
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:** Uriarte Ortiz Enrique Yahir

**Nº de Cuenta:** 318234757

**GRUPO DE LABORATORIO:** 02

**GRUPO DE TEORÍA:** 04

**SEMESTRE 2025-1**

**FECHA DE ENTREGA LÍMITE:** Sábado 31 de Agosto del 2024

**CALIFICACIÓN: _____**

## *REPORTE DE PRÁCTICA:*

1. *Ejecución de los ejercicios.*
    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.*

    Para desarrollar esta actividad decidí realizar un boceto 3D con ayuda de GeoGebra, tomando como base la figura de pirámide que se proporcionó en el código de la práctica, las modificaciones principales a las coordenadas de la pirámide fue poner la base en el plano XZ, en la parte positiva de Z, gracias a la función que da GeoGebra de conocer la distancia de un segmento entre puntos pude verificar que la distancia entre ellos fuera la misma, para que estos fueran equiláteros y la estética de estos no se perdiera.

    Para dimensionar cada triangulo decidí que cada lado debía medir 0.5, la distancia de separación entre uno abajo del otro debía ser de 0.05, para la separación lateral entre los triángulos debía ser de 0.05 igual, pero en las filas donde teníamos unos triángulos dirigidos con su punta hacia arriba y otros hacia abajo decidí agregar una separación extra con referencia al eje Z de 0.05, esta decisión fue a partir de ver la imagen muestra que se proporcionó como guía.

    Una vez definidos los puntos de los 9 triángulos de la cara definí el contorno donde se encontrarían estos triángulos, decidí que el contorno estuviera a una separación de 0.03 de los triángulos.

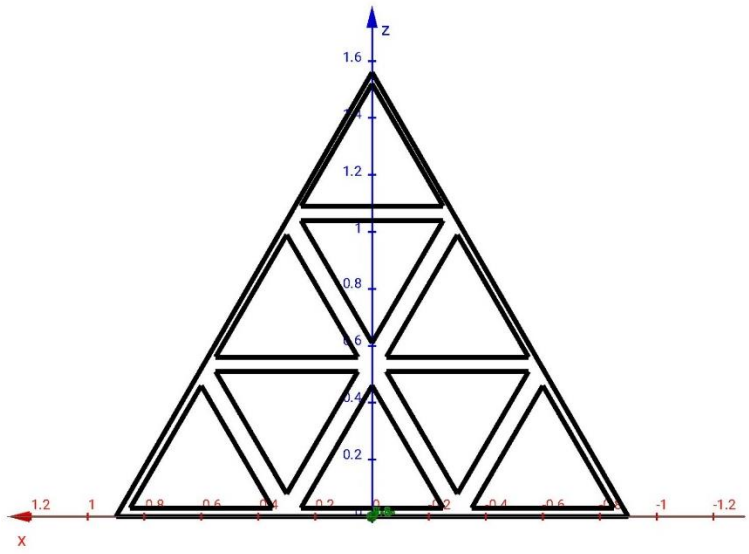


***Fig. 1.*** *Boceto de coordenadas de los triángulos de la primera cara.*

Solamente definí las coordenadas de los puntos de los 9 triángulos de una cara ya que plane apoyarme más adelante en el código de las funciones *“translate”* y *“rotate”* para colocarlas en las demás caras de la pirámide.

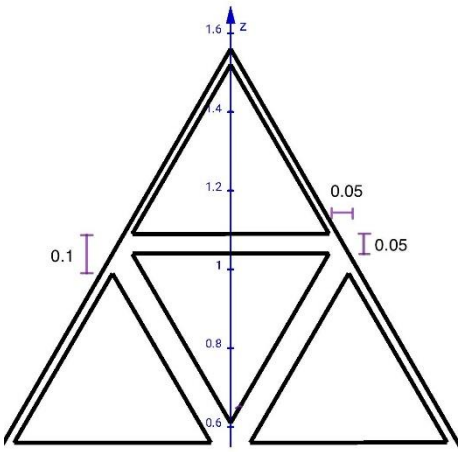


***Fig. 2.*** *Dimensiones entre los triángulos de la primera cara.*

Ya por último definí la altura de la pirámide apoyándome nuevamente de la herramienta de GeoGebra para conocer la distancia de los segmentos entre los puntos, la distancia de cada recta que forma la pirámide es de 1.8.

