



UNIVERSIDAD NACIONAL AUTÓNOMA DE
MÉXICO

FACULTAD DE INGENIERÍA

DIVISIÓN DE INGENIERÍA ELÉCTRICA

INGENIERÍA EN COMPUTACIÓN

LABORATORIO DE COMPUTACIÓN GRÁFICA e
INTERACCIÓN HUMANO COMPUTADORA



EJERCICIOS DE CLASE N° 02

NOMBRE COMPLETO: Aguilar Pérez José Ramón

N° de Cuenta: 317515048

GRUPO DE LABORATORIO: 02

GRUPO DE TEORÍA: 04

SEMESTRE 2025-2

FECHA DE ENTREGA LÍMITE: 19/febrero/2025

CALIFICACIÓN: _____

EJERCICIOS DE SESIÓN:

1. Actividades realizadas.

- Generar las figuras copiando los vértices de triangulorojo y cuadradoverde:
 - triángulo azul
 - triangulo verde (0,0.5,0)
 - cuadrado rojo
 - cuadrado verde
 - cuadrado café (0.478, 0.255, 0.067)

Dentro de la función *CrearLetrasyFiguras()* se añadieron los vértices con sus respectivos valores RGB. Cabe señalar que en el orden en el que se implementaron será su lugar en la lista *meshColor*, por lo que en la posición 0 está el triángulo azul, en la 1 el cuadrado rojo, en la 3 el triángulo verde, en la 4 el cuadrado verde y en la 5 el cuadrado café.

```
//0 Triangulo Azul
GLfloat vertices_letras[] = {
    //X      Y      Z      R      G      B
    -1.0f,  -1.0f,   0.5f,   0.0f,   0.0f,   1.0f,
    1.0f,   -1.0f,   0.5f,   0.0f,   0.0f,   1.0f,
    0.0f,   1.0f,   0.5f,   0.0f,   0.0f,   1.0f,
    /*1.0f,  1.0f,   0.5f,   1.0f,   0.0f,   0.0f,
    -1.0f,  1.0f,   0.5f,   1.0f,   0.0f,   0.0f,
    -1.0f,  -1.0f,   0.5f,   1.0f,   0.0f,   0.0f,*/
};

MeshColor* letras = new MeshColor();
letras->CreateMeshColor(vertices_letras, 18);
meshColorList.push_back(letras);

//1
GLfloat vertices_cuadradorojo[] = {
    //X      Y      Z      R      G      B
    -0.5f,  -0.5f,   0.5f,   1.0f,   0.0f,   0.0f,
    0.5f,   -0.5f,   0.5f,   1.0f,   0.0f,   0.0f,
    0.5f,   0.5f,   0.5f,   1.0f,   0.0f,   0.0f,
    -0.5f,  -0.5f,   0.5f,   1.0f,   0.0f,   0.0f,
    0.5f,   0.5f,   0.5f,   1.0f,   0.0f,   0.0f,
    -0.5f,  0.5f,   0.5f,   1.0f,   0.0f,   0.0f,
};

MeshColor* cuadradorojo = new MeshColor();
cuadradorojo->CreateMeshColor(vertices_cuadradorojo, 36);
meshColorList.push_back(cuadradorojo);
```

```
//2
GLfloat vertices_triangulorojo[] = {
    //X      Y      Z      R      G      B
    -1.0f,  -1.0f,   0.5f,   1.0f,   0.0f,   0.0f,
    1.0f,   -1.0f,   0.5f,   1.0f,   0.0f,   0.0f,
    0.0f,   1.0f,   0.5f,   1.0f,   0.0f,   0.0f,
};
MeshColor* triangulorojo = new MeshColor();
triangulorojo->CreateMeshColor(vertices_triangulorojo, 18);
meshColorList.push_back(triangulorojo);

//3
GLfloat vertices_trianguloverde[] = {
    //X      Y      Z      R      G      B
    -1.0f,  -1.0f,   0.5f,   0.0f,   0.5f,   0.0f,
    1.0f,   -1.0f,   0.5f,   0.0f,   0.5f,   0.0f,
    0.0f,   1.0f,   0.5f,   0.0f,   0.5f,   0.0f,
};
MeshColor* trianguloverde = new MeshColor();
trianguloverde->CreateMeshColor(vertices_trianguloverde, 18);
meshColorList.push_back(trianguloverde);
```

```
//4
GLfloat vertices_cuadradoverde[] = {
    //X      Y      Z      R      G      B
    -0.5f,  -0.5f,   0.5f,   0.0f,   1.0f,   0.0f,
    0.5f,   -0.5f,   0.5f,   0.0f,   1.0f,   0.0f,
    0.5f,   0.5f,   0.5f,   0.0f,   1.0f,   0.0f,
    -0.5f,  -0.5f,   0.5f,   0.0f,   1.0f,   0.0f,
    0.5f,   0.5f,   0.5f,   0.0f,   1.0f,   0.0f,
    -0.5f,  0.5f,   0.5f,   0.0f,   1.0f,   0.0f,
};
MeshColor* cuadradoverde = new MeshColor();
cuadradoverde->CreateMeshColor(vertices_cuadradoverde, 36);
meshColorList.push_back(cuadradoverde);

//5
GLfloat vertices_cuadradocafe[] = {
    //X      Y      Z      R      G      B
    -0.5f,  -0.5f,   0.5f,   0.478f, 0.255f, 0.067f,
    0.5f,   -0.5f,   0.5f,   0.478f, 0.255f, 0.067f,
    0.5f,   0.5f,   0.5f,   0.478f, 0.255f, 0.067f,
    -0.5f,  -0.5f,   0.5f,   0.478f, 0.255f, 0.067f,
    0.5f,   0.5f,   0.5f,   0.478f, 0.255f, 0.067f,
    -0.5f,  0.5f,   0.5f,   0.478f, 0.255f, 0.067f,
};
MeshColor* cuadradocafe = new MeshColor();
cuadradocafe->CreateMeshColor(vertices_cuadradocafe, 36);
meshColorList.push_back(cuadradocafe);
```

