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: Bueno Hernández Héctor Daniel

Nº de Cuenta: 319233148

GRUPO DE LABORATORIO: 02

GRUPO DE TEORÍA: 04

SEMESTRE 2025-2

FECHA DE ENTREGA LÍMITE: 22/02/2025

CALIFICACIÓN: _____

Letras de color diferente

Se solicitó volver a dibujar con triángulos las letras usadas en la práctica 1, pero en esta ocasión cada letra debe tener su propio color. Para poder dar solución a este ejercicio, me basé en el código proporcionado por el profesor y modifiqué los vértices de las figuras, incluyendo tanto las coordenadas como el color de cada uno. Debido que se busca que cada letra sea de un color, todos los vértices de la letra deben ser del mismo color, ya que de lo contrario se apreciaría un degradado en lugar de un color sólido.

Para la letra H se eligió el color naranja, por lo que se establecieron los siguientes vértices:

```
//Letra H
GLfloat vertices_letraH[] = {
    //X         Y           Z           R       G       B
    -0.2f,  0.3f,       0.5f,           0.137f, 0.447f, 0.212f,
    -0.1f,  0.3f,       0.5f,           0.137f, 0.447f, 0.212f,
    -0.1f,  -0.3f,      0.5f,           0.137f, 0.447f, 0.212f,
    -0.2f,  0.3f,       0.5f,           0.137f, 0.447f, 0.212f,
    -0.2f,  -0.3f,      0.5f,           0.137f, 0.447f, 0.212f,
    -0.1f,  -0.3f,      0.5f,           0.137f, 0.447f, 0.212f,

    -0.1f,  0.05f,      0.5f,           0.137f, 0.447f, 0.212f,
    0.1f,   -0.05f,     0.5f,           0.137f, 0.447f, 0.212f,
    0.1f,   0.05f,      0.5f,           0.137f, 0.447f, 0.212f,
    -0.1f,  0.05f,      0.5f,           0.137f, 0.447f, 0.212f,
    0.1f,   -0.05f,     0.5f,           0.137f, 0.447f, 0.212f,
    -0.1f,  -0.05f,     0.5f,           0.137f, 0.447f, 0.212f,

    0.1f,   0.3f,       0.5f,           0.137f, 0.447f, 0.212f,
    0.2f,   -0.3f,      0.5f,           0.137f, 0.447f, 0.212f,
    0.2f,   0.3f,       0.5f,           0.137f, 0.447f, 0.212f,
    0.1f,   0.3f,       0.5f,           0.137f, 0.447f, 0.212f,
    0.2f,   -0.3f,      0.5f,           0.137f, 0.447f, 0.212f,
    0.1f,   -0.3f,      0.5f,           0.137f, 0.447f, 0.212f,

};
MeshColor* letraH = new MeshColor();
letraH->CreateMeshColor(vertices_letraH, 108);
meshColorList.push_back(letraH);
```

