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: Sanchez Villalpando Johan

Nº de Cuenta: 422028657

GRUPO DE LABORATORIO: 04

GRUPO DE TEORÍA: 02

SEMESTRE 2025-2

FECHA DE ENTREGA LÍMITE: 22/02/25

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.

Para esta practica se solicitan 2 ejercicios, el primero de ellos es realizar las letras de nuestro nombre de la práctica 1, pero ahora con colores diferentes, aprovechando el uso del MeshColor y la función CrearFigurasyLetras, visto durante la práctica, y el primer ejercicio, donde además de los vértices de cada figura, incluimos también el vector de los colores RGB, para posteriormente renderizar cada letra con su respectivo color, sacándolos de la meshColorList.

```
std::vector<MeshColor*> meshColorList;
```

```
meshColorList[0]->RenderMeshColor();
```

Comenzando por los vértices de las letras, se reutilizaron los hechos en la práctica 1, de modo que ahora lo que falta es agregar el vector de color para cada vértice y actualizar el valor de la cantidad de valores, la cual a diferencia de las figuras se expande más ya que se utiliza una mayor cantidad de triángulos.

Añadiendo los vértices de cada letra (JSV) y su color en la función CrearLetrasyFiguras, donde también se enlistan en la meshcolorlist.

```
GLfloat verticesj[] = {
    //J
    //X    Y    Z          R   G   B
    -0.9f, 0.4f, 0.0f,     1.0f,0.0f,0.0f,
    -0.7f, 0.4f, 0.0f,     1.0f,0.0f,0.0f,
    -0.7f, 0.3f, 0.0f,     1.0f,0.0f,0.0f,

    -0.9f, 0.4f, 0.0f,     1.0f,0.0f,0.0f,
    -0.9f,0.3f,0.0f,     1.0f,0.0f,0.0f,
    -0.7f,0.3f,0.0f,     1.0f,0.0f,0.0f,

    -0.7f,0.3f,0.0f,     1.0f,0.0f,0.0f,
    -0.7f,0.4f,0.0f,     1.0f,0.0f,0.0f,
    -0.5f,0.4f,0.0f,     1.0f,0.0f,0.0f,

    -0.5f,0.4f,0.0f,     1.0f,0.0f,0.0f,
    -0.5f,0.3f,0.0f,     1.0f,0.0f,0.0f,
    -0.7f,0.3f,0.0f,     1.0f,0.0f,0.0f,

    -0.65f,0.3f,0.0f,     1.0f,0.0f,0.0f,
    -0.65f,0.0f,0.0f,     1.0f,0.0f,0.0f,
    -0.75f,0.3f,0.0f,     1.0f,0.0f,0.0f,

    -0.75f,0.3f,0.0f,     1.0f,0.0f,0.0f,
    -0.65f,0.0f,0.0f,     1.0f,0.0f,0.0f,
    -0.75,0.0f,0.0f,     1.0f,0.0f,0.0f,

    -0.75f,0.0f,0.0f,     1.0f,0.0f,0.0f,
    -0.65f,0.0f,0.0f,     1.0f,0.0f,0.0f,
    -0.65f,-0.1f,0.0f,     1.0f,0.0f,0.0f,

    -0.75f,0.0f,0.0f,     1.0f,0.0f,0.0f,
    -0.75f,-0.1f,0.0f,     1.0f,0.0f,0.0f,
    -0.65f,-0.1f,0.f,     1.0f,0.0f,0.0f,

    -0.75f,0.0f,0.0f,     1.0f,0.0f,0.0f,
    -0.75f,-0.1f,0.0f,     1.0f,0.0f,0.0f,
    -0.9f,0.0f,0.0f,     1.0f,0.0f,0.0f,
};
MeshColor* jota = new MeshColor();
jota->CreateMeshColor(verticesj, 162);
meshColorList.push_back(jota);
```

```
//S
