UNIVERSIDAD NACIONAL AUTÓNOMA DE MÉXICO

FACULTAD DE INGENIERÍA

Ingeniería en Computación

LABORATORIO DE COMPUTACIÓN GRÁFICA
E INTERACCIÓN HUMANO COMPUTADORA

Grupo: 02

Ejercicio de clase

Práctica Número 4

Alumna: Pamela Salgado Fernández Pamela

Número de Cuenta: 313236505

Email: pame501@yahoo.com.mx

Semestre 2019-2

Grupo de teoría: 1

Fecha de entrega límite: 27/02/2019

Contenidos

1 Actividades realizadas **2**
1.1 Instanciar el triángulo varias veces 2
1.1.1 Ejecución del programa 3
1.2 Cambiar color a cada uno de los triángulos 3
1.2.1 ejecución del código 5

2 Código **6**

3 Problemas presentados **6**

4 Conclusiones **6**

1 Actividades realizadas

Para la actividad de clase, el profesor nos solicitó que a partir del código que nos proporcionó, lo modificáramos para crear 9 triángulos, es decir, instanciar varias veces la figura. Para la segunda actividad de clase, fue necesario asignar un color diferente a cada uno de los triángulos.

1.1 Instanciar el triángulo varias veces

Para poder crear varios triángulos solo fue necesario llamar de nuevo a model para inicializar de nuevo la matriz y una vez teniendo la matriz agregamos todos los datos necesarios para que nuestro otro triangulo se vea (escala,translación y rotación) y se los .

En practica4.cpp dentro del while que se encuentra en la linea 151, se agregan los demás triángulos:

```
//*****************************************************************
//creamos los diferentes triangulos
//HACIA ABAJO
//a model le paso una matriz identidad para resetear los datos
model = glm::mat4(1.0);
model = glm::translate(model, glm::vec3(-0.15f, -0.3f, -1.0f));
model = glm::scale(model, glm::vec3(0.3f, 0.3f, 0.3f));
model = glm::rotate(model,  180* toRadians, glm::vec3(1.0f, 0.0f, 0.0f));
glUniformMatrix4fv(uniformModel, 1, GL_FALSE, glm::value_ptr(model));
meshList[1]->RenderMesh();

                //segundo Triangulo
//a model le paso una matriz identidad para resetear los datos
model = glm::mat4(1.0);
model = glm::translate(model, glm::vec3(0.15f, -0.3f, -1.0f));
model = glm::scale(model, glm::vec3(0.3f, 0.3f, 0.3f));
model = glm::rotate(model, 180 * toRadians, glm::vec3(1.0f, 0.0f, 0.0f));
glUniformMatrix4fv(uniformModel, 1, GL_FALSE, glm::value_ptr(model));
meshList[2]->RenderMesh();

                // Tercer triangulo
//a model le paso una matriz identidad para resetear los datos
model = glm::mat4(1.0);
model = glm::translate(model, glm::vec3(0.3f, -0.3f, -1.0f));
model = glm::scale(model, glm::vec3(0.3f, 0.3f, 0.3f));
glUniformMatrix4fv(uniformModel, 1, GL_FALSE, glm::value_ptr(model));
meshList[3]->RenderMesh();

//*****************************************************************
```

Este mismo procedimiento se sigue para los otros triángulos.