Para la letra D se eligió el color verde, por lo que se establecieron los siguientes vértices:

```
//Letra D
GLfloat vertices_letraD[] = {
    //X         Y           Z           R       G       B
    -0.2f,  0.3f,       0.5f,           0.89f,  0.42f,  0.12f,
    -0.1f,  0.3f,       0.5f,           0.89f,  0.42f,  0.12f,
    -0.1f,  -0.3f,      0.5f,           0.89f,  0.42f,  0.12f,
    -0.2f,  0.3f,       0.5f,           0.89f,  0.42f,  0.12f,
    -0.2f,  -0.3f,      0.5f,           0.89f,  0.42f,  0.12f,
    -0.1f,  -0.3f,      0.5f,           0.89f,  0.42f,  0.12f,

    -0.2f,  0.3f,       0.5f,           0.89f,  0.42f,  0.12f,
    0.1f,   0.2f,       0.5f,           0.89f,  0.42f,  0.12f,
    -0.2f,  0.2f,       0.5f,           0.89f,  0.42f,  0.12f,
    -0.2f,  0.3f,       0.5f,           0.89f,  0.42f,  0.12f,
    0.1f,   0.2f,       0.5f,           0.89f,  0.42f,  0.12f,
    0.1f,   0.3f,       0.5f,           0.89f,  0.42f,  0.12f,

    -0.2f,  -0.3f,      0.5f,           0.89f,  0.42f,  0.12f,
    0.1f,   -0.2f,      0.5f,           0.89f,  0.42f,  0.12f,
    -0.2f,  -0.2f,      0.5f,           0.89f,  0.42f,  0.12f,
    -0.2f,  -0.3f,      0.5f,           0.89f,  0.42f,  0.12f,
    0.1f,   -0.2f,      0.5f,           0.89f,  0.42f,  0.12f,
    0.1f,   -0.3f,      0.5f,           0.89f,  0.42f,  0.12f,

    0.1f,   0.1f,       0.5f,           0.89f,  0.42f,  0.12f,
    0.2f,   -0.1f,      0.5f,           0.89f,  0.42f,  0.12f,
    0.2f,   0.1f,       0.5f,           0.89f,  0.42f,  0.12f,
    0.1f,   0.1f,       0.5f,           0.89f,  0.42f,  0.12f,
    0.2f,   -0.1f,      0.5f,           0.89f,  0.42f,  0.12f,
    0.1f,   -0.1f,      0.5f,           0.89f,  0.42f,  0.12f,
```

```
    0.1f,   0.3f,       0.5f,           0.89f,  0.42f,  0.12f,
    0.2f,   0.1f,       0.5f,           0.89f,  0.42f,  0.12f,
    0.11f,  0.07f,      0.5f,           0.89f,  0.42f,  0.12f,
    0.1f,   0.3f,       0.5f,           0.89f,  0.42f,  0.12f,
    0.11f,  0.07f,      0.5f,           0.89f,  0.42f,  0.12f,
    0.01f,  0.27f,      0.5f,           0.89f,  0.42f,  0.12f,

    0.2f,   -0.1f,      0.5f,           0.89f,  0.42f,  0.12f,
    0.1f,   -0.3f,      0.5f,           0.89f,  0.42f,  0.12f,
    0.11f,  -0.07f,     0.5f,           0.89f,  0.42f,  0.12f,
    0.1f,   -0.3f,      0.5f,           0.89f,  0.42f,  0.12f,
    0.11f,  -0.07f,     0.5f,           0.89f,  0.42f,  0.12f,
    0.01f,  -0.27f,     0.5f,           0.89f,  0.42f,  0.12f,

};
MeshColor* letraD = new MeshColor();
letraD->CreateMeshColor(vertices_letraD, 216);
meshColorList.push_back(letraD);
```

