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: Gabriel Patricio Balam Flores

Nº de Cuenta: 320280324

GRUPO DE LABORATORIO: 03

GRUPO DE TEORÍA: 04

SEMESTRE 2026-1

FECHA DE ENTREGA LÍMITE: 31/ago

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.

Ejercicio 1

En el primer ejercicio tenía que utilizar las letras hechas en la práctica pasada, pero ahora cada una debía tener un color diferente. No fue muy complicado, ya que solo consistía en copiar y pegar los vértices de mis letras, asignar el valor RGB deseado y modificar un par de parámetros en el código para dibujar los triángulos.

Ejecución del ejercicio 1

Bloques de código del ejercicio 1

```
void CrearLetrasyFiguras()
{
    GLfloat vertices_letras[] = {
            //X         Y           Z           R       G       B

            //  Para G

            -0.4f, 0.4f, 0.0f,              1.0f,   1.0f,   0.0f,
            -0.4f, 0.2f, 0.0f,              1.0f,   1.0f,   0.0f,
            -0.8f, 0.2f, 0.0f,              1.0f,   1.0f,   0.0f,

            -0.4f, 0.4f, 0.0f,              1.0f,   1.0f,   0.0f,
            -1.0f, 0.1f, 0.0f,              1.0f,   1.0f,   0.0f,
            -1.0f, 0.4f, 0.0f,              1.0f,   1.0f,   0.0f,

            -0.8f, 0.2f, 0.0f,              1.0f,   1.0f,   0.0f,
            -1.0f, 0.1f, 0.0f,              1.0f,   1.0f,   0.0f,
            -0.8f, -0.2f, 0.0f,             1.0f,   1.0f,   0.0f,

            -1.0f, 0.1f, 0.0f,              1.0f,   1.0f,   0.0f,
            -0.8f, -0.2f, 0.0f,             1.0f,   1.0f,   0.0f,
            -1.0f, -0.4f, 0.0f,             1.0f,   1.0f,   0.0f,

            -0.8f, -0.2f, 0.0f,             1.0f,   1.0f,   0.0f,// F
            -1.0f, -0.4f, 0.0f,             1.0f,   1.0f,   0.0f,// G
            -1.0f, -0.6f, 0.0f,             1.0f,   1.0f,   0.0f,// H
            0.4f, -0.55f, 0.0f,             1.0f,   0.0f,   1.0f,// F
            0.5f, -0.6f, 0.0f,              1.0f,   0.0f,   1.0f,// G
            0.4f, 0.2f, 0.0f,               1.0f,   0.0f,   1.0f,// C

            0.6f, 0.2f, 0.0f,               1.0f,   0.0f,   1.0f,// H
            0.5f, -0.6f, 0.0f,              1.0f,   0.0f,   1.0f,// G
            0.4f, 0.2f, 0.0f,               1.0f,   0.0f,   1.0f,// C

            0.6f, -0.6f, 0.0f,              1.0f,   0.0f,   1.0f,// I
            0.6f, 0.2f, 0.0f,               1.0f,   0.0f,   1.0f,// H
            0.5f, -0.6f, 0.0f,              1.0f,   0.0f,   1.0f,// G

            0.6f, 0.0f, 0.0f,               1.0f,   0.0f,   1.0f,// J
            0.8f, -0.1f, 0.0f,              1.0f,   0.0f,   1.0f,// K
            0.6f, -0.2f, 0.0f,              1.0f,   0.0f,   1.0f,// L

            0.75f, 0.0f, 0.0f,              1.0f,   0.0f,   1.0f,// M
            0.6f, 0.0f, 0.0f,               1.0f,   0.0f,   1.0f,// J
            0.8f, -0.1f, 0.0f,              1.0f,   0.0f,   1.0f,// K

            0.75f, -0.2f, 0.0f,             1.0f,   0.0f,   1.0f,// N
            0.8f, -0.1f, 0.0f,              1.0f,   0.0f,   1.0f,// K
            0.6f, -0.2f, 0.0f,              1.0f,   0.0f,   1.0f,// L
    };
    MeshColor *letras = new MeshColor();
    letras->CreateMeshColor(vertices_letras,710);
    meshColorList.push_back(letras);
}
```

```
//Projection: Matriz de Dimensión 4x4 para indicar si vemos en 2D( orthogonal) o en 3D) perspectiva
glm::mat4 projection = glm::ortho(-5.0f, 5.0f, -5.0f, 5.0f, 0.1f, 100.0f);
```

