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: Aguilar Pérez José Ramón

Nº de Cuenta: 317515048

GRUPO DE LABORATORIO: 02

GRUPO DE TEORÍA: 04

SEMESTRE 2025-2

FECHA DE ENTREGA LÍMITE: 22/febrero/2025

CALIFICACIÓN: _____

REPORTE DE PRÁCTICA:

1.- Ejercicios realizados.

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

Para la realización de este ejercicio, se reutilizaron los vértices generados para las letras R, A y P utilizados en la práctica anterior. Se agregan dichos vértices dentro de la función *CrearLetrasyFiguras()*, donde se le agregan los vértices de los colores, en notación RGB, para cada letra. Dentro de la lista *meshColorList,* la R está en la posición 0, la A en la 1 y la P en la 2.

```
void CrearLetrasyFiguras()
{
    //0
    GLfloat vertices_r[] = {
        //X      Y      Z            R      G      B
        -0.9f,  0.5f,   0.0f,        1.0f,  0.0f,  0.0f,
        -0.7f,  0.5f,   0.0f,        1.0f,  0.0f,  0.0f,
        -0.9f,  -0.5f,  0.0f,        1.0f,  0.0f,  0.0f,
        -0.9f,  -0.5f,  0.0f,        1.0f,  0.0f,  0.0f,
        -0.7f,  0.5f,   0.0f,        1.0f,  0.0f,  0.0f,
        -0.7f,  -0.5f,  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,
        -0.5f,  0.5f,   0.0f,        1.0f,  0.0f,  0.0f,
        -0.5f,  0.5f,   0.0f,        1.0f,  0.0f,  0.0f,
        -0.7f,  0.3f,   0.0f,        1.0f,  0.0f,  0.0f,
        -0.4f,  0.3f,   0.0f,        1.0f,  0.0f,  0.0f,
        -0.6f,  0.3f,   0.0f,        1.0f,  0.0f,  0.0f,
        -0.4f,  0.1f,   0.0f,        1.0f,  0.0f,  0.0f,
        -0.4f,  0.3f,   0.0f,        1.0f,  0.0f,  0.0f,
        -0.6f,  0.1f,   0.0f,        1.0f,  0.0f,  0.0f,
        -0.4f,  0.1f,   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,
        -0.4f,  0.1f,   0.0f,        1.0f,  0.0f,  0.0f,
        -0.5f,  -0.1f,  0.0f,        1.0f,  0.0f,  0.0f,
        -0.7f,  0.1f,   0.0f,        1.0f,  0.0f,  0.0f,
        -0.7f,  -0.1f,  0.0f,        1.0f,  0.0f,  0.0f,
        -0.5f,  -0.1f,  0.0f,        1.0f,  0.0f,  0.0f,
        -0.6f,  -0.5f,  0.0f,        1.0f,  0.0f,  0.0f,
        -0.7f,  -0.1f,  0.0f,        1.0f,  0.0f,  0.0f,
        -0.5f,  -0.1f,  0.0f,        1.0f,  0.0f,  0.0f,
        -0.5f,  -0.1f,  0.0f,        1.0f,  0.0f,  0.0f,
        -0.6f,  -0.5f,  0.0f,        1.0f,  0.0f,  0.0f,
        -0.4f,  -0.5f,  0.0f,        1.0f,  0.0f,  0.0f,

    };
    MeshColor *letra_r = new MeshColor();
    letra_r->CreateMeshColor(vertices_r, 180);
    meshColorList.push_back(letra_r);
```

```
GLfloat vertices_a[] = {
    //X      Y      Z            R      G      B
    -0.4f,  -0.5f,  0.0f,        0.0f,  1.0f,  0.0f,
    -0.2f,  -0.5f,  0.0f,        0.0f,  1.0f,  0.0f,
    -0.2f,  0.3f,   0.0f,        0.0f,  1.0f,  0.0f,
