RenderMesh();


// Lado verde
// Piramide punta
shaderList[0].useShader();
uniformModel = shaderList[0].getModelLocation();
uniformProjection = shaderList[0].getProjectLocation();
uniformView = shaderList[0].getViewLocation();
uniformColor = shaderList[0].getColorLocation();
model = glm::mat4(1.0);
model = glm::translate(model, glm::vec3(0.03f, 0.3, -4.02f));
model = glm::scale(model, glm::vec3(0.25f, 0.25f, 0.25));
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));
glUniformMatrix4fv(uniformView, 1, GL_FALSE, glm::value_ptr(camera.calculateViewMatrix()));
color = glm::vec3(0.0f, 1.0f, 0.0f);
glUniform3fv(uniformColor, 1, glm::value_ptr(color)); //para cambiar el color del objetos
meshList[5]->RenderMesh();

// segundo nivel
// Piramide izquierda
shaderList[0].useShader();
uniformModel = shaderList[0].getModelLocation();
uniformProjection = shaderList[0].getProjectLocation();
uniformView = shaderList[0].getViewLocation();
uniformColor = shaderList[0].getColorLocation();
model = glm::mat4(1.0);
model = glm::translate(model, glm::vec3(0.03f, 0.0f, -4.16f));
model = glm::scale(model, glm::vec3(0.25f, 0.25f, 0.25f));
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));
glUniformMatrix4fv(uniformView, 1, GL_FALSE, glm::value_ptr(camera.calculateViewMatrix()));
color = glm::vec3(0.0f, 1.0f, 0.0f);
glUniform3fv(uniformColor, 1, glm::value_ptr(color)); //para cambiar el color del objetos
meshList[5]->RenderMesh();
```

```
// Piramide invertida medio
shaderList[0].useShader();
uniformModel = shaderList[0].getModelLocation();
uniformProjection = shaderList[0].getProjectLocation();
uniformView = shaderList[0].getViewLocation();
uniformColor = shaderList[0].getColorLocation();
model = glm::mat4(1.0);
model = glm::translate(model, glm::vec3(0.11f, 0.0f, -4.06f));
model = glm::rotate(model, 18 * toRadians, glm::vec3(1.0f, 0.0f, 0.0f)); // x
model = glm::rotate(model, 0 * toRadians, glm::vec3(0.0f, 1.0f, 0.0f));  // y
model = glm::rotate(model, -151 * toRadians, glm::vec3(0.0f, 0.0f, 1.0f)); // Z;
model = glm::scale(model, glm::vec3(0.25f, 0.25f, 0.25f));
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));
glUniformMatrix4fv(uniformView, 1, GL_FALSE, glm::value_ptr(camera.calculateViewMatrix()));
color = glm::vec3(0.0f, 1.0f, 0.0f);
glUniform3fv(uniformColor, 1, glm::value_ptr(color)); //para cambiar el color del objetos
meshList[5]->RenderMesh();

// Piramide derecha
shaderList[0].useShader();
uniformModel = shaderList[0].getModelLocation();
uniformProjection = shaderList[0].getProjectLocation();
uniformView = shaderList[0].getViewLocation();
uniformColor = shaderList[0].getColorLocation();
model = glm::mat4(1.0);
model = glm::translate(model, glm::vec3(0.18f, 0.0f, -3.95f));
model = glm::scale(model, glm::vec3(0.25f, 0.25f, 0.25f));
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));
glUniformMatrix4fv(uniformView, 1, GL_FALSE, glm::value_ptr(camera.calculateViewMatrix()));
color = glm::vec3(0.0f, 1.0f, 0.0f);
glUniform3fv(uniformColor, 1, glm::value_ptr(color)); //para cambiar el color del objetos
meshList[5]->RenderMesh();

// Tercer nivel
// Piramide izquierda
shaderList[0].useShader();
uniformModel = shaderList[0].getModelLocation();
uniformProjection = shaderList[0].getProjectLocation();
uniformView = shaderList[0].getViewLocation();
uniformColor = shaderList[0].getColorLocation();
model = glm::mat4(1.0);
model = glm::translate(model, glm::vec3(0.04f, -0.3f, -4.32f));
model = glm::scale(model, glm::vec3(0.25f, 0.25f, 0.25f));
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));
glUniformMatrix4fv(uniformView, 1, GL_FALSE, glm::value_ptr(camera.calculateViewMatrix()));
color = glm::vec3(0.0f, 1.0f, 0.0f);
glUniform3fv(uniformColor, 1, glm::value_ptr(color)); //para cambiar el color del objetos
meshList[5]->RenderMesh();
```

