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 Nº 02

NOMBRE COMPLETO: Barragán Pilar Diana

Nº de Cuenta: 318147981

GRUPO DE LABORATORIO: 03

GRUPO DE TEORÍA: 04

SEMESTRE 2025-1

FECHA DE ENTREGA LÍMITE: 20 de agosto de 2023

CALIFICACIÓN: _____

Ejercicio de clase práctica 2: Proyecciones, Transformaciones geométricas.

1. **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)**

Para conseguir estas figuras lo primero que realice fue copiar y pegar el código que ya se nos había proporcionado con anterioridad, claro que realizando modificaciones en el nombre de las variables y en lo más importante que es el apartado para colocar el código en RGB en la declaración de las matrices de vértices de las figuras.

De igual manera algo esencial a resaltar es que la figura o matriz se va a guardar en una lista que declaramos como `std::vector<MeshColor*> meshColorList`, por lo que el orden en que declaramos las figuras importa pues de este depende el índice que tendrá en la lista y por consiguiente para dibujar la figura en el `while` debemos de poner el índice de la lista `meshColorList` donde se encuentra.

```
//TRIÁNGULO AZUL
//Copiamos el codigo del triángulo rojo cambiando el rgb para que sea azul
GLfloat vertices_trianguloazul[] = {
    //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,

};

MeshColor* trianguloazul = new MeshColor();
trianguloazul->CreateMeshColor(vertices_trianguloazul, 18);
meshColorList.push_back(trianguloazul);// indice 2 en el meshColor

//TRIÁNGULO VERDE
//Copiamos el codigo del triángulo rojo cambiando el rgb para que sea verde dado el que se nos dio
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);// indice 3 en el meshColor
```

Figura 1. Declaración de los triángulos azul y verde.

```
//CUADRADO ROJO
//Copiamos el codigo del cuadrado verde cambiando el rgb para que sea verde dado el que se nos dio
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);// indice 5 en el meshColor

//CUADRADO CAFE
//Copiamos el codigo del cuadrado verde cambiando el rgb para que sea verde dado el que se nos dio
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);// indice 6 en el meshColor
```

Figura 2. Declaración de los cuadrados rojo y café.

```
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);// indice 4 en el meshColor
```

Figura 3. Declaración del cuadrado verde.

Para poder ver la figura en la pantalla primero debemos de inicializar la matriz de transformación que tendrá dimensiones de 4x4, la cual será una matriz identidad que tiene en la diagonal principal el valor de 1 y en lo demás 0. Posteriormente utilice las funciones de translación y escala para que todas las figuras se presenten juntas en una misma pantalla, con `glm::translate` moví los objetos para que se

pudieran presentar en orden y con `glm::scale` logre que los objetos tuvieran casi el mismo tamaño cambiando que tan cerca o lejos están en el plano xyz.

EJECUCIÓN DEL PROGRAMA:

