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: Roberto Aburto López

Nº de Cuenta: 319131996

GRUPO DE LABORATORIO: 03

GRUPO DE TEORÍA: 06

SEMESTRE 2026-1

FECHA DE ENTREGA LÍMITE: 30 de agosto de 2025

CALIFICACIÓN: _____

REPORTE DE PRÁCTICA:

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

Esta parte de la práctica resultó sencilla. Agregué a la lista *meshColorList* nuestras iniciales, que habíamos dibujado en la práctica anterior, pero con la diferencia de que ahora a cada vértice se le asignaba el color con el que sería dibujado.

```
void CrearLetrasyFiguras()
{
    GLfloat vertices_letraR[] = {
        -0.78f, 0.48f, 0.0f,    0.0f,   0.0f,   1.0f,
        -0.78f, 0.36f, 0.0f,    0.0f,   0.0f,   1.0f,
        -0.3f, 0.48f, 0.0f,     0.0f,   0.0f,   1.0f,

        -0.3f, 0.48f, 0.0f,     0.0f,   0.0f,   1.0f,
        -0.3f, 0.36f, 0.0f,     0.0f,   0.0f,   1.0f,
        -0.78f, 0.36f, 0.0f,    0.0f,   0.0f,   1.0f,

        -0.3f, 0.48f, 0.0f,     0.0f,   0.0f,   1.0f,
        -0.3f, 0.3f, 0.0f,      0.0f,   0.0f,   1.0f,
        -0.18f, 0.3f, 0.0f,     0.0f,   0.0f,   1.0f,

        -0.3f, 0.3f, 0.0f,      0.0f,   0.0f,   1.0f,
        -0.18f, 0.3f, 0.0f,     0.0f,   0.0f,   1.0f,
        -0.3f, -0.06f, 0.0f,    0.0f,   0.0f,   1.0f,

        -0.18f, 0.3f, 0.0f,     0.0f,   0.0f,   1.0f,
        -0.18f, 0.12f, 0.0f,    0.0f,   0.0f,   1.0f,
        -0.3f, -0.06f, 0.0f,    0.0f,   0.0f,   1.0f,

        -0.42f, 0.36f, 0.0f,    0.0f,   0.0f,   1.0f,
        -0.3f, 0.36f, 0.0f,     0.0f,   0.0f,   1.0f,
        -0.42f, -0.06f, 0.0f,   0.0f,   0.0f,   1.0f,

        -0.3f, 0.36f, 0.0f,     0.0f,   0.0f,   1.0f,
        -0.3f, -0.06f, 0.0f,    0.0f,   0.0f,   1.0f,
        -0.42f, -0.06f, 0.0f,   0.0f,   0.0f,   1.0f,

        -0.78f, -0.06f, 0.0f,   0.0f,   0.0f,   1.0f,
        -0.42f, 0.12f, 0.0f,    0.0f,   0.0f,   1.0f,
        -0.42f, -0.06f, 0.0f,   0.0f,   0.0f,   1.0f,

        -0.78f, -0.06f, 0.0f,   0.0f,   0.0f,   1.0f,
        -0.78f, 0.12f, 0.0f,    0.0f,   0.0f,   1.0f,
        -0.42f, 0.12f, 0.0f,    0.0f,   0.0f,   1.0f,
```