```
// Piramide invertida izquierda
shaderList[0].useShader();
uniformModel = shaderList[0].getModelLocation();
uniformProjection = shaderList[0].getProjectLocation();
uniformView = shaderList[0].getViewLocation();
uniformColor = shaderList[0].getColorLocation();
model = glm::mat4(1.0);
model = glm::translate(model, glm::vec3(0.12f, -0.3f, -4.21f));
model = glm::rotate(model, 15 * toRadians, glm::vec3(1.0f, 0.0f, 0.0f)); // x
model = glm::rotate(model, 0 * toRadians, glm::vec3(0.0f, 1.0f, 0.0f));  // y
model = glm::rotate(model, -151 * toRadians, glm::vec3(0.0f, 0.0f, 1.0f)); // Z;
model = glm::scale(model, glm::vec3(0.25f, 0.25f, 0.25f));
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));
glUniformMatrix4fv(uniformView, 1, GL_FALSE, glm::value_ptr(camera.calculateViewMatrix()));
color = glm::vec3(0.0f, 1.0f, 0.0f);
glUniform3fv(uniformColor, 1, glm::value_ptr(color)); //para cambiar el color del objetos
meshList[5]->RenderMesh();

// Piramide media
shaderList[0].useShader();
uniformModel = shaderList[0].getModelLocation();
uniformProjection = shaderList[0].getProjectLocation();
uniformView = shaderList[0].getViewLocation();
uniformColor = shaderList[0].getColorLocation();
model = glm::mat4(1.0);
model = glm::translate(model, glm::vec3(0.2f, -0.3f, -4.1f));
model = glm::scale(model, glm::vec3(0.25f, 0.25f, 0.25f));
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));
glUniformMatrix4fv(uniformView, 1, GL_FALSE, glm::value_ptr(camera.calculateViewMatrix()));
color = glm::vec3(0.0f, 1.0f, 0.0f);
glUniform3fv(uniformColor, 1, glm::value_ptr(color)); //para cambiar el color del objetos
meshList[5]->RenderMesh();

