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: Pablo Arturo Gonzalez Cuellar

Nº de Cuenta: 319241013

GRUPO DE LABORATORIO: 02

GRUPO DE TEORÍA: 04

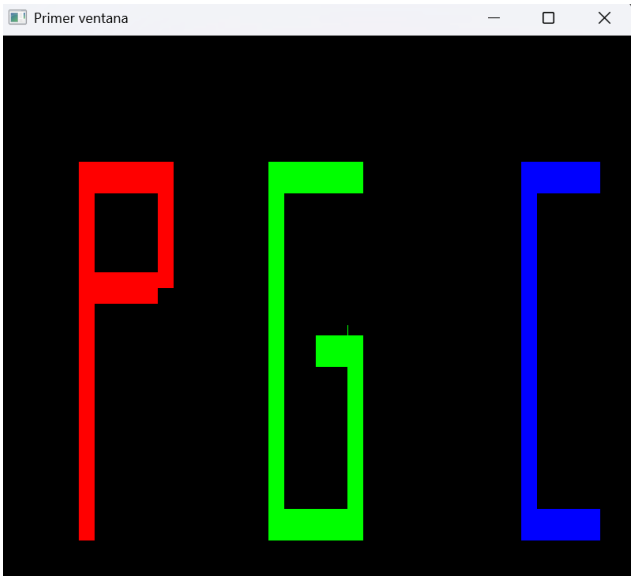
SEMESTRE 2026-1

FECHA DE ENTREGA LÍMITE: 31/08/2026

CALIFICACIÓN: _____

REPORTE DE PRÁCTICA:

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



En este ejercicio se pidió mostrar en la ventana mis iniciales (P, G y C), todas creadas únicamente a partir de triángulos, y además con colores distintos, por lo tanto elegí el rojo para la P, verde para la G y azul para la C.

Para lograr esto, se creó una nueva función llamada `crearIniciales()`, en donde se definió un arreglo de vértices con las coordenadas que forman los triángulos de cada letra. En el caso de la P fueron necesarios 4 rectángulos (24 vértices), para la G se requirieron 5 bloques (30 vértices), y para la C se construyó con 3 rectángulos (18 vértices). En total fueron 72 vértices

```
// LETRA P

-0.75f,  0.60f, 0.0f,   -0.70f,  0.60f, 0.0f,   -0.70f, -0.60f, 0.0f,
-0.75f,  0.60f, 0.0f,   -0.70f, -0.60f, 0.0f,   -0.75f, -0.60f, 0.0f,

-0.75f,  0.60f, 0.0f,   -0.45f,  0.60f, 0.0f,   -0.45f,  0.50f, 0.0f,
-0.75f,  0.60f, 0.0f,   -0.45f,  0.50f, 0.0f,   -0.75f,  0.50f, 0.0f,

-0.50f,  0.60f, 0.0f,   -0.45f,  0.60f, 0.0f,   -0.45f,  0.20f, 0.0f,
-0.50f,  0.60f, 0.0f,   -0.45f,  0.20f, 0.0f,   -0.50f,  0.20f, 0.0f,

-0.75f,  0.25f, 0.0f,   -0.50f,  0.25f, 0.0f,   -0.50f,  0.15f, 0.0f,
-0.75f,  0.25f, 0.0f,   -0.50f,  0.15f, 0.0f,   -0.75f,  0.15f, 0.0f,
```

```
// LETRA G

-0.15f,  0.60f, 0.0f,   -0.10f,  0.60f, 0.0f,   -0.10f, -0.60f, 0.0f,
-0.15f,  0.60f, 0.0f,   -0.10f, -0.60f, 0.0f,   -0.15f, -0.60f, 0.0f,

-0.15f,  0.60f, 0.0f,    0.15f,  0.60f, 0.0f,    0.15f,  0.50f, 0.0f,
-0.15f,  0.60f, 0.0f,    0.15f,  0.50f, 0.0f,   -0.15f,  0.50f, 0.0f,

-0.15f, -0.50f, 0.0f,    0.15f, -0.50f, 0.0f,    0.15f, -0.60f, 0.0f,
-0.15f, -0.50f, 0.0f,    0.15f, -0.60f, 0.0f,   -0.15f, -0.60f, 0.0f,

0.00f,  -0.05f, 0.0f,    0.15f, -0.05f, 0.0f,    0.15f,  0.05f, 0.0f,
0.00f,  -0.05f, 0.0f,    0.15f,  0.05f, 0.0f,    0.00f,  0.05f, 0.0f,

0.10f,   0.04f, 0.0f,    0.15f,  0.02f, 0.0f,    0.15f, -0.60f, 0.0f,
0.10f,   0.10f, 0.0f,    0.15f, -0.60f, 0.0f,    0.10f, -0.60f, 0.0f,
```

```
// LETRA C

0.65f,  0.60f, 0.0f,     0.70f,  0.60f, 0.0f,    0.70f, -0.60f, 0.0f,
0.65f,  0.60f, 0.0f,     0.70f, -0.60f, 0.0f,    0.65f, -0.60f, 0.0f,

0.65f,  0.60f, 0.0f,     0.90f,  0.60f, 0.0f,    0.90f,  0.50f, 0.0f,
0.65f,  0.60f, 0.0f,     0.90f,  0.50f, 0.0f,    0.65f,  0.50f, 0.0f,

0.65f, -0.50f, 0.0f,     0.90f, -0.50f, 0.0f,    0.90f, -0.60f, 0.0f,
0.65f, -0.50f, 0.0f,     0.90f, -0.60f, 0.0f,    0.65f, -0.60f, 0.0f,
```