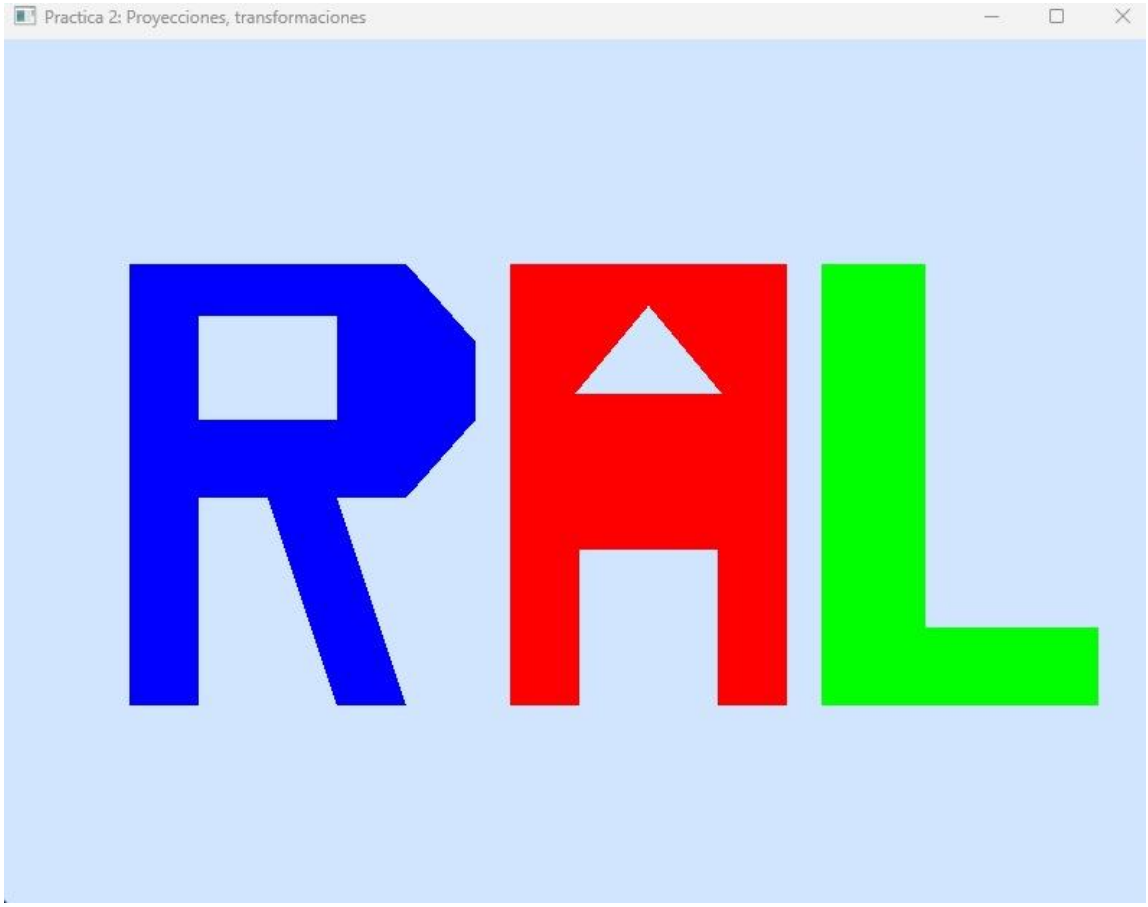
```
    -0.78f, 0.36f, 0.0f,    0.0f,   0.0f,   1.0f,
    -0.78f, 0.12f, 0.0f,    0.0f,   0.0f,   1.0f,
    -0.66f, 0.12f, 0.0f,    0.0f,   0.0f,   1.0f,

    -0.78f, 0.36f, 0.0f,    0.0f,   0.0f,   1.0f,
    -0.66f, 0.36f, 0.0f,    0.0f,   0.0f,   1.0f,
    -0.66f, 0.12f, 0.0f,    0.0f,   0.0f,   1.0f,

    -0.78f, -0.06f, 0.0f,   0.0f,   0.0f,   1.0f,
    -0.78f, -0.54f, 0.0f,   0.0f,   0.0f,   1.0f,
    -0.66f, -0.54f, 0.0f,   0.0f,   0.0f,   1.0f,

    -0.78f, -0.06f, 0.0f,   0.0f,   0.0f,   1.0f,
    -0.66f, -0.06f, 0.0f,   0.0f,   0.0f,   1.0f,
    -0.66f, -0.54f, 0.0f,   0.0f,   0.0f,   1.0f,

    -0.54f, -0.06f, 0.0f,   0.0f,   0.0f,   1.0f,
    -0.42f, -0.54f, 0.0f,   0.0f,   0.0f,   1.0f,
    -0.42f, -0.06f, 0.0f,   0.0f,   0.0f,   1.0f,

    -0.42f, -0.06f, 0.0f,   0.0f,   0.0f,   1.0f,
    -0.42f, -0.54f, 0.0f,   0.0f,   0.0f,   1.0f,
    -0.3f, -0.54f, 0.0f,    0.0f,   0.0f,   1.0f,

};

MeshColor* letraR = new MeshColor();
letraR->CreateMeshColor(vertices_letraR, 270);
meshColorList.push_back(letraR);

GLfloat vertices_letraA[] = {
    //A
    -0.12f, 0.0f, 0.0f,         1.0f,   0.0f,   0.0f,
    -0.12f, 0.48f, 0.0f,        1.0f,   0.0f,   0.0f,
    0.18f, 0.48f, 0.0f,         1.0f,   0.0f,   0.0f,

    0.36f, 0.0f, 0.0f,          1.0f,   0.0f,   0.0f,
    0.36f, 0.48f, 0.0f,         1.0f,   0.0f,   0.0f,
    0.06f, 0.48f, 0.0f,         1.0f,   0.0f,   0.0f,

    -0.12f, -0.18f, 0.0f,       1.0f,   0.0f,   0.0f,
    -0.12f, 0.18f, 0.0f,        1.0f,   0.0f,   0.0f,
    0.36f, 0.18f, 0.0f,         1.0f,   0.0f,   0.0f,
```

```
        0.36f, -0.18f, 0.0f,        1.0f,   0.0f,   0.0f,