// Piramide invertida derechas
shaderList[0].useShader();
uniformModel = shaderList[0].getModelLocation();
uniformProjection = shaderList[0].getProjectLocation();
uniformView = shaderList[0].getViewLocation();
uniformColor = shaderList[0].getColorLocation();
model = glm::mat4(1.0);
model = glm::translate(model, glm::vec3(0.26f, -0.3f, -3.98f));
model = glm::rotate(model, 15 * toRadians, glm::vec3(1.0f, 0.0f, 0.0f)); // x
model = glm::rotate(model, 0 * toRadians, glm::vec3(0.0f, 1.0f, 0.0f));  // y
model = glm::rotate(model, -151 * toRadians, glm::vec3(0.0f, 0.0f, 1.0f)); // Z;
model = glm::scale(model, glm::vec3(0.25f, 0.25f, 0.25f));
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));
glUniformMatrix4fv(uniformView, 1, GL_FALSE, glm::value_ptr(camera.calculateViewMatrix()));
color = glm::vec3(0.0f, 1.0f, 0.0f);
glUniform3fv(uniformColor, 1, glm::value_ptr(color)); //para cambiar el color del objetos
meshList[5]->RenderMesh();
```

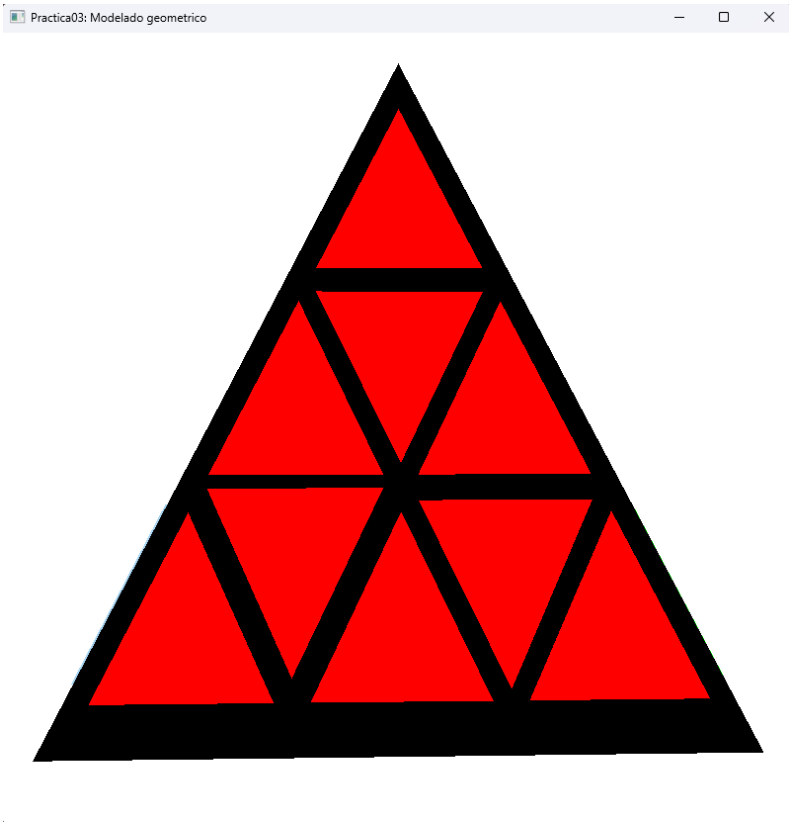
```
// Piramide derecha
shaderList[0].useShader();
uniformModel = shaderList[0].getModelLocation();
uniformProjection = shaderList[0].getProjectLocation();
uniformView = shaderList[0].getViewLocation();
uniformColor = shaderList[0].getColorLocation();
model = glm::mat4(1.0);
model = glm::translate(model, glm::vec3(0.35f, -0.3f, -3.88f));
model = glm::scale(model, glm::vec3(0.25f, 0.25f, 0.25f));
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));
glUniformMatrix4fv(uniformView, 1, GL_FALSE, glm::value_ptr(camera.calculateViewMatrix()));
color = glm::vec3(0.0f, 1.0f, 0.0f);
glUniform3fv(uniformColor, 1, glm::value_ptr(color)); //para cambiar el color del objetos
meshList[5]->RenderMesh();

// Cara Inferior
// Punta
shaderList[0].useShader();
uniformModel = shaderList[0].getModelLocation();
uniformProjection = shaderList[0].getProjectLocation();
uniformView = shaderList[0].getViewLocation();
uniformColor = shaderList[0].getColorLocation();
model = glm::mat4(1.0);
model = glm::translate(model, glm::vec3(0.0f, -0.38f, -4.32f));
model = glm::scale(model, glm::vec3(0.25f, 0.25f, 0.25f));
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));
glUniformMatrix4fv(uniformView, 1, GL_FALSE, glm::value_ptr(camera.calculateViewMatrix()));
color = glm::vec3(1.0f, 1.0f, 0.0f);
glUniform3fv(uniformColor, 1, glm::value_ptr(color)); //para cambiar el color del objetos
meshList[5]->RenderMesh();

//Segundo nivel
// Piramide izquierda
shaderList[0].useShader();
uniformModel = shaderList[0].getModelLocation();
uniformProjection = shaderList[0].getProjectLocation();
uniformView = shaderList[0].getViewLocation();
uniformColor = shaderList[0].getColorLocation();
model = glm::mat4(1.0);
model = glm::translate(model, glm::vec3(0.15f, -0.38f, -4.1f));
model = glm::scale(model, glm::vec3(0.25f, 0.25f, 0.25f));
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));
glUniformMatrix4fv(uniformView, 1, GL_FALSE, glm::value_ptr(camera.calculateViewMatrix()));
color = glm::vec3(1.0f, 1.0f, 0.0f);
glUniform3fv(uniformColor, 1, glm::value_ptr(color)); //para cambiar el color del objetos
meshList[5]->RenderMesh();
```