Para la letra B se eligió el color azul, por lo que se establecieron los siguientes vértices:

```
//Letra B
GLfloat vertices_letraB[] = {
    //X         Y           Z           R       G       B
    -0.2f,  0.3f,       0.5f,           0.302f, 0.541f, 0.545f,
    -0.1f,  0.3f,       0.5f,           0.302f, 0.541f, 0.545f,
    -0.1f,  -0.3f,      0.5f,           0.302f, 0.541f, 0.545f,
    -0.2f,  0.3f,       0.5f,           0.302f, 0.541f, 0.545f,
    -0.2f,  -0.3f,      0.5f,           0.302f, 0.541f, 0.545f,
    -0.1f,  -0.3f,      0.5f,           0.302f, 0.541f, 0.545f,

    -0.2f,  0.3f,       0.5f,           0.302f, 0.541f, 0.545f,
    0.1f,   0.2f,       0.5f,           0.302f, 0.541f, 0.545f,
    -0.2f,  0.2f,       0.5f,           0.302f, 0.541f, 0.545f,
    -0.2f,  0.3f,       0.5f,           0.302f, 0.541f, 0.545f,
    0.1f,   0.2f,       0.5f,           0.302f, 0.541f, 0.545f,
    0.1f,   0.3f,       0.5f,           0.302f, 0.541f, 0.545f,

    -0.2f,  -0.3f,      0.5f,           0.302f, 0.541f, 0.545f,
    0.1f,   -0.2f,      0.5f,           0.302f, 0.541f, 0.545f,
    -0.2f,  -0.2f,      0.5f,           0.302f, 0.541f, 0.545f,
    -0.2f,  -0.3f,      0.5f,           0.302f, 0.541f, 0.545f,
    0.1f,   -0.2f,      0.5f,           0.302f, 0.541f, 0.545f,
    0.1f,   -0.3f,      0.5f,           0.302f, 0.541f, 0.545f,

    -0.1f,  0.05f,      0.5f,           0.302f, 0.541f, 0.545f,
    0.13f,  -0.05f,     0.5f,           0.302f, 0.541f, 0.545f,
    0.13f,  0.05f,      0.5f,           0.302f, 0.541f, 0.545f,
    -0.1f,  0.05f,      0.5f,           0.302f, 0.541f, 0.545f,
    0.13f,  -0.05f,     0.5f,           0.302f, 0.541f, 0.545f,
    -0.1f,  -0.05f,     0.5f,           0.302f, 0.541f, 0.545f,
```

```
0.1f,   0.2f,       0.5f,           0.302f, 0.541f, 0.545f,
0.2f,   0.05f,      0.5f,           0.302f, 0.541f, 0.545f,
0.2f,   0.2f,       0.5f,           0.302f, 0.541f, 0.545f,
0.1f,   0.2f,       0.5f,           0.302f, 0.541f, 0.545f,
0.2f,   0.05f,      0.5f,           0.302f, 0.541f, 0.545f,
0.1f,   0.05f,      0.5f,           0.302f, 0.541f, 0.545f,

0.1f,   -0.05f,     0.5f,           0.302f, 0.541f, 0.545f,
0.2f,   -0.2f,      0.5f,           0.302f, 0.541f, 0.545f,
0.2f,   -0.05f,     0.5f,           0.302f, 0.541f, 0.545f,
0.1f,   -0.05f,     0.5f,           0.302f, 0.541f, 0.545f,
0.2f,   -0.2f,      0.5f,           0.302f, 0.541f, 0.545f,
0.1f,   -0.2f,      0.5f,           0.302f, 0.541f, 0.545f,

0.1f,   0.3f,       0.5f,           0.302f, 0.541f, 0.545f,
0.1f,   0.2f,       0.5f,           0.302f, 0.541f, 0.545f,
0.2f,   0.2f,       0.5f,           0.302f, 0.541f, 0.545f,

0.05f,  0.2f,       0.5f,           0.302f, 0.541f, 0.545f,
0.1f,   0.2f,       0.5f,           0.302f, 0.541f, 0.545f,
0.1f,   0.15f,      0.5f,           0.302f, 0.541f, 0.545f,
0.1f,   0.1f,       0.5f,           0.302f, 0.541f, 0.545f,
0.05f,  0.05f,      0.5f,           0.302f, 0.541f, 0.545f,
0.1f,   0.05f,      0.5f,           0.302f, 0.541f, 0.545f,

0.1f,   -0.3f,      0.5f,           0.302f, 0.541f, 0.545f,
0.1f,   -0.2f,      0.5f,           0.302f, 0.541f, 0.545f,
0.2f,   -0.2f,      0.5f,           0.302f, 0.541f, 0.545f,
```

```
    0.05f,  -0.2f,      0.5f,           0.302f, 0.541f, 0.545f,
    0.1f,   -0.2f,      0.5f,           0.302f, 0.541f, 0.545f,
    0.1f,   -0.15f,     0.5f,           0.302f, 0.541f, 0.545f,
    0.1f,   -0.1f,      0.5f,           0.302f, 0.541f, 0.545f,
    0.05f,  -0.05f,     0.5f,           0.302f, 0.541f, 0.545f,
    0.1f,   -0.05f,     0.5f,           0.302f, 0.541f, 0.545f,

    0.2f,   0.05f,      0.5f,           0.302f, 0.541f, 0.545f,
    0.13f,  0.05f,      0.5f,           0.302f, 0.541f, 0.545f,
    0.13f,  0.0f,       0.5f,           0.302f, 0.541f, 0.545f,
    0.2f,   -0.05f,     0.5f,           0.302f, 0.541f, 0.545f,
    0.13f,  -0.05f,     0.5f,           0.302f, 0.541f, 0.545f,
    0.13f,  -0.0f,      0.5f,           0.302f, 0.541f, 0.545f,

};
MeshColor* letraB = new MeshColor();
letraB->CreateMeshColor(vertices_letraB, 360);
meshColorList.push_back(letraB);
```

Es importante notar que para cada letra se tuvo que especificar el número de vértices creados al llamar la función `CreateMeshColor`, de este modo se generan correctamente todos los vértices y la letra no queda incompleta.