GLfloat verticesS[] = {
    -0.2f,0.3f,0.0f,     0.0f,0.5f,0.0f,
    -0.2f,0.2f,0.0f,     0.0f,0.5f,0.0f,
    0.0f,0.3f,0.0f,      0.0f,0.5f,0.0f,

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

    0.0f,0.3f,0.0f,      0.0f,0.5f,0.0f,
    0.2f,0.2f,0.0f,      0.0f,0.5f,0.0f,
    0.0f,0.2f,0.0f,      0.0f,0.5f,0.0f,

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

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

    -0.1f,0.175f,0.0f,     0.0f,0.5f,0.0f,
    0.1f,0.1f,0.0f,      0.0f,0.5f,0.0f,
    -0.1f,0.1f,0.0f,     0.0f,0.5f,0.0f,

    -0.1f,0.1f,0.0f,     0.0f,0.5f,0.0f,
    0.1f,0.1f,0.0f,      0.0f,0.5f,0.0f,
    0.1f,0.025f,0.0f,     0.0f,0.5f,0.0f,

    0.1f,0.1f,0.0f,      0.0f,0.5f,0.0f,
    0.2f,0.1f,0.0f,      0.0f,0.5f,0.0f,
    0.2f,0.0f,0.0f,      0.0f,0.5f,0.0f,

    0.1f,0.1f,0.0f,      0.0f,0.5f,0.0f,
    0.2f,0.0f,0.0f,      0.0f,0.5f,0.0f,
    0.1f,0.0f,0.0f,      0.0f,0.5f,0.0f,

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

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

    -0.2f,0.0f,0.0f,     0.0f,0.5f,0.0f,
    0.0f,-0.1f,0.0f,     0.0f,0.5f,0.0f,
    0.0f,0.0f,0.0f,      0.0f,0.5f,0.0f,
};
MeshColor* ese = new MeshColor();
ese->CreateMeshColor(verticesS, 216);
meshColorList.push_back(ese);
```

```
//V
GLfloat verticesV[] = {
    0.5f,0.4f,0.0f,     0.5f,0.5f,0.0f,
    0.6f,0.4f,0.0f,     0.5f,0.5f,0.0f,
    0.5f,0.2f,0.0f,     0.5f,0.5f,0.0f,

    0.5f,0.2f,0.0f,     0.5f,0.5f,0.0f,
    0.6f,0.2f,0.0f,     0.5f,0.5f,0.0f,
    0.6f,0.4f,0.0f,     0.5f,0.5f,0.0f,

    0.6f,0.2f,0.0f,     0.5f,0.5f,0.0f,
    0.6f,0.1f,0.0f,     0.5f,0.5f,0.0f,
    0.5f,0.2f,0.0f,     0.5f,0.5f,0.0f,

    0.5f,0.2f,0.0f,     0.5f,0.5f,0.0f,
    0.5f,0.1f,0.0f,     0.5f,0.5f,0.0f,
    0.7f,0.025f,0.0f,     0.5f,0.5f,0.0f,

    0.5f,0.1f,0.0f,     0.5f,0.5f,0.0f,
    0.7f,0.025f,0.0f,     0.5f,0.5f,0.0f,
    0.7f,-0.1f,0.0f,     0.5f,0.5f,0.0f,

    0.7f,0.025f,0.0f,     0.5f,0.5f,0.0f,
    0.7f,-0.1f,0.0f,     0.5f,0.5f,0.0f,
    0.9f,0.1f,0.0f,     0.5f,0.5f,0.0f,

    0.7f,0.025f,0.0f,     0.5f,0.5f,0.0f,
    0.9f,0.2f,0.0f,     0.5f,0.5f,0.0f,
    0.9f,0.1f,0.0f,     0.5f,0.5f,0.0f,

    0.8f,0.1f,0.0f,     0.5f,0.5f,0.0f,
    0.8f,0.2f,0.0f,     0.5f,0.5f,0.0f,
    0.9f,0.2f,0.0f,     0.5f,0.5f,0.0f,

    0.8f,0.4f,0.0f,     0.5f,0.5f,0.0f,
    0.9f,0.4f,0.0f,     0.5f,0.5f,0.0f,
    0.8f,0.2f,0.0f,     0.5f,0.5f,0.0f,

    0.9f,0.4f,0.0f,     0.5f,0.5f,0.0f,
    0.9f,0.2f,0.0f,     0.5f,0.5f,0.0f,
    0.8f,0.2f,0.0f,     0.5f,0.5f,0.0f,
};
MeshColor* uve = new MeshColor();
uve->CreateMeshColor(verticesV, 180);
meshColorList.push_back(uve);
```