```
// Piramide invertida medio
shaderList[0].useShader();
uniformModel = shaderList[0].getModelLocation();
uniformProjection = shaderList[0].getProjectLocation();
uniformView = shaderList[0].getViewLocation();
uniformColor = shaderList[0].getColorLocation();
model = glm::mat4(1.0);
model = glm::translate(model, glm::vec3(0.0f, -0.38f, -4.17f));
model = glm::rotate(model, 0 * toRadians, glm::vec3(1.0f, 0.0f, 0.0f)); // x
model = glm::rotate(model, 180 * toRadians, glm::vec3(0.0f, 1.0f, 0.0f));  // y
model = glm::rotate(model, 0 * toRadians, glm::vec3(0.0f, 0.0f, 1.0f)); // Z;
model = glm::scale(model, glm::vec3(0.25f, 0.25f, 0.25f));
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));
glUniformMatrix4fv(uniformView, 1, GL_FALSE, glm::value_ptr(camera.calculateViewMatrix()));
color = glm::vec3(1.0f, 1.0f, 0.0f);
glUniform3fv(uniformColor, 1, glm::value_ptr(color)); //para cambiar el color del objetos
meshList[5]->RenderMesh();

// Piramide derecha
shaderList[0].useShader();
uniformModel = shaderList[0].getModelLocation();
uniformProjection = shaderList[0].getProjectLocation();
uniformView = shaderList[0].getViewLocation();
uniformColor = shaderList[0].getColorLocation();
model = glm::mat4(1.0);
model = glm::translate(model, glm::vec3(-0.15f, -0.38f, -4.1f));
model = glm::scale(model, glm::vec3(0.25f, 0.25f, 0.25f));
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));
glUniformMatrix4fv(uniformView, 1, GL_FALSE, glm::value_ptr(camera.calculateViewMatrix()));
color = glm::vec3(1.0f, 1.0f, 0.0f);
glUniform3fv(uniformColor, 1, glm::value_ptr(color)); //para cambiar el color del objetos
meshList[5]->RenderMesh();

// tercer nivel
// Piramide izquierda
shaderList[0].useShader();
uniformModel = shaderList[0].getModelLocation();
uniformProjection = shaderList[0].getProjectLocation();
uniformView = shaderList[0].getViewLocation();
uniformColor = shaderList[0].getColorLocation();
model = glm::mat4(1.0);
model = glm::translate(model, glm::vec3(0.30f, -0.38f, -3.88f));
model = glm::scale(model, glm::vec3(0.25f, 0.25f, 0.25f));
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));
glUniformMatrix4fv(uniformView, 1, GL_FALSE, glm::value_ptr(camera.calculateViewMatrix()));
color = glm::vec3(1.0f, 1.0f, 0.0f);
glUniform3fv(uniformColor, 1, glm::value_ptr(color)); //para cambiar el color del objetos
meshList[5]->RenderMesh();
```

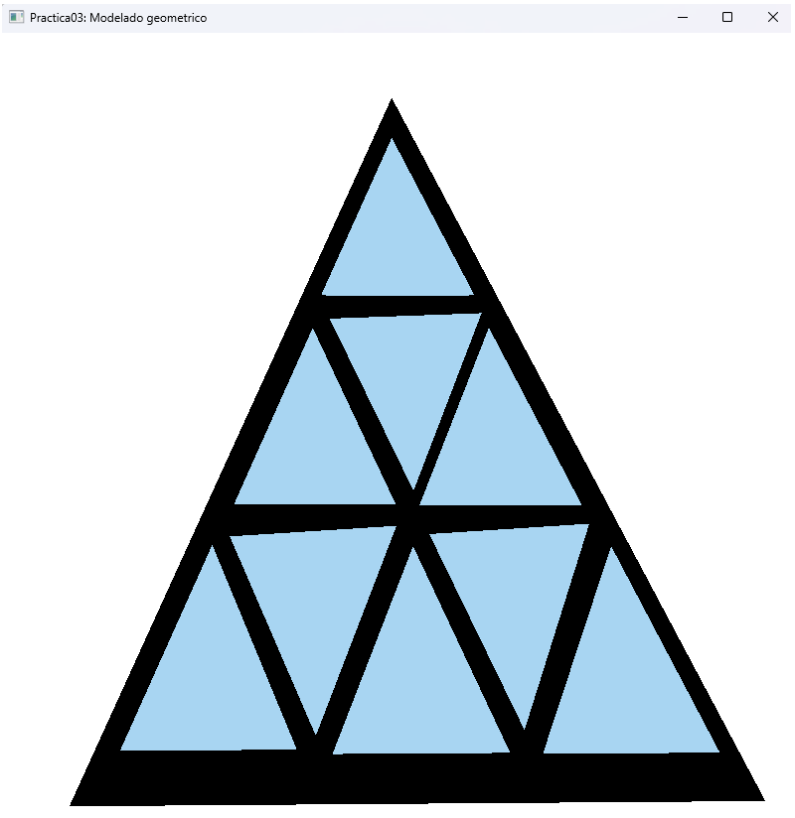
```
// Piramide derecha
shaderList[0].useShader();
uniformModel = shaderList[0].getModelLocation();
uniformProjection = shaderList[0].getProjectLocation();
uniformView = shaderList[0].getViewLocation();
uniformColor = shaderList[0].getColorLocation();
model = glm::mat4(1.0);
model = glm::translate(model, glm::vec3(-0.30f, -0.38f, -3.88f));
model = glm::scale(model, glm::vec3(0.25f, 0.25f, 0.25f));
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));
glUniformMatrix4fv(uniformView, 1, GL_FALSE, glm::value_ptr(camera.calculateViewMatrix()));
color = glm::vec3(1.0f, 1.0f, 0.0f);
glUniform3fv(uniformColor, 1, glm::value_ptr(color)); //para cambiar el color del objetos
meshList[5]->RenderMesh();