        0.36f, 0.18f, 0.0f,         1.0f,   0.0f,   0.0f,
        -0.12f, -0.18f, 0.0f,       1.0f,   0.0f,   0.0f,

        -0.12f, -0.18f, 0.0f,       1.0f,   0.0f,   0.0f,
        -0.12f, -0.54f, 0.0f,       1.0f,   0.0f,   0.0f,
        0.0f, -0.54f, 0.0f,         1.0f,   0.0f,   0.0f,

        0.0f, -0.54f, 0.0f,         1.0f,   0.0f,   0.0f,
        0.0f, -0.18f, 0.0f,         1.0f,   0.0f,   0.0f,
        -0.12f, -0.18f, 0.0f,       1.0f,   0.0f,   0.0f,

        0.24f, -0.18f, 0.0f,        1.0f,   0.0f,   0.0f,
        0.24f, -0.54f, 0.0f,        1.0f,   0.0f,   0.0f,
        0.36f, -0.18f, 0.0f,        1.0f,   0.0f,   0.0f,

        0.36f, -0.18f, 0.0f,        1.0f,   0.0f,   0.0f,
        0.36f, -0.54f, 0.0f,        1.0f,   0.0f,   0.0f,
        0.24f, -0.54f, 0.0f,        1.0f,   0.0f,   0.0f,

    };
    MeshColor* letraA = new MeshColor();
    letraA->CreateMeshColor(vertices_letraA, 144);
    meshColorList.push_back(letraA);


    GLfloat vertices_letraL[] = {
        0.42f, 0.48f, 0.0f,         0.0f,   1.0f,   0.0f,
        0.42f, -0.54f, 0.0f,        0.0f,   1.0f,   0.0f,
        0.6f, 0.48f, 0.0f,          0.0f,   1.0f,   0.0f,

        0.6f, 0.48f, 0.0f,          0.0f,   1.0f,   0.0f,
        0.6f, -0.54f, 0.0f,         0.0f,   1.0f,   0.0f,
        0.42f, -0.54f, 0.0f,        0.0f,   1.0f,   0.0f,

        0.6f, -0.36f, 0.0f,         0.0f,   1.0f,   0.0f,
        0.6f, -0.54f, 0.0f,         0.0f,   1.0f,   0.0f,
        0.9f, -0.36f, 0.0f,         0.0f,   1.0f,   0.0f,

        0.9f, -0.36f, 0.0f,         0.0f,   1.0f,   0.0f,
        0.9f, -0.54f, 0.0f,         0.0f,   1.0f,   0.0f,
        0.6f, -0.54f, 0.0f,         0.0f,   1.0f,   0.0f,

    };
);
```

