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 2

NOMBRE COMPLETO: Araiza Valdés Diego Antonio

Nº de Cuenta: 423032833

GRUPO DE LABORATORIO: 02

GRUPO DE TEORÍA: 06

SEMESTRE: 2026-1

FECHA DE ENTREGA LÍMITE: 31/08/2025

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

```
GLfloat vertices_letras[] = {
// X      Y      Z       R     G    B
-0.85f,  0.40f, 0.0f,   1.0f, 0.0f, 0.0f,
-0.85f, -0.40f, 0.0f,   1.0f, 0.0f, 0.0f,
-0.70f, -0.40f, 0.0f,   1.0f, 0.0f, 0.0f,

-0.85f,  0.40f, 0.0f,   1.0f, 0.0f, 0.0f,
-0.70f, -0.40f, 0.0f,   1.0f, 0.0f, 0.0f,
-0.70f,  0.40f, 0.0f,   1.0f, 0.0f, 0.0f,

-0.70f,  0.40f, 0.0f,   1.0f, 0.0f, 0.0f,
-0.70f,  0.25f, 0.0f,   1.0f, 0.0f, 0.0f,
-0.55f,  0.25f, 0.0f,   1.0f, 0.0f, 0.0f,

-0.70f,  0.40f, 0.0f,   1.0f, 0.0f, 0.0f,
-0.55f,  0.25f, 0.0f,   1.0f, 0.0f, 0.0f,
-0.55f,  0.40f, 0.0f,   1.0f, 0.0f, 0.0f,

-0.625f, 0.25f, 0.0f,   1.0f, 0.0f, 0.0f,
-0.55f,  0.10f, 0.0f,   1.0f, 0.0f, 0.0f,
-0.55f,  0.25f, 0.0f,   1.0f, 0.0f, 0.0f,

-0.55f,  0.40f, 0.0f,   1.0f, 0.0f, 0.0f,
-0.55f,  0.15f, 0.0f,   1.0f, 0.0f, 0.0f,
-0.40f,  0.15f, 0.0f,   1.0f, 0.0f, 0.0f,

-0.55f,  0.15f, 0.0f,   1.0f, 0.0f, 0.0f,
-0.55f, -0.15f, 0.0f,   1.0f, 0.0f, 0.0f,
-0.40f, -0.15f, 0.0f,   1.0f, 0.0f, 0.0f,

-0.55f,  0.15f, 0.0f,   1.0f, 0.0f, 0.0f,
-0.40f, -0.15f, 0.0f,   1.0f, 0.0f, 0.0f,
-0.40f,  0.15f, 0.0f,   1.0f, 0.0f, 0.0f,

-0.70f, -0.25f, 0.0f,   1.0f, 0.0f, 0.0f,
-0.70f, -0.40f, 0.0f,   1.0f, 0.0f, 0.0f,
-0.55f, -0.40f, 0.0f,   1.0f, 0.0f, 0.0f,

-0.70f, -0.25f, 0.0f,   1.0f, 0.0f, 0.0f,
-0.55f, -0.40f, 0.0f,   1.0f, 0.0f, 0.0f,
-0.55f, -0.25f, 0.0f,   1.0f, 0.0f, 0.0f,

-0.625f,-0.25f, 0.0f,   1.0f, 0.0f, 0.0f,
-0.55f, -0.25f, 0.0f,   1.0f, 0.0f, 0.0f,
-0.55f, -0.10f, 0.0f,   1.0f, 0.0f, 0.0f,

-0.55f, -0.15f, 0.0f,   1.0f, 0.0f, 0.0f,
-0.55f, -0.40f, 0.0f,   1.0f, 0.0f, 0.0f,
-0.40f, -0.15f, 0.0f,   1.0f, 0.0f, 0.0f,

//Segunda letra

