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: Lopez Flores Diego Alberto

Nº de Cuenta: 315081143

GRUPO DE LABORATORIO: 11

GRUPO DE TEORÍA: 04

SEMESTRE 2025-1

FECHA DE ENTREGA LÍMITE: 28 de agosto de 2024

CALIFICACIÓN: ________

Actividad 1

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

Utilizando los vértices usados en la anterior practica se ponen en la función de crear figuras y letras, dándole a cada vértice su color, Para la primera letra D que es la que tiene más vértices se acomoda y se le da su color

```
GLfloat vertices_letras[] = {
	//X	Y	Z		R	G	B
	-0.7f, 0.9f,  0.0f,		0.0f,  0.0f,  1.0f,
	-0.5f, 0.9f,  0.0f,		0.0f,  0.0f,  1.0f,
	-0.5f, 0.7f,  0.0f,		0.0f,  0.0f,  1.0f,
	-0.7f, 0.9f,  0.0f,		0.0f,  0.0f,  1.0f,
	-0.5f, 0.7f,  0.0f,		0.0f,  0.0f,  1.0f,
	-0.7f, 0.7f,  0.0f,		0.0f,  0.0f,  1.0f,
	-0.7f, 0.7f,  0.0f,		0.0f,  0.0f,  1.0f,
	-0.6f, 0.7f,  0.0f,		0.0f,  0.0f,  1.0f,
	-0.6f, 0.3f,  0.0f,		0.0f,  0.0f,  1.0f,
	-0.7f, 0.7f,  0.0f,		0.0f,  0.0f,  1.0f,
	-0.6f, 0.3f,  0.0f,		0.0f,  0.0f,  1.0f,
	-0.7f, 0.3f,  0.0f,		0.0f,  0.0f,  1.0f,
	-0.7f, 0.3f,  0.0f,		0.0f,  0.0f,  1.0f,
	-0.7f, 0.1f,  0.0f,		0.0f,  0.0f,  1.0f,
	-0.5f, 0.3f,  0.0f,		0.0f,  0.0f,  1.0f,
	-0.7f, 0.1f,  0.0f,		0.0f,  0.0f,  1.0f,
	-0.5f, 0.3f,  0.0f,		0.0f,  0.0f,  1.0f,
	-0.5f, 0.1f,  0.0f,		0.0f,  0.0f,  1.0f,
	-0.5f, 0.3f,  0.0f,		0.0f,  0.0f,  1.0f,
	-0.3f, 0.3f,  0.0f,		0.0f,  0.0f,  1.0f,
	-0.5f, 0.1f,  0.0f,		0.0f,  0.0f,  1.0f,
	-0.4f, 0.4f,  0.0f,		0.0f,  0.0f,  1.0f,
	-0.5f, 0.3f,  0.0f,		0.0f,  0.0f,  1.0f,
	-0.4f, 0.3f,  0.0f,		0.0f,  0.0f,  1.0f,
	-0.4f, 0.4f,  0.0f,		0.0f,  0.0f,  1.0f,
	-0.4f, 0.3f,  0.0f,		0.0f,  0.0f,  1.0f,
	-0.3f, 0.3f,  0.0f,		0.0f,  0.0f,  1.0f,
	-0.4f, 0.4f,  0.0f,		0.0f,  0.0f,  1.0f,
	-0.3f, 0.4f,  0.0f,		0.0f,  0.0f,  1.0f,
	-0.3f, 0.3f,  0.0f,		0.0f,  0.0f,  1.0f,
	-0.3f, 0.4f,  0.0f,		0.0f,  0.0f,  1.0f,
	-0.3f, 0.3f,  0.0f,		0.0f,  0.0f,  1.0f,
	-0.2f, 0.4f,  0.0f,		0.0f,  0.0f,  1.0f,
	-0.4f, 0.6f,  0.0f,		0.0f,  0.0f,  1.0f,
	-0.4f, 0.4f,  0.0f,		0.0f,  0.0f,  1.0f,
	-0.2f, 0.4f,  0.0f,		0.0f,  0.0f,  1.0f,
	-0.4f, 0.6f,  0.0f,		0.0f,  0.0f,  1.0f,
	-0.2f, 0.4f,  0.0f,		0.0f,  0.0f,  1.0f,
	-0.2f, 0.6f,  0.0f,		0.0f,  0.0f,  1.0f,
	-0.5f, 0.9f,  0.0f,		0.0f,  0.0f,  1.0f,
	-0.5f, 0.7f,  0.0f,		0.0f,  0.0f,  1.0f,
	-0.3f, 0.7f,  0.0f,		0.0f,  0.0f,  1.0f,
	-0.5f, 0.7f,  0.0f,		0.0f,  0.0f,  1.0f,
	-0.4f, 0.6f,  0.0f,		0.0f,  0.0f,  1.0f,
	-0.4f, 0.7f,  0.0f,		0.0f,  0.0f,  1.0f,
        -0.4f, 0.7f,  0.0f,        0.0f,  0.0f,  1.0f,
        -0.4f, 0.6f,  0.0f,        0.0f,  0.0f,  1.0f,
        -0.3f, 0.7f,  0.0f,        0.0f,  0.0f,  1.0f,
        -0.4f, 0.6f,  0.0f,        0.0f,  0.0f,  1.0f,
        -0.3f, 0.6f,  0.0f,        0.0f,  0.0f,  1.0f,
        -0.3f, 0.7f,  0.0f,        0.0f,  0.0f,  1.0f,
        -0.3f, 0.7f,  0.0f,        0.0f,  0.0f,  1.0f,
        -0.3f, 0.6f,  0.0f,        0.0f,  0.0f,  1.0f,
        -0.2f, 0.6f,  0.0f,        0.0f,  0.0f,  1.0f,
        -0.3f, 0.7f,  0.0f,        0.0f,  0.0f,  1.0f,
        -0.3f, 0.6f,  0.0f,        0.0f,  0.0f,  1.0f,
        -0.2f, 0.6f,  0.0f,        0.0f,  0.0f,  1.0f,
    };
    MeshColor *letras = new MeshColor();
    letras->CreateMeshColor(vertices_letras,330);
    meshColorList.push_back(letras);
```