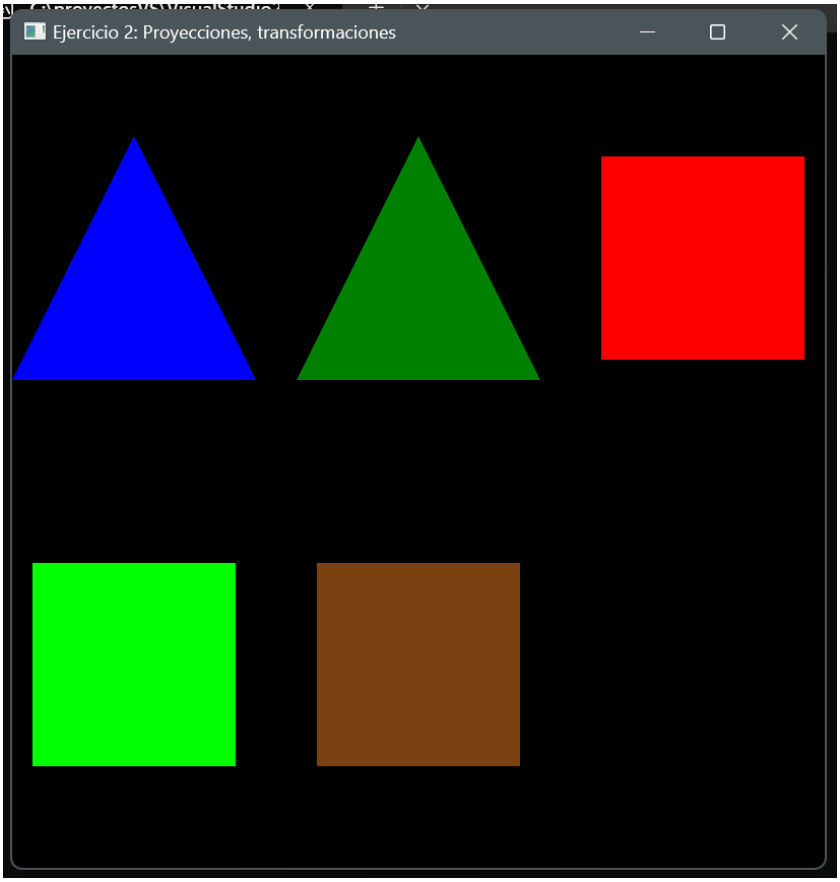

Figura 4. Cuadrados y triángulos dibujados en la pantalla.

2. **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.**

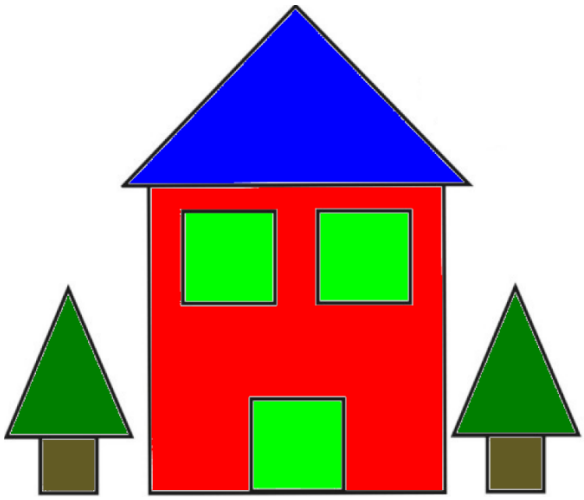

Figura 5. Dibujo a generar.

En este ejercicio al ya tener las figuras hechas y declaradas en el ejercicio anterior lo más importante era definir en que parte de la pantalla se debía establecer cada figura, para ello primero realice un boceto en el cual obtengo las coordenadas de origen en donde se colocan cada una de las formas geométricas.