    -0.2f,  0.3f,   0.0f,        0.0f,  1.0f,  0.0f,
    -0.2f,  -0.5f,  0.0f,        0.0f,  1.0f,  0.0f,
    0.0f,   0.3f,   0.0f,        0.0f,  1.0f,  0.0f,
    -0.2f,  0.3f,   0.0f,        0.0f,  1.0f,  0.0f,
    -0.15f, 0.5f,   0.0f,        0.0f,  1.0f,  0.0f,
    0.15f,  0.5f,   0.0f,        0.0f,  1.0f,  0.0f,
    -0.2f,  0.3f,   0.0f,        0.0f,  1.0f,  0.0f,
    0.2f,   0.3f,   0.0f,        0.0f,  1.0f,  0.0f,
    0.15f,  0.5f,   0.0f,        0.0f,  1.0f,  0.0f,
    0.0f,   0.3f,   0.0f,        0.0f,  1.0f,  0.0f,
    0.2f,   0.3f,   0.0f,        0.0f,  1.0f,  0.0f,
    0.2f,   -0.5f,  0.0f,        0.0f,  1.0f,  0.0f,
    0.4f,   -0.5f,  0.0f,        0.0f,  1.0f,  0.0f,
    0.2f,   0.3f,   0.0f,        0.0f,  1.0f,  0.0f,
    0.2f,   -0.5f,  0.0f,        0.0f,  1.0f,  0.0f,
    -0.1f,  0.0f,   0.0f,        0.0f,  1.0f,  0.0f,
    -0.2f,  -0.2f,  0.0f,        0.0f,  1.0f,  0.0f,
    0.1f,   0.0f,   0.0f,        0.0f,  1.0f,  0.0f,
    0.1f,   0.0f,   0.0f,        0.0f,  1.0f,  0.0f,
    -0.2f,  -0.2f,  0.0f,        0.0f,  1.0f,  0.0f,
    0.2f,   -0.2f,  0.0f,        0.0f,  1.0f,  0.0f,

};

MeshColor* letra_a = new MeshColor();
letra_a->CreateMeshColor(vertices_a, 144);
meshColorList.push_back(letra_a);
```

```
//2
GLfloat vertices_p[] = {
    //X      Y       Z            R      G      B
    0.4f,   -0.5f,  0.0f,        0.0f,  0.0f,  1.0f,
    0.4f,   0.5f,   0.0f,        0.0f,  0.0f,  1.0f,
    0.6f,   0.5f,   0.0f,        0.0f,  0.0f,  1.0f,
    0.6f,   -0.5f,  0.0f,        0.0f,  0.0f,  1.0f,
    0.4f,   -0.5f,  0.0f,        0.0f,  0.0f,  1.0f,
    0.6f,   0.5f,   0.0f,        0.0f,  0.0f,  1.0f,
    0.6f,   0.3f,   0.0f,        0.0f,  0.0f,  1.0f,
    0.8f,   0.5f,   0.0f,        0.0f,  0.0f,  1.0f,
    0.6f,   0.5f,   0.0f,        0.0f,  0.0f,  1.0f,
    0.6f,   0.3f,   0.0f,        0.0f,  0.0f,  1.0f,
    0.8f,   0.5f,   0.0f,        0.0f,  0.0f,  1.0f,
    0.9f,   0.3f,   0.0f,        0.0f,  0.0f,  1.0f,
    0.7f,   .3f,    0.0f,        0.0f,  0.0f,  1.0f,
    0.7f,   0.1f,   0.0f,        0.0f,  0.0f,  1.0f,
    0.9f,   0.3f,   0.0f,        0.0f,  0.0f,  1.0f,
    0.9f,   0.1f,   0.0f,        0.0f,  0.0f,  1.0f,
    0.7f,   0.1f,   0.0f,        0.0f,  0.0f,  1.0f,
    0.9f,   0.3f,   0.0f,        0.0f,  0.0f,  1.0f,
    0.6f,   0.1f,   0.0f,        0.0f,  0.0f,  1.0f,
    0.8f,   -0.1f,  0.0f,        0.0f,  0.0f,  1.0f,
    0.9f,   0.1f,   0.0f,        0.0f,  0.0f,  1.0f,
    0.6f,   -0.1f,  0.0f,        0.0f,  0.0f,  1.0f,
    0.8f,   -0.1f,  0.0f,        0.0f,  0.0f,  1.0f,
    0.6f,   0.1f,   0.0f,        0.0f,  0.0f,  1.0f,

};

MeshColor* letra_p = new MeshColor();
letra_p->CreateMeshColor(vertices_p, 144);
meshColorList.push_back(letra_p);
```