y se le cambia el número de vértices al y por utlimo se coloca su llamada en el while y se acomoda para que se vea bien

```
//Letra 1
model = glm::mat4(1.0);
model = glm::translate(model, glm::vec3(0.0f, -0.1f, -4.0f));
glUniformMatrix4fv(uniformModel, 1, GL_FALSE, glm::value_ptr(model));
glUniformMatrix4fv(uniformProjection, 1, GL_FALSE, glm::value_ptr(projection));
meshColorList[0]->RenderMeshColor();
```

después se coloca la segunda letra en la función de crear figuras y letras y se le da un color diferente

```
GLfloat vertices_letras2[] = {
    //X    Y     Z          R     G     B
    0.4f, 0.7f, 0.0f,       1.0f, 0.0f, 0.0f,
    0.6f, 0.7f, 0.0f,       1.0f, 0.0f, 0.0f,
    0.5f, 0.9f, 0.0f,       1.0f, 0.0f, 0.0f,
    0.3f, 0.5f, 0.0f,       1.0f, 0.0f, 0.0f,
    0.4f, 0.7f, 0.0f,       1.0f, 0.0f, 0.0f,
    0.4f, 0.5f, 0.0f,       1.0f, 0.0f, 0.0f,
    0.4f, 0.7f, 0.0f,       1.0f, 0.0f, 0.0f,
    0.4f, 0.5f, 0.0f,       1.0f, 0.0f, 0.0f,
    0.5f, 0.7f, 0.0f,       1.0f, 0.0f, 0.0f,
    0.5f, 0.7f, 0.0f,       1.0f, 0.0f, 0.0f,
    0.6f, 0.7f, 0.0f,       1.0f, 0.0f, 0.0f,
    0.6f, 0.5f, 0.0f,       1.0f, 0.0f, 0.0f,
    0.6f, 0.7f, 0.0f,       1.0f, 0.0f, 0.0f,
    0.6f, 0.5f, 0.0f,       1.0f, 0.0f, 0.0f,
    0.7f, 0.5f, 0.0f,       1.0f, 0.0f, 0.0f,
    0.2f, 0.3f, 0.0f,       1.0f, 0.0f, 0.0f,
    0.3f, 0.3f, 0.0f,       1.0f, 0.0f, 0.0f,
    0.3f, 0.5f, 0.0f,       1.0f, 0.0f, 0.0f,
    0.3f, 0.5f, 0.0f,       1.0f, 0.0f, 0.0f,
    0.3f, 0.3f, 0.0f,       1.0f, 0.0f, 0.0f,
    0.7f, 0.3f, 0.0f,       1.0f, 0.0f, 0.0f,
    0.1f, 0.1f, 0.0f,       1.0f, 0.0f, 0.0f,
    0.2f, 0.3f, 0.0f,       1.0f, 0.0f, 0.0f,
    0.4f, 0.3f, 0.0f,       1.0f, 0.0f, 0.0f,
    0.1f, 0.1f, 0.0f,       1.0f, 0.0f, 0.0f,
    0.3f, 0.1f, 0.0f,       1.0f, 0.0f, 0.0f,
	0.4f, 0.3f, 0.0f,		1.0f, 0.0f, 0.0f,
	0.3f, 0.5f, 0.0f,		1.0f, 0.0f, 0.0f,
	0.7f, 0.5f, 0.0f,		1.0f, 0.0f, 0.0f,
	0.7f, 0.3f, 0.0f,		1.0f, 0.0f, 0.0f,
	0.7f, 0.5f, 0.0f,		1.0f, 0.0f, 0.0f,
	0.7f, 0.3f, 0.0f,		1.0f, 0.0f, 0.0f,
	0.8f, 0.3f, 0.0f,		1.0f, 0.0f, 0.0f,
	0.6f, 0.3f, 0.0f,		1.0f, 0.0f, 0.0f,
	0.7f, 0.1f, 0.0f,		1.0f, 0.0f, 0.0f,
	0.9f, 0.1f, 0.0f,		1.0f, 0.0f, 0.0f,
	0.6f, 0.3f, 0.0f,		1.0f, 0.0f, 0.0f,
	0.8f, 0.3f, 0.0f,		1.0f, 0.0f, 0.0f,
	0.9f, 0.1f, 0.0f,		1.0f, 0.0f, 0.0f,
};
MeshColor* letras2 = new MeshColor();
letras2->CreateMeshColor(vertices_letras2, 234);
meshColorList.push_back(letras2);
```