// Piramide medio base
shaderList[0].useShader();
uniformModel = shaderList[0].getModelLocation();
uniformProjection = shaderList[0].getProjectLocation();
uniformView = shaderList[0].getViewLocation();
uniformColor = shaderList[0].getColorLocation();
model = glm::mat4(1.0);
model = glm::translate(model, glm::vec3(0.0f, -0.38f, -3.88f));
model = glm::scale(model, glm::vec3(0.25f, 0.25f, 0.25f));
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));
glUniformMatrix4fv(uniformView, 1, GL_FALSE, glm::value_ptr(camera.calculateViewMatrix()));
color = glm::vec3(1.0f, 1.0f, 0.0f);
glUniform3fv(uniformColor, 1, glm::value_ptr(color)); //para cambiar el color del objetos
meshList[5]->RenderMesh();

// Piramide invertida derecha
shaderList[0].useShader();
uniformModel = shaderList[0].getModelLocation();
uniformProjection = shaderList[0].getProjectLocation();
uniformView = shaderList[0].getViewLocation();
uniformColor = shaderList[0].getColorLocation();
model = glm::mat4(1.0);
model = glm::translate(model, glm::vec3(-0.15f, -0.38f, -3.95f));
model = glm::rotate(model, 0 * toRadians, glm::vec3(1.0f, 0.0f, 0.0f)); // x
model = glm::rotate(model, 180 * toRadians, glm::vec3(0.0f, 1.0f, 0.0f));  // y
model = glm::rotate(model, 0 * toRadians, glm::vec3(0.0f, 0.0f, 1.0f)); // Z;
model = glm::scale(model, glm::vec3(0.25f, 0.25f, 0.25f));
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));
glUniformMatrix4fv(uniformView, 1, GL_FALSE, glm::value_ptr(camera.calculateViewMatrix()));
color = glm::vec3(1.0f, 1.0f, 0.0f);
glUniform3fv(uniformColor, 1, glm::value_ptr(color)); //para cambiar el color del objetos
meshList[5]->RenderMesh();
```

