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: Hernández Domínguez Luis Carlos

Nº de Cuenta: 320182668

GRUPO DE LABORATORIO: 03

GRUPO DE TEORÍA: 04

SEMESTRE 2026-1

FECHA DE ENTREGA LÍMITE: 31/08/2025

CALIFICACIÓN: _____

REPORTE DE PRÁCTICA:

1. Ejercicios

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

Bloques de código generado.

```
GLfloat vertices_letraH[] = {
    //X      Y      Z      R      G      B
    -0.9f,   0.8f,   0.0f,   1.0f,   0.0f,   0.0f,
    -0.8f,   0.8f,   0.0f,   1.0f,   0.0f,   0.0f,
    -0.9f,  -0.3f,   0.0f,   1.0f,   0.0f,   0.0f,
    -0.9f,  -0.3f,   0.0f,   1.0f,   0.0f,   0.0f,
    -0.8f,   0.8f,   0.0f,   1.0f,   0.0f,   0.0f,
    -0.8f,  -0.3f,   0.0f,   1.0f,   0.0f,   0.0f,
    -0.8f,   0.3f,   0.0f,   1.0f,   0.0f,   0.0f,
    -0.8f,   0.2f,   0.0f,   1.0f,   0.0f,   0.0f,
    -0.5f,   0.2f,   0.0f,   1.0f,   0.0f,   0.0f,
    -0.8f,   0.3f,   0.0f,   1.0f,   0.0f,   0.0f,
    -0.5f,   0.3f,   0.0f,   1.0f,   0.0f,   0.0f,
    -0.5f,   0.2f,   0.0f,   1.0f,   0.0f,   0.0f,
    -0.5f,   0.8f,   0.0f,   1.0f,   0.0f,   0.0f,
    -0.5f,  -0.3f,   0.0f,   1.0f,   0.0f,   0.0f,
    -0.4f,   0.8f,   0.0f,   1.0f,   0.0f,   0.0f,
    -0.4f,   0.8f,   0.0f,   1.0f,   0.0f,   0.0f,
    -0.5f,  -0.3f,   0.0f,   1.0f,   0.0f,   0.0f,
    -0.4f,  -0.3f,   0.0f,   1.0f,   0.0f,   0.0f,

};

MeshColor* letraH = new MeshColor();
letraH->CreateMeshColor(vertices_letraH, 108);
meshColorList.push_back(letraH);
```

