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: Vargas López Miguel Adán

Nº de Cuenta: 421079522

GRUPO DE LABORATORIO: 3

GRUPO DE TEORÍA: 4

SEMESTRE 2025-1

FECHA DE ENTREGA LÍMITE: 20/08/2024

CALIFICACIÓN: _____

Ejercicio 1. generar las figuras copiando los vértices de triangulorojo y cuadradoverde:

-triángulo azul

-triángulo verde (0,0.5,0)

-cuadrado rojo

-cuadrado verde

-cuadrado café (0.478, 0.255, 0.067)

Primero en el método de CrearLetrasyFiguras añadimos la declaración de cada una de las figuras enlistadas, como los triángulos y los cuadrados.

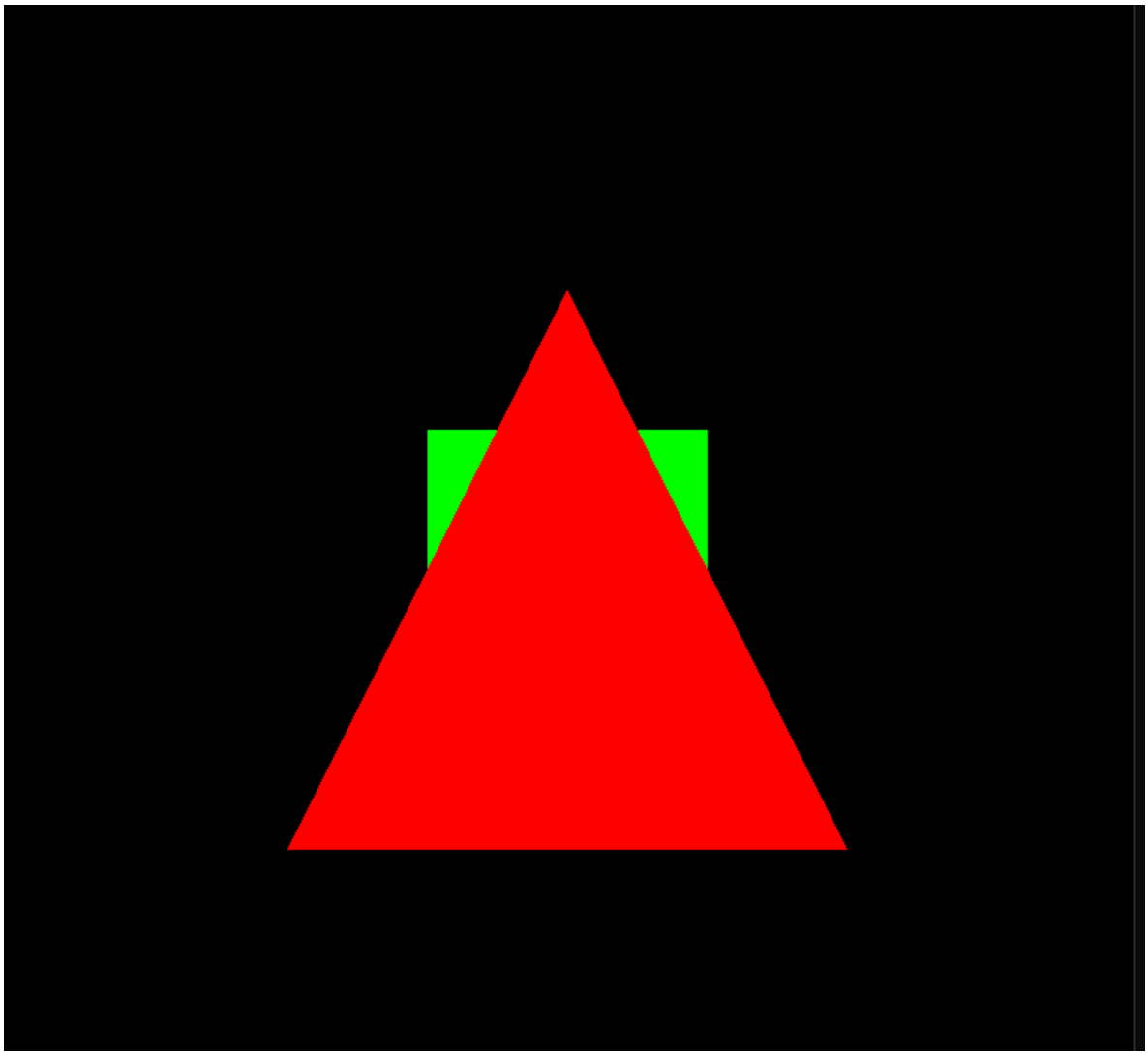
```
GLfloat vertices_triangulorojo[] = {
    //X         Y           Z           R       G       B
    -1.0f,  -1.0f,      0.5f,           1.0f,   0.0f,   0.0f,
    1.0f,   -1.0f,      0.5f,           1.0f,   0.0f,   0.0f,
    0.0f,   1.0f,       0.5f,           1.0f,   0.0f,   0.0f,

};

MeshColor* triangulorojo = new MeshColor();
triangulorojo->CreateMeshColor(vertices_triangulorojo, 18);
meshColorList.push_back(triangulorojo);

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, 36);
meshColorList.push_back(trianguloazul);

GLfloat vertices_trianguloverde[] = {
    //X         Y           Z           R       G       B
    -1.0f,  -1.0f,      0.5f,           0.0f,   1.0f,   0.0f,
    1.0f,   -1.0f,      0.5f,           0.0f,   1.0f,   0.0f,
    0.0f,   1.0f,       0.5f,           0.0f,   1.0f,   0.0f,

};
```

```
MeshColor* cuadradoverde = new MeshColor();
cuadradoverde->CreateMeshColor(vertices_cuadradoverde, 72);
meshColorList.push_back(cuadradoverde);

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, 90);
meshColorList.push_back(cuadradorojo);

GLfloat vertices_cuadradocafe[] = {
    //X         Y           Z           R       G       B
    -0.5f,  -0.5f,      0.5f,           0.478, 0.255, 0.067,
    0.5f,   -0.5f,      0.5f,           0.478, 0.255, 0.067,
    0.5f,   0.5f,       0.5f,           0.478, 0.255, 0.067,
    -0.5f,  -0.5f,      0.5f,           0.478, 0.255, 0.067,
    0.5f,   0.5f,       0.5f,           0.478, 0.255, 0.067,
    -0.5f,  0.5f,       0.5f,           0.478, 0.255, 0.067,

};
```