1.1.1 Ejecución del programa

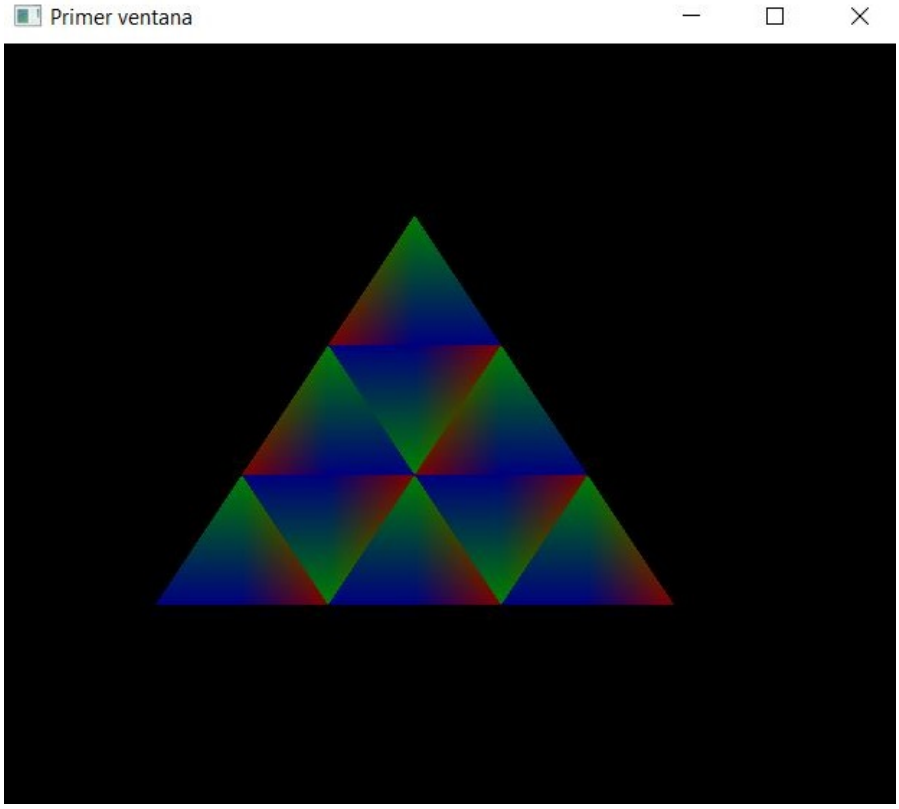

Figura 1

1.2 Cambiar color a cada uno de los triángulos

Para cambiar el color de los triángulos, fue necesario crear 9 VBOs, los cuales contenían el color que se asignaría a cada uno de los triángulos, para ello nos apoyamos también en una función llamada genera_color().

```
//**********************************************************************

// creamos VBO's los cuales nos darán el color de cada vertice
        GLfloat color_triangulo[] = {
                c1, c2, c3, 1.0f,
                c1, c2, c3, 1.0f,
                c1, c2, c3, 1.0f,
                c1, c2, c3, 1.0f,
        };

        GLfloat color_triangulo1[] = {
        c2, c2, c1, 1.0f,
        c2, c2, c1, 1.0f,
        c2, c2, c1, 1.0f,
```

```
        c2, c2, c1, 1.0f,
        };
        genera_color();
        GLfloat color_triangulo2[] = {
        c2, c1, 0.0f, 1.0f,
        c2, c1, 0.0f, 1.0f,
        c2, c1, 0.0f, 1.0f,
        c2, c1, 0.0f, 1.0f,
        };
        genera_color();
        GLfloat color_triangulo3[] = {
        c4, 0, c1, 1.0f,
        c4, 0, c1, 1.0f,
        c4, 0,c1, 1.0f,
        c4, 0, c1, 1.0f,
        };

//**********************************************************************
```

Esto se realiza para cada uno de los triángulos.
La función para cambiar de color es la siguiente:

```
        //**** para cambiar el color *******
float c1= 1.0f, c2=0.0f, c3=0.0, c4=0.8;
void genera_color() {
        c1 = 0;
        c1 = 0.3 + (float)rand() / RAND_MAX;
        c2 = 0;
        c2 = 0.6 + (float)rand() / RAND_MAX;
        c3 = 0;
        c3 = 0.5 + (float)rand() / RAND_MAX;
        c4 = 0;
        c4 = 0.6 + (float)rand() / RAND_MAX;
}
```

