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: Sánchez Trejo Arturo

Nº de Cuenta: 316191809

GRUPO DE LABORATORIO: 02

GRUPO DE TEORÍA: 03

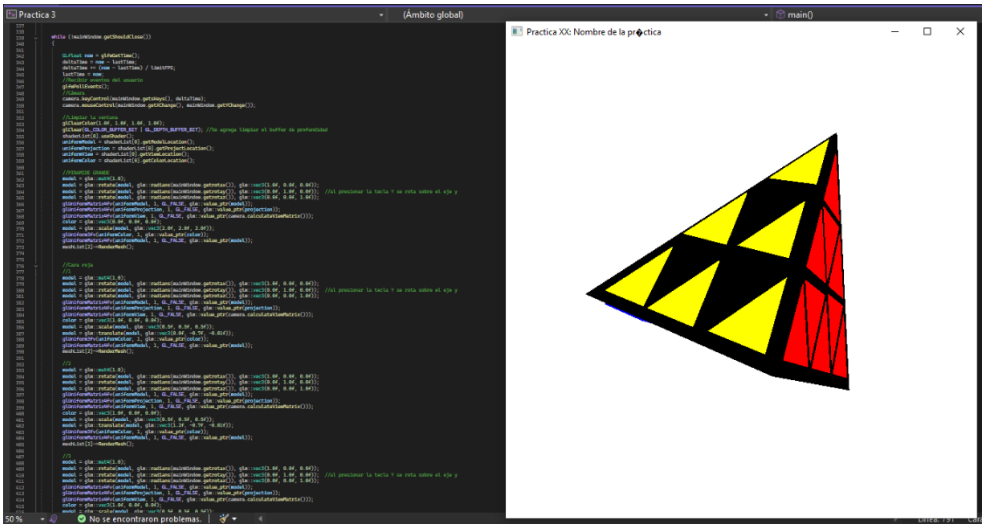
SEMESTRE 2025-1

FECHA DE ENTREGA LÍMITE: 31 DE AGOSTO 2024

CALIFICACIÓN: _____

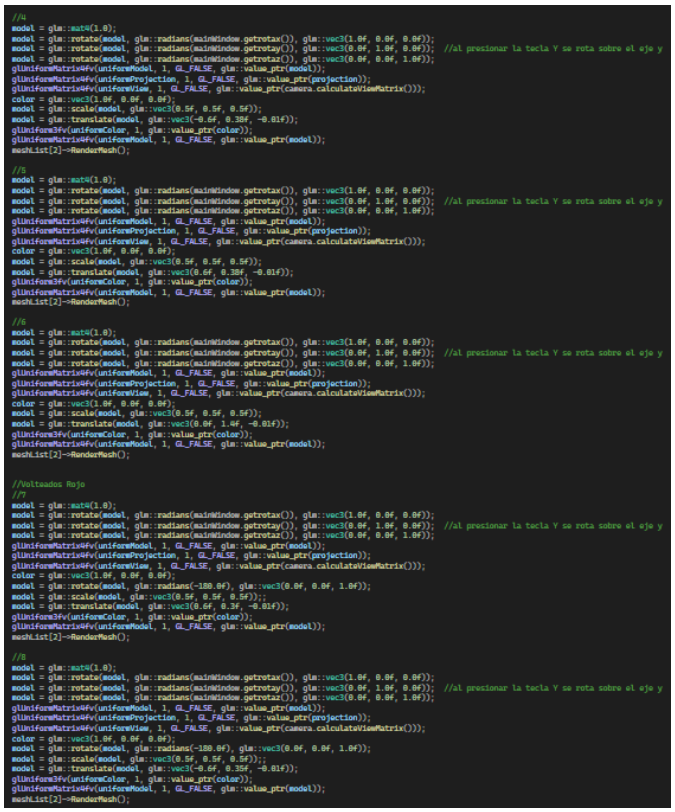
REPORTE DE PRÁCTICA:

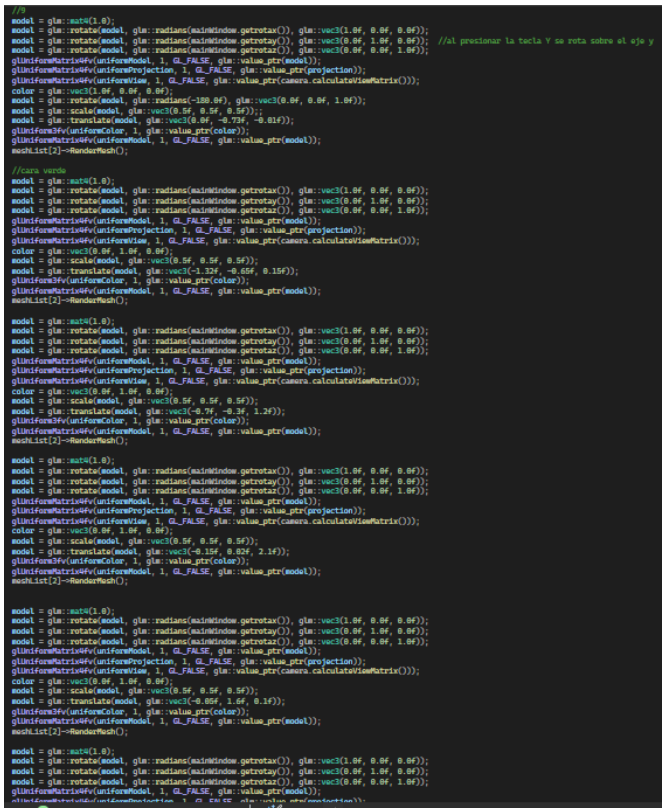
1.- Ejecución de los ejercicios que se dejaron, comentar cada uno y capturas de pantalla de bloques de código generados y de ejecución del programa.

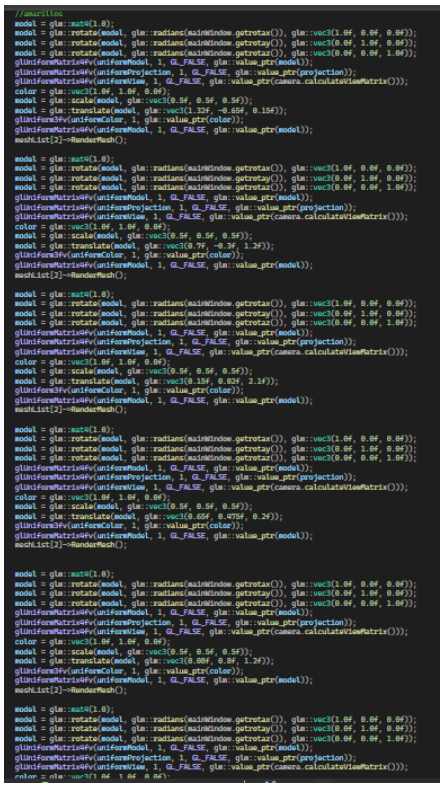



```
void CrearPiramideTriangular()
{
    GLfloat L = 1.0f;
    GLfloat H = sqrt(2.0f / 3.0f) * L;

    unsigned int indices_piramide_triangular[] = {
        0, 1, 2,
        0, 1, 3,
        1, 2, 3,
        2, 0, 3

    };
    GLfloat vertices_piramide_triangular[] = {

       -0.5f * L, -sqrt(3.0f) / 6.0f * L, 0.0f,
        0.5f * L, -sqrt(3.0f) / 6.0f * L, 0.0f,
        0.0f,  sqrt(3.0f) / 3.0f * L, 0.0f,
         0.0f, 0.0f, H

    };
    Mesh* obj1 = new Mesh();
    obj1->CreateMesh(vertices_piramide_triangular, indices_piramide_triangular, 12, 12);
    meshList.push_back(obj1);

}
```

Lo primero que hicimos fue modificar los vértices de crarpiramidetriangular, ya que al utilizar vista en clase creaba de manera deforme nuestro triangulo, es por eso que investigamos llegamos a esta conclusión para modificar y obtener raíces cuadradas para el cuadrado, llegando a este resultado.