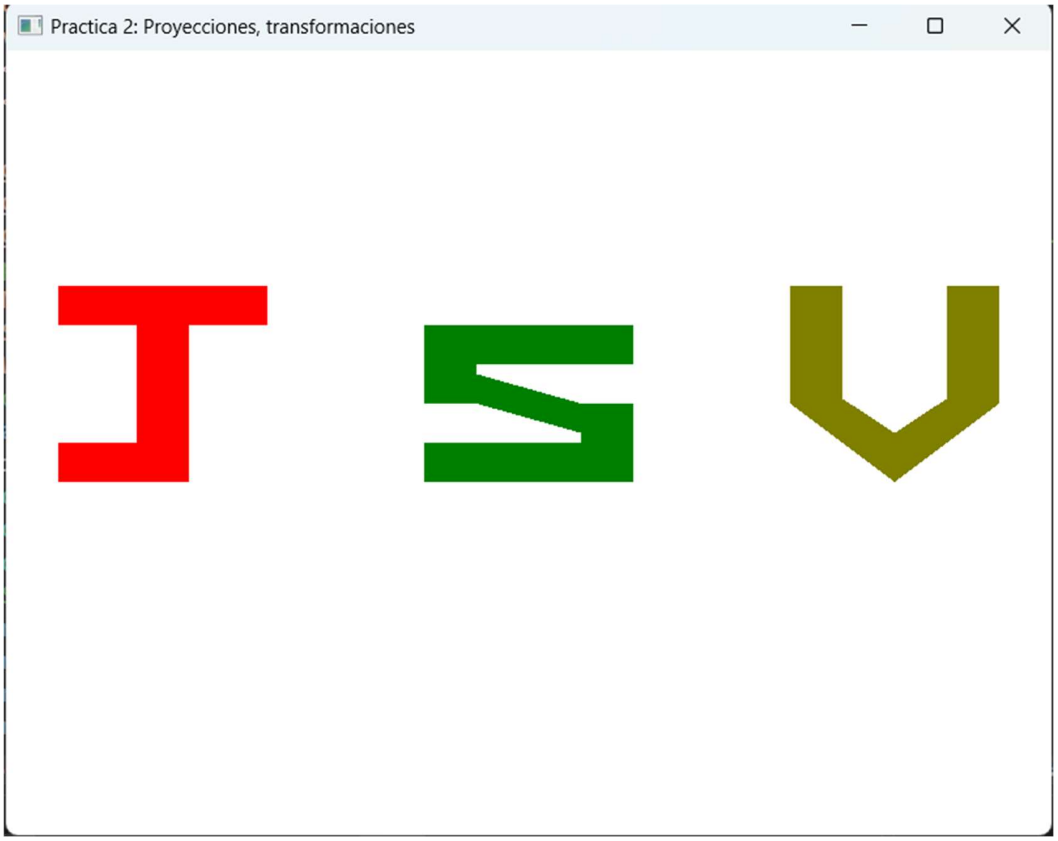
Finalmente se generan en la función main, para este ejercicio utilicé el mismo shader de CrearLetrasyFiguras.

Se llaman los modelos y se renderizan con RenderMeshColor, solo cambiando el índice de la lista meshColorList para cada letra, en este caso a diferencia del ejercicio de clase, no fue necesario utilizar transformaciones, ya que debido a la disposición de los vectores realizada en la práctica 1, las letras ya se encuentran bien posicionadas y en la escala correcta.

```
//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();

//Inicializar matriz de dimensión 4x4 que servirá como matriz de modelo para almacenar
//Generando J
model = glm::mat4(1.0);
model = glm::translate(model, glm::vec3(0.0f, 0.0f, -0.4f));
glUniformMatrix4fv(uniformModel, 1, GL_FALSE, glm::value_ptr(model));//FALSE ES PARA C
glUniformMatrix4fv(uniformProjection, 1, GL_FALSE, glm::value_ptr(projection));
meshColorList[0]->RenderMeshColor();
//Generando S
model = glm::mat4(1.0);
model = glm::translate(model, glm::vec3(0.0f, 0.0f, -0.4f));
glUniformMatrix4fv(uniformModel, 1, GL_FALSE, glm::value_ptr(model));//FALSE ES PARA C
glUniformMatrix4fv(uniformProjection, 1, GL_FALSE, glm::value_ptr(projection));
meshColorList[1]->RenderMeshColor();
//Generando V
model = glm::mat4(1.0);
model = glm::translate(model, glm::vec3(0.0f, 0.0f, -0.4f));
glUniformMatrix4fv(uniformModel, 1, GL_FALSE, glm::value_ptr(model));//FALSE ES PARA C
glUniformMatrix4fv(uniformProjection, 1, GL_FALSE, glm::value_ptr(projection));
meshColorList[2]->RenderMeshColor();
```

Finalmente se obtienen las 3 letras, cada una con colores distintos, adicionalmente se corrigieron errores con respecto a las letras realizadas en la práctica 1, el fondo al igual que la casa del ejercicio de clase, se eligió blanco.