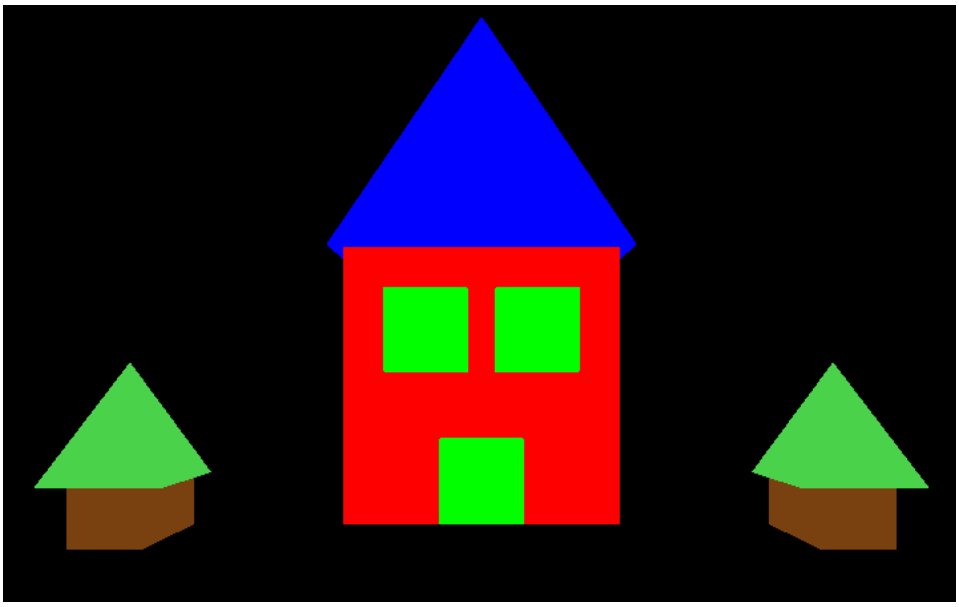
```
//Para las letras hay que usar el segundo set de shaders con índice 1 en ShaderList
shaderList[1].useShader();
uniformModel = shaderList[1].getModelLocation();
uniformProjection = shaderList[1].getProjectLocation();

// Para las letras
model = glm::mat4(1.0);
model = glm::translate(model, glm::vec3(0.0f, 0.0f, -2.0f));
model = glm::scale(model, glm::vec3(2.0f, 2.0f, 2.0f));
//
glUniformMatrix4fv(uniformModel, 1, GL_FALSE, glm::value_ptr(model));//FALSE ES PARA
glUniformMatrix4fv(uniformProjection, 1, GL_FALSE, glm::value_ptr(projection));
meshColorList[0]->RenderMeshColor();
```

Ejercicio 2

En este ejercicio tenía que realizar el mismo dibujo del ejercicio de clase, pero ahora en 3D. Para ello tuve que crear shaders que permitieran asignar el color a cada forma. A simple vista suena sencillo, pero en la práctica implicó crear los archivos, definir los colores, declarar nuevas variables para cada archivo y finalmente agregarlos a la lista de shaders. Después vino la parte más entretenida: colocar las figuras y acomodarlas hasta que se vieran lo mejor posible.