*<u>Como evidencia del trabajo se agrega el archivo el archivo de GeoGebra, para visualizarlo solo cargarlo en GeoGebra 3D online.</u>*

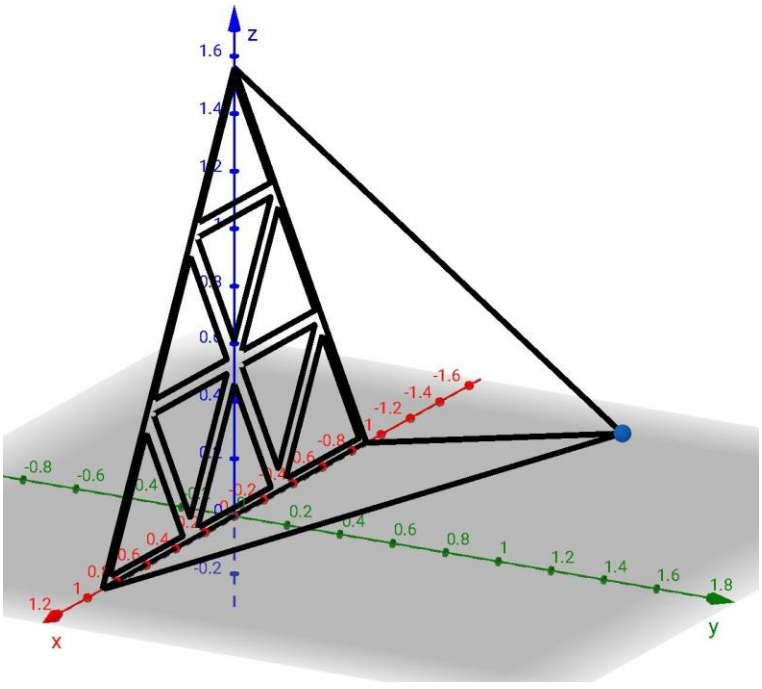


***Fig. 3.*** *Boceto pirámide realizado en GeoGebra.*

Ya con los puntos de cada figura, en el código .cpp principal implemente cada grupo de coordenadas correspondiente a cada figura en una función, obteniendo 10 funciones, una para la pirámide y las otras 9 para los triángulos.

```
void Piramide(){
    unsigned int indices_piramide_triangular[] = {0,1,2,  1,3,2,  3,0,2,  1,0,3};
    GLfloat vertices_piramide_triangular[] = {
        -0.9f,0.0f,0.0f,    0.9f,0.0f,0.0f,     0.0f,0.0f,1.56f,    0.0f,1.47f,0.52f};
    Mesh* obj1 = new Mesh();
    obj1->CreateMesh(vertices_piramide_triangular, indices_piramide_triangular, 12, 12);
    meshList.push_back(obj1);}
```

***Fig. 4.*** *Definición de pirámide.*

```
void T01(){//Triangulo 1.
    unsigned int indices_piramide_triangular[] = { 0,1,2 };
    GLfloat vertices_piramide_triangular[] = {
        0.25f,-0.01f,1.09f,     -0.25f,-0.01f,1.09f,        0.0f,-0.01f,1.52f};
    Mesh* obj1 = new Mesh();
    obj1->CreateMesh(vertices_piramide_triangular, indices_piramide_triangular, 9, 3);
    meshList.push_back(obj1);}
void T02(){//Triangulo 2.
    unsigned int indices_piramide_triangular[] = { 0,1,2 };
    GLfloat vertices_piramide_triangular[] = {
        -0.05f,-0.01f, 0.56f,   -0.55f,-0.01f,0.56f,   -0.3f,-0.01f,0.99f};
    Mesh* obj1 = new Mesh();
    obj1->CreateMesh(vertices_piramide_triangular, indices_piramide_triangular, 9, 3);
    meshList.push_back(obj1);}
void T03(){//Triangulo 3.
    unsigned int indices_piramide_triangular[] = { 0,1,2 };
    GLfloat vertices_piramide_triangular[] = {
        -0.25f,-0.01f,1.04f,    0.25f,-0.01f,1.04f,   0.0f,-0.01f,0.61f};
    Mesh* obj1 = new Mesh();
    obj1->CreateMesh(vertices_piramide_triangular, indices_piramide_triangular, 9, 3);
    meshList.push_back(obj1);}
void T04(){//Triangulo 4.
    unsigned int indices_piramide_triangular[] = { 0,1,2 };
    GLfloat vertices_piramide_triangular[] = {
        0.3f,-0.01f,0.99f,      0.05f,-0.01f,0.56f,     0.55f,-0.01f,0.56f};
    Mesh* obj1 = new Mesh();
    obj1->CreateMesh(vertices_piramide_triangular, indices_piramide_triangular, 9, 3);
    meshList.push_back(obj1);}
void T05(){//Triangulo 5.
    unsigned int indices_piramide_triangular[] = { 0,1,2 };
    GLfloat vertices_piramide_triangular[] = {
        -0.6f,-0.01f, 0.46f,    -0.85f,-0.01f,0.03f,    -0.35f,-0.01f,0.03f};
    Mesh* obj1 = new Mesh();
    obj1->CreateMesh(vertices_piramide_triangular, indices_piramide_triangular, 9, 3);
    meshList.push_back(obj1);}
```

