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 3

NOMBRE COMPLETO: Gómez Enríquez Agustín

Nº de Cuenta: 317031405

GRUPO DE LABORATORIO: 3

GRUPO DE TEORÍA: 5

SEMESTRE 2026-1

FECHA DE ENTREGA LÍMITE: 07 de septiembre del 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.

```
31      // Colores
32      const glm::vec3 COLOR_BLACK(0.00f, 0.00f, 0.00f);
33      const glm::vec3 COLOR_BLUE(0.10f, 0.40f, 0.95f);
34      const glm::vec3 COLOR_PINK(0.95f, 0.30f, 0.65f);
35      const glm::vec3 COLOR_GREEN(0.15f, 0.80f, 0.35f);
36      const glm::vec3 COLOR_YELLOW(0.98f, 0.85f, 0.15f);
37
38      // Borde negro interno en triángulo ABC
39      static void pushTriInset(vector<GLfloat>& V, vector<unsigned int>& I,
40          const glm::vec3& a0, const glm::vec3& b0, const glm::vec3& c0,
41          float inset)
42      {
43          glm::vec3 centroid = (a0 + b0 + c0) / 3.0f;
44          glm::vec3 a = glm::mix(a0, centroid, inset);
45          glm::vec3 b = glm::mix(b0, centroid, inset);
46          glm::vec3 c = glm::mix(c0, centroid, inset);
47
48          unsigned int base = static_cast<unsigned int>(V.size() / 3);
49          V.insert(V.end(), { a.x,a.y,a.z, b.x,b.y,b.z, c.x,c.y,c.z });
50          I.insert(I.end(), { base, base + 1, base + 2 });
51      }
```

Después de definir los colores que vamos a usar para nuestro Pyraminx que en este caso fueron azul, rosa, verde y amarillo. Vamos a trabajar con helpers para definir la geometría de nuestros triángulos. "pushTriInset": recibe un triángulo ABC y crea un triángulo ligeramente dentro de la base, así podemos generar un pequeño borde negro entre piezas al dibujar las caras sobre la base negra. Con “centroid” establecemos el centro de la cara.

```cpp
// Cara subdividida
static Mesh* buildSubdividedFace(const glm::vec3& A, const glm::vec3& B, const glm::vec3& C,
    int N, float inset)
{
    vector<GLfloat> V; V.reserve(N * N * 3 * 3 * 2);
    vector<unsigned int> I; I.reserve(N * N * 3 * 2);

    for (int i = 0; i < N; ++i)
    {
        for (int j = 0; j < N - i; ++j)
        {
            // tri "upright"
            glm::vec3 p0 = gridPoint(A, B, C, i, j, N);
            glm::vec3 p1 = gridPoint(A, B, C, i + 1, j, N);
            glm::vec3 p2 = gridPoint(A, B, C, i, j + 1, N);
            pushTriInset(V, I, p0, p1, p2, inset);

            // tri "inverted"
            if (i + j < N - 1)
            {
                glm::vec3 p3 = gridPoint(A, B, C, i + 1, j, N);
                glm::vec3 p4 = gridPoint(A, B, C, i + 1, j + 1, N);
                glm::vec3 p5 = gridPoint(A, B, C, i, j + 1, N);
                pushTriInset(V, I, p3, p4, p5, inset);
            }
        }
```

Con “gridPoint" establecemos el punto dentro del triángulo ABC, (sirve para construir una rejilla de subdivisión de orden N en cada cara). Para las caras usamos: "buildSubdividedFace" que nos ayuda a genera una cara triangular subdividida en N^2 triángulos. Por cada celda de la rejilla se generan 2 triángulos (upright e inverted) y a cada uno se le aplica "inset".

```cpp
// Tetraedro base (sin subdividir)
static Mesh* buildBaseTetra(const glm::vec3 v[4])
{
    vector<GLfloat> V = {
        v[0].x, v[0].y, v[0].z,
        v[1].x, v[1].y, v[1].z,
        v[2].x, v[2].y, v[2].z,
        v[3].x, v[3].y, v[3].z
    };
    vector<unsigned int> I = {
        0,1,2,
        0,3,1,
        0,2,3,
        1,3,2
    };

    Mesh* m = new Mesh();
    m->CreateMesh(V.data(), I.data(),
        static_cast<unsigned int>(V.size()),
        static_cast<unsigned int>(I.size()));
    return m;
}
```

