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: Tapia Ledesma Angel Hazel

Nº de Cuenta: 320070358

GRUPO DE LABORATORIO: 02

GRUPO DE TEORÍA: 02

SEMESTRE 2026-1

FECHA DE ENTREGA LÍMITE: 31-08-25

CALIFICACIÓN: _____

Actividades realizadas

1.- Dibujar las iniciales de sus nombres, cada letra de un color diferente
2.- 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

Descripción de las actividades

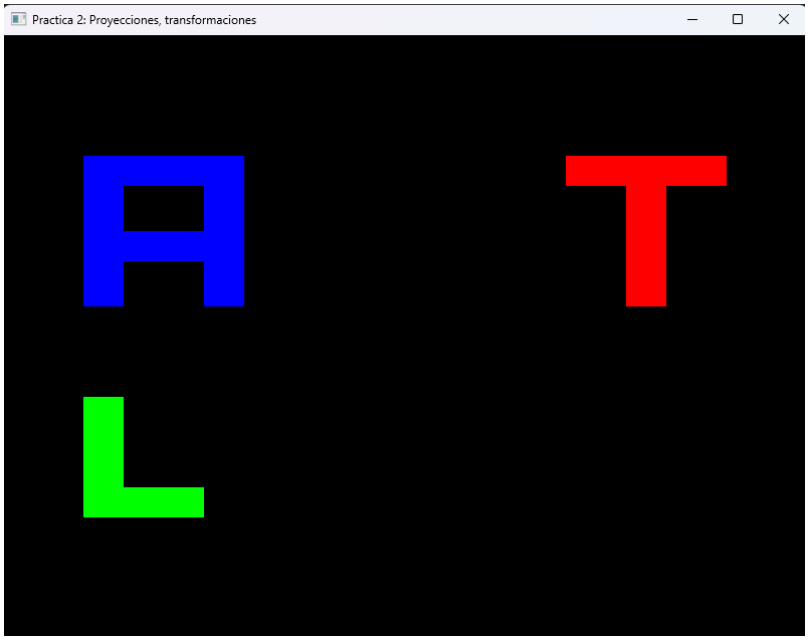
Para el desarrollo de la primera actividad, únicamente copiamos y pegamos las coordenadas de los vértices de las letras (ya obtenido en el primer ejercicio de clase), y además agregamos 3 componentes que representan el color de vértice (RGB), esto en la función "CrearLetrasyFiguras", por lo que cada letra se convirtió en un nuevo objeto de la lista meshColor, tener los objetos por aparte, hizo que pudiéramos darles colores distintos. Una vez realizado este paso, lo que se hizo en el main fue usar el shader correspondiente, en este caso el 1 y después se realizó el dibujado de cada letra con su respectivo modelo.