***Fig. 5.*** *Definición de Triángulos de la primera cara de la pirámide.*

```
void T06(){//Triangulo 6.
    unsigned int indices_piramide_triangular[] = { 0,1,2 };
    GLfloat vertices_piramide_triangular[] = {
        -0.55f,-0.01f,0.51f,    -0.05f,-0.01f,0.51f,    -0.3f,-0.01f,0.08f};
    Mesh* obj1 = new Mesh();
    obj1->CreateMesh(vertices_piramide_triangular, indices_piramide_triangular, 9, 3);
    meshList.push_back(obj1);}
void T07(){//Triangulo 7.
    unsigned int indices_piramide_triangular[] = { 0,1,2 };
    GLfloat vertices_piramide_triangular[] = {
        0.0f,-0.01f, 0.46f,   0.25f,-0.01f,0.03f,  -0.25f ,-0.01f, 0.03f};
    Mesh* obj1 = new Mesh();
    obj1->CreateMesh(vertices_piramide_triangular, indices_piramide_triangular, 9, 3);
    meshList.push_back(obj1);}
void T08(){//Triangulo 8.
    unsigned int indices_piramide_triangular[] = { 0,1,2 };
    GLfloat vertices_piramide_triangular[] = {
        0.55f,-0.01f,0.51f,     0.05f,-0.01f,0.51f,     0.3f,-0.01f,0.08f};
    Mesh* obj1 = new Mesh();
    obj1->CreateMesh(vertices_piramide_triangular, indices_piramide_triangular, 9, 3);
    meshList.push_back(obj1);}
void T09(){//Triangulo 9.
    unsigned int indices_piramide_triangular[] = { 0,1,2 };
    GLfloat vertices_piramide_triangular[] = {
        0.6f,-0.01f,0.46f,      0.85f,-0.01f,0.03f,     0.35f,-0.01f,0.03f};
    Mesh* obj1 = new Mesh();
    obj1->CreateMesh(vertices_piramide_triangular, indices_piramide_triangular, 9, 3);
    meshList.push_back(obj1);}
```

***Fig. 6.*** *Definición de Triángulos de la primera cara de la pirámide.*

En la función principal main agregue las referencias a las llamadas de las demás figuras en un inicio, además de redimensionar la pantalla para que se vieran mejor las figuras. Tomando como ejemplo el video de cómo se debía ver la ejecución, también cambie el fondo a blanco.

```
mainWindow = Window(800, 800);
mainWindow.Initialise();

Piramide();
T01();T02();T03();T04(); T05(); T06(); T07(); T08(); T09();
```

***Fig. 7.*** *Definición de dimensión de pantalla y funciones de las figuras en main.*

```
glClearColor(1.0f, 1.0f, 1.0f, 1.0f);
glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
```

***Fig. 8.*** *Definición de color de fondo blanco.*

Para la impresión de las figuras utilice el código proporcionado pero declarando el numero en *"meshList[]->RenderMesh();"* de acuerdo con el orden declarado en el main.

```
//Piramide principal
model = glm::mat4(1.0);
model = glm::translate(model, glm::vec3(0.0f, -0.3f, -1.5f));
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));
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));
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));
meshList[0]->RenderMesh();
```

