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 02

NOMBRE COMPLETO: Rosas Martinez Francisco Javier

Nº de Cuenta: 320109766

GRUPO DE LABORATORIO: 03

GRUPO DE TEORÍA: 05

SEMESTRE 2026-1

FECHA DE ENTREGA LÍMITE: 30/Agosto/2025

CALIFICACIÓN: _____

REPORTE DE PRÁCTICA:

1.- Ejecución de los ejercicios que se dejaron, comentar cada uno y capturas de pantalla de bloques de código generados y de ejecución del programa.

Ejercicio 1:

1.- Dibujar las iniciales de sus nombres, cada letra de un color diferente

Para resolver el ejercicio tuve que crear cada letra en la función *CrearLetrasyFiguras* utilizando las mismas coordenadas de la práctica pasada agregando los valores de los colores RGB, por último en la función *main* instancia cada figura para poderla mostrar en la ventana.

Bloques de código generado

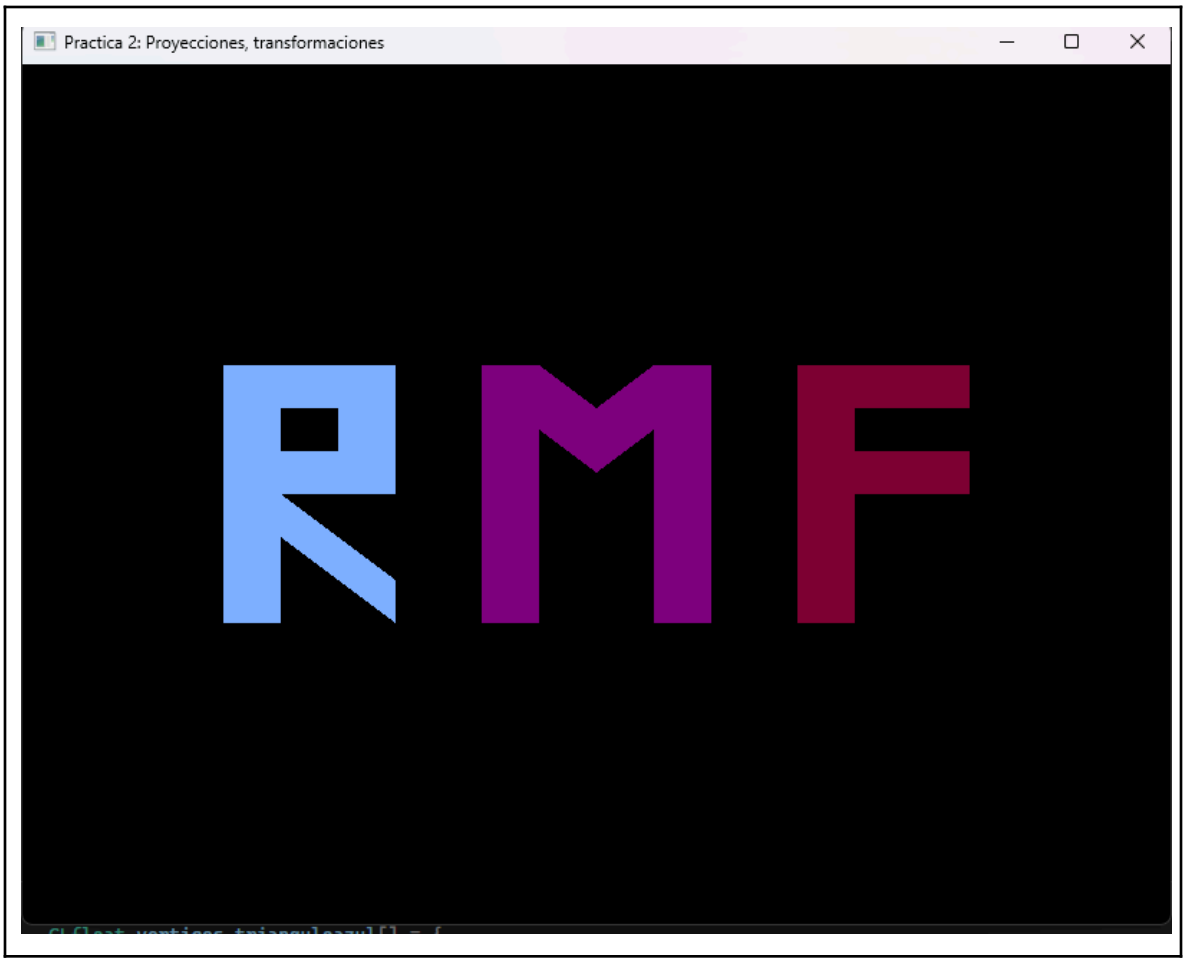
```
GLfloat vertices_F[] = {
                        //  Color vino //
    0.35f,-0.3f,0.5f,   0.5f,0.0f,0.2f,
    0.35f,0.3f,0.5f,    0.5f,0.0f,0.2f,
    0.45f,0.3f,0.5f,    0.5f,0.0f,0.2f,

    0.35f,-0.3f,0.5f,   0.5f,0.0f,0.2f,
    0.45f,0.3f,0.5f,    0.5f,0.0f,0.2f,
    0.45f,-0.3f,0.5f,   0.5f,0.0f,0.2f,

    0.45f,0.0f,0.5f,    0.5f,0.0f,0.2f,
    0.65f,0.0f,0.5f,    0.5f,0.0f,0.2f,
    0.45f,0.1f,0.5f,    0.5f,0.0f,0.2f,

    0.65f,0.0f,0.5f,    0.5f,0.0f,0.2f,
    0.45f,0.1f,0.5f,    0.5f,0.0f,0.2f,
    0.65f,0.1f,0.5f,    0.5f,0.0f,0.2f,

    0.45f,0.2f,0.5f,    0.5f,0.0f,0.2f,
    0.65f,0.2f,0.5f,    0.5f,0.0f,0.2f,
    0.45f,0.3f,0.5f,    0.5f,0.0f,0.2f,

    0.65f,0.2f,0.5f,    0.5f,0.0f,0.2f,
    0.45f,0.3f,0.5f,    0.5f,0.0f,0.2f,
    0.65f,0.3f,0.5f,    0.5f,0.0f,0.2f,

};

MeshColor* letraF = new MeshColor();
letraF->CreateMeshColor(vertices_F, 108);
meshColorList.push_back(letraF);
```

```
GLfloat vertices_R[] = {
                        // Color azul claro //
    -0.65f,-0.3f,0.5f,   0.5f,0.7f,1.0f,