se le cambia los números de vértices totales y se le llama en el while acomodando la letra con el translate

```
//Letra 2
model = glm::mat4(1.0);
model = glm::translate(model, glm::vec3(0.0f, -0.1f, -4.0f));
glUniformMatrix4fv(uniformModel, 1, GL_FALSE, glm::value_ptr(model));
glUniformMatrix4fv(uniformProjection, 1, GL_FALSE, glm::value_ptr(projection));
meshColorList[1]->RenderMeshColor();
```

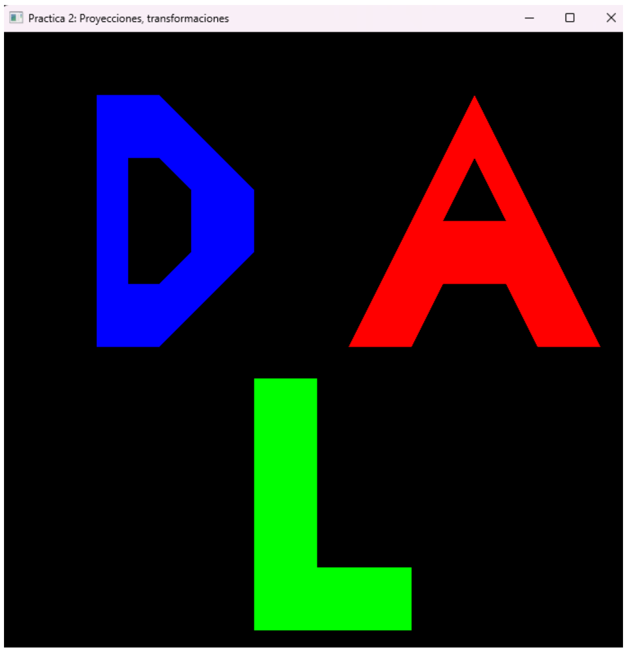
repetimos el proceso con la tercera letra

```
GLfloat vertices_letras3[] = {
	//X	Y	Z		R	G	B
	-0.1f,-0.1f,0.0f,		0.0f, 1.0f, 0.0f,
	-0.1f,-0.7f,0.0f,		0.0f, 1.0f, 0.0f,
	0.1f, -0.1f,0.0f,		0.0f, 1.0f, 0.0f,
	0.1f, -0.7f,0.0f,		0.0f, 1.0f, 0.0f,
	-0.1f,-0.7f,0.0f,		0.0f, 1.0f, 0.0f,
	0.1f, -0.1f,0.0f,		0.0f, 1.0f, 0.0f,
	-0.1f,-0.7f,0.0f,		0.0f, 1.0f, 0.0f,
	-0.1f,-0.9f,0.0f,		0.0f, 1.0f, 0.0f,
	0.1f, -0.9f,0.0f,		0.0f, 1.0f, 0.0f,
	-0.1f,-0.7f,0.0f,		0.0f, 1.0f, 0.0f,
	-0.1f,-0.9f,0.0f,		0.0f, 1.0f, 0.0f,
	0.4f, -0.7f,0.0f,		0.0f, 1.0f, 0.0f,
	-0.1f,-0.9f,0.0f,		0.0f, 1.0f, 0.0f,
	0.4f, -0.7f,0.0f,		0.0f, 1.0f, 0.0f,
	0.4f, -0.9f,0.0f,		0.0f, 1.0f, 0.0f,
};
MeshColor* letras3 = new MeshColor();
letras3->CreateMeshColor(vertices_letras3, 90);
meshColorList.push_back(letras3);
```