Un cambio importante fue el uso de `glUniform3f()` para asignar el color antes de dibujar cada letra. Esto permitió que, en el mismo programa, las letras se mostraran con distintos colores en lugar de compartir un mismo shader de color como en prácticas anteriores.

```
// Letra P
glUniform3f(colorLoc, 1.0f, 0.0f, 0.0f); // rojo
glDrawArrays(GL_TRIANGLES, 0, 24);

// Letra G
glUniform3f(colorLoc, 0.0f, 1.0f, 0.0f); // verde
glDrawArrays(GL_TRIANGLES, 24, 30);

// Letra C
glUniform3f(colorLoc, 0.0f, 0.0f, 1.0f); // azul
glDrawArrays(GL_TRIANGLES, 54, 18);
```

También, para esta práctica decidí utilizar código GLSL del vertex shader y fragment shader dentro del archivo C++. Esto me evitó manejar lectura de archivos externos y rutas en tiempo de ejecución, y me permitió probar rápido los cambios de color/forma.

```
//Codigo GLSL
//Vertex Shader
static const char* vShader = "                    \n\
#version 330                                      \n\
layout (location =0) in vec3 pos;                 \n\
void main()                                       \n\
{                                                 \n\
gl_Position=vec4(pos.x,pos.y,pos.z,1.0f);         \n\
}";

//Fragment Shader
static const char* fShader = "                    \n\
#version 330                                      \n\
out vec4 color;                                   \n\
uniform vec3 uColor;                              \n\
void main()                                       \n\
{                                                 \n\
    color = vec4(uColor,1.0f);                    \n\
}";
```

PROBLEMAS SURGIDOS:

Uno de los principales problemas fue la construcción de la letra G, ya que las coordenadas de algunos triángulos no estaban alineadas correctamente, por lo tanto, para resolver esto ajusté una a una las posiciones de los vértices, probando diferentes valores hasta que se logró una forma más cercana a la letra deseada.

También, como se mencionó, para que cada inicial mostrara un color diferente, tuve que introducir el uso de uniform en el fragment shader y obtener la ubicación del uniform en el main, esto para asignar un color antes de cada dibujo.

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**

Para este segundo ejercicio, el objetivo fue rehacer el dibujo de la casa, pero ahora instanciando una pirámide y varios cubos en lugar de los triángulos del ejemplo original. Además, en vez de usar un único shader con “color clamp”, creé cinco shaders distintos, uno por color: rojo, verde, azul, café y verde oscuro.

Primero modelé las dos figuras necesarias. La pirámide sirve para el techo y los cubos para el cuerpo de la casa, las ventanas, la puerta y los troncos.

```
// Creación del Cubo
void CrearCubo()
{
    unsigned int cubo_indices[] = {

        0,1,2, 2,3,0,
        1,5,6, 6,2,1,
        7,6,5, 5,4,7,
        4,0,3, 3,7,4,
        4,5,1, 1,0,4,
        3,2,6, 6,7,3
    };

    GLfloat cubo_vertices[] = {

        -0.5f, -0.5f,  0.5f,
         0.5f, -0.5f,  0.5f,
         0.5f,  0.5f,  0.5f,
        -0.5f,  0.5f,  0.5f,
        -0.5f, -0.5f, -0.5f,
         0.5f, -0.5f, -0.5f,
         0.5f,  0.5f, -0.5f,
        -0.5f,  0.5f, -0.5f
    };
```

```cpp
//Creacion de Pirámide
void CreaPiramide()
{
    unsigned int indices[] = {
        0,1,2,
        1,3,2,
        3,0,2,
        1,0,3
    };
    GLfloat vertices[] = {
        -0.5f, -0.5f,  0.0f, // 0  base
         0.5f, -0.5f,  0.0f, // 1
         0.0f,  0.5f, -0.25f,// 2
         0.0f, -0.5f, -0.5f  // 3
    };
    Mesh* obj = new Mesh();
    obj->CreateMesh(vertices, indices, 12, 12);
    meshList.push_back(obj);
}
```