***Fig. 9.*** *Sección de color de pirámide.*

Para la impresión de los 9 triángulos únicamente se colocaron las líneas a partir de *"color = glm:: vec3 …"* ya que no se hacía ninguna modificación a los demás valores de translación, escala y rotación.

```
//Triangulos de la Primera Cara - Rojos
color = glm::vec3(1.0f, 0.0f, 0.0f);
glUniform3fv(uniformColor, 1, glm::value_ptr(color));
meshList[1]->RenderMesh(); meshList[2]->RenderMesh(); meshList[3]->RenderMesh();
meshList[4]->RenderMesh(); meshList[5]->RenderMesh(); meshList[6]->RenderMesh();
meshList[7]->RenderMesh(); meshList[8]->RenderMesh(); meshList[9]->RenderMesh();
```

***Fig. 10.*** *Sección de color rojo para los triángulos de la primera cara.*

Para las demás caras decidí utilizar las funciones *"rotate"* y *"translate"* para acomodar los triángulos. Para la segunda cara analice la figura y me di cuenta de que haciendo una rotación con respecto al eje X de estas figuras podía colocarlas en la siguiente cara, para ello calcule el ángulo de inclinación que se puede observar en GeoGebra de acuerdo con el plano YZ. Aplicando los cálculos:

$$c = \sqrt{a^2 + b^2} = \sqrt{1.47^2 + 0.52^2} = 1.599$$

$$Sin^{-1} = \frac{0.52}{1.599} = 18.977°$$

$$270° + 18.977° = 288.977°$$

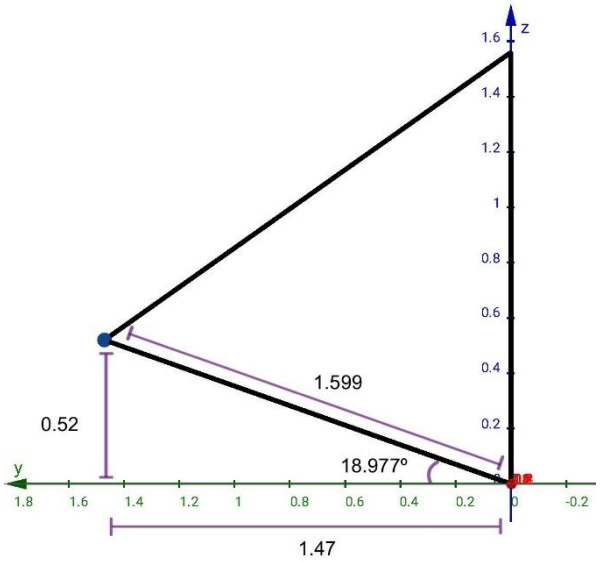


***Fig. 11.*** *Cálculos del ángulo para la segunda cara de la pirámide.*

Por lo tanto, el ángulo de inclinación para colocar los triángulos es de 288.977° con respecto al eje X, pero después de esta rotación los triángulos quedaban un poco metidos dentro de la pirámide, por lo que agregue un poco de desplazamiento en los ejes Y y Z, además de cambiar el color de los triángulos a verde

```
//Triangulos de la Segunda Cara - Verde
glm::mat4 rotatedModel = model;
rotatedModel = glm::translate(rotatedModel, glm::vec3(0.0f, 0.01f, -0.008f));
rotatedModel = glm::rotate(rotatedModel, glm::radians(288.785f), glm::vec3(1.0f, 0.0f, 0.0f));
glUniformMatrix4fv(uniformModel, 1, GL_FALSE, glm::value_ptr(rotatedModel));
color = glm::vec3(0.0f, 1.0f, 0.0f);
glUniform3fv(uniformColor, 1, glm::value_ptr(color));
meshList[1]->RenderMesh(); meshList[2]->RenderMesh(); meshList[3]->RenderMesh();
meshList[4]->RenderMesh(); meshList[5]->RenderMesh(); meshList[6]->RenderMesh();
meshList[7]->RenderMesh(); meshList[8]->RenderMesh(); meshList[9]->RenderMesh();
```