Ejecución del ejercicio 2



Bloques de código del ejercicio 2

```
// Para las formas
static const char* vShaderRed = "shaders/shaderRed.vert";
static const char* fShaderRed = "shaders/shaderRed.frag";

static const char* vShaderGreen = "shaders/shaderGreen.vert";
static const char* fShaderGreen = "shaders/shaderGreen.frag";

static const char* vShaderBlue = "shaders/shaderBlue.vert";
static const char* fShaderBlue = "shaders/shaderBlue.frag";

static const char* vShaderBrown = "shaders/shaderBrown.vert";
static const char* fShaderBrown = "shaders/shaderBrown.frag";

static const char* vShaderDarkGreen = "shaders/shaderDarkGreen.vert";
static const char* fShaderDarkGreen = "shaders/shaderDarkGreen.frag";
```

```
void CreateShaders()
{

    Shader *shader1 = new Shader(); //shader para usar índices: objetos: cubo y  pirámide
    shader1->CreateFromFiles(vShaderRed, fShaderRed);
    shaderList.push_back(*shader1);

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

    Shader* shader3 = new Shader(); //shader para usar índices: objetos: cubo y  pirámide
    shader3->CreateFromFiles(vShaderGreen, fShaderGreen);
    shaderList.push_back(*shader3);

    Shader* shader4 = new Shader(); //shader para usar índices: objetos: cubo y  pirámide
    shader4->CreateFromFiles(vShaderBlue, fShaderBlue);
    shaderList.push_back(*shader4);

    Shader* shader5 = new Shader(); //shader para usar índices: objetos: cubo y  pirámide
    shader5->CreateFromFiles(vShaderBrown, fShaderBrown);
    shaderList.push_back(*shader5);

    Shader* shader6 = new Shader(); //shader para usar índices: objetos: cubo y  pirámide
    shader6->CreateFromFiles(vShaderDarkGreen, fShaderDarkGreen);
    shaderList.push_back(*shader6);
}
```

```
// Cubo Rojo
shaderList[0].useShader();
uniformModel = shaderList[0].getModelLocation();
uniformProjection = shaderList[0].getProjectLocation();

model = glm::mat4(1.0);
model = glm::translate(model, glm::vec3(0.0f, 0.0f, -3.0f));
glUniformMatrix4fv(uniformModel, 1, GL_FALSE, glm::value_ptr(model));
glUniformMatrix4fv(uniformProjection, 1, GL_FALSE, glm::value_ptr(projection));
meshList[1]->RenderMesh();


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

model = glm::mat4(1.0);
model = glm::translate(model, glm::vec3(0.2f, 0.2f, -2.6f));
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();
```

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

model = glm::mat4(1.0);
model = glm::translate(model, glm::vec3(-0.2f, 0.2f, -2.6f));
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();


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

model = glm::mat4(1.0);
model = glm::translate(model, glm::vec3(0.0f, -0.3385f, -2.7f));
model = glm::scale(model, glm::vec3(0.3f, 0.3f, 0.5f));
glUniformMatrix4fv(uniformModel, 1, GL_FALSE, glm::value_ptr(model));
glUniformMatrix4fv(uniformProjection, 1, GL_FALSE, glm::value_ptr(projection));
meshList[1]->RenderMesh();
```

```
// Piramide Azul
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, -3.0f));
model = glm::scale(model, glm::vec3(1.1f, 1.1f, 1.1f));
glUniformMatrix4fv(uniformModel, 1, GL_FALSE, glm::value_ptr(model));
glUniformMatrix4fv(uniformProjection, 1, GL_FALSE, glm::value_ptr(projection));
meshList[0]->RenderMesh();


// Cubo Cafe izquierda
shaderList[4].useShader();
uniformModel = shaderList[4].getModelLocation();
uniformProjection = shaderList[4].getProjectLocation();

model = glm::mat4(1.0);
model = glm::translate(model, glm::vec3(-1.5f, -0.5f, -3.0f));
model = glm::scale(model, glm::vec3(0.3f, 0.3f, 0.5f));
glUniformMatrix4fv(uniformModel, 1, GL_FALSE, glm::value_ptr(model));
glUniformMatrix4fv(uniformProjection, 1, GL_FALSE, glm::value_ptr(projection));
meshList[1]->RenderMesh();
```