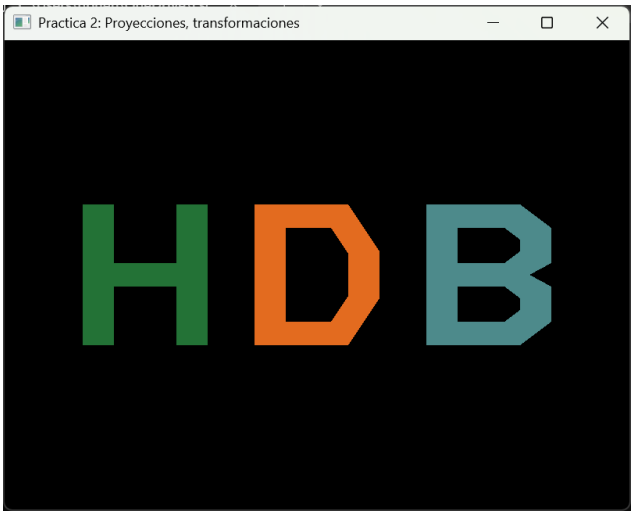
Con los vértices previamente mostrados, las letras se generan en el origen de la pantalla, por lo que al momento de instanciarlas fue necesario utilizar matrices de traslación para moverlas a su posición adecuada:

```
//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();

//Letra H
model = glm::mat4(1.0);
model = glm::translate(model, glm::vec3(-0.55f, 0.0f, -4.0f));
//
glUniformMatrix4fv(uniformModel, 1, GL_FALSE, glm::value_ptr(model));//FALSE ES PARA QUE NO SEA TRANSPUESTA
glUniformMatrix4fv(uniformProjection, 1, GL_FALSE, glm::value_ptr(projection));
meshColorList[0]->RenderMeshColor();
```

```
//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();

//Letra H
model = glm::mat4(1.0);
model = glm::translate(model, glm::vec3(-0.55f, 0.0f, -4.0f));
//
glUniformMatrix4fv(uniformModel, 1, GL_FALSE, glm::value_ptr(model));//FALSE ES PARA QUE NO SEA TRANSPUESTA
glUniformMatrix4fv(uniformProjection, 1, GL_FALSE, glm::value_ptr(projection));
meshColorList[0]->RenderMeshColor();

//Letra D
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
meshColorList[1]->RenderMeshColor();

//Letra B
model = glm::mat4(1.0);
model = glm::translate(model, glm::vec3(0.55f, 0.0f, -4.0f));
glUniformMatrix4fv(uniformModel, 1, GL_FALSE, glm::value_ptr(model));//FALSE ES PARA QUE NO SEA TRANSPUESTA
meshColorList[2]->RenderMeshColor();
```

Estas instrucciones primero inicializan una matriz de 4x4 para almacenar las matrices de transformaciones, después establece la matriz de traslación para mover la figura a la posición deseada dependiendo de los valores establecidos en los argumentos para cada eje. Después se envía la matriz al *shader* y, por último, se elige el índice dentro de la lista `meshColorList` de la figura que se quiere mostrar en pantalla. En este caso solo se tienen tres letras, por lo que los índices de `meshColorList` van de 0 al 2.

La ejecución del código se ve de la siguiente forma:



Casa con cubos y pirámides

Adicionalmente, se solicitó realizar generar una casa como la del ejercicio de clase, pero esta vez utilizando instancias de pirámides y cubos en lugar de triángulos. Para ello, era necesario poder generar figuras tridimensionales de distintos colores.