Código generado:

```
void CrearLetrasyFiguras()
{
    GLfloat vertices_letraA[] = {
        // A
        //x     y       z       R       G       B
        -0.7f, 0.6f,0.0f, 0.0f, 0.0f, 1.0f,
        -0.8f,0.6f, 0.0f, 0.0f, 0.0f, 1.0f,
        -0.8f,0.1f,0.0f, 0.0f, 0.0f, 1.0f,
        -0.8f,0.1f,0.0f, 0.0f, 0.0f, 1.0f,
        -0.7f,0.1f,0.0f, 0.0f, 0.0f, 1.0f,
        -0.7f, 0.6f,0.0f, 0.0f, 0.0f, 1.0f,
        -0.5f, 0.35f,0.0f, 0.0f, 0.0f, 1.0f,
        -0.7f, 0.35f,0.0f, 0.0f, 0.0f, 1.0f,
        -0.7f, 0.25f,0.0f, 0.0f, 0.0f, 1.0f,
        -0.5f, 0.35f,0.0f, 0.0f, 0.0f, 1.0f,
        -0.7f, 0.25f,0.0f, 0.0f, 0.0f, 1.0f,
        -0.5f, 0.25f,0.0f, 0.0f, 0.0f, 1.0f,
        -0.5f, 0.1f,0.0f, 0.0f, 0.0f, 1.0f,
        -0.4f, 0.1f,0.0f, 0.0f, 0.0f, 1.0f,
        -0.4f, 0.6f,0.0f, 0.0f, 0.0f, 1.0f,
        -0.4f, 0.6f,0.0f, 0.0f, 0.0f, 1.0f,
        -0.5f, 0.6f,0.0f, 0.0f, 0.0f, 1.0f,
        -0.5f, 0.1f,0.0f, 0.0f, 0.0f, 1.0f,
        -0.5f, 0.6f,0.0f, 0.0f, 0.0f, 1.0f,
        -0.7f, 0.6f,0.0f, 0.0f, 0.0f, 1.0f,
        -0.7f, 0.5f,0.0f, 0.0f, 0.0f, 1.0f,
        -0.7f, 0.5f,0.0f, 0.0f, 0.0f, 1.0f,
        -0.5f, 0.5f,0.0f, 0.0f, 0.0f, 1.0f,
        -0.5f, 0.6f,0.0f, 0.0f, 0.0f, 1.0f,
    };
    MeshColor* letraA = new MeshColor();
    letraA->CreateMeshColor(vertices_letraA, 144);
    meshColorList.push_back(letraA);
```

```
    GLfloat vertices_letraT[] = {
        // T
        //x     y       z       R       G       B
        0.8f, 0.6f,0.0f, 1.0f, 0.0f, 0.0f,
        0.4f, 0.6f,0.0f, 1.0f, 0.0f, 0.0f,
        0.4f, 0.5f,0.0f, 1.0f, 0.0f, 0.0f,
        0.4f, 0.5f,0.0f, 1.0f, 0.0f, 0.0f,
        0.8f, 0.5f,0.0f, 1.0f, 0.0f, 0.0f,
        0.8f, 0.6f,0.0f, 1.0f, 0.0f, 0.0f,
        0.55f, 0.5f,0.0f, 1.0f, 0.0f, 0.0f,
        0.55f, 0.1f,0.0f, 1.0f, 0.0f, 0.0f,
        0.65f, 0.1f,0.0f, 1.0f, 0.0f, 0.0f,
        0.55f, 0.5f,0.0f, 1.0f, 0.0f, 0.0f,
        0.65f, 0.5f,0.0f, 1.0f, 0.0f, 0.0f,
        0.65f, 0.1f,0.0f, 1.0f, 0.0f, 0.0f,
    };
    MeshColor* letraT = new MeshColor();
    letraT->CreateMeshColor(vertices_letraT, 77);
    meshColorList.push_back(letraT);

    GLfloat vertices_letraL[] = {
        // L
        -0.8f, -0.2f,0.0f, 0.0f, 1.0f, 0.0f,
        -0.8f, -0.6f,0.0f, 0.0f, 1.0f, 0.0f,
        -0.7f, -0.6f,0.0f, 0.0f, 1.0f, 0.0f,
        -0.8f, -0.2f,0.0f, 0.0f, 1.0f, 0.0f,
        -0.7f, -0.2f,0.0f, 0.0f, 1.0f, 0.0f,
        -0.7f, -0.6f,0.0f, 0.0f, 1.0f, 0.0f,
        -0.5f, -0.5f,0.0f, 0.0f, 1.0f, 0.0f,
        -0.7f, -0.5f,0.0f, 0.0f, 1.0f, 0.0f,
        -0.7f, -0.6f,0.0f, 0.0f, 1.0f, 0.0f,
        -0.5f, -0.5f,0.0f, 0.0f, 1.0f, 0.0f,
        -0.5f, -0.6f,0.0f, 0.0f, 1.0f, 0.0f,
        -0.7f, -0.6f,0.0f, 0.0f, 1.0f, 0.0f

    };
    MeshColor* letraL = new MeshColor();
    letraL->CreateMeshColor(vertices_letraL, 77);
    meshColorList.push_back(letraL);

}
```

Ejecución



Para el 2do punto, lo primero fue hacer los shaders correspondientes a cada color, en este caso en el .frag, en estos archivos fue donde se creó la lógica para el color de cada shader, después se modificó el .vert (el mismo para todos los shaders), ya que este tenía la línea de color dentro de su código, pero cómo ahora estábamos usando el color en cada .frag, eliminé esta línea. Posteriormente a esto, en el main, se tuvieron que crear las variables para cada shader y añadir su ruta; Después, en la función "createShaders", se tuvo que realizar el proceso que seguían los shaders que ya teníamos, para agregarlos a la lista de shaders, finalmente, dentro del while, le dimos a cada figura su shader y mesh correspondiente, para que se generara la figura con el color que se buscaba, también se le agregó la rotación, para que si queremos observarlo con la proyección de perspectiva, podamos ver un poco más de la figura.

