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: Barragán Pilar Diana

Nº de Cuenta: 318147981

GRUPO DE LABORATORIO: 03

GRUPO DE TEORÍA: 04

SEMESTRE 2025-1

FECHA DE ENTREGA LÍMITE: 31 de agosto de 2024

CALIFICACIÓN: _____

Práctica 3: Modelado Geométrico y Cámara Sintética

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)**

En este ejercicio para lograr crear la pirámide compuesta de pirámides más pequeñas lo que realice fue una pirámide mayor que sería la base y pequeñas pirámides acomodadas de colores distintos en cada cara de tal forma que se formará una pyraminx.

Lo primero que realice fue generar una nueva forma de pirámide que tuviera otros valores de vértice para que al momento de rotarla y acomodarla esta encajará de mejor forma en la pirámide más grande, para ello copie la forma de la función `CrearPiramideTriangular()` y cambie la posición de uno de los vértices, la nueva función la llame `CrearPiramideTriangularC()` y la declare ultima al mandar a llamar a las funciones en el main, así que su índice en la `MeshList` fue de 5.

```
void CrearPiramideTriangularC()
{
    unsigned int indices_piramide_triangular[] = {
            0,1,2,
            1,3,2,
            3,0,2,
            1,0,3

    };
    GLfloat vertices_piramide_triangular[] = {
        -0.5f, -0.5f,0.0f,  //0
        0.5f,-0.5f,0.0f,    //1
        0.0f,0.5f, -0.2f,   //2
        //0.0f,0.5f, 0.0f,  //2
        0.0f,-0.5f,-0.5f,   //3

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

}
```

Figura 1. Función `CrearPiramideTriangularC()`.

Posteriormente en el `main` cree la figura principal de la pirámide y la cambie de color al negro según el RGB para que las líneas que permiten diferenciar las pirámides sean de este color, con ayuda de la función `scale` hice que la figura se dibujará cinco veces más grande en ‘x’ y ‘y’ y nueve veces mayor en ‘z’.

```
//PIRAMIDE TRIANGULAR DE BASE
model = glm::mat4(1.0);
//Traslación inicial para posicionar en -Z a los objetos

//otras transformaciones para el objeto
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));

model = glm::translate(model, glm::vec3(0.0f, 0.0f, 0.0f));
model = glm::scale(model, glm::vec3(5.2f, 5.2f, 9.2f));
glUniformMatrix4fv(uniformModel, 1, GL_FALSE, glm::value_ptr(model));
//la línea de proyección solo se manda una vez a menos que en tiempo de ejecución
//se programe cambio entre proyección ortogonal y perspectiva
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[1]->RenderMesh(); //dibuja cubo y pirámide triangular
//meshList[3]->RenderMeshGeometry(); //dibuja las figuras geométricas cilindro, cono, pirámide
//sp.render(); //dibuja esfera
```

Figura 2. Pirámide base.

Para acomodar los triángulos pequeños en sus respectivos lugares primero utilice la función `translate` para que la figura dejara de dibujarse en medio, comencé con la pirámide en la esquina inferior derecha y con un `scale` la modifique para que quedará de un tamaño adecuado para que cupieran todas las pirámides pequeñas dentro de la mayor, al final le cambie el color al que llevaría ese lado de la pirámide.

Con el siguiente triangulo como debía estar de cabeza utilicé la pirámide que cree con la función `CrearPiramideTriangularC()` que se encentra en el `MeshList` cinco, luego aplique `rotate` en el eje z para lograrlo y con otro `rotate` en el eje x para que se acomodará mejor con la pirámide mayor.

```
//CARA 1 PISO 1
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));
model = glm::translate(model, glm::vec3(1.6f, -1.6f, -0.15f));
model = glm::scale(model, glm::vec3(1.3f, 1.3f, 2.7f));
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(1.0f, 0.0f, 0.0f);
glUniform3fv(uniformColor, 1, glm::value_ptr(color)); //para cambiar el color del objetos
meshList[5]->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));
model = glm::translate(model, glm::vec3(0.8f, -1.6f, -0.1f));
model = glm::scale(model, glm::vec3(1.3f, 1.3f, 2.7f));
model = glm::rotate(model, 180 * toRadians, glm::vec3(0.0f, 0.0f, 1.0f));
model = glm::rotate(model, 26 * toRadians, glm::vec3(1.0f, 0.0f, 0.0f));
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(1.0f, 0.0f, 0.0f);
glUniform3fv(uniformColor, 1, glm::value_ptr(color)); //para cambiar el color del objetos
meshList[1]->RenderMesh();
```

Figura 3. Código de la cara 1 uno de la pirámide.

```
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));
model = glm::translate(model, glm::vec3(-0.02f, -1.6f, -0.15f));
model = glm::scale(model, glm::vec3(1.3f, 1.3f, 2.7f));
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(1.0f, 0.0f, 0.0f);
glUniform3fv(uniformColor, 1, glm::value_ptr(color)); //para cambiar el color del objetos
meshList[5]->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));
model = glm::translate(model, glm::vec3(-0.8f, -1.6f, -0.1f));
model = glm::scale(model, glm::vec3(1.3f, 1.3f, 2.7f));
model = glm::rotate(model, 180 * toRadians, glm::vec3(0.0f, 0.0f, 1.0f));
model = glm::rotate(model, 26 * toRadians, glm::vec3(1.0f, 0.0f, 0.0f));
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(1.0f, 0.0f, 0.0f);
glUniform3fv(uniformColor, 1, glm::value_ptr(color)); //para cambiar el color del objetos
meshList[1]->RenderMesh();
```

Figura 4. Código de la cara 1 uno de la pirámide.

Para el resto simplemente hice lo mismo solo cambiando la posición con `scale` según fuera el caso y con `rotate` en los ejes tanto ‘x’, ‘y’ y ‘z’ para acomodar bien las pirámides en la pirámide mayor.

Al compilar el código de esta práctica no se me presento ningún inconveniente respecto a la programación de este, puesto que todo funciono con normalidad.

CONCLUSIÓN:

En la elaboración de esta práctica reforcé las nociones que tenía respecto al dibujado de las figuras geométricas como también al uso de las funciones `translate` y `rotate` puesto que al utilizarlas en tantas ocasiones para acomodar cada una de las pirámides más pequeñas su uso se me fue facilitando más, de igual forma con el dibujado de las figuras geométricas pues comprendí mejor que hace cada una de las funciones para generarlas.

La práctica sin duda me pareció muy buena para afianzar los conocimientos respecto a las figuras geométricas e igual la cámara, sin embargo, de la forma en que yo elegí realizar esta actividad es muy ineficiente pues se ocupa mucho código, así como tiempo para ir acomodando cada pirámide en su respectivo lugar.