- Usando la proyección ortogonal generar el siguiente dibujo a partir de instancias de las figuras anteriormente creadas, recordar que todos se dibujan en el origen y por transformaciones geométricas se desplazan.

Se establece que se usará la proyección ortogonal para la generación de las figuras. Además, se cambió el color del fondo a blanco para distinguir mejor las figuras.

```
//Proyección: Matriz de Dimensión 4x4 para indicar si vemos en 2D( ortogonal)
glm::mat4 projection = glm::ortho(-1.0f, 1.0f, -1.0f, 1.0f, 0.1f, 100.0f); //
```

```
//Recibir eventos del usuario
glfwPollEvents();
//Limpiar la ventana
glClearColor(1.0f, 1.0f, 1.0f, 1.0f); //Fondo blanco
glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT); //
```

Dentro del ciclo *while*, se empiezan a generar las figuras. Cada vez que se quiera crear una figura diferente, se debe reiniciar la matriz 4x4 (*model=glm::mat4(1.0)*). Dependiendo del número asignado a *meshColorList*, será la figura que se generará. Las figuras se movieron de lugar con ayuda de *translate*, mientras que *scale* permitió cambiar el tamaño de cada una de las figuras.

```
//Generando triángulo azul para el techo de la casa
model = glm::mat4(1.0);
model = glm::translate(model, glm::vec3(0.0f, 0.5f, -3.0f)); //Traslacion de posicionamiento
model = glm::scale(model, glm::vec3(0.60f, 0.25f, 0.0f)); //Escalamiento de la figura
//
glUniformMatrix4fv(uniformModel, 1, GL_FALSE, glm::value_ptr(model)); //FALSE ES PARA QUE NO SE
glUniformMatrix4fv(uniformProjection, 1, GL_FALSE, glm::value_ptr(projection));
meshColorList[0]->RenderMeshColor();

//Generando cuadrado rojo para la casa
model = glm::mat4(1.0);
model = glm::translate(model, glm::vec3(0.0f, -0.2f, -4.0f)); //Traslacion de posicionamiento
model = glm::scale(model, glm::vec3(0.95f, 1.0f, 0.0f)); //Escalamiento de la figura
//
glUniformMatrix4fv(uniformModel, 1, GL_FALSE, glm::value_ptr(model)); //FALSE ES PARA QUE NO SE
glUniformMatrix4fv(uniformProjection, 1, GL_FALSE, glm::value_ptr(projection));
meshColorList[1]->RenderMeshColor();
```

```
//Generando los cuadrados verdes (ventanas y puerta)
//Ventana izquierda
model = glm::mat4(1.0);
model = glm::translate(model, glm::vec3(-0.2f, 0.0f, -3.0f)); //Traslacion de pos
model = glm::scale(model, glm::vec3(0.33f, 0.33f, 0.0f)); //Escalamiento de la f
//
glUniformMatrix4fv(uniformModel, 1, GL_FALSE, glm::value_ptr(model)); //FALSE ES
glUniformMatrix4fv(uniformProjection, 1, GL_FALSE, glm::value_ptr(projection));
meshColorList[4]->RenderMeshColor();
//Ventana derecha
model = glm::mat4(1.0);
model = glm::translate(model, glm::vec3(0.2f, 0.0f, -3.0f)); //Traslacion de pos
model = glm::scale(model, glm::vec3(0.33f, 0.33f, 0.0f)); //Escalamiento de la f
//
glUniformMatrix4fv(uniformModel, 1, GL_FALSE, glm::value_ptr(model)); //FALSE ES
glUniformMatrix4fv(uniformProjection, 1, GL_FALSE, glm::value_ptr(projection));
meshColorList[4]->RenderMeshColor();
//Puerta
model = glm::mat4(1.0);
model = glm::translate(model, glm::vec3(0.0f, -0.535f, -3.0f)); //Traslacion de p
model = glm::scale(model, glm::vec3(0.33f, 0.33f, 0.0f)); //Escalamiento de la f
//
glUniformMatrix4fv(uniformModel, 1, GL_FALSE, glm::value_ptr(model)); //FALSE ES