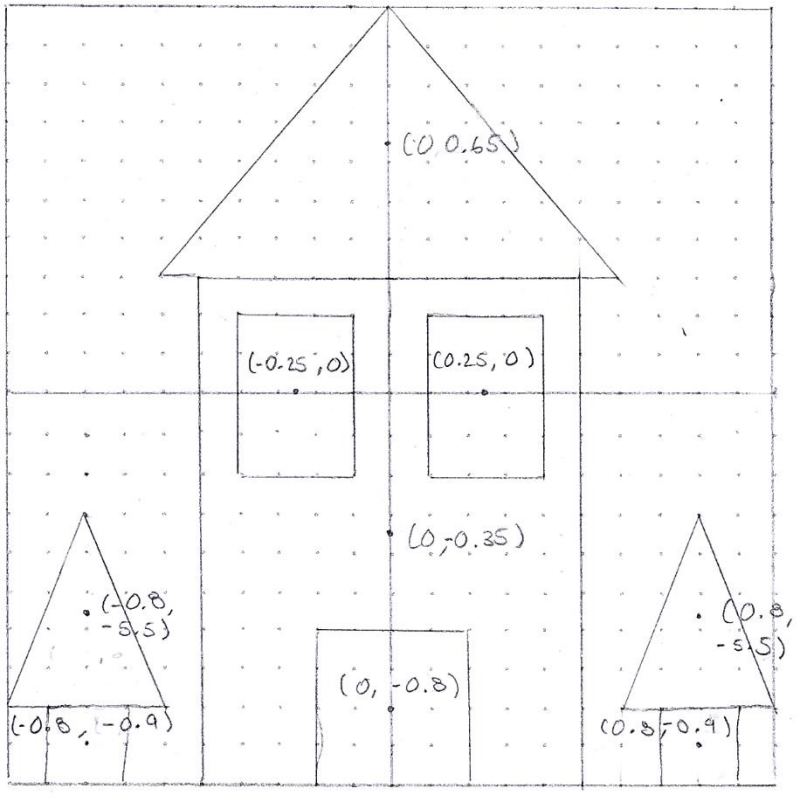

Figura 6. Dibujo preliminar con coordenadas.

Para lograr que las figuras se dibujen en el lugar correcto utilice la función de `glm::translate` en la que debemos pasar la matriz de transformaciones geométricas llamada `model` y un vector de coordenadas que es la posición hacía la que se va a mover nuestra figura, después para que las figuras tuvieran el tamaño que se requiere para formar el dibujo utilice la función `glm::scale` a la que igual se le pasa la matriz `model` y un vector en el cual el numero que se ponga es al que se va a aumentar o disminuir el tamaño de la figura geométrica. Finalmente dibujamos la figura con la línea `meshColorList[2]->RenderMeshColor()` recordando cual es el índice en la lista `meshColor`.

```
//Para las letras hay que usar el segundo set de shaders con índice 1 en ShaderList
shaderList[1].useShader();
uniformModel = shaderList[1].getModelLocation();
uniformProjection = shaderList[1].getProjectLocation();

//***TRIÁNGULO AZUL***
//Inicializar matriz de dimensión 4x4 que servirá como matriz de modelo para almacenar
//las transformaciones geométricas
model = glm::mat4(1.0); //Se crea una matriz de 4x4 identidad es decir que la diagonal
//principal es de 1 y lo demas 0
model = glm::translate(model, glm::vec3(0.0f, 0.65f, -4.0f));//Se translada la matriz model
//es decir el objeto se translada a otro lado dependiendo de las coordenadas del vector
model = glm::scale(model, glm::vec3(0.7f, 0.3f, 0.3f));//Con esta matriz hacemos que un
//objeto se vea mas lejano o cercano

glUniformMatrix4fv(uniformModel, 1, GL_FALSE, glm::value_ptr(model));
//FALSE ES PARA QUE NO SEA TRANSPUESTA y se envían al shader como variables de tipo uniform
glUniformMatrix4fv(uniformProjection, 1, GL_FALSE, glm::value_ptr(projection));
meshColorList[2]->RenderMeshColor();
```

Figura 7. Translación y escalado del triángulo azul.

```
//***CUADRADO VERDE***
model = glm::mat4(1.0);
model = glm::translate(model, glm::vec3(-0.25f, 0.0f, -2.0f));
model = glm::scale(model, glm::vec3(0.4f, 0.4f, 0.4f));

glUniformMatrix4fv(uniformModel, 1, GL_FALSE, glm::value_ptr(model));
glUniformMatrix4fv(uniformProjection, 1, GL_FALSE, glm::value_ptr(projection));
meshColorList[4]->RenderMeshColor();

model = glm::mat4(1.0);
model = glm::translate(model, glm::vec3(0.25f, 0.0f, -2.0f));
model = glm::scale(model, glm::vec3(0.4f, 0.4f, 0.4f));

glUniformMatrix4fv(uniformModel, 1, GL_FALSE, glm::value_ptr(model));
glUniformMatrix4fv(uniformProjection, 1, GL_FALSE, glm::value_ptr(projection));
meshColorList[4]->RenderMeshColor();

model = glm::mat4(1.0);
model = glm::translate(model, glm::vec3(0.0f, -0.8f, -2.0f));
model = glm::scale(model, glm::vec3(0.4f, 0.4f, 0.4f));

glUniformMatrix4fv(uniformModel, 1, GL_FALSE, glm::value_ptr(model));
glUniformMatrix4fv(uniformProjection, 1, GL_FALSE, glm::value_ptr(projection));
meshColorList[4]->RenderMeshColor();
```