y se vuelve a colocar en el while acomodándola con el translate

```
//Letra 3
model = glm::mat4(1.0);
model = glm::translate(model, glm::vec3(-0.1f, 0.0f, -3.0f));
glUniformMatrix4fv(uniformModel, 1, GL_FALSE, glm::value_ptr(model));
glUniformMatrix4fv(uniformProjection, 1, GL_FALSE, glm::value_ptr(projection));
meshColorList[2]->RenderMeshColor();
```

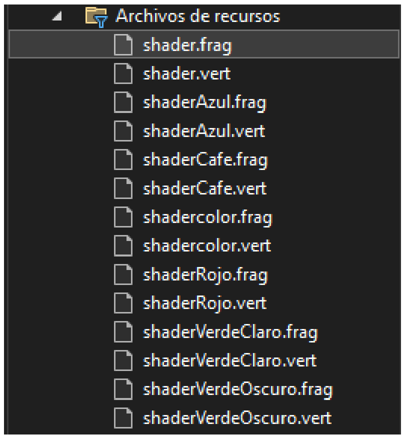
y el resultado es el esperado



Actividad 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

Primero se crearon nuevos shadders pera para colorear las figuras



```
Archivos de recursos
    shader.frag
    shader.vert
    shaderAzul.frag
    shaderAzul.vert
    shaderCafe.frag
    shaderCafe.vert
    shadercolor.frag
    shadercolor.vert
    shaderRojo.frag
    shaderRojo.vert
    shaderVerdeClaro.frag
    shaderVerdeClaro.vert
    shaderVerdeOscuro.frag
    shaderVerdeOscuro.vert
```

y se llaman en el main para ser utilizado

```
static const char* vShaderRojo = "shaders/shaderRojo.vert";
static const char* fShaderRojo = "shaders/shaderRojo.frag";
static const char* vShaderAzul = "shaders/shaderAzul.vert";
static const char* fShaderAzul = "shaders/shaderAzul.frag";
static const char* vShaderCafe = "shaders/shaderCafe.vert";
static const char* fShaderCafe = "shaders/shaderCafe.frag";
static const char* vShaderVerdeClaro = "shaders/shaderVerdeClaro.vert";
static const char* fShaderVerdeClaro = "shaders/shaderVerdeClaro.frag";
static const char* vShaderVerdeOscuro = "shaders/shaderVerdeOscuro.vert";
static const char* fShaderVerdeOscuro = "shaders/shaderVerdeOscuro.frag";
```

en la función CreateShaders se llamarán los archivos de los shaders y se harán push a la lista de shaders

