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: Carandia Lorenzo Brenda Fernanda

Nº de Cuenta: 319018961

GRUPO DE LABORATORIO: 03

GRUPO DE TEORÍA: 05

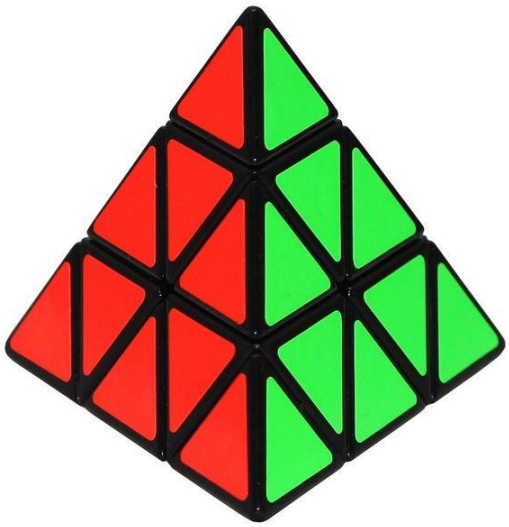
SEMESTRE 2025-2

FECHA DE ENTREGA LÍMITE: 1 de marzo de 2024

CALIFICACIÓN: _____

EJERCICIO

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



Bloque de código

```
void CrearPiramidesTriangulares() {
    unsigned int indices[] = {
        0, 1, 2,   // Cara magenta
        3, 4, 5,   // Cara verde
        6, 7, 8,   // Cara amarilla
        9, 10, 11 // Cara cian
    };

    // Vértices para una pirámide normal (colores variados)
    GLfloat vertices[] = {
        // Cara magenta
       -0.5f, -0.5f, 0.0f,   1.0f, 0.0f, 1.0f,  // Vértice 0 (Magenta)
        0.5f, -0.5f, 0.0f,   1.0f, 0.0f, 1.0f,  // Vértice 1 (Magenta)
        0.0f,  0.5f, -0.25f, 1.0f, 0.0f, 1.0f,  // Vértice 2 (Magenta)

        // Cara verde
        0.5f, -0.5f, 0.0f,   0.0f, 1.0f, 0.0f,  // Vértice 3 (Verde)
        0.0f, -0.5f, -0.5f,   0.0f, 1.0f, 0.0f,  // Vértice 4 (Verde)
        0.0f,  0.5f, -0.25f,  0.0f, 1.0f, 0.0f,  // Vértice 5 (Verde)

        // Cara amarilla
         0.0f, -0.5f, -0.5f,   1.0f, 1.0f, 0.0f,  // Vértice 6 (Amarillo)
        -0.5f, -0.5f, 0.0f,    1.0f, 1.0f, 0.0f,  // Vértice 7 (Amarillo)
         0.0f,  0.5f, -0.25f,  1.0f, 1.0f, 0.0f,  // Vértice 8 (Amarillo)

         // Cara cian
        -0.5f, -0.5f, 0.0f,   0.0f, 1.0f, 1.0f,  // Vértice 9 (Cian)
         0.5f, -0.5f, 0.0f,    0.0f, 1.0f, 1.0f,  // Vértice 10 (Cian)
         0.0f, -0.5f, -0.5f,   0.0f, 1.0f, 1.0f   // Vértice 11 (Cian)
    };
```

```
// Vértices para una pirámide negra
GLfloat verticesNegra[] = {
    // Cara 1 (Negra)
    -0.5f, -0.5f, 0.0f,   0.0f, 0.0f, 0.0f,  // Vértice 0 (Negro)
     0.5f, -0.5f, 0.0f,   0.0f, 0.0f, 0.0f,  // Vértice 1 (Negro)
     0.0f,  0.5f, -0.25f, 0.0f, 0.0f, 0.0f,  // Vértice 2 (Negro)

     // Cara 2 (Negra)
      0.5f, -0.5f, 0.0f,   0.0f, 0.0f, 0.0f,  // Vértice 3 (Negro)
      0.0f, -0.5f, -0.5f,   0.0f, 0.0f, 0.0f,  // Vértice 4 (Negro)
      0.0f,  0.5f, -0.25f,  0.0f, 0.0f, 0.0f,  // Vértice 5 (Negro)

      // Cara 3 (Negra)
       0.0f, -0.5f, -0.5f,   0.0f, 0.0f, 0.0f,  // Vértice 6 (Negro)
      -0.5f, -0.5f, 0.0f,    0.0f, 0.0f, 0.0f,  // Vértice 7 (Negro)
       0.0f,  0.5f, -0.25f,  0.0f, 0.0f, 0.0f,  // Vértice 8 (Negro)

       // Cara 4 (Negra)
       -0.5f, -0.5f, 0.0f,   0.0f, 0.0f, 0.0f,  // Vértice 9 (Negro)
        0.5f, -0.5f, 0.0f,    0.0f, 0.0f, 0.0f,  // Vértice 10 (Negro)
        0.0f, -0.5f, -0.5f,   0.0f, 0.0f, 0.0f   // Vértice 11 (Negro)
};
```

```
for (int i = 0; i < 15; i++) {
    Mesh* obj = new Mesh();
    obj->CreateMesh(vertices, indices, 72, 12); // 72 vértices (12 vértices * 6 componentes), 12 índices
    meshList.push_back(obj);
}

// Crear 1 pirámide negra
Mesh* piramideNegra = new Mesh();
piramideNegra->CreateMesh(verticesNegra, indices, 72, 12);
meshList.push_back(piramideNegra);
```