Para el segundo ejercicio, es similar a lo visto en el ejercicio de la práctica, dibujar una casa, pero en vez de usar cuadrados y triángulos, debe ser utilizando cubos y pirámides, los cuales ya tenemos creados con la función CrearPiramide y CrearCubo incluidos en la práctica, pero en este caso se deben usar shaders distintos para cada color, por ello se toma como base el shader original para el cubo y pirámide, con la modificación de que en lugar del color clamp, se tiene que poner el color sólido de cada figura.

```
#version 330
layout (location =0) in vec3 pos;
layout (location =1) in vec3 color;
out vec4 vColor;
//uniform vec3 colors; //color uniforme
uniform mat4 model;
uniform mat4 projection;
void main()
{
    gl_Position=projection*model*vec4(pos.x,pos.y,pos.z,1.0f);
    vColor=vec4(1.0f,0.0f,0.0f,1.0f);
}
```

Se modifica el vColor, para que el shader muestre el color rojo en este caso, la variable uniform “colors” fue hecha para una prueba de cambio de color en un mismo shader, sin embargo, para esta práctica se usarán shaders independientes.

Para los nuevos shaders, se añaden en el código, para el caso del color rojo se reutilizó el shader original del cubo y la pirámide:

```
//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
//azul para triangulo
static const char* vTriazul = "shaders/vTriazul.vert";
static const char* fTriazul = "shaders/fTriazul.frag";
//verde para ventana y puerta
static const char* vVerde = "shaders/vVerde.vert";
static const char* fVerde = "shaders/fVerde.frag";
//verdoso para piramide
static const char* vVerdoso = "shaders/vVerdoso.vert";
static const char* fVerdoso = "shaders/fVerdoso.frag";
//cafe para piramide
static const char* vCafe = "shaders/vCafe.vert";
static const char* fCafe = "shaders/fCafe.frag";
```

Tambíen es necesario crear los shaders usando la función CreateShaders:

```
//shader para cuadrado rojo
Shader* shader1 = new Shader(); //shader para usar índices: objetos: cubo y  pirámide
shader1->CreateFromFiles(vShader, fShader); //indice 0
shaderList.push_back(*shader1);

Shader* shader2 = new Shader();//shader para usar color como parte del VAO: letras
shader2->CreateFromFiles(vShaderColor, fShaderColor); //índice 1
shaderList.push_back(*shader2);

//CREACIÓN de SHADERS figuras 3d con color
//shader para triangulo azul
Shader* shader3 = new Shader();
shader3->CreateFromFiles(vTriazul, fTriazul); //índice 2
shaderList.push_back(*shader3);
//shader para cuadrado verde
Shader* shader4 = new Shader();
shader4->CreateFromFiles(vVerde, fVerde);//índice 3
shaderList.push_back(*shader4);
//shader para triangulo verde
Shader* shader5 = new Shader();
shader5->CreateFromFiles(vVerdoso, fVerdoso);//índice 4
shaderList.push_back(*shader5);
//shader para cuadrado café
Shader* shader6 = new Shader();
shader6->CreateFromFiles(vCafe, fCafe);//índice 5
shaderList.push_back(*shader6);
```

Así mismo, crear los archivos de shaders en la carpeta shaders.

Todos los shaders son similares, con el único cambio del color en su main, para tener un shader de cada color que se necesita en el dibujo, modificando el vColor=vec4

```
gl_Position=projection*model*vec4(pos.x,pos.y,pos.z,1.0f);
vColor=vec4(0.0f,1.0f,0.0f,1.0f);
```

Finalmente, en el main, antes de crear cada figura 3D, se selecciona el shader que se usará, según el índice de la shaderList, para el caso del cubo rojo se usa el del índice 0, la sección comentada es para el experimento del cambio de color a través de variables uniform.