Finalmente, se agregaron las instancias para generar cada una de las letras dentro del ciclo *while* del programa. Es importante tener en cuenta la posición de las letras dentro de la lista *meshColorList,* para que así se obtengan las iniciales correctas.

```
//Generando las letras de colores
//Letra R
model = glm::mat4(1.0);
model = glm::translate(model, glm::vec3(0.0f, 0.0f, -4.0f)); //Traslacion inicial
//
glUniformMatrix4fv(uniformModel, 1, GL_FALSE, glm::value_ptr(model));//FALSE ES PA
glUniformMatrix4fv(uniformProjection, 1, GL_FALSE, glm::value_ptr(projection));
meshColorList[0]->RenderMeshColor();

//Letra A
model = glm::mat4(1.0);
model = glm::translate(model, glm::vec3(0.0f, 0.0f, -4.0f));  //Traslacion inicial
//
glUniformMatrix4fv(uniformModel, 1, GL_FALSE, glm::value_ptr(model)); //FALSE ES P
glUniformMatrix4fv(uniformProjection, 1, GL_FALSE, glm::value_ptr(projection));
meshColorList[1]->RenderMeshColor();

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

- **Generar el dibujo de la casa de la clase, pero en lugar de instanciar triángulos y cuadrados será instanciando pirámides 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, se elaboraron los shaders para colorear la casa, en este caso se hicieron 5 shaders diferetens (rojo, verde, azul, verde de los árboles y café). Dentro de los *.vert,* específicamente en la función *vColor=vec4(R,G,B,alfa),* se fue modificando el valor RGB al color correspondiente del shader. En cambio, los *.frag* se dejaron sin modificación alguna.

```
static const char* vShaderRojo = "shaders/shaderrojo.vert"; //Para el color rojo
static const char* fShaderRojo = "shaders/shaderrojo.frag";

static const char* vShaderAzul = "shaders/shaderazul.vert"; //Para el color azul
static const char* fShaderAzul = "shaders/shaderazul.frag";

static const char* vShaderVerde = "shaders/shaderverde.vert"; //Para el color verde
static const char* fShaderVerde = "shaders/shaderverde.frag";

static const char* vShaderArbol = "shaders/shaderarbol.vert"; //Para el color de los arbolitos
static const char* fShaderArbol = "shaders/shaderarbol.frag";

static const char* vShaderCafe = "shaders/shadercafe.vert"; //Para el color cafe
static const char* fShaderCafe = "shaders/shadercafe.frag";
```

```
#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(1.0, 0.0, 0.0, 1.0f);
}
```

Después, se fueron agregando los nuevos shaders a la lista *ShaderList*, esto dentro de la función *CreateShaders.* Además, se comentó la posición en la lista de cada shader para facilitar las consultas futuras.