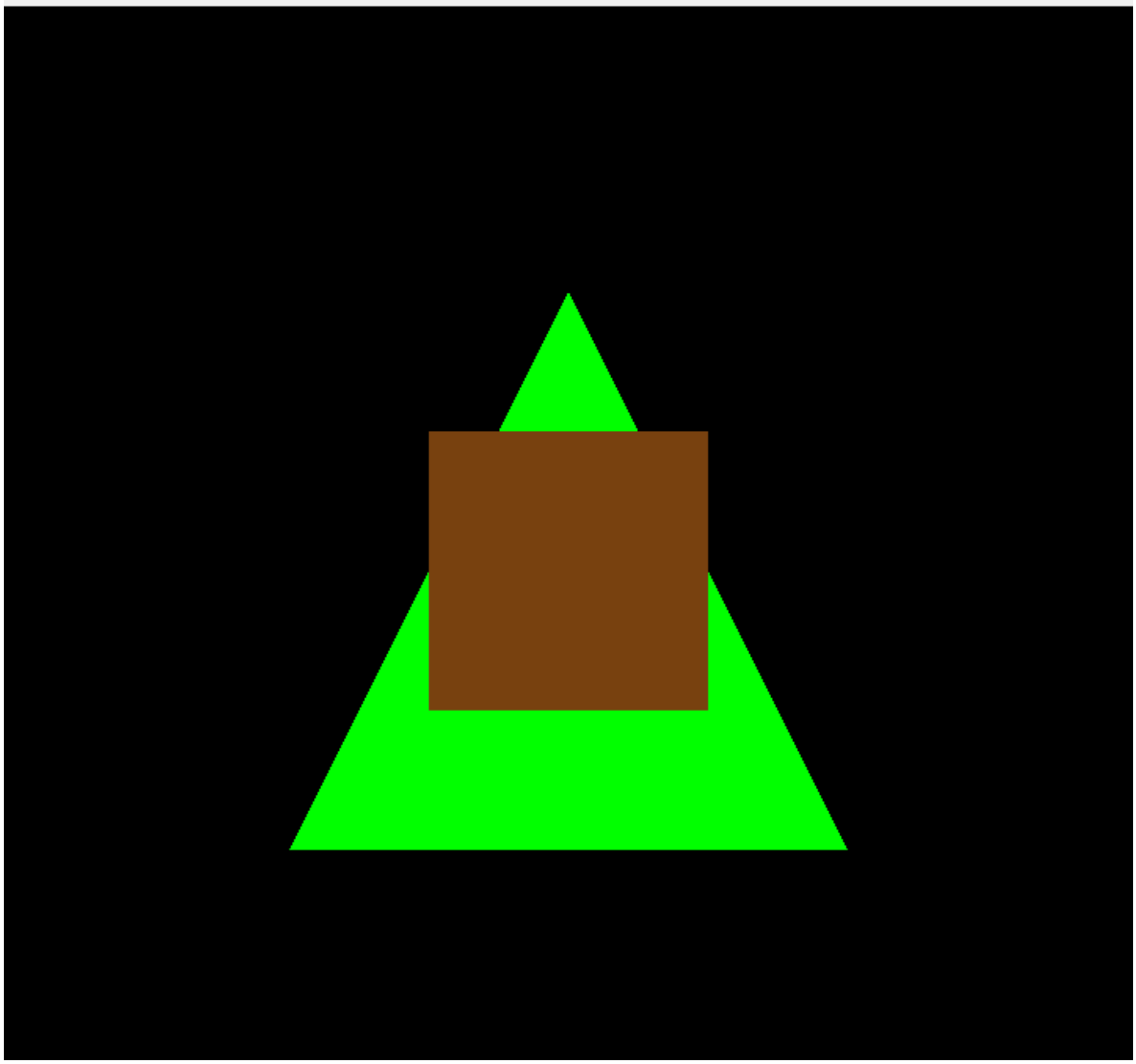
Y después en el bucle while del main, renderizamos cada una de estas figuras, a través de sus índices, que corresponden al orden de cómo las declaramos anteriormente.

```
// Renderizar letras y figuras
meshColorList[0]->RenderMeshColor(); // Triangulo rojo
meshColorList[1]->RenderMeshColor(); // Triangulo azul
meshColorList[2]->RenderMeshColor(); // Triangulo verde
meshColorList[3]->RenderMeshColor(); // Cuadrado verde
meshColorList[4]->RenderMeshColor(); // Cuadrado rojo
meshColorList[5]->RenderMeshColor(); // Cuadrado cafe
```