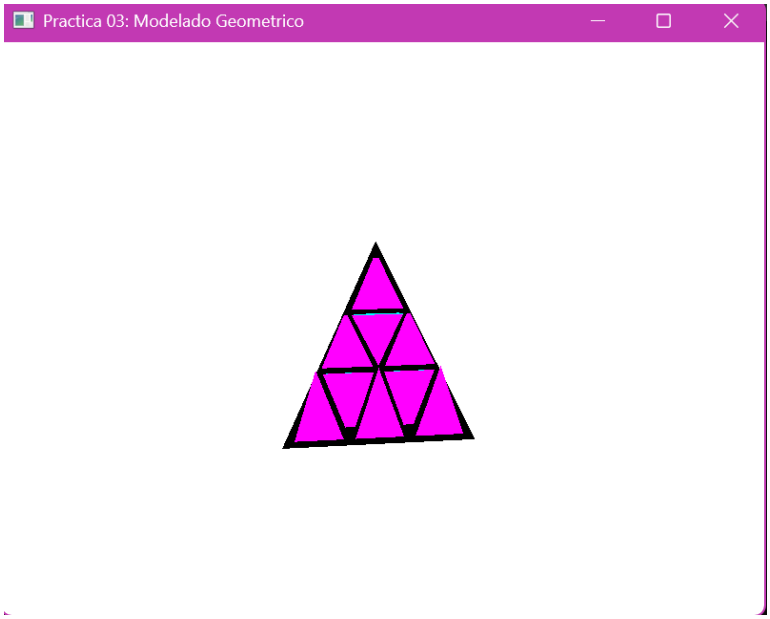
En esta parte del código declaramos vértices que reciban el color de la cara de la pirámide, por eso se usan vértices duplicados. Para que cada cara de la pirámide tenga su color uniforme, además agregamos dentro de la función un arreglo para la pirámide negra, la cual servirá para los vértices; y se los pasamos a meshList, los índices cambiados y el número total de vértices.

```
 // Pirámide grande (negra)
modelNegra = glm::mat4(1.0);
modelNegra = glm::translate(modelNegra, glm::vec3(0.0f, -0.5f, 0.0f)); // Centrada y abajo
modelNegra = glm::scale(modelNegra, glm::vec3(4.0f, 4.0f, 4.0f)); // Escalada
glUniformMatrix4fv(uniformModel, 1, GL_FALSE, glm::value_ptr(modelNegra));
glUniformMatrix4fv(uniformProjection, 1, GL_FALSE, glm::value_ptr(projection));
glUniformMatrix4fv(uniformView, 1, GL_FALSE, glm::value_ptr(camera.calculateViewMatrix()));
meshList[15]->RenderMesh(); // Dibuja la pirámide grande

// Ajustes para las pirámides pequeñas (magenta)
float baseY = -1.6f; // Posición Y base de las pirámides pequeñas
float segundoNivelY = -0.3f; // Posición Y del segundo nivel
float puntaY = 0.8f; // Posición Y de la pirámide en la punta
float escalaBase = 1.5f; // Escala base para las pirámides pequeñas
float escalaSegundoNivel = 1.0f; // Escala para el segundo nivel
float escalaPunta = 1.0f; // Escala para la pirámide en la punta
```

```
// Pirámides en la base (5 pirámides)
float offsetsXBase[] = { -1.24f, -0.6f, 0.0f, 0.6f, 1.24f }; // Posiciones en X
for (int i = 0; i < 5; i++) {
    model = glm::mat4(1.0);
    model = glm::translate(model, glm::vec3(offsetsXBase[i], baseY, -0.01f)); // Posición en la base
    model = glm::scale(model, glm::vec3(1.0f, escalaBase, 1.0f)); // Escala

    // Ajustar rotación para las pirámides 2 y 4 de la base
    if (i == 1 || i == 3) { // Pirámides 2 y 4 (índices 1 y 3)
        model = glm::rotate(model, -15 * toRadians, glm::vec3(1.0f, 0.0f, 0.0f)); // Rotación en X
        model = glm::rotate(model, glm::radians(180.0f), glm::vec3(0.0f, 0.0f, 1.0f)); // Rotación en Z
        model = glm::rotate(model, 5 * toRadians, glm::vec3(1.0f, 0.0f, 0.0f)); // Rotación en X
    }

    glUniformMatrix4fv(uniformModel, 1, GL_FALSE, glm::value_ptr(model));
    glUniformMatrix4fv(uniformView, 1, GL_FALSE, glm::value_ptr(camera.calculateViewMatrix()));
    meshList[0]->RenderMesh(); // Dibuja la pirámide pequeña
}

// Pirámides en el segundo nivel (3 pirámides)
float offsetsXSegundoNivel[] = { -0.62f, 0.0f, 0.62f }; // Posiciones en X
for (int i = 0; i < 3; i++) {
    model = glm::mat4(1.0);
    model = glm::translate(model, glm::vec3(offsetsXSegundoNivel[i], segundoNivelY, -0.24f)); // Posición en el segundo nivel