    -0.65f,0.3f,0.5f,    0.5f,0.7f,1.0f,
    -0.55f,0.3f,0.5f,    0.5f,0.7f,1.0f,

    -0.65f,-0.3f,0.5f,   0.5f,0.7f,1.0f,
    -0.55f,0.3f,0.5f,    0.5f,0.7f,1.0f,
    -0.55f,-0.3f,0.5f,   0.5f,0.7f,1.0f,

    -0.55f,-0.1f,0.5f,   0.5f,0.7f,1.0f,
    -0.35f,-0.3f,0.5f,   0.5f,0.7f,1.0f,
    -0.35f,-0.2f,0.5f,   0.5f,0.7f,1.0f,

    -0.35f,-0.2f,0.5f,   0.5f,0.7f,1.0f,
    -0.55f,-0.1f,0.5f,   0.5f,0.7f,1.0f,
    -0.55f,-0.0f,0.5f,   0.5f,0.7f,1.0f,

    -0.55f,-0.0f,0.5f,   0.5f,0.7f,1.0f,
    -0.35f,-0.0f,0.5f,   0.5f,0.7f,1.0f,
    -0.55f,0.1f,0.5f,    0.5f,0.7f,1.0f,

    -0.35f,-0.0f,0.5f,   0.5f,0.7f,1.0f,
    -0.55f,0.1f,0.5f,    0.5f,0.7f,1.0f,
    -0.35f,0.1f,0.5f,    0.5f,0.7f,1.0f,

    -0.55f,0.2f,0.5f,    0.5f,0.7f,1.0f,
    -0.35f,0.2f,0.5f,    0.5f,0.7f,1.0f,
    -0.55f,0.3f,0.5f,    0.5f,0.7f,1.0f,

    -0.35f,0.2f,0.5f,    0.5f,0.7f,1.0f,
    -0.55f,0.3f,0.5f,    0.5f,0.7f,1.0f,
    -0.35f,0.3f,0.5f,    0.5f,0.7f,1.0f,

    -0.35f,0.1f,0.5f,    0.5f,0.7f,1.0f,
    -0.45f,0.1f,0.5f,    0.5f,0.7f,1.0f,
    -0.45f,0.2f,0.5f,    0.5f,0.7f,1.0f,

    -0.35f,0.1f,0.5f,    0.5f,0.7f,1.0f,
    -0.35f,0.2f,0.5f,    0.5f,0.7f,1.0f,
    -0.45f,0.2f,0.5f,    0.5f,0.7f,1.0f,

};

MeshColor* letraR = new MeshColor();
letraR->CreateMeshColor(vertices_R, 180);
meshColorList.push_back(letraR);
```

```
float vertices_M[] = {
    -0.2f,-0.3f,0.5f,   0.5f,0.0f,0.5f,
    -0.1f,-0.3f,0.5f,   0.5f,0.0f,0.5f,
    -0.2f, 0.3f,0.5f,   0.5f,0.0f,0.5f,

    -0.1f,-0.3f,0.5f,   0.5f,0.0f,0.5f,
    -0.2f, 0.3f,0.5f,   0.5f,0.0f,0.5f,
    -0.1f, 0.3f,0.5f,   0.5f,0.0f,0.5f,

    -0.1f, 0.3f,0.5f,   0.5f,0.0f,0.5f,
     0.0f, 0.2f,0.5f,   0.5f,0.0f,0.5f,
    -0.1f,0.15f,0.5f,   0.5f,0.0f,0.5f,

     0.0f, 0.2f,0.5f,   0.5f,0.0f,0.5f,
    -0.1f,0.15f,0.5f,   0.5f,0.0f,0.5f,
     0.0f,0.05f,0.5f,   0.5f,0.0f,0.5f,

     0.2f,-0.3f,0.5f,   0.5f,0.0f,0.5f,
     0.1f,-0.3f,0.5f,   0.5f,0.0f,0.5f,