-0.25f, -0.40f, 0.0f,   0.0f, 1.0f, 0.0f,
-0.10f, -0.40f, 0.0f,   0.0f, 1.0f, 0.0f,
-0.10f,  0.40f, 0.0f,   0.0f, 1.0f, 0.0f,

-0.10f,  0.25f, 0.0f,   0.0f, 1.0f, 0.0f,
-0.10f, -0.40f, 0.0f,   0.0f, 1.0f, 0.0f,
 0.00f,  0.25f, 0.0f,   0.0f, 1.0f, 0.0f,

 0.00f,  0.25f, 0.0f,   0.0f, 1.0f, 0.0f,
 0.10f, -0.40f, 0.0f,   0.0f, 1.0f, 0.0f,
 0.10f,  0.25f, 0.0f,   0.0f, 1.0f, 0.0f,

 0.10f,  0.40f, 0.0f,   0.0f, 1.0f, 0.0f,
 0.10f, -0.40f, 0.0f,   0.0f, 1.0f, 0.0f,
 0.25f, -0.40f, 0.0f,   0.0f, 1.0f, 0.0f,

-0.10f,  0.40f, 0.0f,   0.0f, 1.0f, 0.0f,
-0.10f,  0.25f, 0.0f,   0.0f, 1.0f, 0.0f,
 0.10f,  0.25f, 0.0f,   0.0f, 1.0f, 0.0f,

-0.10f,  0.40f, 0.0f,   0.0f, 1.0f, 0.0f,
 0.10f,  0.25f, 0.0f,   0.0f, 1.0f, 0.0f,
 0.10f,  0.40f, 0.0f,   0.0f, 1.0f, 0.0f,

-0.10f, -0.10f, 0.0f,   0.0f, 1.0f, 0.0f,
-0.10f, -0.20f, 0.0f,   0.0f, 1.0f, 0.0f,
 0.10f, -0.20f, 0.0f,   0.0f, 1.0f, 0.0f,

-0.10f, -0.10f, 0.0f,   0.0f, 1.0f, 0.0f,
 0.10f, -0.20f, 0.0f,   0.0f, 1.0f, 0.0f,
 0.10f, -0.10f, 0.0f,   0.0f, 1.0f, 0.0f,

//tercera letra
 0.35f,  0.40f, 0.0f,   0.0f, 0.0f, 1.0f,
 0.50f, -0.40f, 0.0f,   0.0f, 0.0f, 1.0f,
 0.50f,  0.40f, 0.0f,   0.0f, 0.0f, 1.0f,

 0.50f,  0.40f, 0.0f,   0.0f, 0.0f, 1.0f,
 0.50f, -0.25f, 0.0f,   0.0f, 0.0f, 1.0f,
 0.60f, -0.25f, 0.0f,   0.0f, 0.0f, 1.0f,

 0.60f, -0.25f, 0.0f,   0.0f, 0.0f, 1.0f,
 0.70f, -0.25f, 0.0f,   0.0f, 0.0f, 1.0f,
 0.70f,  0.40f, 0.0f,   0.0f, 0.0f, 1.0f,

 0.70f,  0.40f, 0.0f,   0.0f, 0.0f, 1.0f,
 0.70f, -0.40f, 0.0f,   0.0f, 0.0f, 1.0f,
 0.85f,  0.40f, 0.0f,   0.0f, 0.0f, 1.0f,

 0.50f, -0.25f, 0.0f,   0.0f, 0.0f, 1.0f,
 0.50f, -0.40f, 0.0f,   0.0f, 0.0f, 1.0f,
 0.70f, -0.40f, 0.0f,   0.0f, 0.0f, 1.0f,

 0.50f, -0.25f, 0.0f,   0.0f, 0.0f, 1.0f,
 0.70f, -0.40f, 0.0f,   0.0f, 0.0f, 1.0f,
 0.70f, -0.25f, 0.0f,   0.0f, 0.0f, 1.0f,

};

MeshColor *letras = new MeshColor();
letras->CreateMeshColor(vertices_letras,468);
meshColorList.push_back(letras);
```

