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 01

NOMBRE COMPLETO: Sanchez Calvillo Saida Mayela

Nº de Cuenta: 318164481

GRUPO DE LABORATORIO: 02

GRUPO DE TEORÍA: 04

SEMESTRE 2025-2

FECHA DE ENTREGA LÍMITE: 22 de febrero 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.**

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

Para poder plasmar las letras, he usado los vértices que ya habíamos generado en la Practica 1. Para cada letra vamos a crear un tipo de dato GLfloat dentro de la función *CrearLetrasyFiguras,* donde los vértices que tenemos serán la posición X, Y y Z, mas el color RGB que nosotros elijamos. Ya que tenemos eso definido, lo vamos a agregar a la lista meshColorList, ya que esta es la lista donde no se encuentra la interpolación de colores.

Así con cada letra de las iniciales que nos pide el ejercicio.

```
GLfloat vertices_s[] = {
    //X        Y         Z        R       G       B
    -0.833f,  0.332f,    0.0f,     1.0f,   0.655f,   0.937f, // S
    - 0.415f,  0.5f,      0.0f,     1.0f,   0.655f,   0.937f,
    - 0.666f,  0.5f,      0.0f,     1.0f,   0.655f,   0.937f,

    - 0.833f,  0.332f,    0.0f,     1.0f,   0.655f,   0.937f,
    - 0.415f,  0.332f,    0.0f,     1.0f,   0.655f,   0.937f,
    - 0.415f,  0.5f,      0.0f,     1.0f,   0.655f,   0.937f,

    - 0.833f,  0.166f,    0.0f,     1.0f,   0.655f,   0.937f,
    - 0.666f,  0.332f,    0.0f,     1.0f,   0.655f,   0.937f,
    - 0.833f,  0.332f,    0.0f,     1.0f,   0.655f,   0.937f,

    - 0.833f,  0.166f,    0.0f,     1.0f,   0.655f,   0.937f,
    - 0.666f,  0.166f,    0.0f,     1.0f,   0.655f,   0.937f,
    - 0.666f,  0.332f,    0.0f,     1.0f,   0.655f,   0.937f,

    - 0.583f,  0.0f,      0.0f,     1.0f,   0.655f,   0.937f,
    - 0.666f,  0.166f,    0.0f,     1.0f,   0.655f,   0.937f,
    - 0.833f,  0.166f,    0.0f,     1.0f,   0.655f,   0.937f,

    - 0.583f,  0.0f,      0.0f,     1.0f,   0.655f,   0.937f,
    - 0.415f,  0.0f,      0.0f,     1.0f,   0.655f,   0.937f,
    - 0.666f,  0.166f,    0.0f,     1.0f,   0.655f,   0.937f,

    - 0.583f, -0.166f,    0.0f,     1.0f,   0.655f,   0.937f,
    - 0.415f,  0.0f,      0.0f,     1.0f,   0.655f,   0.937f,
    - 0.583f,  0.0f,      0.0f,     1.0f,   0.655f,   0.937f,

    - 0.583f, -0.166f,    0.0f,     1.0f,   0.655f,   0.937f,
    - 0.415f, -0.166f,    0.0f,     1.0f,   0.655f,   0.937f,
    - 0.415f,  0.0f,      0.0f,     1.0f,   0.655f,   0.937f,

    - 0.833f, -0.332f,    0.0f,     1.0f,   0.655f,   0.937f,
    - 0.415f, -0.166f,    0.0f,     1.0f,   0.655f,   0.937f,
    - 0.833f, -0.166f,    0.0f,     1.0f,   0.655f,   0.937f,

    - 0.833f, -0.332f,    0.0f,     1.0f,   0.655f,   0.937f,
    - 0.583f, -0.332f,    0.0f,     1.0f,   0.655f,   0.937f,
    - 0.415f, -0.166f,    0.0f,     1.0f,   0.655f,   0.937f,
};
MeshColor* saida = new MeshColor();
saida->CreateMeshColor(vertices_s, 180);
meshColorList.push_back(saida);
```

```
GLfloat vertices_m[] = {
    //X        Y         Z        R       G       B
    -0.332f,  -0.332f,    0.0f,     0.761f,   0.878f,   0.961f,    //M
    -0.166f,   0.5f,      0.0f,     0.761f,   0.878f,   0.961f,
    -0.332f,   0.5f,      0.0f,     0.761f,   0.878f,   0.961f,

    -0.332f,  -0.332f,    0.0f,     0.761f,   0.878f,   0.961f,
    -0.166f,  -0.332f,    0.0f,     0.761f,   0.878f,   0.961f,
    -0.166f,   0.5f,      0.0f,     0.761f,   0.878f,   0.961f,

    -0.166f,   0.249f,    0.0f,     0.761f,   0.878f,   0.961f,
    -0.0f,     0.249f,    0.0f,     0.761f,   0.878f,   0.961f,
    -0.166f,   0.5f,      0.0f,     0.761f,   0.878f,   0.961f,

    -0.166f,   0.249f,    0.0f,     0.761f,   0.878f,   0.961f,
    0.0f,      0.0f,      0.0f,     0.761f,   0.878f,   0.961f,
    0.0f,      0.249f,    0.0f,     0.761f,   0.878f,   0.961f,

    0.0f,      0.0f,      0.0f,     0.761f,   0.878f,   0.961f,
    0.166f,    0.249f,    0.0f,     0.761f,   0.878f,   0.961f,
    0.0f,      0.249f,    0.0f,     0.761f,   0.878f,   0.961f,

    0.0f,      0.249f,    0.0f,     0.761f,   0.878f,   0.961f,
    0.166f,    0.249f,    0.0f,     0.761f,   0.878f,   0.961f,
    0.166f,    0.5f,      0.0f,     0.761f,   0.878f,   0.961f,

    0.166f,   -0.332f,    0.0f,     0.761f,   0.878f,   0.961f,
    0.332f,    0.5f,      0.0f,     0.761f,   0.878f,   0.961f,
    0.166f,    0.5f,      0.0f,     0.761f,   0.878f,   0.961f,

    0.166f,   -0.332f,    0.0f,     0.761f,   0.878f,   0.961f,
    0.332f,   -0.332f,    0.0f,     0.761f,   0.878f,   0.961f,
    0.332f,    0.5f,      0.0f,     0.761f,   0.878f,   0.961f,
};
MeshColor* mayela = new MeshColor();
mayela->CreateMeshColor(vertices_m, 144);
meshColorList.push_back(mayela);
```