Código generado

verdeOscuro.frag

```
#version 330 core
in vec4 vColor;
out vec4 color;
void main()
{
    color = vec4(0.0f, 0.5f, 0.0f, 1.0f);
}
```

verde.frag

```
#version 330 core
in vec4 vColor;
out vec4 color;
void main()
{
    color = vec4(0.0f, 1.0f, 0.0f, 1.0f);
}
```

cafe.frag

```
#version 330 core
in vec4 vColor;
out vec4 color;
void main()
{
    color = vec4(0.478f, 0.255f, 0.067f, 1.0f);
}
```

Azul.frag

```
#version 330 core
in vec4 vColor;
out vec4 color;
void main()
{
    color = vec4(0.0f, 0.0f, 1.0f, 1.0f);
}
```

Rojo.frag

```
#version 330 core
in vec4 vColor;
out vec4 color;
void main()
{
    color = vec4(1.0f, 0.0f, 0.0f, 1.0f);
}
```

Shader.vert

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

Main:

```
// EJERCICIO 2

// Cuadrado rojo grande
shaderList[2].useShader(); // Asegurarse de que el shader rojo esté activo
uniformModel = shaderList[2].getModelLocation();
uniformProjection = shaderList[2].getProjectLocation();
model = glm::mat4(1.0);
model = glm::translate(model, glm::vec3(0.0f, -0.345f, -4.0f));
model = glm::rotate(model, angulo * toRadians, glm::vec3(0.0f, 1.0f, 0.0f));
model = glm::scale(model, glm::vec3(1.0f, 1.29f, 1.0f));
glUniformMatrix4fv(uniformModel, 1, GL_FALSE, glm::value_ptr(model));//FALSE ES PARA QUE NO SEA TRANSPUESTA y se envían al shader como variables de tipo uniform
glUniformMatrix4fv(uniformProjection, 1, GL_FALSE, glm::value_ptr(projection));
meshList[1]->RenderMesh();

// Triangulo azul
shaderList[4].useShader(); // Se llama la shader en la ubicacion 1 dentro de la lista de shader (shader color en este caso)
uniformModel = shaderList[4].getModelLocation();
uniformProjection = shaderList[4].getProjectLocation();
model = glm::mat4(1.0);
model = glm::translate(model, glm::vec3(0.0f, 0.65f, -4.0f));
model = glm::rotate(model, angulo * toRadians, glm::vec3(0.0f, 1.0f, 0.0f));
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 y se envían al shader como variables de tipo uniform
glUniformMatrix4fv(uniformProjection, 1, GL_FALSE, glm::value_ptr(projection));
meshList[0]->RenderMesh();

// Cuadrado verde izquierda
shaderList[3].useShader(); // Se llama la shader en la ubicacion 1 dentro de la lista de shader (shader color en este caso)
uniformModel = shaderList[3].getModelLocation();
uniformProjection = shaderList[3].getProjectLocation();
model = glm::mat4(1.0);
model = glm::translate(model, glm::vec3(-0.25f, 0.0, -3.9f));
model = glm::rotate(model, angulo * toRadians, glm::vec3(0.0f, 1.0f, 0.0f));
model = glm::scale(model, glm::vec3(0.30f, 0.35f, 1.0f));
glUniformMatrix4fv(uniformModel, 1, GL_FALSE, glm::value_ptr(model));//FALSE ES PARA QUE NO SEA TRANSPUESTA y se envían al shader como variables de tipo uniform
glUniformMatrix4fv(uniformProjection, 1, GL_FALSE, glm::value_ptr(projection));
meshList[1]->RenderMesh();

// Cuadrado verde derecha
shaderList[3].useShader(); // Se llama la shader en la ubicacion 1 dentro de la lista de shader (shader color en este caso)
uniformModel = shaderList[3].getModelLocation();
uniformProjection = shaderList[3].getProjectLocation();
model = glm::mat4(1.0);
model = glm::translate(model, glm::vec3(0.25f, 0.0, -3.9f));
model = glm::rotate(model, angulo * toRadians, glm::vec3(0.0f, 1.0f, 0.0f));
model = glm::scale(model, glm::vec3(0.30f, 0.35f, 1.0f));
glUniformMatrix4fv(uniformModel, 1, GL_FALSE, glm::value_ptr(model));//FALSE ES PARA QUE NO SEA TRANSPUESTA y se envían al shader como variables de tipo uniform
glUniformMatrix4fv(uniformProjection, 1, GL_FALSE, glm::value_ptr(projection));
meshList[1]->RenderMesh();
```