Para las letras se copiaron los vértices de la práctica anterior y se les asigno un color RGB específico, D = Rojo, A = Verde, V = Azul. Además, como en la práctica anterior contamos con 78 conjunto de 3 vértices, y en esta práctica cada valor de los vectores cuenta por 1, se multiplicó los 78 por 6 dando un total de 468, valor que colocamos en el CreateMeshColor

```
//EJERCICIO 1
    shaderList[1].useShader();
    uniformModel = shaderList[1].getModelLocation();
    uniformProjection = shaderList[1].getProjectLocation();
    model = glm::mat4(1.0);
    model = glm::translate(model, glm::vec3(0.0f, 0.0f, -4.0f));
    glUniformMatrix4fv(uniformModel, 1, GL_FALSE, glm::value_ptr(model));
    glUniformMatrix4fv(uniformProjection, 1, GL_FALSE, glm::value_ptr(projection));
    meshColorList[0]->RenderMeshColor();
```

Finalmente, dentro del while seleccionamos el shader correspondiente a las letras y las dibujamos. En este caso solo se hizo uso de la función translate para mover las letras en el eje Z negativo.

1.2. 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

```
#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(0.478f, 0.255f, 0.067f, 1.0);
}
```

```
#version 330
in vec4 vColor;
out vec4 color;
void main()
{
    color= vColor;
}
```

Para crear los shaders se crearon 2 archivos por cada uno. En el ejemplo se muestran los del color café, nombrados shaderC.frag y shaderC.vert. Para los demás colores solo fue cuestión de repetir el proceso cambiando los valores del vector que recibe vColor y la 'C' del nombre de los archivos por R, V, A, o VO.

```
//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* vShaderR = "shaders/shaderR.vert";
static const char* fShaderR = "shaders/shaderR.frag";
static const char* vShaderV = "shaders/shaderV.vert";
static const char* fShaderV = "shaders/shaderV.frag";
static const char* vShaderA = "shaders/shaderA.vert";
static const char* fShaderA = "shaders/shaderA.frag";
static const char* vShaderC = "shaders/shaderC.vert";
static const char* fShaderC = "shaders/shaderC.frag";
static const char* vShaderVO = "shaders/shaderVO.vert";
static const char* fShaderVO = "shaders/shaderVO.frag";
```

Posteriormente, dentro del archivo inicializamos las variables que hacen referencia a los shaders previamente creados.

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

    Shader* shaderV = new Shader(); //shader para color verde
    shaderV->CreateFromFiles(vShaderV, fShaderV);
    shaderList.push_back(*shaderV);

    Shader* shaderA = new Shader(); //shader para color azul
    shaderA->CreateFromFiles(vShaderA, fShaderA);
    shaderList.push_back(*shaderA);

    Shader* shaderC = new Shader(); //shader para color café
    shaderC->CreateFromFiles(vShaderC, fShaderC);
    shaderList.push_back(*shaderC);

    Shader* shaderVO = new Shader(); //shader para color verde oscuro
    shaderVO->CreateFromFiles(vShaderVO, fShaderVO);
    shaderList.push_back(*shaderVO);
}
```

Por consiguiente, dentro de la función CreateShader creamos los nuevos shaders, los enlazamos a sus respectivos archivos y los agregamos a la shaderList

```
//EJERCICIO 2
//ARBOL IZQUIERDO
    //Triangulo verde
    shaderList[3].useShader();
    uniformModel = shaderList[3].getModelLocation();
    uniformProjection = shaderList[3].getProjectLocation();
    model = glm::mat4(1.0f);
    model = glm::translate(model, glm::vec3(-0.69f, -0.453f, -4.0f));
    model = glm::scale(model, glm::vec3(0.38f, 0.62f, 1.0f));
    glUniformMatrix4fv(uniformModel, 1, GL_FALSE, glm::value_ptr(model));
    glUniformMatrix4fv(uniformProjection, 1, GL_FALSE, glm::value_ptr(projection));
    meshList[0]->RenderMesh();

    //Cuadrado café
    shaderList[5].useShader();
    uniformModel = shaderList[5].getModelLocation();
    uniformProjection = shaderList[5].getProjectLocation();
    model = glm::mat4(1.0f);
    model = glm::translate(model, glm::vec3(-0.69f, -0.88f, -4.0f));
    model = glm::scale(model, glm::vec3(0.18f, 0.23f, 1.0f));
    glUniformMatrix4fv(uniformModel, 1, GL_FALSE, glm::value_ptr(model));
    glUniformMatrix4fv(uniformProjection, 1, GL_FALSE, glm::value_ptr(projection));
    meshList[1]->RenderMesh();
```