    model = glm::scale(model, glm::vec3(1.0f, escalaSegundoNivel, 1.8f)); // Escala
    //model = glm::rotate(model, 2 * toRadians, glm::vec3(0.0f, 1.0f, 0.0f)); // Rotación en X
    //model = glm::rotate(model, -3 * toRadians, glm::vec3(1.0f, 0.0f, 0.0f)); // Rotación en X

    // Ajustar rotación para la pirámide 2 del segundo nivel
    if (i == 1) { // Pirámide 2 (índice 1)
        model = glm::rotate(model, -15 * toRadians, glm::vec3(1.0f, 0.0f, 0.0f)); // Rotación en X
        model = glm::rotate(model, glm::radians(180.0f), glm::vec3(0.0f, 0.0f, 1.0f)); // Rotación en Z
        model = glm::rotate(model, 5 * toRadians, glm::vec3(1.0f, 0.0f, 0.0f)); // Rotación en X
    }

    glUniformMatrix4fv(uniformModel, 1, GL_FALSE, glm::value_ptr(model));
    glUniformMatrix4fv(uniformView, 1, GL_FALSE, glm::value_ptr(camera.calculateViewMatrix()));
    meshList[0]->RenderMesh(); // Dibuja la pirámide pequeña
}


    // Pirámide en la punta (1 pirámide)
    model = glm::mat4(1.0);
    model = glm::translate(model, glm::vec3(0.0f, puntaY, -0.52f)); // Posición en la punta

    model = glm::scale(model, glm::vec3(1.0f, escalaPunta, 1.8f)); // Escala

    //model = glm::rotate(model, 5 * toRadians, glm::vec3(1.0f, 0.0f, 0.0f)); // Rotación en X
    glUniformMatrix4fv(uniformModel, 1, GL_FALSE, glm::value_ptr(model));
    glUniformMatrix4fv(uniformView, 1, GL_FALSE, glm::value_ptr(camera.calculateViewMatrix()));
    meshList[0]->RenderMesh(); // Dibuja la pirámide en la punta

    // Pirame de atras verde y amarillo segundo
    model = glm::mat4(1.0);
    model = glm::translate(model, glm::vec3(0.0f, segundoNivelY, -0.80f)); // Posición en la punta
    model = glm::scale(model, glm::vec3(1.0f, escalaSegundoNivel, 1.8f)); // Escala
    model = glm::rotate(model, 5 * toRadians, glm::vec3(1.0f, 0.0f, 0.0f)); // Rotación en X

    glUniformMatrix4fv(uniformModel, 1, GL_FALSE, glm::value_ptr(model));
    glUniformMatrix4fv(uniformView, 1, GL_FALSE, glm::value_ptr(camera.calculateViewMatrix()));
    meshList[0]->RenderMesh(); // Dibuja la pirámide en la punta con el color correspondiente

    // Pirame de atras verde y amarillo tercer
    model = glm::mat4(1.0);
    model = glm::translate(model, glm::vec3(0.0f, -1.6, -0.80f)); // Posición en la punta
    model = glm::scale(model, glm::vec3(1.0f, 1.5, 1.8f)); // Escala
    model = glm::rotate(model, 5 * toRadians, glm::vec3(1.0f, 0.0f, 0.0f)); // Rotación en X

    glUniformMatrix4fv(uniformModel, 1, GL_FALSE, glm::value_ptr(model));
    glUniformMatrix4fv(uniformView, 1, GL_FALSE, glm::value_ptr(camera.calculateViewMatrix()));
    meshList[0]->RenderMesh(); // Dibuja la pirámide en la punta con el color correspondiente

        glUseProgram(0);
        mainWindow.swapBuffers();
    }
    return 0;
}
```