Una vez creado el cubo, se añaden transformaciones de traslación y escala, para acoplarlo al tamaño requerido para el dibujo, de forma que después a través de RenderMesh, se pueda generar en pantalla.

```
//Para el cubo rojo
shaderList[0].useShader();
uniformModel = shaderList[0].getModelLocation();
uniformProjection = shaderList[0].getProjectLocation();
//uniformColor = shaderList[0].getColorLocation(); //uso de color propio.
//glUniform3f(uniformColor, 1.0f, 0.0f, 0.0f);
//angulo += 0.1;
//Inicializar matriz de dimensión 4x4 que servirá como matriz de modelo para almacenar las transformaciones
model = glm::mat4(1.0);
model = glm::translate(model, glm::vec3(0.0f, -0.4f, -5.0f));
model = glm::scale(model, glm::vec3(1.0, 1.3f, 1.0));
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();
```

Se realiza lo mismo con las demás formas, cambiando el índice de la meshList para alternar entre cubo y pirámide, el índice del shaderList para alternar entre los diferentes shaders y aplicando transformaciones en escala y translado para

acomodarlos correctamente, en algunos casos se aplican transformaciones de rotación, esto como parte de un experimento propio para apreciarlos mejor en la proyección de perspectiva.

```
//PARA DIBUJAR A LA piramide azul
shaderList[2].useShader();
uniformModel = shaderList[2].getModelLocation();
uniformProjection = shaderList[2].getProjectLocation();
model = glm::mat4(1.0);
model = glm::translate(model, glm::vec3(0.0f, 0.55, -4.0f));
model = glm::scale(model, glm::vec3(1.2f, 0.75f, 1.0));
glUniformMatrix4fv(uniformModel, 1, GL_FALSE, glm::value_ptr(model));
glUniformMatrix4fv(uniformProjection, 1, GL_FALSE, glm::value_ptr(projection));
meshList[0]->RenderMesh();

//PARA DIBUJAR A LA piramide verde derecha
shaderList[4].useShader();
uniformModel = shaderList[4].getModelLocation();
uniformProjection = shaderList[4].getProjectLocation();
model = glm::mat4(1.0);
model = glm::translate(model, glm::vec3(0.75f, -0.5, -4.0f));
model = glm::scale(model, glm::vec3(0.35, 0.6f, -1.0));
glUniformMatrix4fv(uniformModel, 1, GL_FALSE, glm::value_ptr(model));
glUniformMatrix4fv(uniformProjection, 1, GL_FALSE, glm::value_ptr(projection));
meshList[0]->RenderMesh();
```

```
//PARA DIBUJAR A LA piramide verde izquierda
shaderList[4].useShader();
uniformModel = shaderList[4].getModelLocation();
uniformProjection = shaderList[4].getProjectLocation();
model = glm::mat4(1.0);
model = glm::translate(model, glm::vec3(-0.75f, -0.5, -4.0f));
model = glm::scale(model, glm::vec3(0.35, 0.6f, -1.0));
glUniformMatrix4fv(uniformModel, 1, GL_FALSE, glm::value_ptr(model));
glUniformMatrix4fv(uniformProjection, 1, GL_FALSE, glm::value_ptr(projection));
meshList[0]->RenderMesh();

//PARA DIBUJAR ventana verde izquierda
shaderList[3].useShader();
uniformModel = shaderList[3].getModelLocation();
uniformProjection = shaderList[3].getProjectLocation();
model = glm::mat4(1.0);
model = glm::translate(model, glm::vec3(-0.25f, -0.08f, -4.0f));
model = glm::scale(model, glm::vec3(0.3, 0.4f, 0.25f));
glUniformMatrix4fv(uniformModel, 1, GL_FALSE, glm::value_ptr(model));
glUniformMatrix4fv(uniformProjection, 1, GL_FALSE, glm::value_ptr(projection));
meshList[1]->RenderMesh();
```