Ahora vamos a construir la base principal de la pirámide que además de ser la pirámide base, también nos ayuda a delimitar bordes para los triángulos mas pequeños. Con "buildBaseTetra" el tetraedro base (sin subdividir) que se dibuja en negro hace que esta malla sea la que queda por detrás y del efecto de borde.

```cpp
// Crea: 0) base negra, 4) caras subdivididas
static void CreatePyraminxMeshes(int N, float inset)
{
    // Vértices de un tetra centrado
    glm::vec3 v[4] = {
        glm::vec3(1,  1,  1),
        glm::vec3(-1, -1,  1),
        glm::vec3(-1,  1, -1),
        glm::vec3(1, -1, -1),
    };
    for (int i = 0; i < 4; ++i) v[i] = glm::normalize(v[i]);

    // 0: base negra
    meshList.push_back(buildBaseTetra(v));

    // Caras subdivididas
    meshList.push_back(buildSubdividedFace(v[0], v[1], v[2], N, inset)); // cara 1
    meshList.push_back(buildSubdividedFace(v[0], v[3], v[1], N, inset)); // cara 2
    meshList.push_back(buildSubdividedFace(v[0], v[2], v[3], N, inset)); // cara 3
    meshList.push_back(buildSubdividedFace(v[1], v[3], v[2], N, inset)); // cara 4
}
```

Para crear las imágenes de cada cara hacemos que "CreatePyraminxMeshes" cree las 5 mallas que se usan en el render, [0] base negra (tetra sin subdividir) y [1..4] una malla por cada cara subdividida.

```cpp
    // Geometría: orden 3, inset = grosor de borde
    const int   ORDER = 3;
    const float INSET = 0.085f;
    CreatePyraminxMeshes(ORDER, INSET);

    // Uniforms (matrices)
    GLuint uModel = 0, uProj = 0, uView = 0;
```

Dentro del main vamos a trabajar la geometría de las figuras, estableciendo el grosor de color negro para la base.

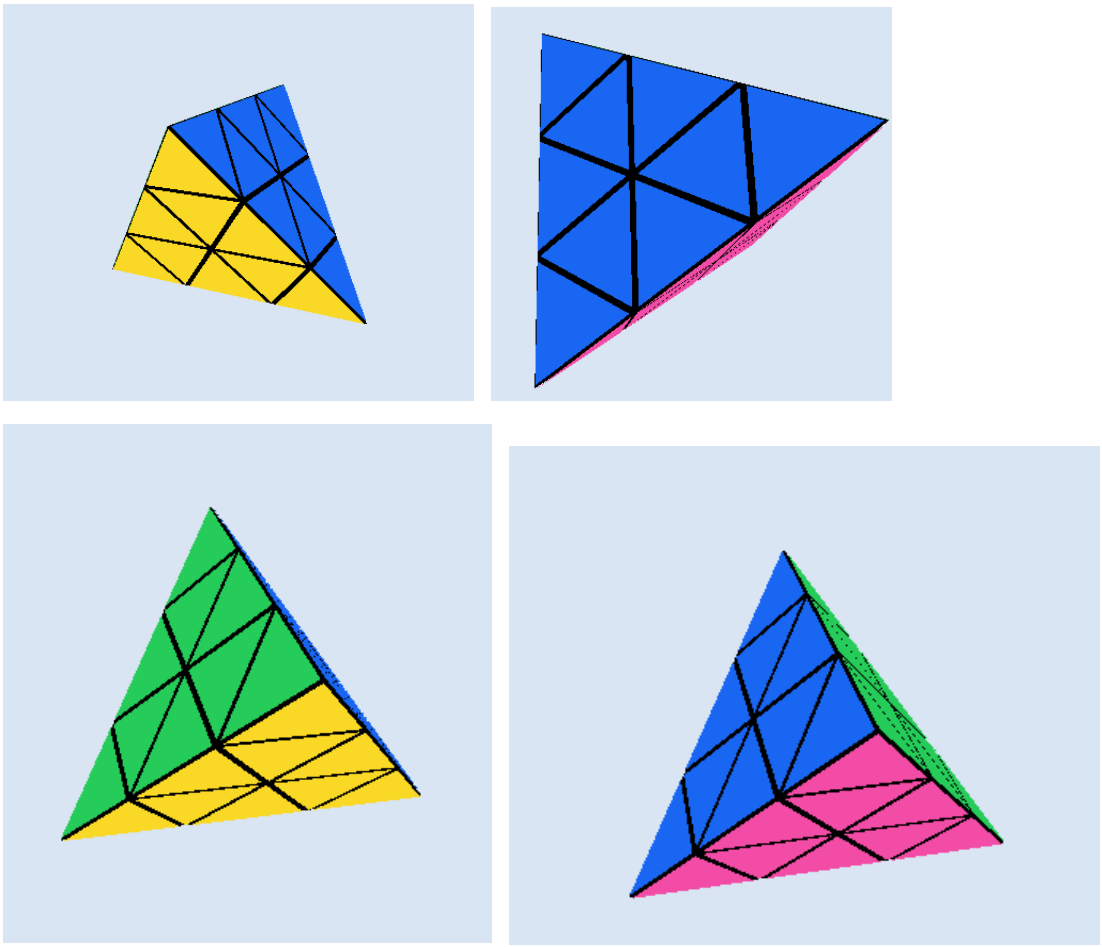
```cpp
        glm::mat4 baseModel(1.0f);
        baseModel = glm::rotate(baseModel, now * 0.6f, glm::vec3(0, 1, 0));

        // 0) Base negra MÁS PEQUEÑA para que se vean las caras encima
        {
            glm::mat4 model = glm::scale(baseModel, glm::vec3(0.985f));
            glUniformMatrix4fv(uModel, 1, GL_FALSE, glm::value_ptr(model));
            setColorUniform(COLOR_BLACK);
            meshList[0]->RenderMesh();
        }

        // 4) Caras con color
        const glm::vec3 faceColors[4] = { COLOR_BLUE, COLOR_PINK, COLOR_GREEN, COLOR_YELLOW };

        for (int f = 1; f <= 4; ++f)
        {
            glUniformMatrix4fv(uModel, 1, GL_FALSE, glm::value_ptr(baseModel));
            setColorUniform(faceColors[f - 1]);
            meshList[f]->RenderMesh();
        }
```