Figura 8. Translación y escalado del cuadrado verde.

```
//***CUADRADO ROJO***
model = glm::mat4(1.0);
model = glm::translate(model, glm::vec3(0.0f, -0.35f, -4.0f));
model = glm::scale(model, glm::vec3(1.0f, 1.4f, 1.0f));

glUniformMatrix4fv(uniformModel, 1, GL_FALSE, glm::value_ptr(model));
glUniformMatrix4fv(uniformProjection, 1, GL_FALSE, glm::value_ptr(projection));
meshColorList[5]->RenderMeshColor();

//***TRIÁNGULO VERDE***
model = glm::mat4(1.0);
model = glm::translate(model, glm::vec3(-0.8f, -0.55f, -4.0f));
model = glm::scale(model, glm::vec3(0.2f, 0.3f, 0.3f));

glUniformMatrix4fv(uniformModel, 1, GL_FALSE, glm::value_ptr(model));
glUniformMatrix4fv(uniformProjection, 1, GL_FALSE, glm::value_ptr(projection));
meshColorList[3]->RenderMeshColor();

model = glm::mat4(1.0);
model = glm::translate(model, glm::vec3(0.8f, -0.55f, -4.0f));
model = glm::scale(model, glm::vec3(0.2f, 0.3f, 0.3f));

glUniformMatrix4fv(uniformModel, 1, GL_FALSE, glm::value_ptr(model));
glUniformMatrix4fv(uniformProjection, 1, GL_FALSE, glm::value_ptr(projection));
meshColorList[3]->RenderMeshColor();
```

Figura 9. Translación y escalado del cuadrado rojo y triangulo verde.

```
//***CUADRADO CAFE***
model = glm::mat4(1.0);
model = glm::translate(model, glm::vec3(-0.8f, -0.9f, -4.0f));
model = glm::scale(model, glm::vec3(0.2f, 0.2f, 0.2f));

glUniformMatrix4fv(uniformModel, 1, GL_FALSE, glm::value_ptr(model));
glUniformMatrix4fv(uniformProjection, 1, GL_FALSE, glm::value_ptr(projection));
meshColorList[6]->RenderMeshColor();

model = glm::mat4(1.0);
model = glm::translate(model, glm::vec3(0.8f, -0.9f, -4.0f));
model = glm::scale(model, glm::vec3(0.2f, 0.2f, 0.2f));

glUniformMatrix4fv(uniformModel, 1, GL_FALSE, glm::value_ptr(model));
glUniformMatrix4fv(uniformProjection, 1, GL_FALSE, glm::value_ptr(projection));
meshColorList[6]->RenderMeshColor();
```

Figura 10. Translación y escalado del cuadrado café.

Para este ejercicio solo tuve un inconveniente puesto que al compilar el programa una primera vez, el proceso si se realizaba correctamente, sin embargo, al querer repetir este proceso sin cambiarle nada al código me decía que el archivo contenía errores y al revisarlo todo estaba en orden, por lo que elimine el proyecto y solo copie y pegue el mismo código en el nuevo proyecto en el que volvió a compilar correctamente.

CONCLUSIONES:

Los ejercicios que se dejaron para que realizáramos después de haber sido explicado el código, me parecieron muy acordes al nivel que vimos en la sesión de laboratorio, puesto que me permitieron comprender mejor la funcionalidad tanto de la declaración de figuras con las matrices de vértices y el guardar estas en una lista, así como la manera en que estas figuras se dibujaban, con ello el funcionamiento de las funciones de translación y escalado para poder ubicar varias figuras en una misma pantalla en distintos puntos, cosa en que difiere de lo antes ejecutado pues hasta este momento solo había estado presente un solo tipo de figura que eran los triángulos.

El tener que editar todas estas partes del código antes mencionadas para lograr cumplir con las actividades establecidas satisfactoriamente me ayudo a familiarizarme más con el código y entender algunas transformaciones geométricas.