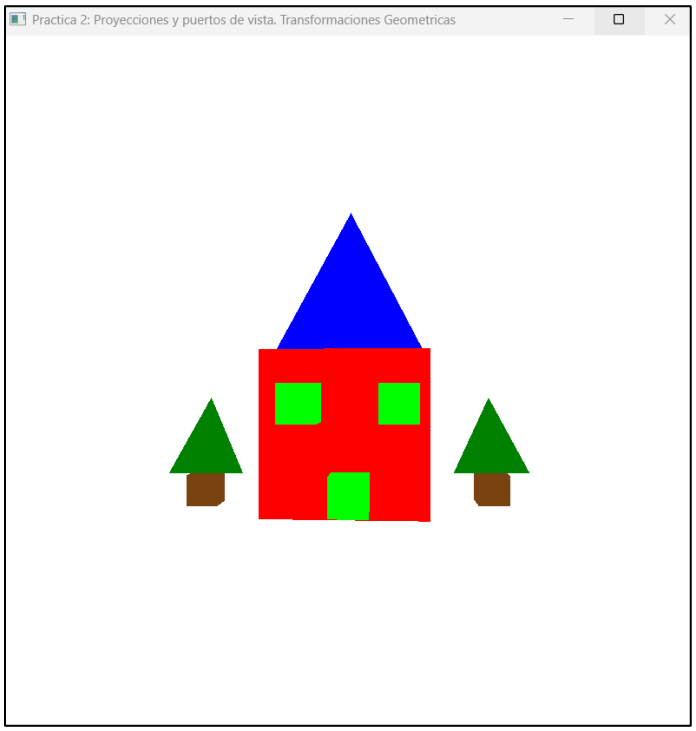
```
GLfloat vertices_c[] = {
    //X        Y         Z          R         G        B
    0.415f,   0.332f,    0.0f,     0.702f,   0.91f,   0.694f, // C
    0.833f,   0.5f,      0.0f,     0.702f,   0.91f,   0.694f,
    0.583f,   0.5f,      0.0f,     0.702f,   0.91f,   0.694f,

    0.415f,   0.332f,    0.0f,     0.702f,   0.91f,   0.694f,
    0.833f,   0.332f,    0.0f,     0.702f,   0.91f,   0.694f,
    0.833f,   0.5f,      0.0f,     0.702f,   0.91f,   0.694f,

    0.415f,  -0.166f,    0.0f,     0.702f,   0.91f,   0.694f,
    0.583f,   0.332f,    0.0f,     0.702f,   0.91f,   0.694f,
    0.415f,   0.332f,    0.0f,     0.702f,   0.91f,   0.694f,

    0.415f,  -0.166f,    0.0f,     0.702f,   0.91f,   0.694f,
    0.583f,  -0.166f,    0.0f,     0.702f,   0.91f,   0.694f,
    0.583f,   0.332f,    0.0f,     0.702f,   0.91f,   0.694f,

    0.583f,  -0.332f,    0.0f,     0.702f,   0.91f,   0.694f,
    0.833f,  -0.166f,    0.0f,     0.702f,   0.91f,   0.694f,
    0.415f,  -0.166f,    0.0f,     0.702f,   0.91f,   0.694f,

    0.583f,  -0.332f,    0.0f,     0.702f,   0.91f,   0.694f,
    0.833f,  -0.332f,    0.0f,     0.702f,   0.91f,   0.694f,
    0.833f,  -0.166f,    0.0f,     0.702f,   0.91f,   0.694f,
};
MeshColor* calvillo = new MeshColor();
calvillo->CreateMeshColor(vertices_c, 108);
meshColorList.push_back(calvillo);
```