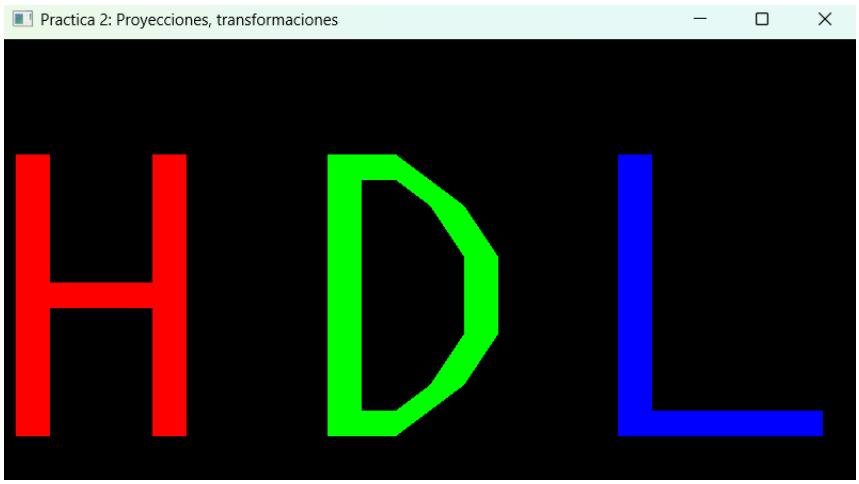
```
GLfloat vertices_letraL[] = {
    //X      Y      Z      R      G      B
    0.3f,   0.8f,   0.0f,   0.0f,   0.0f,   1.0f,
    0.4f,   0.8f,   0.0f,   0.0f,   0.0f,   1.0f,
    0.3f,  -0.3f,   0.0f,   0.0f,   0.0f,   1.0f,
    0.3f,  -0.3f,   0.0f,   0.0f,   0.0f,   1.0f,
    0.4f,   0.8f,   0.0f,   0.0f,   0.0f,   1.0f,
    0.4f,  -0.3f,   0.0f,   0.0f,   0.0f,   1.0f,
    0.4f,  -0.2f,   0.0f,   0.0f,   0.0f,   1.0f,
    0.4f,  -0.3f,   0.0f,   0.0f,   0.0f,   1.0f,
    0.9f,  -0.3f,   0.0f,   0.0f,   0.0f,   1.0f,
    0.4f,  -0.2f,   0.0f,   0.0f,   0.0f,   1.0f,
    0.9f,  -0.2f,   0.0f,   0.0f,   0.0f,   1.0f,
    0.9f,  -0.3f,   0.0f,   0.0f,   0.0f,   1.0f,
```

```
//Letra H
model = glm::mat4(1.0);
model = glm::translate(model, glm::vec3(-0.25f, 0.0f, -4.0f));
model = glm::scale(model, glm::vec3(0.8f, 0.8f, 0.85f));
glUniformMatrix4fv(uniformModel, 1, GL_FALSE, glm::value_ptr(model));//FALSE ES
glUniformMatrix4fv(uniformProjection, 1, GL_FALSE, glm::value_ptr(projection));
meshColorList[6]->RenderMeshColor();


//Letra D
model = glm::mat4(1.0);
model = glm::translate(model, glm::vec3(0.0f, 0.0f, -4.0f));
model = glm::scale(model, glm::vec3(0.8f, 0.8f, 0.85f));
glUniformMatrix4fv(uniformModel, 1, GL_FALSE, glm::value_ptr(model));//FALSE ES
glUniformMatrix4fv(uniformProjection, 1, GL_FALSE, glm::value_ptr(projection));
meshColorList[7]->RenderMeshColor();

//Letra L
model = glm::mat4(1.0);
model = glm::translate(model, glm::vec3(0.2f, 0.0f, -4.0f));
model = glm::scale(model, glm::vec3(0.8f, 0.8f, 0.85f));
glUniformMatrix4fv(uniformModel, 1, GL_FALSE, glm::value_ptr(model));//FALSE ES
glUniformMatrix4fv(uniformProjection, 1, GL_FALSE, glm::value_ptr(projection));
meshColorList[8]->RenderMeshColor();
```

Ejecución



Para este primer ejercicio se definieron nuevos vértices en la función CrearLetrasyFiguras(), posteriormente se posicionaron y escalaron usando los comandos translate y scale respectivamente

Ejercicio 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.

Bloques de código generado.

```
28      static const char* vShader2 = "shaders/shader2.vert";
29      static const char* vShader3 = "shaders/shader3.vert";
30      static const char* vShader4 = "shaders/shader4.vert";
31      static const char* vShader5 = "shaders/shader5.vert";
32      static const char* vShader6 = "shaders/shader6.vert";
```

```
    vColor=vec4(1.0f,0.0f,0.0f,1.0f);
    //vColor=vec4(clamp(pos,0.0f,1.0f),1.0f);
```