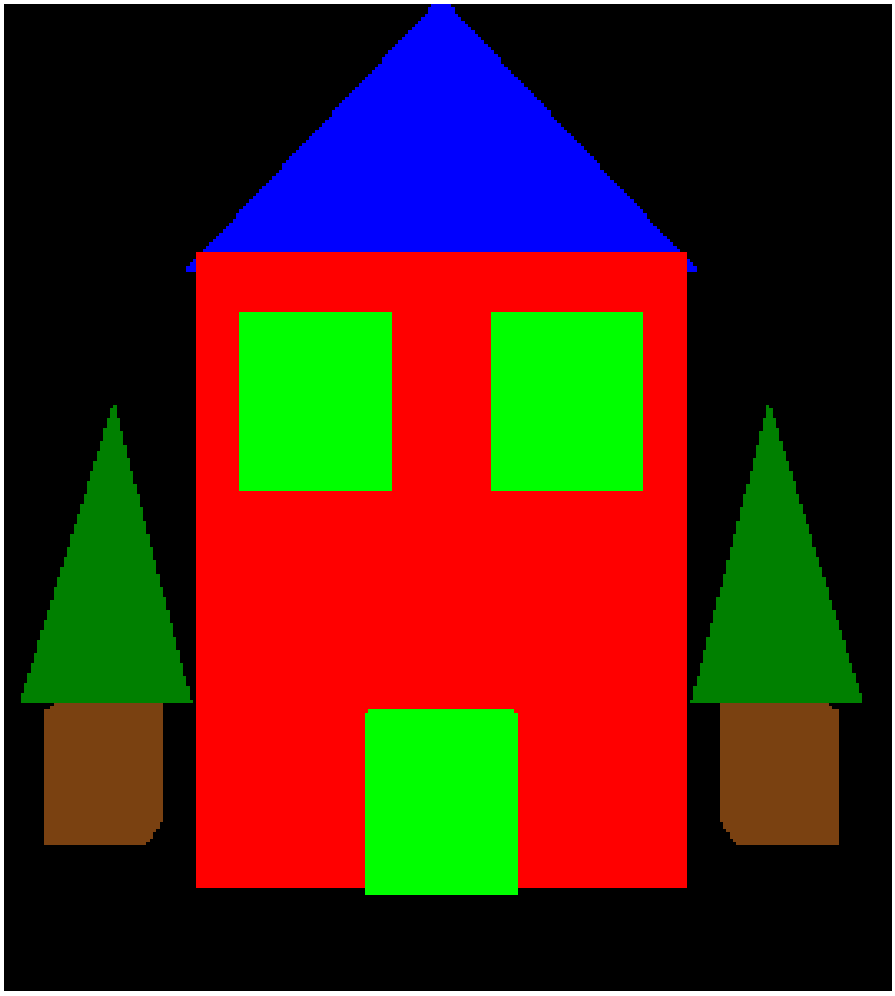
```
// Cuadrado verde abajo
shaderList[3].useShader(); // Se llama la shader en la ubicacion 1 dentro de la lista de shader (shader color en este caso)
uniformModel = shaderList[3].getModelLocation();
uniformProjection = shaderList[3].getProjectLocation();
model = glm::mat4(1.0);
model = glm::translate(model, glm::vec3(0.0f, -0.8, -3.9f));
model = glm::rotate(model, angulo * toRadians, glm::vec3(0.0f, 1.0f, 0.0f));
model = glm::scale(model, glm::vec3(0.30f, 0.35f, 1.0f));
glUniformMatrix4fv(uniformModel, 1, GL_FALSE, glm::value_ptr(model));//FALSE ES PARA QUE NO SEA TRANSPUESTA y se envían al shader como variables de tipo uniform
glUniformMatrix4fv(uniformProjection, 1, GL_FALSE, glm::value_ptr(projection));
meshList[1]->RenderMesh();

// Cuadrado cafe izquierda
shaderList[5].useShader(); // Se llama la shader en la ubicacion 1 dentro de la lista de shader (shader color en este caso)
uniformModel = shaderList[5].getModelLocation();
uniformProjection = shaderList[5].getProjectLocation();
model = glm::mat4(1.0);
model = glm::translate(model, glm::vec3(-0.78f, -0.85, -4.0f));
model = glm::rotate(model, angulo * toRadians, glm::vec3(0.0f, 1.0f, 0.0f));
model = glm::scale(model, glm::vec3(0.23f, 0.30f, 0.23f));
glUniformMatrix4fv(uniformModel, 1, GL_FALSE, glm::value_ptr(model));//FALSE ES PARA QUE NO SEA TRANSPUESTA y se envían al shader como variables de tipo uniform
glUniformMatrix4fv(uniformProjection, 1, GL_FALSE, glm::value_ptr(projection));
meshList[1]->RenderMesh();

// Cuadrado cafe derecha
shaderList[5].useShader(); // Se llama la shader en la ubicacion 1 dentro de la lista de shader (shader color en este caso)
uniformModel = shaderList[5].getModelLocation();
uniformProjection = shaderList[5].getProjectLocation();
model = glm::mat4(1.0);
model = glm::translate(model, glm::vec3(0.78f, -0.85, -4.0f));
model = glm::rotate(model, angulo * toRadians, glm::vec3(0.0f, 1.0f, 0.0f));
model = glm::scale(model, glm::vec3(0.23f, 0.3f, 0.23f));
glUniformMatrix4fv(uniformModel, 1, GL_FALSE, glm::value_ptr(model));//FALSE ES PARA QUE NO SEA TRANSPUESTA y se envían al shader como variables de tipo uniform
glUniformMatrix4fv(uniformProjection, 1, GL_FALSE, glm::value_ptr(projection));
meshList[1]->RenderMesh();

// Triangulo verde izquierda
shaderList[6].useShader(); // Se llama la shader en la ubicacion 1 dentro de la lista de shader (shader color en este caso)
uniformModel = shaderList[6].getModelLocation();
uniformProjection = shaderList[6].getProjectLocation();
model = glm::mat4(1.0);
model = glm::translate(model, glm::vec3(-0.78f, -0.35, -4.0f));
model = glm::rotate(model, angulo * toRadians, glm::vec3(0.0f, 1.0f, 0.0f));
model = glm::scale(model, glm::vec3(0.4f, 0.7f, 0.4f));
glUniformMatrix4fv(uniformModel, 1, GL_FALSE, glm::value_ptr(model));//FALSE ES PARA QUE NO SEA TRANSPUESTA y se envían al shader como variables de tipo uniform
glUniformMatrix4fv(uniformProjection, 1, GL_FALSE, glm::value_ptr(projection));
meshList[0]->RenderMesh();
```

```
// Triangulo verde derecha
shaderList[6].useShader(); // Se llama la shader en la ubicacion 1 dentro de la lista de shader (shader color en este caso)
uniformModel = shaderList[6].getModelLocation();
uniformProjection = shaderList[6].getProjectLocation();
model = glm::mat4(1.0);
model = glm::translate(model, glm::vec3(0.78f, -0.35, -4.0f));
model = glm::rotate(model, angulo * toRadians, glm::vec3(0.0f, 1.0f, 0.0f));
model = glm::scale(model, glm::vec3(0.4f, 0.7f, 0.4f));
glUniformMatrix4fv(uniformModel, 1, GL_FALSE, glm::value_ptr(model));//FALSE ES PARA QUE NO SEA TRANSPUESTA y se envían al shader como variables de tipo uniform
glUniformMatrix4fv(uniformProjection, 1, GL_FALSE, glm::value_ptr(projection));
meshList[0]->RenderMesh();
```