glUniformMatrix4fv(uniformProjection, 1, GL_FALSE, glm::value_ptr(projection));
meshColorList[4]->RenderMeshColor();
```

```

//Generando los triangulos verde 0.5f
//Arbol izquierdo
model = glm::mat4(1.0);
model = glm::translate(model, glm::vec3(-0.75f, -0.25f, -3.0f)); //Traslacion de posicionamiento
model = glm::scale(model, glm::vec3(0.2f, 0.25f, 0.0f)); //Escalamiento de la figura
//
glUniformMatrix4fv(uniformModel, 1, GL_FALSE, glm::value_ptr(model)); //FALSE ES PARA QUE NO SEA
glUniformMatrix4fv(uniformProjection, 1, GL_FALSE, glm::value_ptr(projection));
meshColorList[3]->RenderMeshColor();
//Arbol derecho
model = glm::mat4(1.0);
model = glm::translate(model, glm::vec3(0.75f, -0.25f, -3.0f)); //Traslacion de posicionamiento
model = glm::scale(model, glm::vec3(0.2f, 0.25f, 0.0f)); //Escalamiento de la figura
//
glUniformMatrix4fv(uniformModel, 1, GL_FALSE, glm::value_ptr(model)); //FALSE ES PARA QUE NO SEA
glUniformMatrix4fv(uniformProjection, 1, GL_FALSE, glm::value_ptr(projection));
meshColorList[3]->RenderMeshColor();

//Generando los cuadrados cafes
//Tronco izquierdo
model = glm::mat4(1.0);
model = glm::translate(model, glm::vec3(-0.75f, -0.6f, -3.0f)); //Traslacion de posicionamiento
model = glm::scale(model, glm::vec3(0.2f, 0.2f, 0.0f)); //Escalamiento de la figura
//
glUniformMatrix4fv(uniformModel, 1, GL_FALSE, glm::value_ptr(model)); //FALSE ES PARA QUE NO SEA
glUniformMatrix4fv(uniformProjection, 1, GL_FALSE, glm::value_ptr(projection));
meshColorList[5]->RenderMeshColor();
//Tronco derecho
//Tronco izquierdo
model = glm::mat4(1.0);
model = glm::translate(model, glm::vec3(0.75f, -0.6f, -3.0f)); //Traslacion de posicionamiento
model = glm::scale(model, glm::vec3(0.2f, 0.2f, 0.0f)); //Escalamiento de la figura
//
glUniformMatrix4fv(uniformModel, 1, GL_FALSE, glm::value_ptr(model)); //FALSE ES PARA QUE NO SEA
glUniformMatrix4fv(uniformProjection, 1, GL_FALSE, glm::value_ptr(projection));
meshColorList[5]->RenderMeshColor();

```

Finalmente, se obtuvo el dibujo solicitado de una casa con ventanas, puerta y dos arbolitos al lado.



2. Problemas presentados.

No se presentaron problemas a la hora de realizar el ejercicio.

3. Conclusión

a. Los ejercicios de la clase: Complejidad, explicación

Los ejercicios solicitados se me hicieron de una complejidad razonable, ya que la explicación del funcionamiento de las listas y los índices para generar las figuras fue precisa y clara, por lo que a la hora de implementar más figuras no hubo contratiempos notorios. Además, gracias a que se proporcionó la estructura de las funciones que permiten la traslación y escalamiento de las figuras, se pudo llevar a cabo el ejercicio de manera exitosa.

b. Comentarios generales: Faltó explicar a detalle, ir más lento en alguna explicación, otros comentarios y sugerencias.

La explicación general del funcionamiento del programa me pareció buena, ya que fue preciso y claro sobre cómo se generan los vértices e índices para la construcción de las figuras. Me pareció interesante como se empiezan a generar estas figuras a través de matrices 4x4 y también, como afecta el orden en el que se declaran los vértices de las figuras a la hora de llenar la lista de *meshColor*.