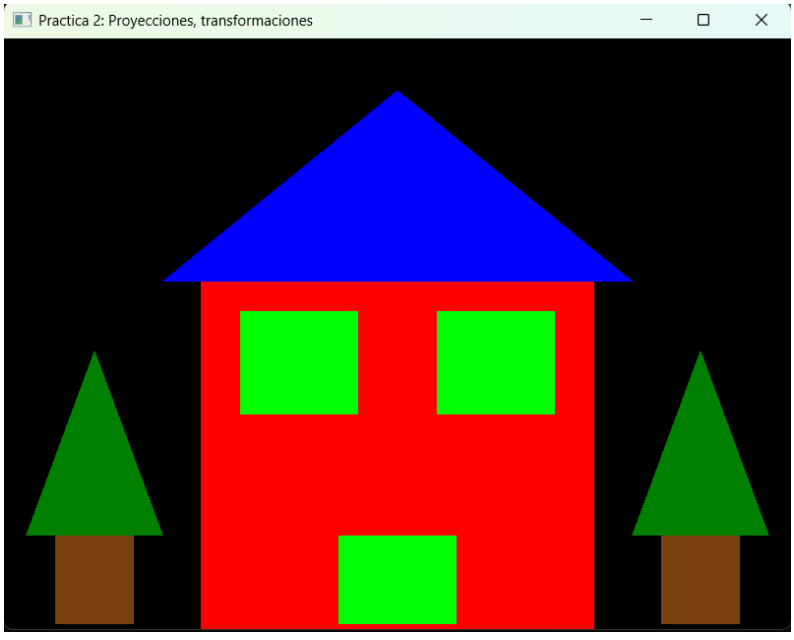
```
313         Shader  *shader3 = new Shader();
314         shader3->CreateFromFiles(vShader2, fShader);
315         shaderList.push_back(*shader3);
316
317         Shader *shader4 = new Shader();
318         shader4->CreateFromFiles(vShader3, fShader);
319         shaderList.push_back(*shader4);
320
321         Shader *shader5 = new Shader();
322         shader5->CreateFromFiles(vShader4, fShader);
323         shaderList.push_back(*shader5);
324
325         Shader* shader6 = new Shader();
326         shader6->CreateFromFiles(vShader5, fShader);
327         shaderList.push_back(*shader6);
328
329         Shader* shader7 = new Shader();
330         shader7->CreateFromFiles(vShader6, fShader);
331         shaderList.push_back(*shader7);
```

```
462         //Cubos cafes
463
464         shaderList[6].useShader();
465         uniformModel = shaderList[6].getModelLocation();
466         uniformProjection = shaderList[6].getProjectLocation();
467         angulo += 0.01;
468         //Inicializar matriz de dimensión 4x4 que servirá como matriz de modelo para al
469         model = glm::mat4(1.0);
470         model = glm::translate(model, glm::vec3(-0.77f, -0.835f, -2.0f));
471         model = glm::scale(model, glm::vec3(0.2f, 0.3f, 0.3f));
472         glUniformMatrix4fv(uniformModel, 1, GL_FALSE, glm::value_ptr(model));//FALSE ES
473         glUniformMatrix4fv(uniformProjection, 1, GL_FALSE, glm::value_ptr(projection));
474         meshList[1]->RenderMesh();
475
476         shaderList[6].useShader();
477         uniformModel = shaderList[6].getModelLocation();
478         uniformProjection = shaderList[6].getProjectLocation();
479         angulo += 0.01;
480         //Inicializar matriz de dimensión 4x4 que servirá como matriz de modelo para al
481         model = glm::mat4(1.0);
482         model = glm::translate(model, glm::vec3(0.77f, -0.835f, -2.0f));
483         model = glm::scale(model, glm::vec3(0.2f, 0.3f, 0.3f));
484         glUniformMatrix4fv(uniformModel, 1, GL_FALSE, glm::value_ptr(model));//FALSE ES
485         glUniformMatrix4fv(uniformProjection, 1, GL_FALSE, glm::value_ptr(projection));
486         meshList[1]->RenderMesh();
```

```
488         //piramides verdes
489
490         shaderList[5].useShader();
491         uniformModel = shaderList[5].getModelLocation();
492         uniformProjection = shaderList[5].getProjectLocation();
493         angulo += 0.01;
494         //Inicializar matriz de dimensión 4x4 que servirá como matriz de modelo para alm
495         model = glm::mat4(1.0);
496         model = glm::translate(model, glm::vec3(-0.77f, -0.37f, -2.0f));
497         model = glm::scale(model, glm::vec3(0.35f, 0.63f, 0.3f));
498         glUniformMatrix4fv(uniformModel, 1, GL_FALSE, glm::value_ptr(model));//FALSE ES
499         glUniformMatrix4fv(uniformProjection, 1, GL_FALSE, glm::value_ptr(projection));
500         meshList[0]->RenderMesh();
501
502         shaderList[5].useShader();
503         uniformModel = shaderList[5].getModelLocation();