```
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: letras
    shader2->CreateFromFiles(vShaderColor, fShaderColor);
    shaderList.push_back(*shader2);
    //2
    Shader* shader3 = new Shader();  //shader para el color rojo
    shader3->CreateFromFiles(vShaderRojo, fShaderRojo);
    shaderList.push_back(*shader3);
    //3
    Shader* shader4 = new Shader();  //shader para el color azul
    shader4->CreateFromFiles(vShaderAzul, fShaderAzul);
    shaderList.push_back(*shader4);
    //4
    Shader* shader5 = new Shader();  //shader para el color verde
    shader5->CreateFromFiles(vShaderVerde, fShaderVerde);
    shaderList.push_back(*shader5);
    //5
    Shader* shader6 = new Shader();  //shader para el color de los arbolitos
    shader6->CreateFromFiles(vShaderArbol, fShaderArbol);
    shaderList.push_back(*shader6);
    //6
    Shader* shader7 = new Shader();  //shader para el color cafe
    shader7->CreateFromFiles(vShaderCafe, fShaderCafe);
    shaderList.push_back(*shader7);
}
```

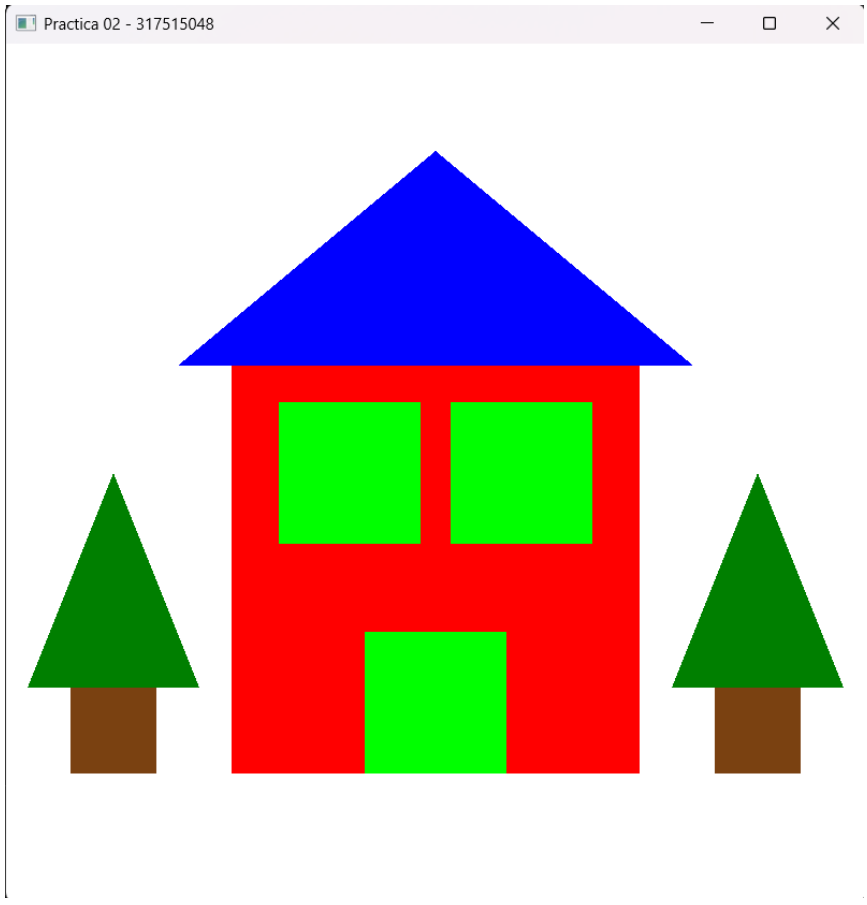
Para generar las pirámides y los cubos, se debe usar ahora la lista *meshList* (en la posición 0 se encuentra la formación de la pirámide, mientras que en la 1 se encuentra la del cubo). Dependiendo del color necesario, se cambiará el índice en la *shaderList*, obteniendo así el color deseado para cada parte de la casa dibujada.

Finalmente, con ayuda de las funciones de traslación y escala, se modificaron las figuras para que quedaran en la forma de la casa solicitada.

```
//Generando la casita
//Generando 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, -0.2f, -0.4f));
model = glm::scale(model, glm::vec3(0.95f, 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(); //cubo

//Generando a la piramide azul del techo
shaderList[3].useShader();
uniformModel = shaderList[3].getModelLocation();
uniformProjection = shaderList[3].getProjectLocation();

model = glm::mat4(1.0);
model = glm::translate(model, glm::vec3(0.0f, 0.5f, -0.3f)); //Traslacion de posicionamiento
model = glm::scale(model, glm::vec3(0.60f, 0.25f, 0.0f)); //Escalamiento de la figura
//
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(); //piramide
```

```
//Generando ventanas y puerta
shaderList[4].useShader();
uniformModel = shaderList[4].getModelLocation();
uniformProjection = shaderList[4].getProjectLocation();

//Ventana izquierda
model = glm::mat4(1.0);
model = glm::translate(model, glm::vec3(-0.2f, 0.0f, -0.2f));  //Traslacion de po
model = glm::scale(model, glm::vec3(0.33f, 0.33f, 0.0f));  //Escalamiento de la f
//
glUniformMatrix4fv(uniformModel, 1, GL_FALSE, glm::value_ptr(model)); //FALSE ES
glUniformMatrix4fv(uniformProjection, 1, GL_FALSE, glm::value_ptr(projection));
meshList[1]->RenderMesh();
//Ventana derecha
model = glm::mat4(1.0);
model = glm::translate(model, glm::vec3(0.2f, 0.0f, -0.2f));  //Traslacion de pos
model = glm::scale(model, glm::vec3(0.33f, 0.33f, 0.0f));  //Escalamiento de la f
//
glUniformMatrix4fv(uniformModel, 1, GL_FALSE, glm::value_ptr(model));  //FALSE ES
glUniformMatrix4fv(uniformProjection, 1, GL_FALSE, glm::value_ptr(projection));
meshList[1]->RenderMesh();
//Puerta
model = glm::mat4(1.0);
model = glm::translate(model, glm::vec3(0.0f, -0.535f, -0.2));  //Traslacion de p
model = glm::scale(model, glm::vec3(0.33f, 0.33f, 0.0f));  //Escalamiento de la f
//
glUniformMatrix4fv(uniformModel, 1, GL_FALSE, glm::value_ptr(model));  //FALSE ES
glUniformMatrix4fv(uniformProjection, 1, GL_FALSE, glm::value_ptr(projection));
meshList[1]->RenderMesh();
```