```
// Piramide Verde obscuro izquierda
shaderList[5].useShader();
uniformModel = shaderList[5].getModelLocation();
uniformProjection = shaderList[5].getProjectLocation();

model = glm::mat4(1.0);
model = glm::translate(model, glm::vec3(-1.5f, -0.4f, -2.95f));
model = glm::scale(model, glm::vec3(0.5f, 0.5f, 0.5f));
glUniformMatrix4fv(uniformModel, 1, GL_FALSE, glm::value_ptr(model));
glUniformMatrix4fv(uniformProjection, 1, GL_FALSE, glm::value_ptr(projection));
meshList[0]->RenderMesh();


// Cubo Cafe Derecha
shaderList[4].useShader();
uniformModel = shaderList[4].getModelLocation();
uniformProjection = shaderList[4].getProjectLocation();

model = glm::mat4(1.0);
model = glm::translate(model, glm::vec3(1.5f, -0.5f, -3.0f));
model = glm::scale(model, glm::vec3(0.3f, 0.3f, 0.5f));
glUniformMatrix4fv(uniformModel, 1, GL_FALSE, glm::value_ptr(model));
glUniformMatrix4fv(uniformProjection, 1, GL_FALSE, glm::value_ptr(projection));
meshList[1]->RenderMesh();
```

```
// Piramide Verde obscuro Derecho
shaderList[5].useShader();
uniformModel = shaderList[5].getModelLocation();
uniformProjection = shaderList[5].getProjectLocation();

model = glm::mat4(1.0);
model = glm::translate(model, glm::vec3(1.5f, -0.4f, -2.95f));
model = glm::scale(model, glm::vec3(0.5f, 0.5f, 0.5f));
glUniformMatrix4fv(uniformModel, 1, GL_FALSE, glm::value_ptr(model));
glUniformMatrix4fv(uniformProjection, 1, GL_FALSE, glm::value_ptr(projection));
meshList[0]->RenderMesh();
```

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.

Contratiempos

En esta práctica tuve algunos contratiempos relacionados con la explicación del profesor. Durante la clase, mientras él explicaba cómo realizar la práctica, yo no presté mucha atención porque estaba concentrado en terminar el ejercicio de clase. Esto me generó cierta inseguridad al comenzar, aunque después de analizar el código por un rato fui encontrando la manera de resolverlo.

En el primer ejercicio tuve algunas complicaciones con la cantidad de vértices, ya que olvidé por completo que era necesario aumentarla.

El segundo ejercicio resultó un poco más complejo que el primero. A primera vista parecía más laborioso que difícil, ya que no solo se trataba de cambiar cosas en el código, sino de agregar varias nuevas. Al principio eso me generó algo de miedo, porque para añadir cosas se necesita entender un poco más el funcionamiento del código. Sin embargo, una vez que comprendí qué debía agregar y en dónde, el ejercicio se volvió mucho más sencillo.

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

1. Bibliografía en formato APA

Conclusión

Esta práctica me resultó más útil que la primera, ya que me permitió reforzar conocimientos que antes no me habían quedado del todo claros y poco a poco me estoy familiarizando con la estructura y el uso de OpenGL. En cuanto a complejidad, considero que los ejercicios fueron interesantes, especialmente al tener que agregar nuevas partes al código en lugar de solo modificarlas, lo que me obligó a comprender mejor su funcionamiento.

En general, esta práctica fue un paso importante para ganar confianza con OpenGL y espero seguir aprendiendo y mejorando en las próximas.

Bibliografía

- Página principal. (26 de septiembre de 2018). *Wiki de OpenGL* . Recuperado el 29 de agosto de 2025 de http://www.khronos.org/opengl/wiki_opengl/index.php?title=Main_Page&oldid=14430 .