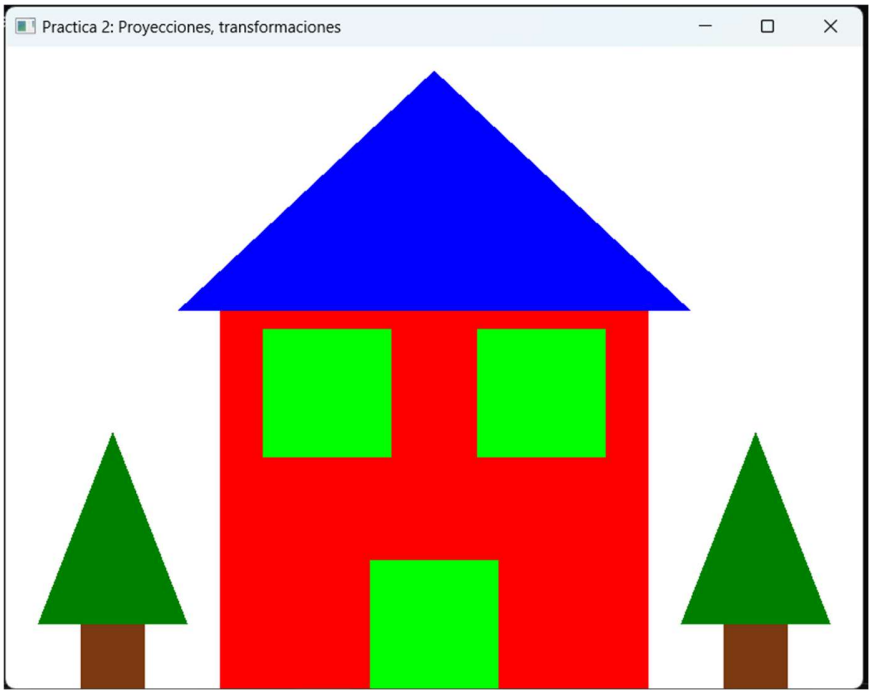
```
//PARA DIBUJAR ventana verde derecha
shaderList[3].useShader();
uniformModel = shaderList[3].getModelLocation();
uniformProjection = shaderList[3].getProjectLocation();
model = glm::mat4(1.0);
model = glm::translate(model, glm::vec3(0.25f, -0.08f, -4.0f));
model = glm::scale(model, glm::vec3(0.3, 0.4f, 0.25f));
glUniformMatrix4fv(uniformModel, 1, GL_FALSE, glm::value_ptr(model));
glUniformMatrix4fv(uniformProjection, 1, GL_FALSE, glm::value_ptr(projection));
meshList[1]->RenderMesh();

//PARA DIBUJAR puerta verde
shaderList[3].useShader();
uniformModel = shaderList[3].getModelLocation();
uniformProjection = shaderList[3].getProjectLocation();
model = glm::mat4(1.0);
model = glm::translate(model, glm::vec3(0.0f, -0.8f, -4.0f));
model = glm::scale(model, glm::vec3(0.3, 0.4f, 0.25f));
glUniformMatrix4fv(uniformModel, 1, GL_FALSE, glm::value_ptr(model));
glUniformMatrix4fv(uniformProjection, 1, GL_FALSE, glm::value_ptr(projection));
meshList[1]->RenderMesh();
```