***Fig. 12.*** *Sección de color verde para los triángulos de la segunda cara.*

Para las 2 ultimas caras tuve que emplear más las funciones de *“rotate”* y *“translate”*, primero los desplace hacia los ejes Y y Z, después rote 90° en con respecto al eje Y, luego otra rotación de 90° con respecto a su nuevo eje Y, desplace la figura nuevamente pero esta vez con respecto al eje X la figura, hasta este punto los triángulos ya están ajustados en el centro, con las siguientes funciones se declara el ángulo para definir hacia que cara se encuentran, se rotan 30° hacia su eje Z, para luego rotar unos 18.785° hacia X, lo único que falta es realizar un desplazamiento, ya que de nuevo las figuras se encuentran un poco metidas en la pirámide.
Para la cara restante son las mismas funciones de *“rotate”* y *“translate”*, hasta la instrucción de rotar 30° hacia su eje Z, ya que como es la cara contraria, esta debe rotar hacia -30° hacia su eje Z y después rotar unos -18.785° hacia X, también realiza un desplazamiento, pero únicamente es el eje Y que cambia de signo.

```
//Triangulos de la Tercer Cara - Azul
glm::mat4 rotatedModel1 = model;
rotatedModel1 = glm::translate(rotatedModel1, glm::vec3(0.0f, 0.01f, -0.008f));
rotatedModel1 = glm::rotate(rotatedModel1, glm::radians(90.0f), glm::vec3(0.0f, 0.0f, 1.0f));
rotatedModel1 = glm::rotate(rotatedModel1, glm::radians(90.0f), glm::vec3(0.0f, 1.0f, 0.0f));
rotatedModel1 = glm::translate(rotatedModel1, glm::vec3(-0.69f, 0.0f, 0.0f));
rotatedModel1 = glm::rotate(rotatedModel1, glm::radians(30.0f), glm::vec3(0.0f, 0.0f, 1.0f));
rotatedModel1 = glm::rotate(rotatedModel1, glm::radians(18.785f), glm::vec3(1.0f, 0.0f, 0.0f));
rotatedModel1 = glm::translate(rotatedModel1, glm::vec3(0.15f, 0.425f, -0.15f));
glUniformMatrix4fv(uniformModel, 1, GL_FALSE, glm::value_ptr(rotatedModel1));
color = glm::vec3(0.0f, 0.0f, 1.0f);
glUniform3fv(uniformColor, 1, glm::value_ptr(color));
meshList[1]->RenderMesh(); meshList[2]->RenderMesh(); meshList[3]->RenderMesh();
meshList[4]->RenderMesh(); meshList[5]->RenderMesh(); meshList[6]->RenderMesh();
meshList[7]->RenderMesh(); meshList[8]->RenderMesh(); meshList[9]->RenderMesh();
```

***Fig. 13.*** *Sección de color azul para los triángulos de la tercera cara.*

```
//Triangulos de la Tercer Cara - Amarillo
glm::mat4 rotatedModel2 = model;
rotatedModel2 = glm::translate(rotatedModel2, glm::vec3(0.0f, 0.01f, -0.008f));
rotatedModel2 = glm::rotate(rotatedModel2, glm::radians(90.0f), glm::vec3(0.0f, 0.0f, 1.0f));
rotatedModel2 = glm::rotate(rotatedModel2, glm::radians(90.0f), glm::vec3(0.0f, 1.0f, 0.0f));
rotatedModel2 = glm::translate(rotatedModel2, glm::vec3(-0.69f, 0.0f, 0.0f));
rotatedModel2 = glm::rotate(rotatedModel2, glm::radians(-30.0f), glm::vec3(0.0f, 0.0f, 1.0f));
rotatedModel2 = glm::rotate(rotatedModel2, glm::radians(-18.785f), glm::vec3(1.0f, 0.0f, 0.0f));
rotatedModel2 = glm::translate(rotatedModel2, glm::vec3(0.15f, -0.405f, -0.15f));
glUniformMatrix4fv(uniformModel, 1, GL_FALSE, glm::value_ptr(rotatedModel2));
color = glm::vec3(1.0f, 1.0f, 0.0f);
glUniform3fv(uniformColor, 1, glm::value_ptr(color));
meshList[1]->RenderMesh(); meshList[2]->RenderMesh(); meshList[3]->RenderMesh();
meshList[4]->RenderMesh(); meshList[5]->RenderMesh(); meshList[6]->RenderMesh();
meshList[7]->RenderMesh(); meshList[8]->RenderMesh(); meshList[9]->RenderMesh();
```