Lo primero que hice fue generar el código de distintos *shaders*, uno para cada color necesario: rojo, verde, azul, café y verde oscuro. Haciendo uso del *vertex shader* proporcionado por el profesor, creé cinco nuevos archivos en los que modifiqué el valor de `vColor` para que mostrara el color deseado:

```
#version 330
layout (location =0) in vec3 pos;
out vec4 vColor;
uniform mat4 model;
uniform mat4 projection;
void main()
{
    gl_Position=projection*model*vec4(pos,1.0f);
    //vColor=vec4(color,1.0f);
    vColor=vec4(1.0f,0.0f,0.0f,0.0f);
}
```

Shader rojo

```
#version 330
layout (location =0) in vec3 pos;
out vec4 vColor;
uniform mat4 model;
uniform mat4 projection;
void main()
{
    gl_Position=projection*model*vec4(pos,1.0f);
    //vColor=vec4(color,1.0f);
    vColor=vec4(0.0f,1.0f,0.0f,0.0f);
}
```

Shader verde

```
#version 330
layout (location =0) in vec3 pos;
out vec4 vColor;
uniform mat4 model;
uniform mat4 projection;
void main()
{
    gl_Position=projection*model*vec4(pos,1.0f);
    //vColor=vec4(color,1.0f);
    vColor=vec4(0.0f,0.0f,1.0f,0.0f);
}
```

Shader azul

```
#version 330
layout (location =0) in vec3 pos;
out vec4 vColor;
uniform mat4 model;
uniform mat4 projection;
void main()
{
    gl_Position=projection*model*vec4(pos,1.0f);
    //vColor=vec4(color,1.0f);
    vColor=vec4(0.478f,0.255f,0.067f,0.0f);
}
```

Shader café

```
#version 330
layout (location =0) in vec3 pos;
out vec4 vColor;
uniform mat4 model;
uniform mat4 projection;
void main()
{
    gl_Position=projection*model*vec4(pos,1.0f);
    //vColor=vec4(color,1.0f);
    vColor=vec4(0.0f,0.5f,0.0f,0.0f);
}
```

Shader verde oscuro

Posteriormente, ya dentro del *main*, primero fue necesario declarar todos los shaders nuevos, indicando la ruta de cada uno de ellos. Todos los archivos generados se encuentran dentro de la carpeta shaders del proyecto, por lo tanto, lo único que varía entre estos es el nombre del archivo como tal:

```
//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";

//shaders nuevos se crearían acá
static const char* vShaderRojo = "shaders/shaderRojo.vert";
static const char* vShaderVerde = "shaders/shaderVerde.vert";
static const char* vShaderAzul = "shaders/shaderAzul.vert";
static const char* vShaderCafe = "shaders/shaderCafe.vert";
static const char* vShaderVerdeOscuro = "shaders/shaderVerdeOscuro.vert";
```

Después, dentro de la función `CreateShaders` se crearon nuevos objetos de la clase `Shader` para que estos se pudieran agregar a la lista `shaderList` y así poderlos usar al momento de instanciar las figuras. Lo único que cambia en la declaración de los objetos, además del nombre del objeto mismo, es el nombre del *vertex shader* usado. El *fragment shader* es el mismo para todos como se puede observar a continuación:

```
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);

    Shader* shader3 = new Shader();//shader para usar color rojo
    shader3->CreateFromFiles(vShaderRojo, fShader);
    shaderList.push_back(*shader3);

    Shader* shader4 = new Shader();//shader para usar color verde
    shader4->CreateFromFiles(vShaderVerde, fShader);
    shaderList.push_back(*shader4);

    Shader* shader5 = new Shader();//shader para usar color azul
    shader5->CreateFromFiles(vShaderAzul, fShader);
    shaderList.push_back(*shader5);

    Shader* shader6 = new Shader();//shader para usar color cafe
    shader6->CreateFromFiles(vShaderCafe, fShader);
    shaderList.push_back(*shader6);

    Shader* shader7 = new Shader();//shader para usar color verde oscuro
    shader7->CreateFromFiles(vShaderVerdeOscuro, fShader);
    shaderList.push_back(*shader7);
}
```

Finalmente, las instancias de las figuras fueron declaradas una por una teniendo cuidado de usar el índice correcto de `shaderList` para que esta la figura fuese del color deseado. Así mismo, es importante recordar que dentro de `meshList`, el índice ‘0’ representa una pirámide y ‘1’ representa un cubo.

Al igual que con las letras, las figuras se generan en el origen, por lo que es necesario usar una matriz de traslación para ponerlas en la posición deseada. Además, debido a que estaremos creando instancias de las mismas figuras con medidas ya establecidas, es necesario usar matrices de escalamiento para que las figuras tengan el tamaño deseado.