Asimismo, en la visualización del programa se verán las figuras en el orden en que fueron renderizados, los primeros de cada figura serán los primeros en verse, por ejemplo, con el anterior orden, el resultado sería el siguiente:



Pero si cambiáramos el orden en que se renderizan, sería el resultado siguiente:



Donde ahora el cuadrado se va al frente al ser el primero en renderizar.

Ejercicio 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

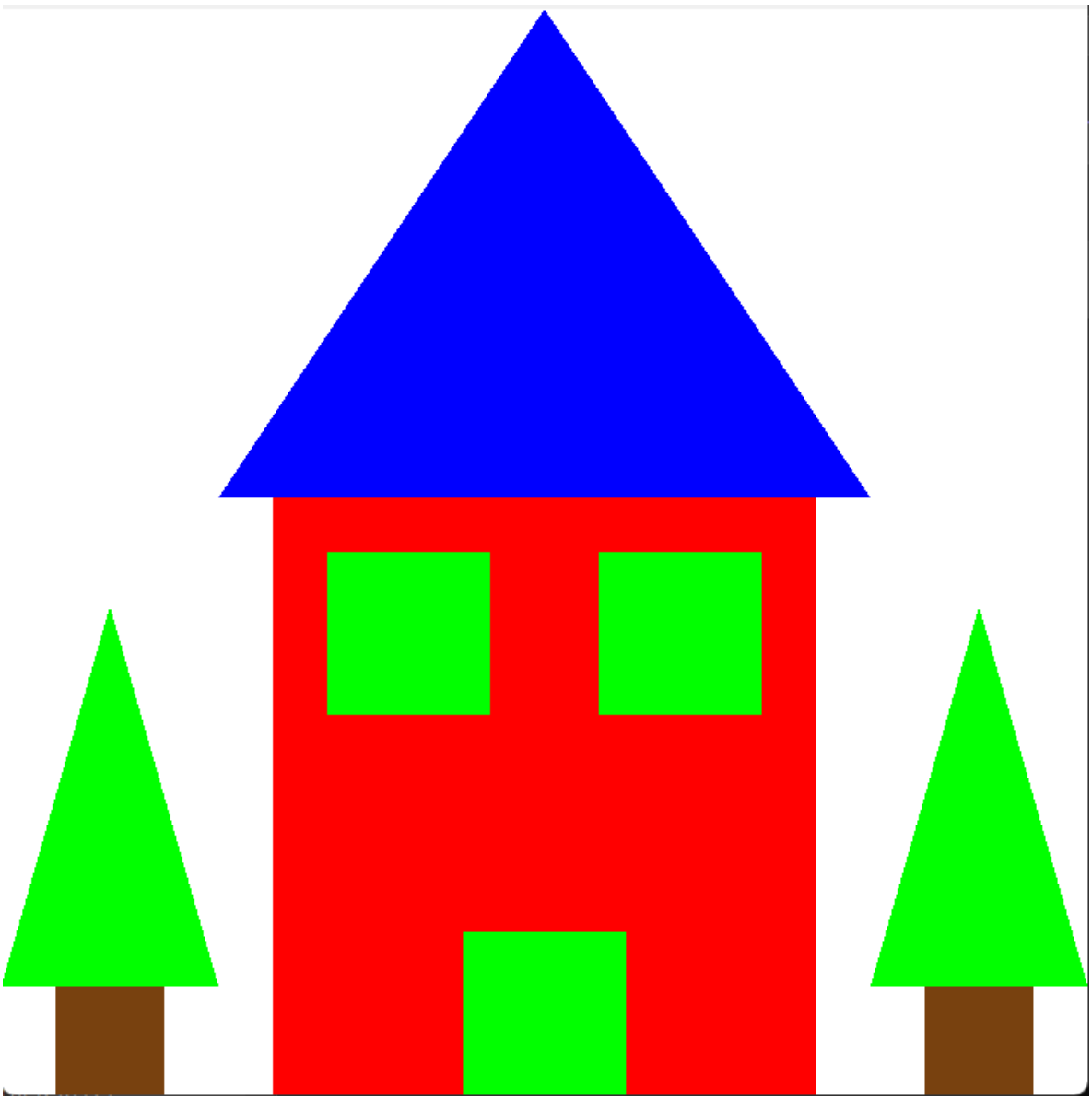
Primero dentro del método de CrearLetrasYFiguras definimos cada figura que va dibujarse con su respectivo color, con el único detalle de que los cuadrados verdes deben ser antes del cuadrado rojo ya que ese debe estar al fondo.

```
GLfloat vertices_trianguloazul[] = { ... }

MeshColor* trianguloazul = new MeshColor();
trianguloazul->CreateMeshColor(vertices_trianguloazul, 18);
meshColorList.push_back(trianguloazul);

GLfloat vertices_cuadradocafe1[] = { ... }

MeshColor* cuadradocafe1 = new MeshColor();
cuadradocafe1->CreateMeshColor(vertices_cuadradocafe1, 54);
meshColorList.push_back(cuadradocafe1);

GLfloat vertices_cuadradocafe2[] = { ... }

MeshColor* cuadradocafe2 = new MeshColor();
cuadradocafe2->CreateMeshColor(vertices_cuadradocafe2, 90);
meshColorList.push_back(cuadradocafe2);