     0.2f, 0.3f,0.5f,   0.5f,0.0f,0.5f,

     0.1f,-0.3f,0.5f,   0.5f,0.0f,0.5f,
     0.2f, 0.3f,0.5f,   0.5f,0.0f,0.5f,
     0.1f, 0.3f,0.5f,   0.5f,0.0f,0.5f,

     0.1f, 0.3f,0.5f,   0.5f,0.0f,0.5f,
     0.0f, 0.2f,0.5f,   0.5f,0.0f,0.5f,
     0.1f,0.15f,0.5f,   0.5f,0.0f,0.5f,

     0.0f, 0.2f,0.5f,   0.5f,0.0f,0.5f,
     0.1f,0.15f,0.5f,   0.5f,0.0f,0.5f,
     0.0f,0.05f,0.5f,   0.5f,0.0f,0.5f
};


MeshColor* letraM = new MeshColor();
letraM->CreateMeshColor(vertices_M, 144);
meshColorList.push_back(letraM);
```

```
//Letra F
shaderList[1].useShader();
uniformModel = shaderList[1].getModelLocation();
uniformProjection = shaderList[1].getProjectLocation();

//Inicializar matriz de dimensión 4x4 que servirá como matriz de modelo para almac
model = glm::mat4(1.0);
model = glm::translate(model, glm::vec3(-0.0f, -0.0f, -4.0f));

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


//Letra R
shaderList[1].useShader();
uniformModel = shaderList[1].getModelLocation();
uniformProjection = shaderList[1].getProjectLocation();

//Inicializar matriz de dimensión 4x4 que servirá como matriz de modelo para almac
model = glm::mat4(1.0);
model = glm::translate(model, glm::vec3(-0.0f, -0.0f, -4.0f));
//
glUniformMatrix4fv(uniformModel, 1, GL_FALSE, glm::value_ptr(model));//FALSE ES PA
glUniformMatrix4fv(uniformProjection, 1, GL_FALSE, glm::value_ptr(projection));
meshColorList[7]->RenderMeshColor();

//Letra M
shaderList[1].useShader();
uniformModel = shaderList[1].getModelLocation();
uniformProjection = shaderList[1].getProjectLocation();

//Inicializar matriz de dimensión 4x4 que servirá como matriz de modelo para almac
model = glm::mat4(1.0);
model = glm::translate(model, glm::vec3(-0.0f, -0.0f, -4.0f));
//
glUniformMatrix4fv(uniformModel, 1, GL_FALSE, glm::value_ptr(model));//FALSE ES PA
glUniformMatrix4fv(uniformProjection, 1, GL_FALSE, glm::value_ptr(projection));
meshColorList[8]->RenderMeshColor();
```