```
// Piramide invertida izquierda
shaderList[0].useShader();
uniformModel = shaderList[0].getModelLocation();
uniformProjection = shaderList[0].getProjectLocation();
uniformView = shaderList[0].getViewLocation();
uniformColor = shaderList[0].getColorLocation();
model = glm::mat4(1.0);
model = glm::translate(model, glm::vec3(0.15f, -0.38f, -3.95f));
model = glm::rotate(model, 0 * toRadians, glm::vec3(1.0f, 0.0f, 0.0f)); // x
model = glm::rotate(model, 180 * toRadians, glm::vec3(0.0f, 1.0f, 0.0f));  // y
model = glm::rotate(model, 0 * toRadians, glm::vec3(0.0f, 0.0f, 1.0f)); // Z;
model = glm::scale(model, glm::vec3(0.25f, 0.25f, 0.25f));
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));
glUniformMatrix4fv(uniformView, 1, GL_FALSE, glm::value_ptr(camera.calculateViewMatrix()));
color = glm::vec3(1.0f, 1.0f, 0.0f);
glUniform3fv(uniformColor, 1, glm::value_ptr(color)); //para cambiar el color del objetos
meshList[5]->RenderMesh();
```

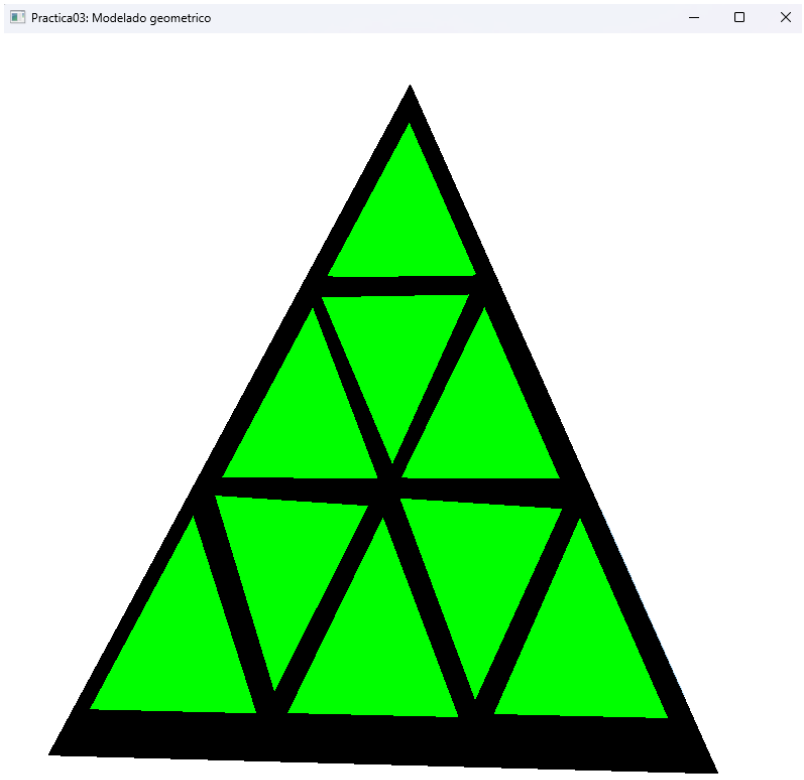
Ejecución
Cara frontal



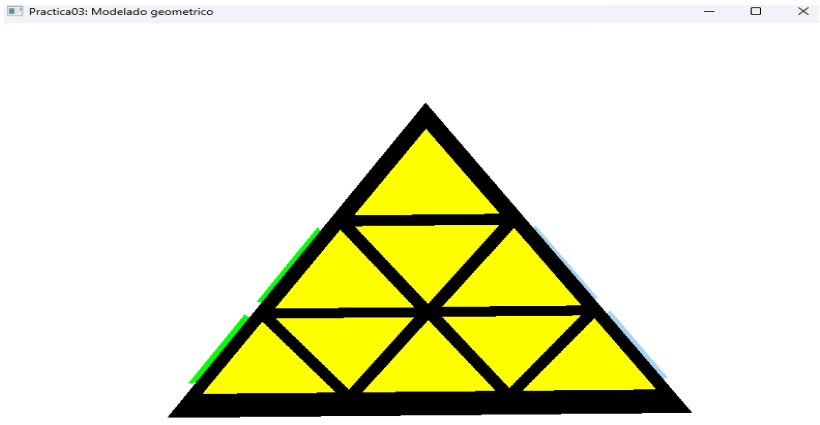
Cara lateral izquierda



Cara lateral derecha



Cara inferior

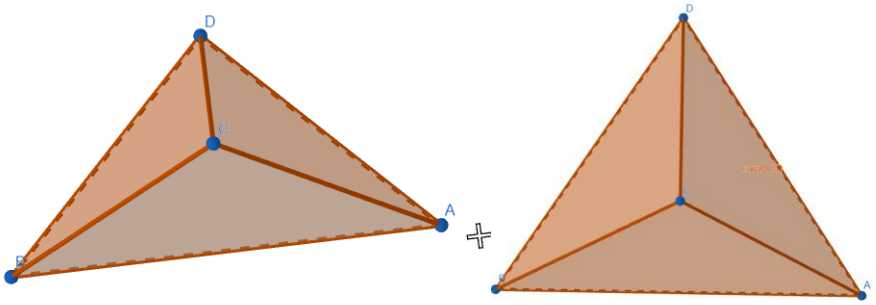


Video:

https://youtu.be/B4ipOby5dUU

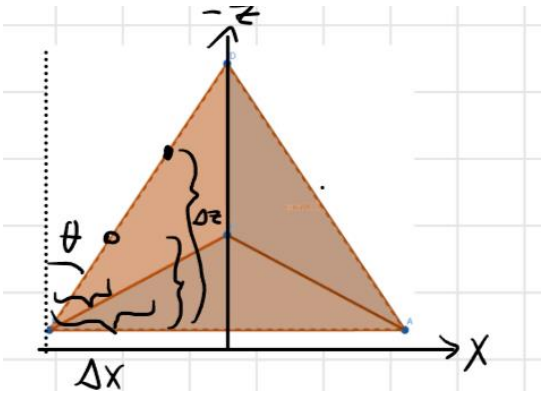
Problemas presentados

El primer problema fue que la pirámide que se daba en la implementación del laboratorio no era la adecuada para realizar la figura, ya que era una pirámide con base de un triángulo rectángulo, y se necesitaba un isósceles por lo que tuvimos ese problema, para solucionarlo, tuvimos que modificar esta geometría.



Pirámide de implementación Pirámide adaptada

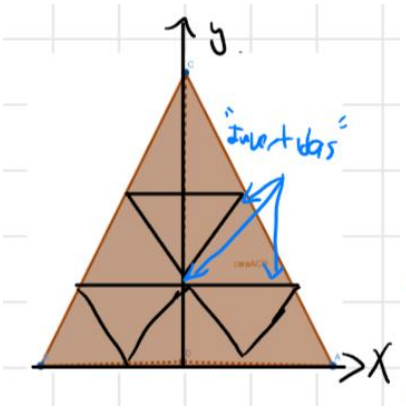
El 2do problema fue calcular la posición de las caras laterales, ya que, como se ve en el diagrama de abajo, para calcular la posición en X de la cara frontal, es sencillo, pues solo se calcula una vez, y para las demás pirámides es la misma, sin embargo, para las caras laterales ya hay un ángulo implícito, por lo que es un poco más complicado calcularlos, agregando que para todas las caras, menos la inferior, también hay cambio en Y, pero no es tan complejo esto, ya que para todas las caras cambia lo mismo, para esto lo que se hizo fue solo calcular las de enfrente y una cara y para la otra cara era hacer una especie de simetría con respecto al eje Z.