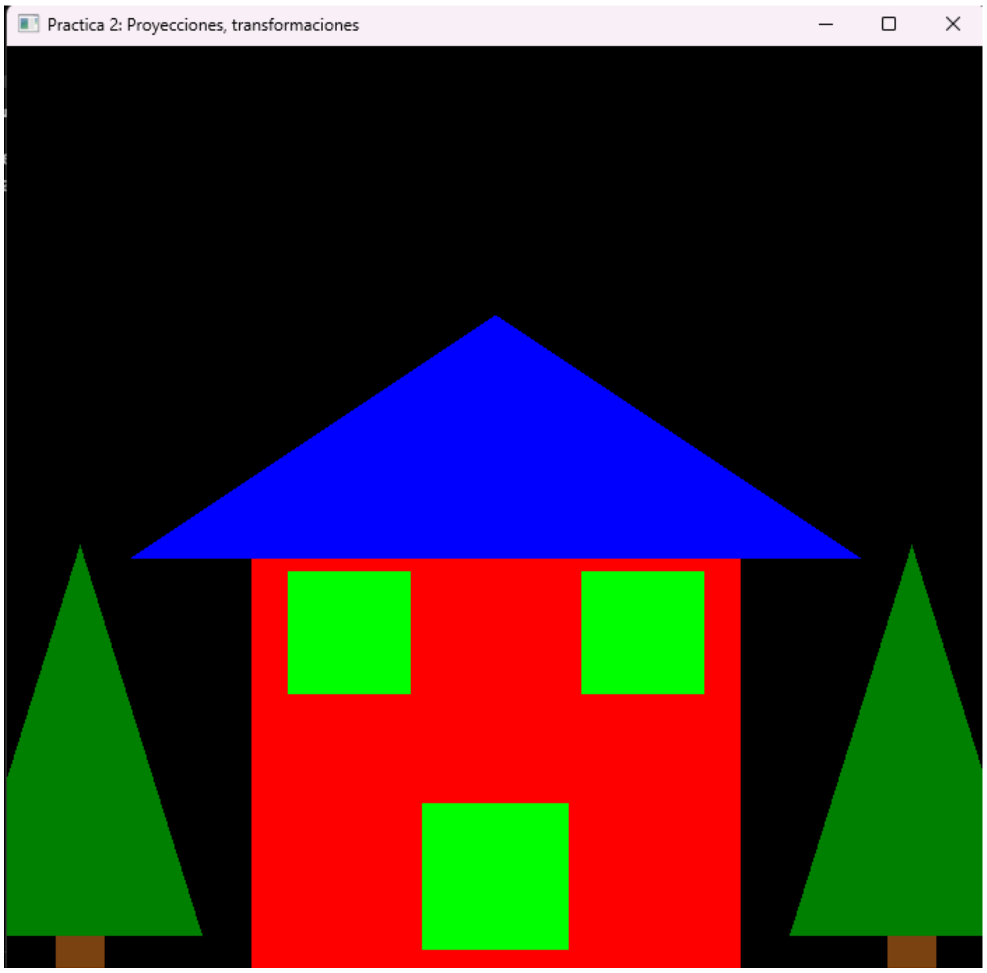
```
//se agregan los nuevos shader aqui
Shader* shader3 = new Shader();
shader3->CreateFromFiles(vShaderRojo, fShaderRojo); //Shader Rojo
shaderList.push_back(*shader3);

Shader* shader4 = new Shader();
shader4->CreateFromFiles(vShaderAzul, fShaderAzul); //Shader Azul
shaderList.push_back(*shader4);

Shader* shader5 = new Shader();
shader5->CreateFromFiles(vShaderCafe, fShaderCafe); //Shader Cafe
shaderList.push_back(*shader5);

Shader* shader6 = new Shader();
shader6->CreateFromFiles(vShaderVerdeClaro, fShaderVerdeClaro); //Shader Verde
Claro
shaderList.push_back(*shader6);

Shader* shader7 = new Shader();
shader7->CreateFromFiles(vShaderVerdeOscuro, fShaderVerdeOscuro); //Shader
Verde Oscuro
shaderList.push_back(*shader7);
```