Y como hemos impreso las figuras para formar la casa, también vamos a imprimir las letras dentro del main y por medio de transformaciones geométricas, solo le daremos valor a Z ya que necesitamos que nuestras letras estén en el origen.

Como son letras en 2D, dejaremos la proyección ortogonal.

```
glm::mat4 projection = glm::ortho(-1.0f, 1.0f, -1.0f, 1.0f, 0.1f, 100.0f);
```

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



Para este ejercicio cambiaremos de proyección y pondremos la perspectiva. Esto para poder ver que los objetos tienen profundidad y den apariencia de están en 3D.

```
glm::mat4 projection = glm::perspective(glm::radians(60.0f), mainWindow.getBufferWidth() / mainWindow.getBufferHeight(), 0.1f, 100.0f);
```

Luego, vamos a crear archivos shaders para cada color definido y podérselo dar a cada figura el lugar de que cada vértice tenga un color y exista la interpolación de colores. Se tomo *shader.vert* como base y se comento el *clamp*, de descomentó la línea anterior y así definimos nosotros que color queríamos.

shaderverde.vert | practica2main.cpp

```
1   #version 330
2   layout (location =0) in vec3 pos;
3   out vec4 vColor;
4   uniform mat4 model;
5   uniform mat4 projection;
6   void main()
7   {
8       //gl_Position=projection*model*vec4(pos,1.0f);
9       gl_Position=projection*model*vec4(pos.x, pos.y, pos.z, 1.0f);
10      vColor = vec4(0.0f, 1.0f, 0.0f, 1.0f);      // verde
11      //vColor=vec4(clamp(pos,0.0f,1.0f),1.0f);
12  }
```

En la función shaders también vamos a crear un objeto shader por cada archivo que hemos creado y lo agregaremos a la lista de shaders para poder llamarlos después.

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

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

    //2
    Shader* shader3 = new Shader(); //shader para color verde
    shader3->CreateFromFiles(vShaderVerde, fShader);
    shaderList.push_back(*shader3);

    //3
    Shader* shader4 = new Shader(); //shader para color verde oscuro
    shader4->CreateFromFiles(vShaderVerdeOscuro, fShader);
    shaderList.push_back(*shader4);

    //4
    Shader* shader5 = new Shader(); //shader para color cafe
    shader5->CreateFromFiles(vShaderCafe, fShader);
    shaderList.push_back(*shader5);

    //5
    Shader* shader6 = new Shader(); //shader para color rojo
    shader6->CreateFromFiles(vShaderRojo, fShader);
    shaderList.push_back(*shader6);

    //6
    Shader* shader7 = new Shader(); //shader para color azul
    shader7->CreateFromFiles(vShaderAzul, fShader);
    shaderList.push_back(*shader7);
}
```

Ya con eso creado, vamos a declarar y definir las rutas de archivos que contienen los shaders que vamos a utilizar.