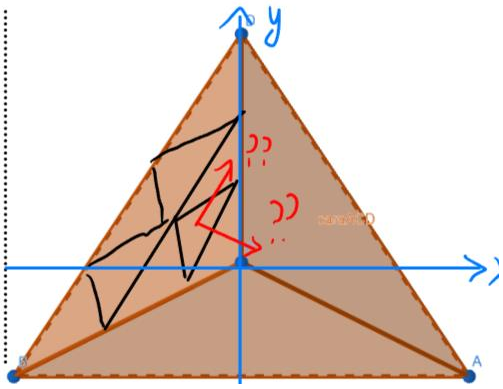
Cambio en x, z

El tercer problema surge también con respecto a las pirámides “invertidas” de la figura, es ya que, cuanto se rota la pirámide, la punta, queda abajo, y el ángulo que antes cerraba hacia arriba, ahora cierra hacia abajo, haciendo que lo que sobresalga, sea la base y lo que quede dentro de la figura, sea la punta. Para la cara frontal, este problema se resuelve fácil, ya que solo hay que rotar al rededor del eje x, para que tenga el mismo ángulo que la pirámide, sin embargo, para las caras laterales se complica, ya que al no seguir una trayectoria “recta” como se expone en el caso

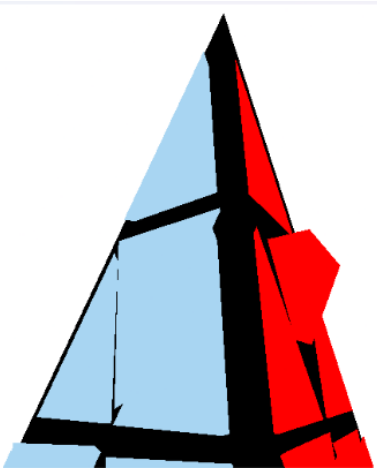
anterior también, ya que lo que se realiza para invertir la pirámide en la cara frontal, fue “rotar” al rededor del eje x y z para adaptar la orientación y el ángulo, sin embargo, cuando nos movemos a una cara lateral, ya los ejes de rotación, no están ni de forma perpendicular, ni paralela, cómo en el caso de la cara frontal, si no que están diagonalmente, por lo que realizar el giro para ajustar el ángulo de inclinación se vuelve una tarea muy complicada, para resolver esto lo que se hizo fue ajustar primero la cara para “invertirla” y después ir rotando poco a poco el ángulo para que quedara alineada con la pirámide grande e ir corrigiendo el error de forma manual.
El tercer y último problema que se tuvo fue la replicación de caras, ya que lo que se planeaba en el inicio, era realizar solo 1 cara, y después rotar sus componentes para cada lado, sin embargo, no se consiguió, lo que se intentó para hacer esto, fue usar los parámetros TRS(Translation, rotation and scaling), cómo RTS, para así una vez que teníamos generada la figura de forma correcta rotar con respecto al origen 120 grados, para que los componentes de la “cara” que ya estaba hecha, estuvieran ahora “en frente”, y una vez conseguido esto, trasladar el elemento, de nuevo a la posición donde debía estar , sin embargo, hacer eso, no resulto de la manera correcta, ya que no nos generaba el resultado deseado, por lo que la solución que se siguió fue la de calcular manualmente con las visualizaciones, las rotaciones de ángulo que había que realizar.