```
//Generando arbolitos
shaderList[5].useShader();
uniformModel = shaderList[5].getModelLocation();
uniformProjection = shaderList[5].getProjectLocation();

//Arbol izquierdo
model = glm::mat4(1.0);
model = glm::translate(model, glm::vec3(-0.75f, -0.25f, -3.0f)); //Traslacion de po
model = glm::scale(model, glm::vec3(0.2f, 0.25f, 0.0f)); //Escalamiento de la figur
//
glUniformMatrix4fv(uniformModel, 1, GL_FALSE, glm::value_ptr(model)); //FALSE ES PA
glUniformMatrix4fv(uniformProjection, 1, GL_FALSE, glm::value_ptr(projection));
meshList[0]->RenderMesh();
//Arbol derecho
model = glm::mat4(1.0);
model = glm::translate(model, glm::vec3(0.75f, -0.25f, -3.0f)); //Traslacion de pos
model = glm::scale(model, glm::vec3(0.2f, 0.25f, 0.0f)); //Escalamiento de la figur
//
glUniformMatrix4fv(uniformModel, 1, GL_FALSE, glm::value_ptr(model)); //FALSE ES PA
glUniformMatrix4fv(uniformProjection, 1, GL_FALSE, glm::value_ptr(projection));
meshList[0]->RenderMesh();

//Troncos
shaderList[6].useShader();
uniformModel = shaderList[6].getModelLocation();
uniformProjection = shaderList[6].getProjectLocation();

//Tronco izquierdo
model = glm::mat4(1.0);
model = glm::translate(model, glm::vec3(-0.75f, -0.6f, -3.0f)); //Traslacion de pos
model = glm::scale(model, glm::vec3(0.2f, 0.2f, 0.0f)); //Escalamiento de la figura
//
glUniformMatrix4fv(uniformModel, 1, GL_FALSE, glm::value_ptr(model));  //FALSE ES P
glUniformMatrix4fv(uniformProjection, 1, GL_FALSE, glm::value_ptr(projection));
meshList[1]->RenderMesh();
//Tronco derecho
model = glm::mat4(1.0);
model = glm::translate(model, glm::vec3(0.75f, -0.6f, -3.0f)); //Traslacion de posi
model = glm::scale(model, glm::vec3(0.2f, 0.2f, 0.0f)); //Escalamiento de la figura
//
glUniformMatrix4fv(uniformModel, 1, GL_FALSE, glm::value_ptr(model)); //FALSE ES PA
glUniformMatrix4fv(uniformProjection, 1, GL_FALSE, glm::value_ptr(projection));
meshList[1]->RenderMesh();
```



2.- Problemas presentados

No se presentaron problemas significativos a la hora de realizar los ejercicios.

3.- Conclusión:

a. Los ejercicios del reporte: Complejidad, Explicación.

Los ejercicios presentados para esta práctica fueron interesantes de realizar, ya que se tuvieron que aplicar los conocimientos y datos adquiridos durante la practica pasada, por lo que se mantiene fresca la información, además de que permite reutilizar información como lo son los vetores.

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

La explicación del funcionamiento y generación de los shaders, así como la explicación de las funciones para colorear las figuras, fue bastante clara y precisa.

c. Conclusión

Esta práctica me permitió comprender mejor cómo funcionan los shaders para colorear distintas figuras, como la pirámide y el cubo, el shader “principal” es el vertex, ya que en este se especifica el color que adquirirá la figura, mientras que en el fragment no se hicieron cambios. Como se viene viendo desde la practica pasada, uno de los puntos más importantes son los vértices, por lo que es importante no cometer errores a la hora de declararlos.

1. Bibliografía en formato APA

- OpenGL Programming/Basics/Color. (s/f). Wikibooks.org. Recuperado el 14 de febrero de 2025, de https://en.wikibooks.org/wiki/OpenGL_Programming/Basics/Color

- Tutorial 8: Shading básico. (s/f). Opengl-tutorial.org. Recuperado el 23 de febrero de 2025, de http://www.opengl-tutorial.org/es/beginners-tutorials/tutorial-8-basic-shading/