```
//ARBOL DERECHO
    shaderList[3].useShader();
    uniformModel = shaderList[3].getModelLocation();
    uniformProjection = shaderList[3].getProjectLocation();
    model = glm::mat4(1.0f);
    model = glm::translate(model, glm::vec3(0.69f, -0.453f, -4.0f));
    model = glm::scale(model, glm::vec3(0.38f, 0.62f, 1.0f));
    glUniformMatrix4fv(uniformModel, 1, GL_FALSE, glm::value_ptr(model));
    glUniformMatrix4fv(uniformProjection, 1, GL_FALSE, glm::value_ptr(projection));
    meshList[0]->RenderMesh();

    //Cuadrado café
    shaderList[5].useShader();
    uniformModel = shaderList[5].getModelLocation();
    uniformProjection = shaderList[5].getProjectLocation();
    model = glm::mat4(1.0f);
    model = glm::translate(model, glm::vec3(0.69f, -0.88f, -4.0f));
    model = glm::scale(model, glm::vec3(0.18f, 0.23f, 1.0f));
    glUniformMatrix4fv(uniformModel, 1, GL_FALSE, glm::value_ptr(model));
    glUniformMatrix4fv(uniformProjection, 1, GL_FALSE, glm::value_ptr(projection));
    meshList[1]->RenderMesh();

//CASA
    //Tejado triangulo azul
    shaderList[4].useShader();
    uniformModel = shaderList[4].getModelLocation();
    uniformProjection = shaderList[4].getProjectLocation();
    model = glm::mat4(1.0f);
    model = glm::translate(model, glm::vec3(0.0f, 0.627f, -4.0f));
    model = glm::scale(model, glm::vec3(1.05f, 0.73f, 1.0f));
    glUniformMatrix4fv(uniformModel, 1, GL_FALSE, glm::value_ptr(model));
    glUniformMatrix4fv(uniformProjection, 1, GL_FALSE, glm::value_ptr(projection));
    meshList[0]->RenderMesh();

    //Cuerpo de la casa cuadrado rojo
    shaderList[2].useShader();
    uniformModel = shaderList[2].getModelLocation();
    uniformProjection = shaderList[2].getProjectLocation();
    model = glm::mat4(1.0f);
    model = glm::translate(model, glm::vec3(0.0f, -0.365f, -4.0f));
    model = glm::scale(model, glm::vec3(0.9f, 1.26f, 1.0f));
    glUniformMatrix4fv(uniformModel, 1, GL_FALSE, glm::value_ptr(model));
    glUniformMatrix4fv(uniformProjection, 1, GL_FALSE, glm::value_ptr(projection));
    meshList[1]->RenderMesh();

    //Puerta cuadrado verde
    shaderList[3].useShader();
    uniformModel = shaderList[3].getModelLocation();
    uniformProjection = shaderList[3].getProjectLocation();
    model = glm::mat4(1.0f);
    model = glm::translate(model, glm::vec3(0.0f, -0.793f, -3.9f));
    model = glm::scale(model, glm::vec3(0.3f, 0.4f, 1.0f));
    glUniformMatrix4fv(uniformModel, 1, GL_FALSE, glm::value_ptr(model));
    glUniformMatrix4fv(uniformProjection, 1, GL_FALSE, glm::value_ptr(projection));
    meshList[1]->RenderMesh();
```

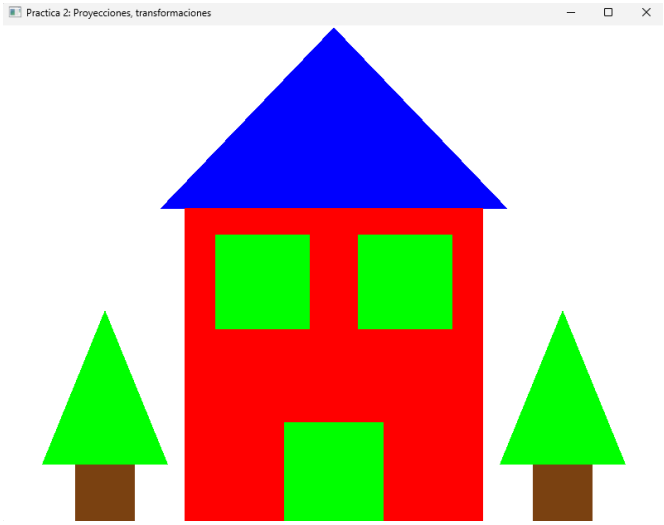
```
//Ventana izquierda cuadrado verde
shaderList[3].useShader();
uniformModel = shaderList[3].getModelLocation();
uniformProjection = shaderList[3].getProjectLocation();
model = glm::mat4(1.0f);
model = glm::translate(model, glm::vec3(-0.215f, -0.03f, -3.9f));
model = glm::scale(model, glm::vec3(0.285f, 0.38f, 1.0f));
glUniformMatrix4fv(uniformModel, 1, GL_FALSE, glm::value_ptr(model));
glUniformMatrix4fv(uniformProjection, 1, GL_FALSE, glm::value_ptr(projection));
meshList[1]->RenderMesh();