Después llamamos nuestras figuras dentro del *while* con su respectivo *shader*, donde el índice 1 de la lista de *shaders* que corresponde a nuestras letras.

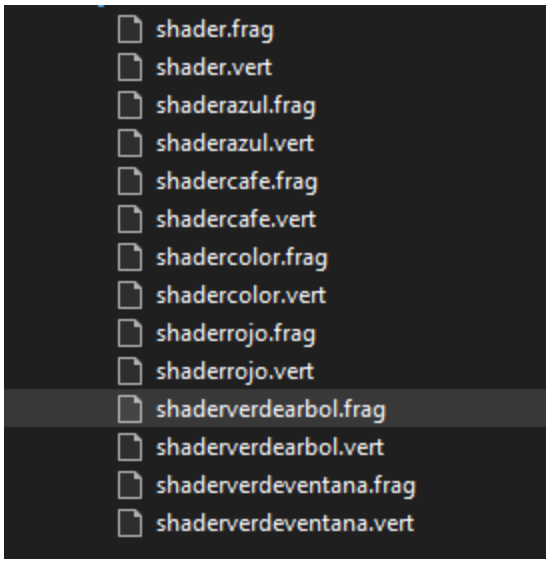
```
     shaderList[1].useShader();
    uniformModel = shaderList[1].getModelLocation();
    uniformProjection = shaderList[1].getProjectLocation();

    //Inicializar matriz de dimensión 4x4 que servirá como matriz de modelo para almacenar las transformaciones geométricas
    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 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[0]->RenderMeshColor();

    //Inicializar matriz de dimensión 4x4 que servirá como matriz de modelo para almacenar las transformaciones geométricas
    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 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[1]->RenderMeshColor();

    //Inicializar matriz de dimensión 4x4 que servirá como matriz de modelo para almacenar las transformaciones geométricas
    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 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();
```

Resultado:



2.- Generar el dibujo de la casa de la clase, pero en lugar de instanciar triangulos y cuadrados será instanciando piramides 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

Esta parte de la práctica fue más desafiante para mí, primero creé los shaders correspondientes para los diferentes colores que tenemos.



Donde cada archivo .frag sólo variaba en el color RGB, y los archivos.vert son iguales.

Archivo .vert

```
#version 330 core

layout (location =0) in vec3 pos;

uniform mat4 model;
uniform mat4 projection;

void main()
{
    gl_Position=projection*model*vec4(pos,1.0f);
}
```

Aquí fue importante poner la versión core para que se colocaran los colores correctamente.