La instancia creada del cubo rojo fue la siguiente:

```
//Cubo rojo
shaderList[2].useShader();
uniformModel = shaderList[0].getModelLocation();
uniformProjection = shaderList[0].getProjectLocation();
//angulo += 0.1;
//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, -0.35f, -3.0f));
model = glm::scale(model, glm::vec3(1.0f, 1.2f, 1.0f));
glUniformMatrix4fv(uniformModel, 1, GL_FALSE, glm::value_ptr(model));//FALSE ES PARA QUE NO SEA TRANSPUESTA
glUniformMatrix4fv(uniformProjection, 1, GL_FALSE, glm::value_ptr(projection));
meshList[1]->RenderMesh();
```

Las instancias creadas del cubo verde fueron las siguientes:

```
//Cubos verdes
shaderList[3].useShader();
uniformModel = shaderList[0].getModelLocation();
uniformProjection = shaderList[0].getProjectLocation();
//angulo += 0.1;
//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(0.25f, -0.02f, -2.9f));
model = glm::scale(model, glm::vec3(0.3f, 0.35f, 1.0f));
glUniformMatrix4fv(uniformModel, 1, GL_FALSE, glm::value_ptr(model));//FALSE ES PARA QUE NO SEA TRANSPUESTA
glUniformMatrix4fv(uniformProjection, 1, GL_FALSE, glm::value_ptr(projection));
meshList[1]->RenderMesh();

model = glm::mat4(1.0);
model = glm::translate(model, glm::vec3(-0.25f, -0.02f, -2.9f));
model = glm::scale(model, glm::vec3(0.3f, 0.35f, 1.0f));
glUniformMatrix4fv(uniformModel, 1, GL_FALSE, glm::value_ptr(model));//FALSE ES PARA QUE NO SEA TRANSPUESTA
meshList[1]->RenderMesh();

model = glm::mat4(1.0);
model = glm::translate(model, glm::vec3(0.0f, -0.75f, -2.9f));
model = glm::scale(model, glm::vec3(0.3f, 0.4f, 1.0f));
glUniformMatrix4fv(uniformModel, 1, GL_FALSE, glm::value_ptr(model));//FALSE ES PARA QUE NO SEA TRANSPUESTA
meshList[1]->RenderMesh();
```

La instancia creada de la pirámide azul fue la siguiente:

```
//Pirámide azul
shaderList[4].useShader();
uniformModel = shaderList[0].getModelLocation();
uniformProjection = shaderList[0].getProjectLocation();
//angulo += 0.1;
//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, 0.6f, -3.0f));
model = glm::scale(model, glm::vec3(1.2f, 0.7f, 1.0f));
glUniformMatrix4fv(uniformModel, 1, GL_FALSE, glm::value_ptr(model));//FALSE ES PARA QUE NO SEA TRANSPUESTA
glUniformMatrix4fv(uniformProjection, 1, GL_FALSE, glm::value_ptr(projection));
meshList[0]->RenderMesh();
```

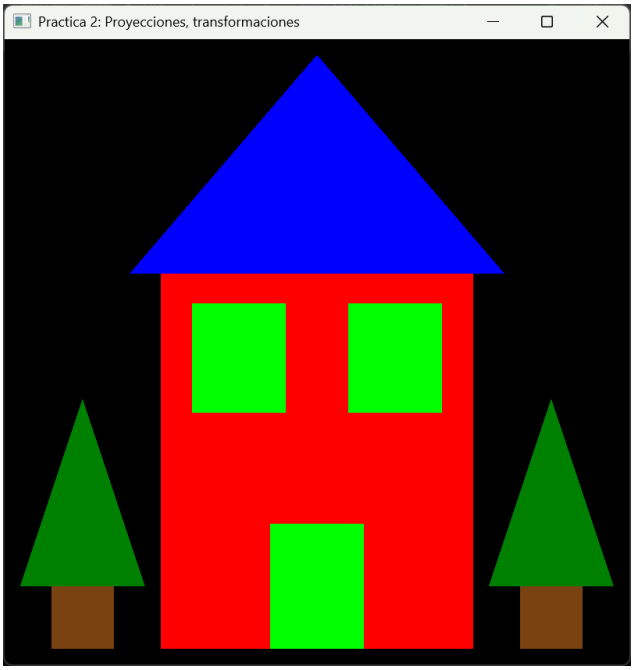
Las instancias creadas del cubo café fueron las siguientes:

```
//Cubos cafés
shaderList[5].useShader();
uniformModel = shaderList[0].getModelLocation();
uniformProjection = shaderList[0].getProjectLocation();
//angulo += 0.1;
//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(0.75f, -0.85f, -3.0f));
model = glm::scale(model, glm::vec3(0.2f, 0.2f, 1.0f));
glUniformMatrix4fv(uniformModel, 1, GL_FALSE, glm::value_ptr(model));//FALSE ES PARA QUE NO SEA TRANSPUESTA
glUniformMatrix4fv(uniformProjection, 1, GL_FALSE, glm::value_ptr(projection));
meshList[1]->RenderMesh();

model = glm::mat4(1.0);
model = glm::translate(model, glm::vec3(-0.75f, -0.85f, -3.0f));
model = glm::scale(model, glm::vec3(0.2f, 0.2f, 1.0f));
glUniformMatrix4fv(uniformModel, 1, GL_FALSE, glm::value_ptr(model));//FALSE ES PARA QUE NO SEA TRANSPUESTA
meshList[1]->RenderMesh();
```