Ejecución:

Ortho

Perspective



Problemas presentados

La práctica tuvo 2 principales problemas, enfocados en el 2do ejercicio, el primer problema fueron los shaders, ya que no tenía tanto conocimiento de esto, a pesar de que lo vimos en clase, no me quedó muy claro, así que tuve un error muy grande, ya que pensaba que el color se debía seguir manejando en el vertex shader, e implementé una lógica muy extraña, así que después de ver que no funcionaba tuve que investigar otra forma, así, solucionándolo con la que se comentó anteriormente.
El 2do problema fue en la forma de rotar las figuras, ya que puse la traslación, rotación y escala en ordenes distintos, lo cual me daba rotaciones o movimientos que no eran los que quería, finalmente me di cuenta de cuál era la forma de ponerlos (TRS) y así generar el giro que se deseaba.

Conclusión:

Esta práctica se me hizo demasiado buena, especialmente para reforzar lo aprendido en el aula a cerca de las matrices de proyección, los shaders que hay, y seguir practicando todos los conceptos que teníamos de antes. En cuanto a la complejidad, creo que, si fue algo elevada, o al menos a mí la parte de creación de shaders se me complicó, ya que en

clase se vio que hacían los archivos de .frag y .vert, pero no se mostró un caso de cómo crear varios para los colores, fuera de eso, creo que la actividad 1 estuvo muy bien, y la 2 a pesar de la dificultad, estuvo excelente para practicar y entender mejor los shaders.

La única recomendación que daría para la práctica es hacer un ejemplo de creación de shaders en la clase, para que este punto no se dificulte tanto a la hora de la práctica. Fue una práctica algo larga, pero sin duda muy interesante para aprender y aplicar todos los conceptos vistos en clase.

Bibliografía

- Khronos. (noviembre, 2020). “Fragment Shader”. Recuperado el 30 de agosto de 2025 de: https://www.khronos.org/opengl/wiki/Fragment_Shader
- LearnOpenGL. (s.f). “Transformations”. Recuperado el 30 de agosto de 2025 de: https://learnopengl.com/Getting-started/Transformations