```
//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* vShaderVerde = "shaders/shaderverde.vert";
static const char* fShaderVerde = "shaders/shaderverde.frag";
static const char* vShaderVerdeOscuro = "shaders/shaderverdeoscuro.vert";
static const char* vShaderCafe = "shaders/shadercafe.vert";
static const char* vShaderRojo = "shaders/shaderrojo.vert";
static const char* vShaderAzul = "shaders/shaderazul.vert";
```

Para el shader frag vamos a utilizar el ya existente shader.frag y funcionara de la misma forma

Para poder llamar los cubos y pirámides con sus colores respectivos, debemos recordar en que índices de cada lista se han guardado los datos y así poder llamarlos. Por ejemplo: la pirámide esta guardada en la posición 0 de meshList y el cubo en la posidion 1 de la misma lista.

Para poder elegir el color tenemos que cambiar el shader que estamos utilizando cada que cambiemos de figura y así podemos llamar el color que necesitamos. Por ejemplo: para el color rojo llamamos a la shaderList en la posición 5 y así tendremos un cubo rojo.

```
//Cubo rojo
shaderList[5].useShader();
uniformModel = shaderList[5].getModelLocation();
uniformProjection = shaderList[5].getProjectLocation();
angulo += 0.02; //Cambia la velocidad de la rotacion
//Inicializar matriz de dimensión 4x4 que servirá como matriz de modelo para almacenar las transformaciones geométricas
model = glm::mat4(1.0);
//model = glm::scale(model, glm::vec3(0.2f, 0.2f, 0.2f));
//Primero se traslada y luego rota para que el objeto se mueva en su propio centro.
model = glm::translate(model, glm::vec3(0.0f, -0.3f, -4.0f));
model = glm::rotate(model, glm::radians(angulo), glm::vec3(0.0f, 1.0f, 0.0f));
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();

//Piramide azul
shaderList[6].useShader();
uniformModel = shaderList[6].getModelLocation();
uniformProjection = shaderList[6].getProjectLocation();
model = glm::mat4(1.0);
//model = glm::scale(model, glm::vec3(0.5f, 0.5f, 0.0f));
model = glm::translate(model, glm::vec3(0.0f, 0.7f, -4.0f)); //piramide
model = glm::rotate(model, glm::radians(angulo), glm::vec3(0.0f, 1.0f, 0.0f)); //rotar en el eje y
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();
```

Recordemos que las figuras que hemos hecho se encuentran en el origen, ya con transformaciones geométricas las desplazaremos para darles un lugar y así formar la casita.

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**

Un problema que tuve fue que había creado los archivos de sharder vertex, pero no los había colocado en la ubicación que había definido al principio y eso me mantuvo revisando un buen rato que estaba mal en mi código poque las pirámides ni los cubos de imprimían. Ya después movi de lugar los archivos y se imprimieron bine las figuras.

Otro problema que tuve fue que no supe donde poner la línea de código donde nos escalaba la figura, por lo que cuando la ponía la figura tenia el mismo tamaño. Para resolverlo solo cambie de línea varias veces y después funciono la escalabilidad.

3. **Conclusión:**
    a. Los ejercicios del reporte: Complejidad, Explicación.

No fue tan complicado una vez que entiendas que todo lo que movías se metía dentro de una lista y podías llamarla después. El ejercicio de esta práctica ayudo mucho a entender el código y así poder resolver esta práctica.

    b. Comentarios generales: Faltó explicar a detalle, ir más lento en alguna explicación, otros comentarios y sugerencias para mejorar desarrollo de la práctica

Si me gustaría que el profesor fuera más lento o que nos diera chance de nosotros poder preguntar más cosas en el laboratorio.

    c. Conclusión

Esta practica estuvo entretenida e interesante. En general, se entendió cómo los shaders que creamos controlan el color y la apariencia de los objetos en una escena 3D.