Usando lo proyección ortogonal dibujaran las las figuras tomando en y se usara el translate y el Scale para darle un tamaño adecuado y colorarlas en una posición correcta

```
//3D
//Piramide azul
shaderList[3].useShader();
uniformModel = shaderList[3].getModelLocation();
uniformProjection = shaderList[3].getProjectLocation();
angulo += 0.1;

model = glm::mat4(1.0);
model = glm::translate(model, glm::vec3(0.0f, 0.2f, -1.0f));
model = glm::scale(model, glm::vec3(1.5f, 0.5f, 1.5f));

glUniformMatrix4fv(uniformModel, 1, GL_FALSE, glm::value_ptr(model));
glUniformMatrix4fv(uniformProjection, 1, GL_FALSE, glm::value_ptr(projection));
meshList[0]->RenderMesh();

//Cubo rojo
shaderList[2].useShader();
uniformModel = shaderList[2].getModelLocation();
uniformProjection = shaderList[2].getProjectLocation();

model = glm::mat4(1.0);
model = glm::translate(model, glm::vec3(0.0f, -0.5f, -2.0f));
glUniformMatrix4fv(uniformModel, 1, GL_FALSE, glm::value_ptr(model));
glUniformMatrix4fv(uniformProjection, 1, GL_FALSE, glm::value_ptr(projection));
meshList[1]->RenderMesh();

//Cubos verdes
shaderList[5].useShader();
uniformModel = shaderList[5].getModelLocation();
uniformProjection = shaderList[5].getProjectLocation();

model = glm::mat4(1.0);
model = glm::translate(model, glm::vec3(0.0f, -0.7f, -1.6f));
model = glm::scale(model, glm::vec3(0.3f, 0.3f, 0.3f));
glUniformMatrix4fv(uniformModel, 1, GL_FALSE, glm::value_ptr(model));
glUniformMatrix4fv(uniformProjection, 1, GL_FALSE, glm::value_ptr(projection));
meshList[1]->RenderMesh();

model = glm::mat4(1.0);
model = glm::translate(model, glm::vec3(-0.3f, -0.2f, -1.6f));
model = glm::scale(model, glm::vec3(0.25f, 0.25f, 0.25f));
glUniformMatrix4fv(uniformModel, 1, GL_FALSE, glm::value_ptr(model));
meshList[1]->RenderMesh();

model = glm::mat4(1.0);
model = glm::translate(model, glm::vec3(0.3f, -0.2f, -1.6f));
model = glm::scale(model, glm::vec3(0.25f, 0.25f, 0.25f));
glUniformMatrix4fv(uniformModel, 1, GL_FALSE, glm::value_ptr(model));
meshList[1]->RenderMesh();

//Cubos cafes
shaderList[4].useShader();
uniformModel = shaderList[4].getModelLocation();
uniformProjection = shaderList[4].getProjectLocation();

model = glm::mat4(1.0);
model = glm::translate(model, glm::vec3(0.85f, -0.85f, -1.65f));
model = glm::scale(model, glm::vec3(0.1f, 0.1f, 0.09f));
glUniformMatrix4fv(uniformModel, 1, GL_FALSE, glm::value_ptr(model));
glUniformMatrix4fv(uniformProjection, 1, GL_FALSE, glm::value_ptr(projection));
meshList[1]->RenderMesh();

model = glm::mat4(1.0);
model = glm::translate(model, glm::vec3(-0.85f, -0.85f, -1.65f));
model = glm::scale(model, glm::vec3(0.1f, 0.1f, 0.09f));
glUniformMatrix4fv(uniformModel, 1, GL_FALSE, glm::value_ptr(model));
meshList[1]->RenderMesh();
```

```
//Piramides verdes oscuros
shaderList[6].useShader();
uniformModel = shaderList[6].getModelLocation();
uniformProjection = shaderList[6].getProjectLocation();

model = glm::mat4(1.0);
model = glm::translate(model, glm::vec3(0.85f, -0.42f, -1.6f));
model = glm::scale(model, glm::vec3(0.5f, 0.8f, 0.5f));
glUniformMatrix4fv(uniformModel, 1, GL_FALSE, glm::value_ptr(model));
glUniformMatrix4fv(uniformProjection, 1, GL_FALSE, glm::value_ptr(projection));
meshList[0]->RenderMesh();

model = glm::mat4(1.0);
model = glm::translate(model, glm::vec3(-0.85f, -0.42f, -1.6f));
model = glm::scale(model, glm::vec3(0.5f, 0.8f, 0.5f));
glUniformMatrix4fv(uniformModel, 1, GL_FALSE, glm::value_ptr(model));
meshList[0]->RenderMesh();
```

al ejecutar el código se puede observar una ejecución correcta de lo esperado, se mostró en la ventana emergente la estructura deseada



Conclusión:

Se implementó el uso de cubos y pirámides como formas básicas para construir la estructura. Se reemplazó el uso del shader con color por shaders personalizados que permiten una mayor flexibilidad en la representación de colores específicos: rojo, verde, azul, café y verde oscuro. Lo cual se me dificulto mucho por que se me olvido llamar los shaders en la función de crear shaders

Esta aproximación no solo enriquece la visualización de la casa al dar una mayor diversidad cromática, sino que también mejora la comprensión del pipeline de gráficos y la importancia de los shaders en el renderizado de escenas 3D.

La creación de shaders individuales proporciona una experiencia práctica en la manipulación de los colores a nivel de fragment shader, subrayando la relevancia del control preciso sobre los aspectos visuales en el desarrollo de aplicaciones gráficas interactivas.