También se modificó mesh.cpp para que recibiera el VAO del color.

```
void Mesh::CreateMesh(GLfloat *vertices, unsigned int *indices, unsigned
↪  int numOfVertices, unsigned int numberOfIndices, GLfloat *color,
↪  unsigned int tam_color)
{

        indexCount = numberOfIndices;
        glGenVertexArrays(1, &VAO); //generar 1 VAO
        glBindVertexArray(VAO);//asignar VAO

        glGenBuffers(1, &IBO);
        glBindBuffer(GL_ELEMENT_ARRAY_BUFFER, IBO);
```

```
        glBufferData(GL_ELEMENT_ARRAY_BUFFER, sizeof(indices[0]) *
        ↪  numberOfIndices, indices, GL_STATIC_DRAW);

        glGenBuffers(1, &VBO);
        glBindBuffer(GL_ARRAY_BUFFER, VBO);
        glBufferData(GL_ARRAY_BUFFER, sizeof(vertices[0]) *
        ↪  numOfVertices, vertices, GL_STATIC_DRAW);


        glVertexAttribPointer(0, 3, GL_FLOAT, GL_FALSE, 0, 0);
        glEnableVertexAttribArray(0);
        // VBO con el color
        glGenBuffers(1, &VBO);
        glBindBuffer(GL_ARRAY_BUFFER, VBO);
        glBufferData(GL_ARRAY_BUFFER, sizeof(color[0]) * tam_color,
        ↪  color, GL_STATIC_DRAW);
        glVertexAttribPointer(1, 4, GL_FLOAT, GL_FALSE, 0, 0);
        glEnableVertexAttribArray(1);

        glBindBuffer(GL_ARRAY_BUFFER, 0);
        glBindBuffer(GL_ELEMENT_ARRAY_BUFFER, 0);
        glBindVertexArray(0);

}
```

1.2.1 ejecución del código

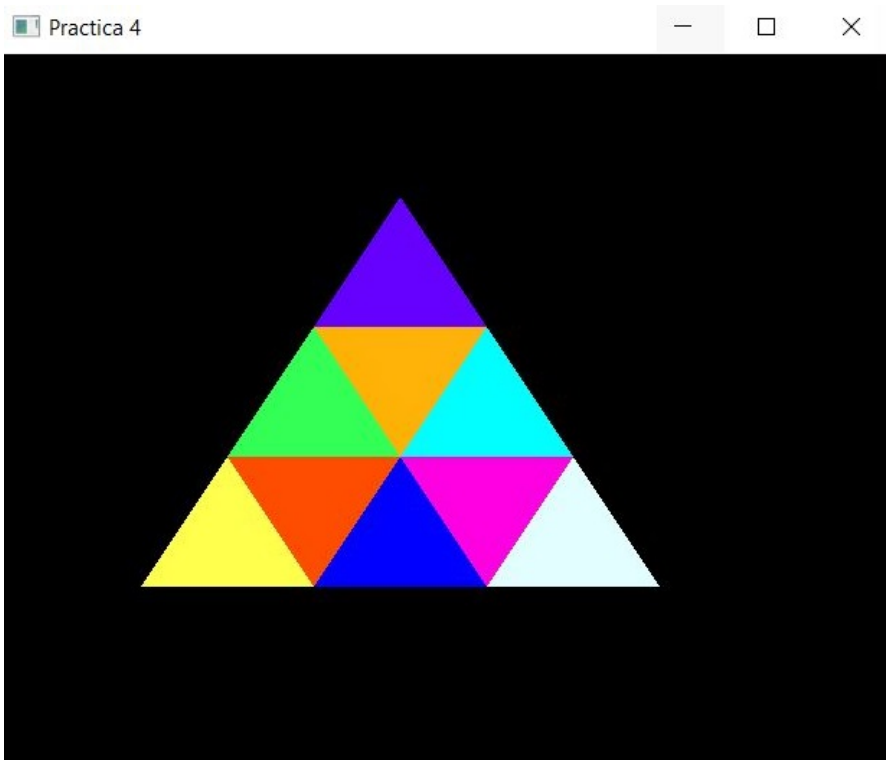

Figura 2

2 Código

Anexo a este documento, se encuentran los códigos creados para realizar esta práctica, tanto como los archivos .cpp como los .h.

3 Problemas presentados

No se presentaron problemas durante esta práctica.

4 Conclusiones

La realización de estos ejercicios no fue difícil, ya que utilizamos todo lo aprendido, lo único nuevo fue la creación de más objetos y la función ortho.

La parte que costó más trabajo fue la realización de la pirámide con los 9 triángulos ya que teníamos que desplazarlos y rotarlos de tal forma que no hubieran espacios en blanco.