Código en ejecución



Ejercicio 2:

Generar el dibujo de la casa de la clase, pero en lugar de instanciar triángulos y cuadrados será instanciando pirámides y cubos, para esto se requiere crear shaders diferentes de los colores: rojo, verde, azul, café y verde oscuro en lugar de usar el shader con el color clamp

En este ejercicio, tuve que crear nuevos archivos para cada shader distinto. Con los shaders creados, los agregué a la función *CreateShaders*. Posteriormente, instancié cada cubo y pirámide utilizando el índice *shaderList[i],* donde *i* corresponde al color deseado, y seleccioné la figura que quería instanciar de la lista *meshList[]*. Además, con la ayuda de las funciones *translate* y *scale*, logré darle la forma solicitada a cada figura.

Bloques de código generado

```
#version 330 core
out vec4 FragColor;

void main()
{
    FragColor = vec4(0.0, 0.0, 1.0, 1.0); // azul
}
```

```
#version 330 core
out vec4 FragColor;

void main()
{
    FragColor = vec4(0.478f, 0.255f, 0.067f, 1.0); // café
}
```

```
#version 330 core
out vec4 FragColor;

void main()
{
    FragColor = vec4(1.0, 0.0, 0.0, 1.0); // rojo
}
```

```
#version 330 core
out vec4 FragColor;

void main()
{
    FragColor = vec4(0.0, 1.0, 0.0, 1.0); // verde
}
```

```
#version 330 core
out vec4 FragColor;

void main()
{
    FragColor = vec4(0.0, 0.5, 0.0, 1.0); // verde oscuro
}
```

```
void CreateShaders()
{

    Shader *shader1 = new Shader(); //shader para usar índices: objetos: cubo y  pirámide
    shader1->CreateFromFiles(vShader, fShader);
    shaderList.push_back(*shader1);

    Shader *shader2 = new Shader();//shader para usar color como parte del VAO: letras
    shader2->CreateFromFiles(vShaderColor, fShaderColor);
    shaderList.push_back(*shader2);

    // Rojo 2
    Shader *shaderRojo = new Shader();
    shaderRojo->CreateFromFiles("shaders/shader.vert", "shaders/fShaderRojo.frag");
    shaderList.push_back(*shaderRojo);

    // Verde 3
    Shader *shaderVerde = new Shader();
    shaderVerde->CreateFromFiles("shaders/shader.vert", "shaders/fShaderVerde.frag");
    shaderList.push_back(*shaderVerde);

    // Azul 4
    Shader *shaderAzul = new Shader();
    shaderAzul->CreateFromFiles("shaders/shader.vert", "shaders/fShaderAzul.frag");
    shaderList.push_back(*shaderAzul);

    // Café 5
    Shader *shaderCafe = new Shader();
    shaderCafe->CreateFromFiles("shaders/shader.vert", "shaders/fShaderCafe.frag");
    shaderList.push_back(*shaderCafe);

    // Verde oscuro 6
    Shader *shaderVerdeOscuro = new Shader();
    shaderVerdeOscuro->CreateFromFiles("shaders/shader.vert", "shaders/fShaderVerdeOscuro.frag");
    shaderList.push_back(*shaderVerdeOscuro);
}
```