***Fig. 14.*** *Sección de color amarillo para los triángulos de la cuarta cara.*

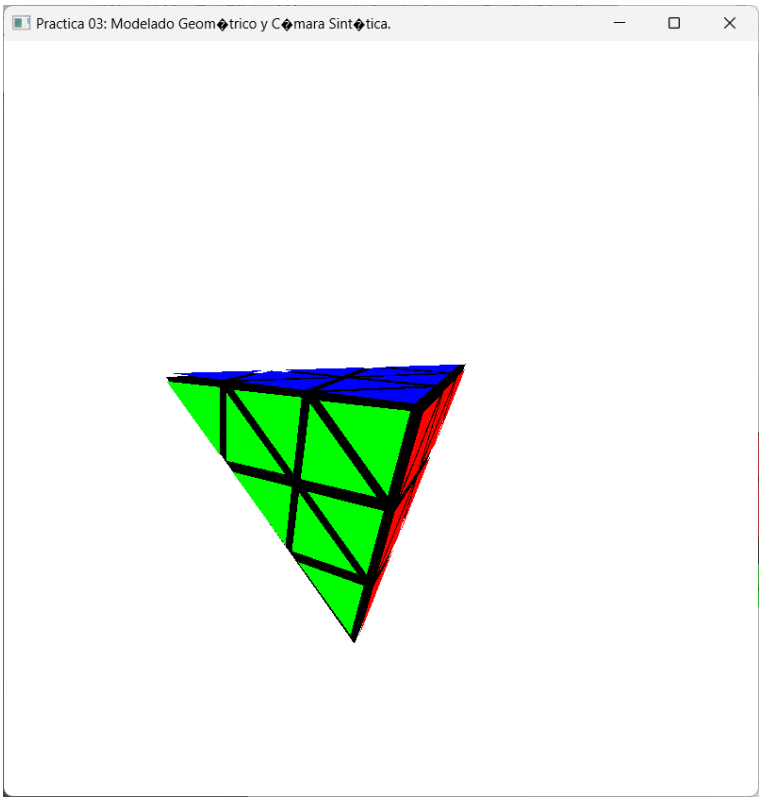


***Fig. 15.*** *Ejecución del código.*

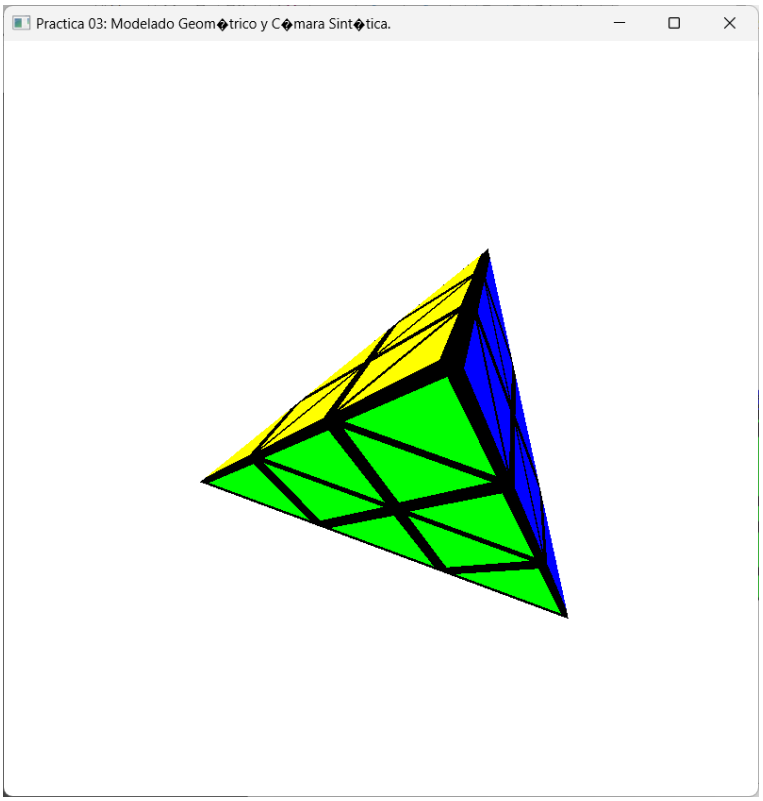


***Fig. 16.*** *Ejecución del código.*

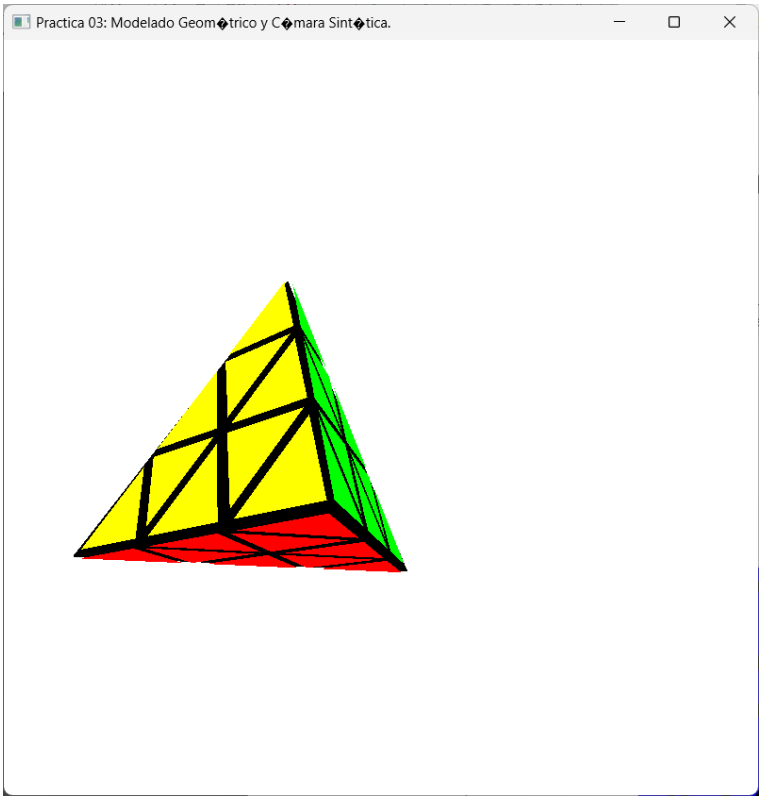


***Fig. 17.*** *Ejecución del código.*

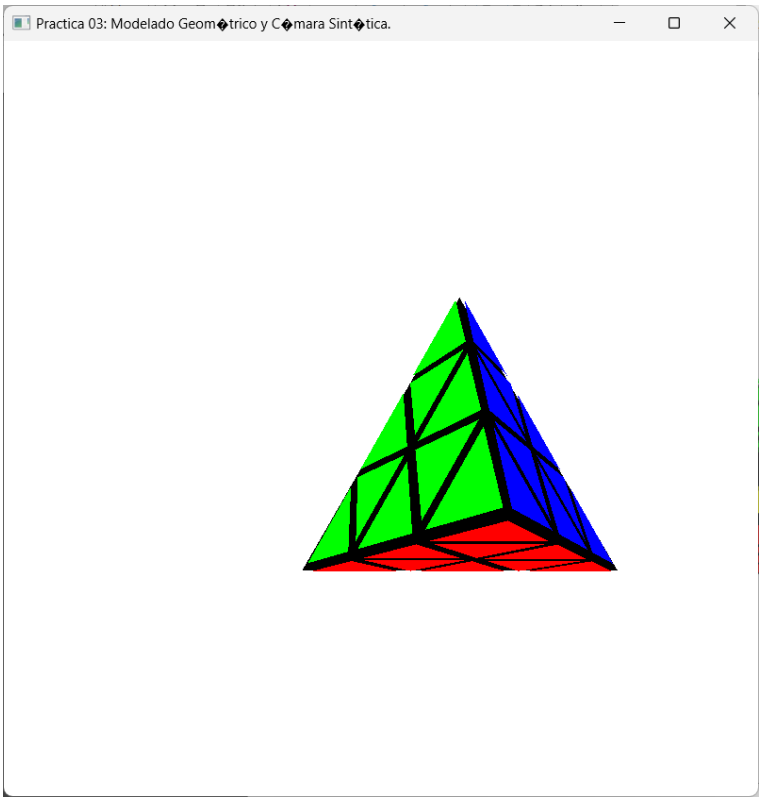


***Fig. 18.*** *Ejecución del código.*

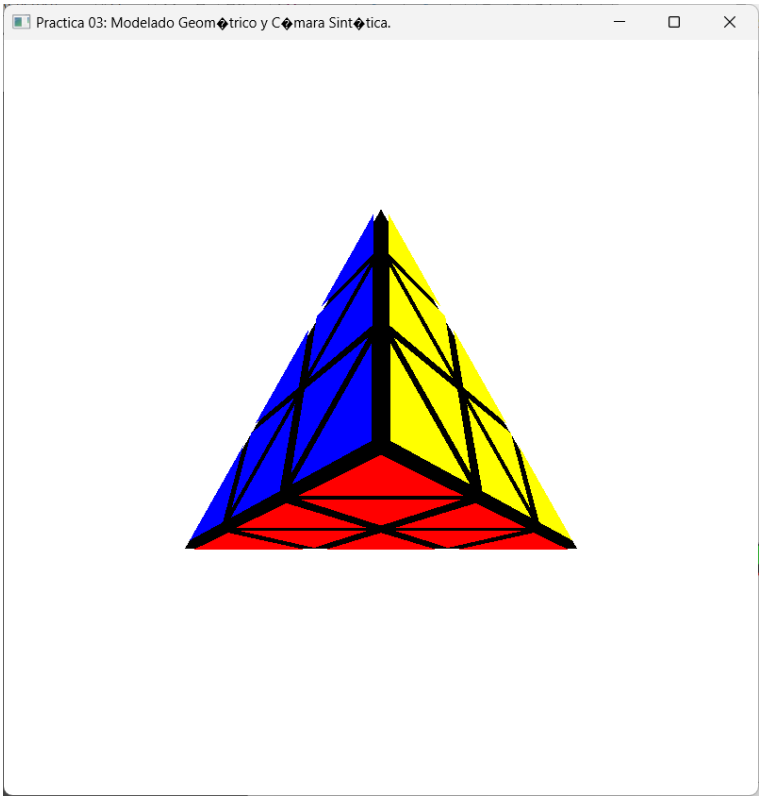


***Fig. 19.*** *Ejecución del código.*

2. *Problemas presentados.*
   Durante el desarrollo de la actividad en un inicio tenia planeado definir una por una las coordenadas de los 9 triángulos en cada cara con ayuda de GeoGebra, sin embargo, después de realizar la primera cara que era la que se