504         uniformProjection = shaderList[5].getProjectLocation();
505         angulo += 0.01;
506         //Inicializar matriz de dimensión 4x4 que servirá como matriz de modelo para alm
507         model = glm::mat4(1.0);
508         model = glm::translate(model, glm::vec3(0.77f, -0.37f, -2.0f));
509         model = glm::scale(model, glm::vec3(0.35f, 0.63f, 0.3f));
510         glUniformMatrix4fv(uniformModel, 1, GL_FALSE, glm::value_ptr(model));//FALSE ES
511         glUniformMatrix4fv(uniformProjection, 1, GL_FALSE, glm::value_ptr(projection));
512         meshList[0]->RenderMesh();
```

```
397         //Cubo rojo
398         shaderList[2].useShader();
399         uniformModel = shaderList[2].getModelLocation();
400         uniformProjection = shaderList[2].getProjectLocation();
401         angulo += 0.01;
402         //Inicializar matriz de dimensión 4x4 que servirá como matriz de modelo para alma
403         model = glm::mat4(1.0);
404         model = glm::translate(model, glm::vec3(0.0f, -0.49f, -3.0f));
405         model = glm::scale(model, glm::vec3(1.0f, 1.5f, 1.0f));
406         glUniformMatrix4fv(uniformModel, 1, GL_FALSE, glm::value_ptr(model));//FALSE ES
407         glUniformMatrix4fv(uniformProjection, 1, GL_FALSE, glm::value_ptr(projection));
408         meshList[1]->RenderMesh();
409
410         //Cubos verdes
411         shaderList[3].useShader();
412         uniformModel = shaderList[3].getModelLocation();
413         uniformProjection = shaderList[3].getProjectLocation();
414         angulo += 0.01;
415         //Inicializar matriz de dimensión 4x4 que servirá como matriz de modelo para alma
416         model = glm::mat4(1.0);
417         model = glm::translate(model, glm::vec3(0.0f, -0.835f, -2.0f));
418         model = glm::scale(model, glm::vec3(0.3f, 0.3f, 0.3f));
419         glUniformMatrix4fv(uniformModel, 1, GL_FALSE, glm::value_ptr(model));//FALSE ES
420         glUniformMatrix4fv(uniformProjection, 1, GL_FALSE, glm::value_ptr(projection));
421         meshList[1]->RenderMesh();
```

Ejecución.



Para realizar este ejercicio fue necesario crear nuevos shaders, en forma de los archivos .vert. Y de igual forma, añadiéndolos a la lista de los shaders, tras este proceso solo fue cuestión de dibujar los cubos y pirámides a las distancias y tamaños correctos para recrear la figura.

2. Problemas presentados.

No se presentó ningún problema a la hora de ejecutar código.

3.- Conclusión:

a. El primer ejercicio fue relativamente sencillo, debido a que, contaba con los vértices definidos en la práctica anterior. En el caso del segundo ejercicio, su solución requería de una inspección un poco más detallada del código, sin embargo, una vez identificada la parte a modificar y agregar no fue complicado.
b. Los conceptos y explicaciones dadas en clase fueron correctas y suficientes para la resolución de esta práctica.
c. En conclusión, esta práctica me pareció tener un nivel de dificultad adecuado y hace un buen trabajo al introducir los shaders y el dibujo de figuras geométricas como el cubo o la pirámide.