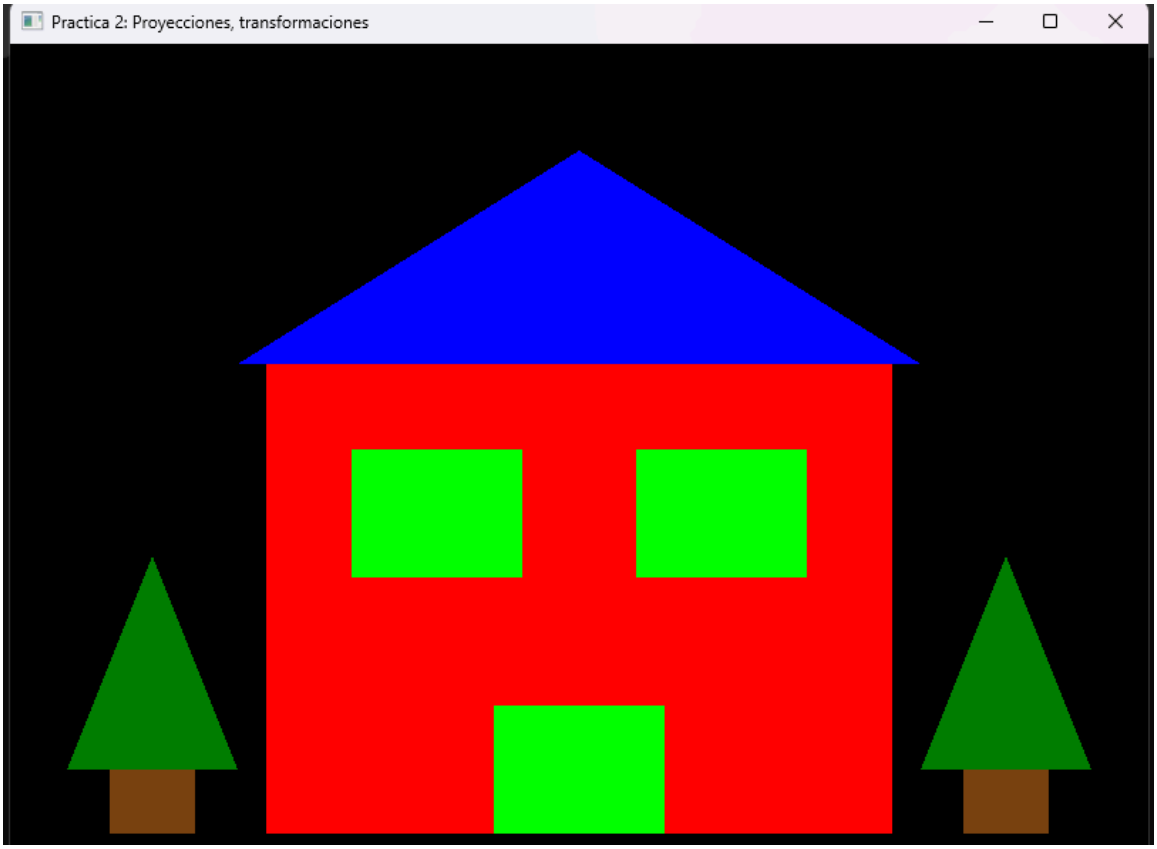
```
//Cubo rojo
shaderList[2].useShader();
uniformModel = shaderList[2].getModelLocation();
uniformProjection = shaderList[2].getProjectLocation();
angulo += 0.01;
//Inicializar matriz de dimensión 4x4 que servirá como matriz de modelo para almacenar
model = glm::mat4(1.0);
model = glm::translate(model, glm::vec3(0.0f, -0.3f, -3.0f));
model = glm::scale(model, glm::vec3(1.1f, 1.1f, 1.0f)); // escalar en X, Y, Z
glUniformMatrix4fv(uniformModel, 1, GL_FALSE, glm::value_ptr(model));//FALSE ES PARA C
glUniformMatrix4fv(uniformProjection, 1, GL_FALSE, glm::value_ptr(projection));
meshList[1]->RenderMesh();



//Piramide azul
shaderList[4].useShader();
uniformModel = shaderList[4].getModelLocation();
uniformProjection = shaderList[4].getProjectLocation();
angulo += 0.01;
//Inicializar matriz de dimensión 4x4 que servirá como matriz de modelo para almacenar
model = glm::mat4(1.0);
model = glm::translate(model, glm::vec3(0.0f, 0.5f, -3.0f));
model = glm::scale(model, glm::vec3(1.2f, 0.5f, 1.0f)); // escalar en X, Y, Z
glUniformMatrix4fv(uniformModel, 1, GL_FALSE, glm::value_ptr(model));//FALSE ES PARA C
glUniformMatrix4fv(uniformProjection, 1, GL_FALSE, glm::value_ptr(projection));
meshList[0]->RenderMesh();
```

```
//Cubo verde arriba izq
shaderList[3].useShader();
uniformModel = shaderList[3].getModelLocation();
uniformProjection = shaderList[3].getProjectLocation();
angulo += 0.01;
//Inicializar matriz de dimensión 4x4 que servirá como matriz de modelo para almac
model = glm::mat4(1.0);
model = glm::translate(model, glm::vec3(-0.25f, -0.1f, -3.0f));
model = glm::scale(model, glm::vec3(0.3f, 0.3f, 2.0f)); // escalar en X, Y, Z
glUniformMatrix4fv(uniformModel, 1, GL_FALSE, glm::value_ptr(model));//FALSE ES PA
glUniformMatrix4fv(uniformProjection, 1, GL_FALSE, glm::value_ptr(projection));
meshList[1]->RenderMesh();


//Cubo verde arriba der
shaderList[3].useShader();
uniformModel = shaderList[3].getModelLocation();
uniformProjection = shaderList[3].getProjectLocation();
angulo += 0.01;
//Inicializar matriz de dimensión 4x4 que servirá como matriz de modelo para almac
model = glm::mat4(1.0);
model = glm::translate(model, glm::vec3(0.25f, -0.1f, -3.0f));
model = glm::scale(model, glm::vec3(0.3f, 0.3f, 2.0f)); // escalar en X, Y, Z
glUniformMatrix4fv(uniformModel, 1, GL_FALSE, glm::value_ptr(model));//FALSE ES PA
glUniformMatrix4fv(uniformProjection, 1, GL_FALSE, glm::value_ptr(projection));
meshList[1]->RenderMesh();
```

```
//Cubo verde abajo
shaderList[3].useShader();
uniformModel = shaderList[3].getModelLocation();
uniformProjection = shaderList[3].getProjectLocation();
angulo += 0.01;
//Inicializar matriz de dimensión 4x4 que servirá como matriz de modelo para almac
model = glm::mat4(1.0);