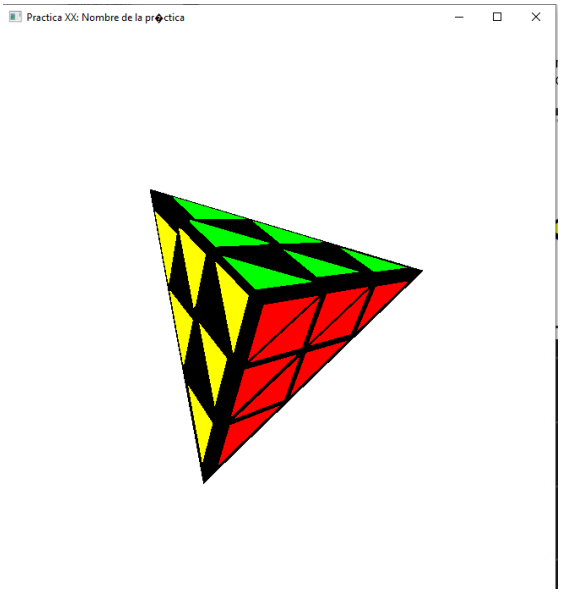


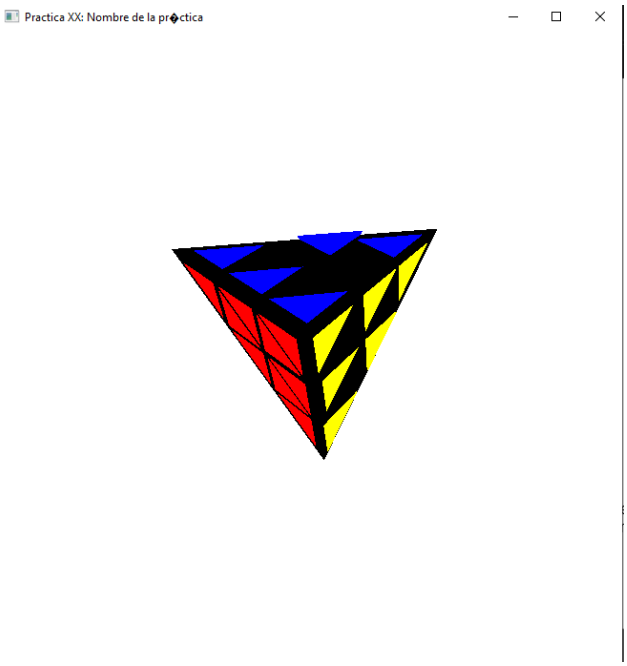


En estos códigos lo que hicimos fue cambiar al mesh por el 2, esto para hacer llamar a la pirámide triangular, ya con esto, reescalamos el cuadrado central negro a x2 y lo cambiamos de color negro, ahora para las caras lo que hicimos fue reescalar el triangulo a .3 para que sea una tercera parte del tamaño de nuestro triangulo negro, con esto volvemos utilizamos el translate para cambiar de lugar y crear las caras de cada una, con esto hicimos 6 triángulos en la misma dirección que el triangulo principal, nuestro principal problema se presento cuando tratamos de colocar los triangulo volteador, lo que hicimos fue utilizar el rotate para voltearlo de cabeza y conseguir ese efecto, y lo colocamos en su posición correcta, esta cara que fue la roja fue la mas sencilla, el problema es que nuestro triangulo principal nos lo construyo de una manera inclinada por lo tanto nos provoco problemas el hacer la cara verde ya que las coordenadas de x,y,z se comportaban de manera muy extraña, ya que si lo subimos un poco nos deformaba mucho el eje x, z, y viceversa si modificábamos el eje x alteraba el eje y, z.