encontraba en el plano XZ me fue más complicado definir la segunda cara que se encontraba en el plano XY, mas que nada se me complico el darle ese efecto de inclinación para que las coordenadas salieran en la cara de la pirámide y no perdieran sus dimensiones, cuando comencé con las otras 2 caras me di cuenta de lo complicado y tardado que me seria este proceso con esas caras ya que en este caso no tenia un plano de referencia definido, por lo que después de analizar que podía hacer, recordé la implementación de *"rotate"* y *"translate"*, pero para ello tenía que sumarle todas las coordenadas en el eje Z 0.69 para que pudiera utilizar mejor la rotación de las figuras, lo que provocó que al final se pudiera realizar el ejercicio como se solicitó.

3. *Conclusión:*
   Tras realizar el ejercicio de la practica me parece que la complejidad era engañosa, dadas las instrucciones del profesor parecía muy complicada, pero una vez analizado las funciones que podríamos utilizar, la complejidad a mi parecer bajo, ya que lo único que se tenía que hacer era analizar y calcular los ángulos y dimensiones de las figuras para qué obtuviéramos la pirámide rubik que se pedía, considero que la actividad fue buena para poner en práctica las funciones de rotar y desplazar. Al final consideró que esta práctica cumplió su objetivo al implementar diversas figuras en 3D analizando el espacio, siendo esto a mi parecer una buena práctica para el desarrollo del proyecto final.

4. *Bibliografía.*

   - *No empleada*