Archivos .frag

```
#version 330 core
out vec4 fragColor;

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

Aquí solamente varía el color RGB.

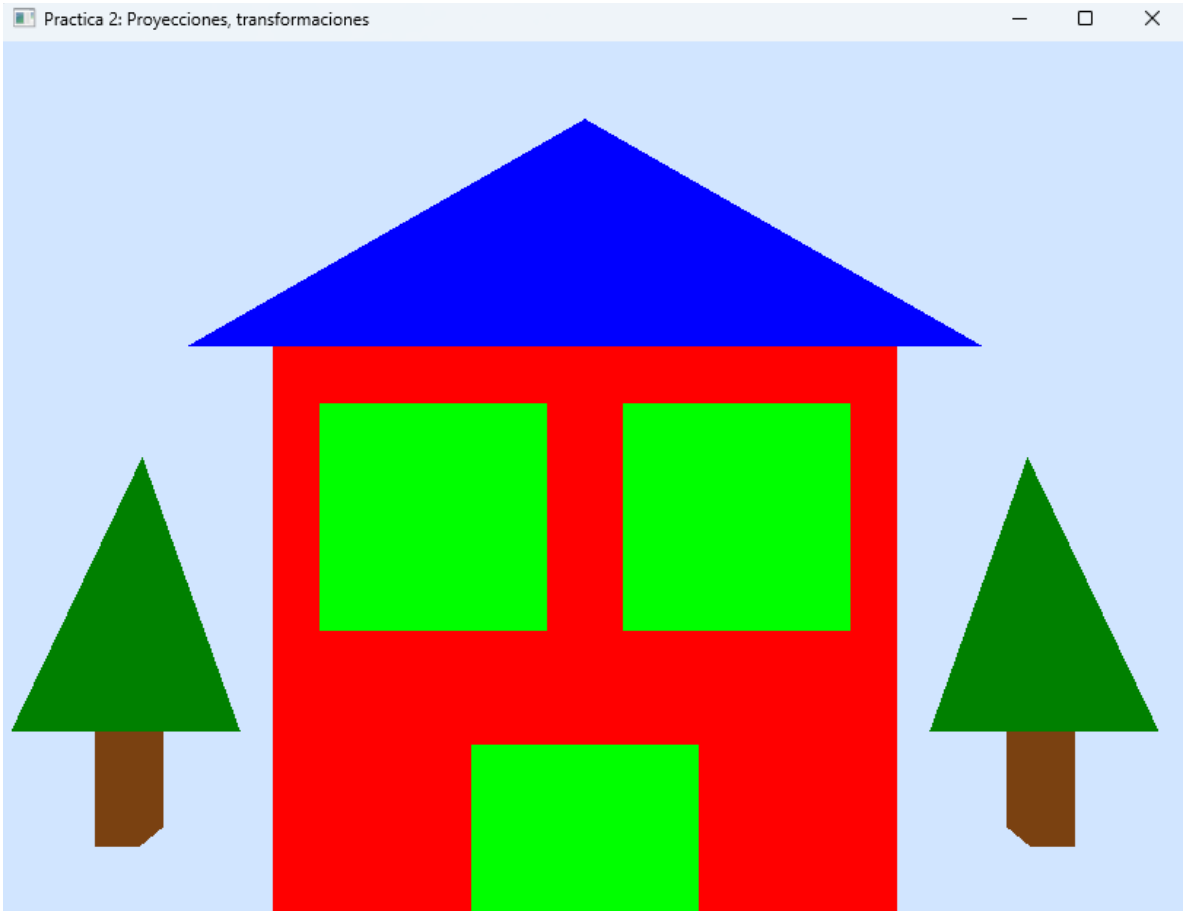
Despues, definí las rutas de los distintos shaders (vertex y fragment) que se usarán en el programa, cada uno con un color o función específica (letras, verde, rojo, café, etc.).

```
//Vertex Shader
static const char* vShader      = "shaders/shader.vert";
static const char* fShader      = "shaders/shader.frag";
static const char* vShaderColor = "shaders/shadercolor.vert";
static const char* fShaderColor = "shaders/shadercolor.frag";
static const char* vVerde = "shaders/shaderverdeventana.vert";
static const char* fverde = "shaders/shaderverdeventana.frag";
static const char* vRojo        = "shaders/shaderrojo.vert";
static const char* fRojo        = "shaders/shaderrojo.frag";
static const char* vCafe        = "shaders/shadercafe.vert";
static const char* fCafe        = "shaders/shadercafe.frag";
static const char* vVarbol      = "shaders/shaderverdearbol.vert";
static const char* fVarbol      = "shaders/shaderverdearbol.frag";
static const char* vAzul = "shaders/shaderazul.vert";
static const char* fAzul = "shaders/shaderazul.frag";
```

Creé distintos objetos *Shader* a partir de los archivos .vert y .frag definidos antes, y los guarda en la lista shaderList para luego usarlos al dibujar diferentes figuras y colores en la escena.

Por último, en el *while* activé el *shader* que define su color o estilo, obtenemos las variables de transformación, aplica traslaciones, escalas o rotaciones según se requiera y luego envía las matrices correspondientes al *shader*. Finalmente, se renderiza la figura seleccionada desde la lista de mallas.

Resultado:



Conclusión:

Gracias a esta práctica aprendí a utilizar diferentes shaders para asignar colores a las figuras, reforcé aplicar transformaciones como traslación y escala. Aunque en el último ejercicio no se ve en 3D aún, es porque lo estamos viendo desde enfrente, entonces no podemos ver las otras caras de la casa, aunque se puede observar un poco en los árboles que ya estoy usando prismas.