De igual manera la cara azul eran coordenadas totalmente diferente ya que se encontraba debajo de nuestra figura principal, para este tratamos de utilizar las caras los triángulos de las otras figuras y modificar un poco para bajarlas un poco mas o moverlas en el eje x, z.

```
//azul
model = glm::mat4(1.0);
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));
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, 1.0f);
model = glm::scale(model, glm::vec3(0.5f, 0.5f, 0.5f));
model = glm::translate(model, glm::vec3(0.0f, -0.8f, 0.2f));
glUniform3fv(uniformColor, 1, glm::value_ptr(color));
glUniformMatrix4fv(uniformModel, 1, GL_FALSE, glm::value_ptr(model));
meshList[2]->RenderMesh();


model = glm::mat4(1.0);
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, 1.0f);
model = glm::scale(model, glm::vec3(0.5f, 0.5f, 0.5f));
model = glm::translate(model, glm::vec3(1.2f, -0.8f, 0.2f));
glUniform3fv(uniformColor, 1, glm::value_ptr(color));
glUniformMatrix4fv(uniformModel, 1, GL_FALSE, glm::value_ptr(model));
meshList[2]->RenderMesh();

model = glm::mat4(1.0);
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, 1.0f);
model = glm::scale(model, glm::vec3(0.5f, 0.5f, 0.5f));
model = glm::translate(model, glm::vec3(-1.2f, -0.8f, 0.2f));
glUniform3fv(uniformColor, 1, glm::value_ptr(color));
glUniformMatrix4fv(uniformModel, 1, GL_FALSE, glm::value_ptr(model));
meshList[2]->RenderMesh();

model = glm::mat4(1.0);
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, 1.0f);
model = glm::scale(model, glm::vec3(0.5f, 0.5f, 0.5f));
model = glm::translate(model, glm::vec3(0.1f, -0.15f, 2.1f));
glUniform3fv(uniformColor, 1, glm::value_ptr(color));
glUniformMatrix4fv(uniformModel, 1, GL_FALSE, glm::value_ptr(model));
meshList[2]->RenderMesh();

model = glm::mat4(1.0);
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, 1.0f);
model = glm::scale(model, glm::vec3(0.5f, 0.5f, 0.5f));
model = glm::translate(model, glm::vec3(-0.6f, -0.4, 1.5f));
glUniform3fv(uniformColor, 1, glm::value_ptr(color));
glUniformMatrix4fv(uniformModel, 1, GL_FALSE, glm::value_ptr(model));
meshList[2]->RenderMesh();

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





2.- Liste los problemas que tuvo a la hora

El principal erro fue el crear nuestra pirámide central de manera inclinada pudimos resolverlo con un rotate en el eje x pero cuando nos dimos cuenta de esto ya estábamos muy avanzados.

El otro fue el hacer los triángulos invertidos se comporto de manera diferente en cada cara es por eso que solo la pudimos colocar en la cara roja.

3.- Conclusión:

a. Los ejercicios del reporte: No debió ser mucho problema el hacer esta figura, pero creo que hay algunos comando importantes que deben dedicar un poca más de tiempo como lo fue en el model = glm::mat4(1.0);, o como se comporta el rotate en nuestras figuras creadas. Además, no encontré el motivo del porque mis pirámides se crearon de manera inclinada, sin embargo, pareciera que era un practica sencilla pero al colocar cada triangulo si me llevaba mucho tiempo ya que el posicionarla con cada figura era modificar cada y renderizar para cada posición.

1. Bibliografía en formato APA

Wright, R. S., Haemel, N., Sellers, G., & Lipchak, B. (2010). OpenGL SuperBible:
Comprehensive tutorial and reference (5th ed.). Addison-Wesley.

Khronos Group. (n.d.). OpenGL: A powerful graphics API for rendering 2D and 3D
vector graphics. Retrieved from https://www.opengl.org/