//Ventana derecha cuadrado verde
shaderList[3].useShader();
uniformModel = shaderList[3].getModelLocation();
uniformProjection = shaderList[3].getProjectLocation();
model = glm::mat4(1.0f);
model = glm::translate(model, glm::vec3(0.215f, -0.03f, -3.9f));
model = glm::scale(model, glm::vec3(0.285f, 0.38f, 1.0f));
glUniformMatrix4fv(uniformModel, 1, GL_FALSE, glm::value_ptr(model));
glUniformMatrix4fv(uniformProjection, 1, GL_FALSE, glm::value_ptr(projection));
meshList[1]->RenderMesh();

glUseProgram(0);
mainWindow.swapBuffers();
```

Finalmente, de igual forma a como hicimos en el ejercicio del laboratorio, dibujamos las formas, sin embargo, ahora seleccionamos su shader correspondiente para cada una de ellas. De manera que, para cada figura seleccionamos su respectivo índice del shader, inicializamos su matriz y posteriormente trasladaremos y escalaremos cada figura, por consiguiente, enviamos la matriz del modelo y la matriz de proyección al shader, para finalmente dibujarla la forma usando el mismo.

1.1. Evidencias:

1.1.1. Ejercicio 1

1.1.2. Ejercicio 2



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
   R: No hubo ningún problema a la hora de realizar los ejercicios.

3. Conclusión:
   1. Los ejercicios del reporte: Complejidad, Explicación.
      Respecto a los ejercicios realizados, creo que fueron bastante adecuados al nivel de lo visto en el laboratorio y los ejercicios realizados previamente, ya que, en realidad, las actividades solo fueron realizar unos pequeños cambios, como crear archivos (basándonos en los que ya teníamos), cambiar, cambiar triángulos y cuadrados por pirámides y cubos, escalar y trasladar un poco de ellos y poco más.
   2. Comentarios generales: Faltó explicar a detalle, ir más lento en alguna explicación, otros comentarios y sugerencias para mejorar desarrollo de la práctica.
      Al inicio no entendí muy bien los archivos .frag y .vert por lo que me perdí un poco, sin embargo, al final de la clase al explicar la práctica los terminé de entender, por lo que no hubo mayor problema.
   3. Conclusión

      Personalmente creo que se cumplieron los objetivos de la práctica. Con los ejercicios tuve un acercamiento a los dos tipos de proyecciones, aplicar transformaciones y delimité mi espacio de trabajo. Aprendí la principal diferencia entre la proyección ortogonal y en perspectiva y en que casos usar cada una. Aprendí a declarar figuras en 3D y 2D, a usar índices a la hora de delcararlas y realizar transformaciones sobre ellas, como escalado, rotación y traslación. Además, aprendí a crear shaders básicos y a trabajar

con ellos para colorear formas, haciendo mención de la función clamp, que permite hacer degradados restringiendo un valor en un rango específico.

Bibliografía en formato APA

GLFW. (2024, 23 de febrero). *GLFW: Introduction*. Recuperado de https://www.glfw.org/docs/latest/

GLM. (s. f.). *OpenGL Mathematics (GLM) documentation – Version 0.9.9*. Recuperado de https://glm.g-truc.net/0.9.9/index.html