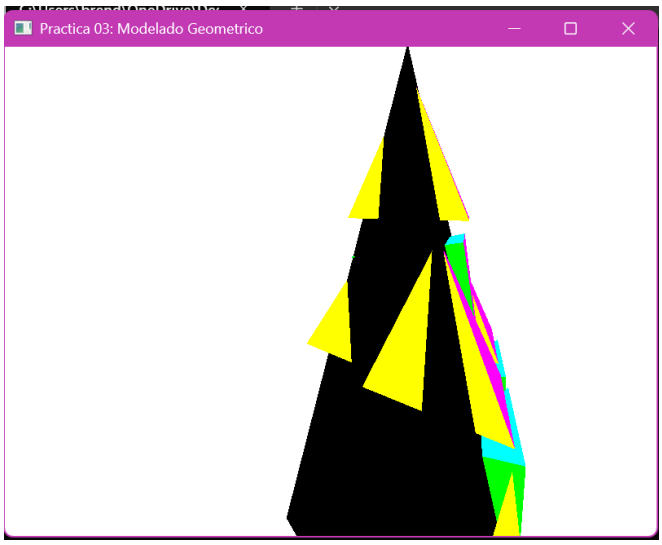
En el main realizamos las transformaciones necesarias para la cara de magenta-

Ejecución del programa



2. Problemas presentados

En este ejercicio tuve muchos problemas para lograr acomodar las pirámides y que tuvieran su borde, puesto que si realizaba transformaciones, o quedaba muy inclinada la pirámide o muy en fondo, chueca, por mas que intentaba cambiar la posición muchas veces no lo logre, puesto que las pirámides de las esquina eran parte fundamental de la estructura, no pude lograr que se observan ambos lados, ya que solamente se podía apreciar una. Tipo así



3. Conclusión

Sin duda este ejercicio, fue uno de los más complicados para mí, pues debía estar jugando con las rotaciones de las pirámides, ya que debían estar alineadas para que una pirámide pudiera compartir sus caras, que estuvieran aproximadamente a 45 grados. No fue complicado el tema de instanciar

las pirámides, si no el lugar en donde debían estar, y la cámara juega un papel fundamental para la perspectiva. Fue un ejercicio que no pude completar, pero espero aprender de esto y mejorar,

Bibliografía

- *Visualizacion de geometria*. (s. f.). https://cphoto.uji.es/grafica/_p2/
- *Tutorial 8 : Shading básico*. (s. f.). https://www.opengl-tutorial.org/es/beginners-tutorials/tutorial-8-basic-shading/