model = glm::translate(model, glm::vec3(0.0f, -0.7f, -3.0f));
model = glm::scale(model, glm::vec3(0.3f, 0.3f, 2.0f)); // escalar en X, Y, Z
glUniformMatrix4fv(uniformModel, 1, GL_FALSE, glm::value_ptr(model));//FALSE ES PA
glUniformMatrix4fv(uniformProjection, 1, GL_FALSE, glm::value_ptr(projection));
meshList[1]->RenderMesh();


//Cubo cafe izq
shaderList[5].useShader();
uniformModel = shaderList[5].getModelLocation();
uniformProjection = shaderList[5].getProjectLocation();
angulo += 0.01;
//Inicializar matriz de dimensión 4x4 que servirá como matriz de modelo para almac
model = glm::mat4(1.0);
model = glm::translate(model, glm::vec3(-0.75f, -0.775f, -3.0f));
model = glm::scale(model, glm::vec3(0.15f, 0.15f, 1.0f)); // escalar en X, Y, Z
glUniformMatrix4fv(uniformModel, 1, GL_FALSE, glm::value_ptr(model));//FALSE ES PA
glUniformMatrix4fv(uniformProjection, 1, GL_FALSE, glm::value_ptr(projection));
meshList[1]->RenderMesh();
```

```
//Cubo cafe der
shaderList[5].useShader();
uniformModel = shaderList[5].getModelLocation();
uniformProjection = shaderList[5].getProjectLocation();
angulo += 0.01;
//Inicializar matriz de dimensión 4x4 que servirá como matriz de modelo para almacen
model = glm::mat4(1.0);
model = glm::translate(model, glm::vec3(0.75f, -0.775f, -3.0f));
model = glm::scale(model, glm::vec3(0.15f, 0.15f, 1.0f)); // escalar en X, Y, Z
glUniformMatrix4fv(uniformModel, 1, GL_FALSE, glm::value_ptr(model));//FALSE ES PARA
glUniformMatrix4fv(uniformProjection, 1, GL_FALSE, glm::value_ptr(projection));
meshList[1]->RenderMesh();


//Piramide verde izq
shaderList[6].useShader();
uniformModel = shaderList[6].getModelLocation();
uniformProjection = shaderList[6].getProjectLocation();
angulo += 0.01;
//Inicializar matriz de dimensión 4x4 que servirá como matriz de modelo para almacen
model = glm::mat4(1.0);
model = glm::translate(model, glm::vec3(-0.75f, -0.45f, -3.0f));
model = glm::scale(model, glm::vec3(0.3f, 0.5f, 1.0f)); // escalar en X, Y, Z
glUniformMatrix4fv(uniformModel, 1, GL_FALSE, glm::value_ptr(model));//FALSE ES PARA
glUniformMatrix4fv(uniformProjection, 1, GL_FALSE, glm::value_ptr(projection));
meshList[0]->RenderMesh();


//Piramide verde izq
shaderList[6].useShader();
uniformModel = shaderList[6].getModelLocation();
uniformProjection = shaderList[6].getProjectLocation();
angulo += 0.01;
//Inicializar matriz de dimensión 4x4 que servirá como matriz de modelo para almacen
model = glm::mat4(1.0);
model = glm::translate(model, glm::vec3(0.75f, -0.45f, -3.0f));
model = glm::scale(model, glm::vec3(0.3f, 0.5f, 1.0f)); // escalar en X, Y, Z
glUniformMatrix4fv(uniformModel, 1, GL_FALSE, glm::value_ptr(model));//FALSE ES PARA
glUniformMatrix4fv(uniformProjection, 1, GL_FALSE, glm::value_ptr(projection));
meshList[0]->RenderMesh();
```