Las instancias creadas de la pirámide verde oscuro fueron las siguientes:

```
//Pirámides verde oscuro
shaderList[6].useShader();
uniformModel = shaderList[0].getModelLocation();
uniformProjection = shaderList[0].getProjectLocation();
//angulo += 0.1;
//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(0.75f, -0.45f, -3.0f));
model = glm::scale(model, glm::vec3(0.4f, 0.6f, 1.0f));
glUniformMatrix4fv(uniformModel, 1, GL_FALSE, glm::value_ptr(model));//FALSE ES PARA QUE NO SEA TRANSPUESTA
glUniformMatrix4fv(uniformProjection, 1, GL_FALSE, glm::value_ptr(projection));
meshList[0]->RenderMesh();
```

Finalmente, la ejecución del código ya con las figuras escaladas y trasladadas es la siguiente:



Problemas presentados

En realidad, no tuve mayores problemas al momento de realizar las actividades solicitadas. Considero que sí se tuvo un salto importante en la dificultad de esta a comparación de la primera práctica por la naturaleza del código utilizado, pero una vez que entendí el funcionamiento del código, no fue difícil identificar las partes que debía modificar o en donde agregar nuevas línea de código.

Si bien el poder hacer la casa me llevó un poco de tiempo ya que tuve que trasladar las figuras un poco al tanteo, no fue difícil ya que solamente fue cuestión de calcular las coordenadas. Del mismo modo, no tuve problemas para ejecutar el código ya que usé como base el código proporcionado por el profesor y solo comenté las partes que no se iban a utilizar en cierto momento.

Conclusiones

Considero que la dificultad de la práctica fue bastante adecuada. A pesar de que fue una práctica más compleja que la anterior debido a que el código esta vez era mucho más largo y con más componentes como los *vertex shader* y *fragment shader*, el funcionamiento general de este era bastante similar al usado en la práctica 1.

Los ejercicios me ayudaron a familiarizarme con el uso de los distintos shaders para poder proyectar distintos colores en pantalla y me ayudó también a comprender que no es necesario crear muchas figuras, sino que basta con instanciarlas múltiples veces cuando se sea necesario para reutilizar los objetos declarados.

La explicación del profesor me pareció bien ya que cubrió todas las partes del código y los distintos archivos del proyecto de Visual Studio, aunque considero que la explicación fue un poco rápida al inicio de la clase ya que empezó a trabajar sobre el mismo proyecto que usamos anteriormente. Independientemente de esto, pude comprender bastante bien lo que se estaba diciendo y pude realizar la práctica sin problemas.

Bibliografía

#4d8a8b código de color hex. (s. f.). Recuperado 21 de febrero de 2025, de https://encycolorpedia.es/4d8a8b

#237236 código de color hex. (s. f.). Recuperado 21 de febrero de 2025, de https://encycolorpedia.es/237236

#e36b1f código de color hex. (s. f.). Recuperado 21 de febrero de 2025, de https://encycolorpedia.es/e36b1f