Para la ejecución y repetir el efecto del video que se mostro como ejemplo usamos rotate (así se ve una rotación suave del pyraminx). La base se dibuja un más pequeña (0.985) que las caras de color para que no tape las caras (por depth test), pero se asomen las pequeñas pirámides internas.

```
224         // 4) Caras con color
225         const glm::vec3 faceColors[4] = { COLOR_BLUE, COLOR_PINK, COLOR_GREEN, COLOR_YELLOW };
226
227         for (int f = 1; f <= 4; ++f)
228         {
229             glUniformMatrix4fv(uModel, 1, GL_FALSE, glm::value_ptr(baseModel));
230             setColorUniform(faceColors[f - 1]);
231             meshList[f]->RenderMesh();
232         }
233
234         glUseProgram(0);
235         mainWindow.swapBuffers();
236     }
```

Por ultimo coloreamos cada una de las caras haciendo uso de funciones y así optimizar el código.

Ejecución



Gracias a la transición de rotación es posible girar sobre el eje Y para darnos diferentes perspectivas de la pirámide, aun así, se puede interactuar con las teclas y el mouse para mover la cámara y hacer acercamientos o alejarnos.

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

La parte más difícil fue hacer que los bordes negros se notaran para delimitar los triángulos internos. Pero haciendo un poco de trampa pude investigar que era posible lograr este efecto visual subdividir cada cara en triángulos y “contraerlos” (inset) hacia su centroide, más dibujar un tetra negro ligeramente más pequeño detrás.

3.- Conclusión:

Esta practica resulto un verdadero reto en creatividad, pues fue fácil hacer una pirámide de 4 colores, porque se hacía una pirámide pequeña con un color de cara y este se repetía 9 veces para dar el efecto de una cara, pero al colocar la base negra esta absorbe el color de las caras y solo se veía una pirámide de color negro. Fue así como investigando la estructura del pyraminx vi que podía reducir un pequeño porcentaje de la base de color negro y hacer más pequeños los triángulos internos para causar ese efecto de bordes.

Ya por último quise dar el efecto de rotación como se mostraba en el video, y recordando el uso de los shaders pude modificar la perspectiva para hacer esa transición de rotación. Esta práctica fue de gran ayuda para conocer las profundidades geométricas c0on las que se trabajan en un plano 3D.

Bibliografía en formato APA

- *Cómo: Girar un objeto*. (n.d.). Microsoft.com. Retrieved September 6, 2025, from https://learn.microsoft.com/es-es/dotnet/desktop/wpf/graphics-multimedia/how-to-rotate-an-object
- *Dibujar y editar formas y rutas de acceso con el Diseñador XAML en Blend - Blend for Visual Studio*. (n.d.). Microsoft.com. Retrieved September 6, 2025, from https://learn.microsoft.com/es-es/visualstudio/xaml-tools/draw-shapes-and-paths?view=vs-2022
-