Código en ejecución



2.- Liste los problemas que tuvo a la hora de hacer estos ejercicios y si los resolvió explicar cómo fue, en caso de error adjuntar captura de pantalla

- Tuve muchas complicaciones para entender cómo funcionaba el código del ejercicio 2. Me tomó bastante tiempo comprender cómo crear los nuevos shaders y cómo instanciar cada figura con el color y la forma solicitados. Para solucionarlo, tuve que investigar cómo crear un shader y cómo instanciarlo correctamente en la ventana. En general, ese fue el único problema que tuve para resolver los ejercicios planteados.

Conclusión:

Con esta práctica logré, en el primer ejercicio, imprimir en la ventana las iniciales de mi nombre, pero asignando un color distinto a cada letra. En el segundo ejercicio, pude formar la misma figura del ejercicio anterior, pero utilizando nuevos shaders para los colores y con instancias de cubos y pirámides. Además, gracias al uso de las funciones *translate* y *scale*, logré dar forma correctamente a la figura. Fue una práctica que me presentó algunas

dificultades, pero con la ayuda de la bibliografía y las explicaciones de mis compañeros y del profesor, pude completarla satisfactoriamente.

Bibliografía :

Shreiner, D., Sellers, G., Kessenich, J., & Licea-Kane, B. (2013). OpenGL Programming Guide: The Official Guide to Learning OpenGL, Version 4.3 (8th ed.). Addison-Wesley Professional.