GLfloat vertices_cuadradoverde[] = { ... }

MeshColor* cuadradoverde = new MeshColor();
cuadradoverde->CreateMeshColor(vertices_cuadradoverde, 126);
meshColorList.push_back(cuadradoverde);

GLfloat vertices_cuadradoverde2[] = { ... }

MeshColor* cuadradoverde2 = new MeshColor();
cuadradoverde2->CreateMeshColor(vertices_cuadradoverde2, 162);
meshColorList.push_back(cuadradoverde2);

GLfloat vertices_cuadradoverde3[] = { ... }
```

Y después renderizarlos en orden.

```
// Renderizar letras y figuras

meshColorList[0]->RenderMeshColor();
meshColorList[1]->RenderMeshColor();
meshColorList[2]->RenderMeshColor();
meshColorList[3]->RenderMeshColor();
meshColorList[4]->RenderMeshColor();
meshColorList[5]->RenderMeshColor();
meshColorList[6]->RenderMeshColor();
meshColorList[7]->RenderMeshColor();
meshColorList[8]->RenderMeshColor();

glUseProgram(0);
```

Para tener el siguiente resultado:



Conclusión:

Si bien este reporte no tuvo gran complejidad, dejó entrever algunos datos interesantes para trabajar con OpenGL, empezamos a dibujar más figuras y con ello entender que es relevante tanto la proyección con la cuál estamos trabajando, en este caso de ejercicios la proyección ortogonales resulta útil para renderizados en 2D, además de ver que el orden en el que son renderizados los objetos puede afectar a la vista final.