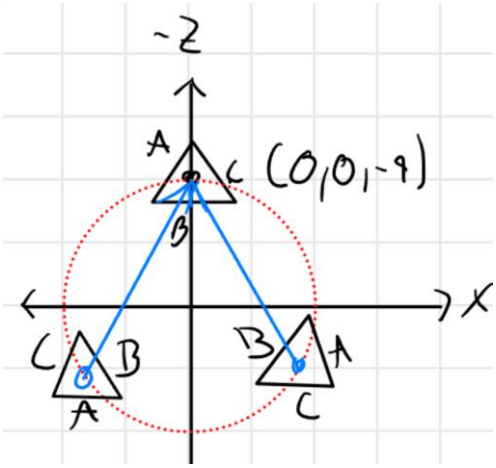


Pirámides invertidas



Problema al rotar e invertir


Problema de forma visual



Rotación a partir del origen (RTS)

Conclusión:

La práctica fue buena, me parece que fue bastante desafiante, por lo que implicó mucho tiempo trabajando en ella, lo cual ayudó mucho a afinar las habilidades sobre la traslación, rotación y escalamiento de las figuras, así como las geometrías y el cambio de color, por lo que considero que es una en las cuales se comprendió mucho más el espacio 3D y cómo funciona, además de las implicaciones que se tienen cuando trabajamos con estos objetos.

En cuanto a la complejidad, me parece que fue una práctica bastante más complicada con respecto a las anteriores, y tiene sentido, ya que la variación de ángulos que había por ejemplo en las alturas de cada "nivel" de la pirámide o los ángulos que existían en las caras laterales, hacían que la tarea de rotación fuera algo común, agregando a esto, que por ejemplo en las caras laterales, estaban en diagonal con respecto a los ejes de rotación, hacían muy poco predecible saber lo que pasaría al rotar las figuras, por lo que al menos a mí, este aspecto se me complicó mucho, la parte de calcular las posiciones de traslación, no fue tan complejo. En general, la dificultad si la sentí algo más elevada, y que al menos de la manera en que la resolví, que fue manualmente calculando cada mini pirámide, fue muy tardada.

La única recomendación que puedo dar para la práctica sería dar más teoría o herramientas para replicar y trasladar las caras, ya que me imagino que debe haber formas de hacerlo, sin embargo, no se han visto aun en laboratorio, por lo que tampoco sabía si era válido aplicarlo, así que lo hice todo con cada mini pirámide, y también me imagino que hay alguna forma de realizar la rotación de las caras para posicionarlas más sencillo, así que hubiera sido de gran ayuda ver esa parte de teoría.

Bibliografía

- Rodo P. (diciembre, 2021). "Traslación de vectores". Recuperado el 5 de septiembre de 2025 de: https://economipedia.com/definiciones/matematicas/traslacion-de-vectores.html