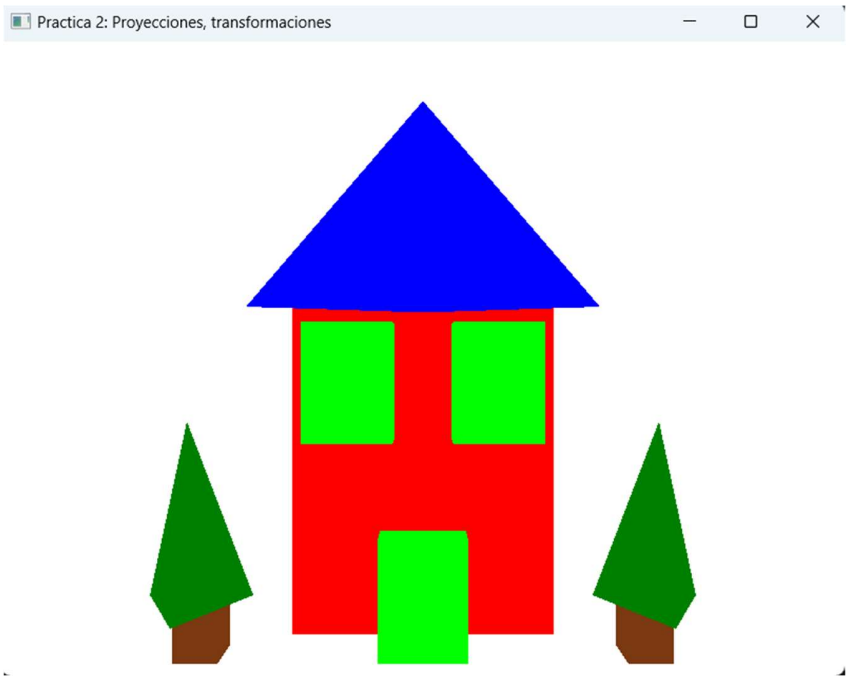
```
//PARA DIBUJAR cuadrado cafe izquierda
shaderList[5].useShader();
uniformModel = shaderList[5].getModelLocation();
uniformProjection = shaderList[5].getProjectLocation();
model = glm::mat4(1.0);
model = glm::translate(model, glm::vec3(-0.75f, -0.9f, -4.0f));
model = glm::scale(model, glm::vec3(0.15, 0.20f, 0.25f));
glUniformMatrix4fv(uniformModel, 1, GL_FALSE, glm::value_ptr(model));
glUniformMatrix4fv(uniformProjection, 1, GL_FALSE, glm::value_ptr(projection));
meshList[1]->RenderMesh();

//PARA DIBUJAR cuadrado cafe derecha
shaderList[5].useShader();
uniformModel = shaderList[5].getModelLocation();
uniformProjection = shaderList[5].getProjectLocation();
model = glm::mat4(1.0);
model = glm::translate(model, glm::vec3(0.75f, -0.9f, -4.0f));
model = glm::scale(model, glm::vec3(0.15, 0.20f, 0.25f));
glUniformMatrix4fv(uniformModel, 1, GL_FALSE, glm::value_ptr(model));
glUniformMatrix4fv(uniformProjection, 1, GL_FALSE, glm::value_ptr(projection));
meshList[1]->RenderMesh();
```

El resultado que se obtiene es el dibujo de la casa pero esta vez con cubos y pirámides.



Se adjunta además el mismo dibujo, pero utilizando la proyección de perspectiva, donde se aplicó rotación en las piramides para que se puedan apreciar mejor.



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.

Para esta práctica el principal problema fue entender los shaders, las funciones que los crean y el contenido del vertex y el fragment, ya que a pesar de haberlos visto en clase, no logré entenderlo muy bien, después de estudiar el código y el funcionamiento de los shaders se solucionó este problema e incluso pude probar trabajar con variables uniform.

3.- Conclusión:

a. Los ejercicios del reporte: Complejidad, Explicación.
b. Comentarios generales: Faltó explicar a detalle, ir más lento en alguna explicación, otros comentarios y sugerencias para mejorar desarrollo de la práctica
c. Conclusión

Esta práctica fue algo más compleja comparado con la anterior, sin embargo, no se siente un cambio de dificultad que sea complicado de manejar, por lo que los conceptos anteriores y los nuevos se juntan lo suficientemente bien para hacerlo entendible, los ejercicios fueron simples una vez se entendía como funcionaban, ya que una vez se genera la primera letra/cubo/pirámide de manera correcta, basta con repetir el procedimiento con los demás.

Pienso que faltó explicar un poco más o ir más lento en la parte de los shaders, en particular se me dificultó un poco más entender la diferencia entre meshColor y mesh, pero repasando un poco lo pudo entender. De ahí en más, la explicación fue suficiente para desarrollar la práctica.

Como conclusión puedo decir que la práctica fue interesante y se empieza a trabajar con cosas más complejas, sin embargo, se sigue sintiendo acorde a lo visto en el curso, me parece muy útil lo aprendido, como poder aplicar transformaciones para reutilizar figuras y formas, el uso de índices para optimizar la cantidad de vértices, además de el acercamiento al uso de shaders y el uso de objetos 3D para realizar dibujos simples, aunque por ahora no se aprecian tan al completo debido a la limitación de que solamente se aprecian de frente en la proyección de perspectiva.

Bibliografía en formato APA

- *LearnOpenGL - shaders*. (s/f). Learnopengl.com. Recuperado el 19 de febrero de 2025, de https://learnopengl.com/Getting-started/Shaders
- *LearnOpenGL - mesh*. (s/f). Learnopengl.com. Recuperado el 19 de febrero de 2025, de https://learnopengl.com/Model-Loading/Mesh