Luego preparé un shader por color. Cada shader se carga desde archivos `.vert/.frag` y se guarda en `shaderList` en un orden fijo para llamar al que necesito antes de dibujar cada parte, el orden fue el siguiente: verde=0, rojo=1, azul=2, café=3, verde oscuro=4.

```cpp
// verde
Shader* s0 = new Shader(); s0->CreateFromFiles(vgreenShader, fgreenShader); shaderList.push_back(*s0);
// rojo
Shader* s1 = new Shader(); s1->CreateFromFiles(vredShader, fredShader);   shaderList.push_back(*s1);
// azul
Shader* s2 = new Shader(); s2->CreateFromFiles(vblueShader, fblueShader);  shaderList.push_back(*s2);
// café
Shader* s3 = new Shader(); s3->CreateFromFiles(vbrownShader, fbrownShader); shaderList.push_back(*s3);
// verde oscuro
Shader* s4 = new Shader(); s4->CreateFromFiles(vdarkgreenShader, fdarkgreenShader); shaderList.push_back(*s4);
```

También, para que el escenario se viera limpio, modelé una línea de “suelo” y a partir de ahí coloqué todo “apoyado”. El cubo rojo de la casa se escaló y trasladó para que su base toque el suelo; el techo se movió justo encima del cubo. Todo esto con `translate` y `scale`.

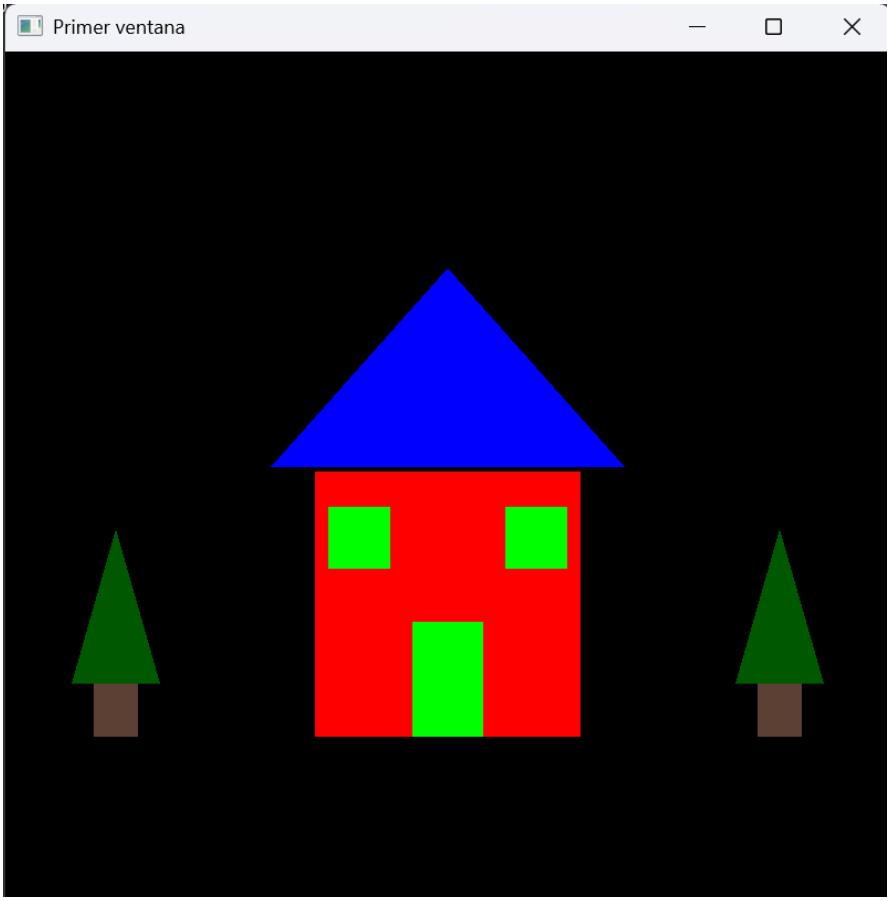
```cpp
glm::mat4 projection = glm::ortho(-1.0f, 1.0f, -1.0f, 1.0f, 0.1f, 100.0f);
glm::mat4 model(1.0f);

const float Z_HOUSE = -5.0f;          // profundidad
const float GROUND_Y = -0.55f;
```

Por último, una cosa sencilla pero importante fue el ajuste de las ventanas y puerta, ya que comparten la cara frontal del cubo. Debido a esto y para que no se traslapen por estar en el mismo plano, las dibujé un poquito más cerca de la cámara y muy delgadas:

```
//Profundidad de ventanas y puerta
const float zFront = Z_HOUSE + 0.5f * BODY_S.z;
const float zOverlay = zFront + 0.01f;
```

Capturas de ejecución:



PROBLEMAS SURGIDOS:

Al principio el techo y los árboles parecían volar debido a que estaba trasladando por el centro de los objetos sin considerar dónde queda su base real. Lo resolví definiendo un valor fijo de suelo y calculando el centro en Y de cada pieza para que su base coincidiera exactamente con esa línea.

También, la práctica pedía no usar un solo shader con color, sino cinco shaders diferentes. Por lo tante creé un par vertex/fragment por color y, justo antes de dibujar cada parte, activo el shader correspondiente con `useShader()`. Lo cual es un cambio pequeño, pero se logra el objetivo de la práctica sobre que cada color debe ser un programa distinto y así ya no dependemos de un uniform para el color.

CONCLUSIÓN

En esta práctica me familiaricé mejor con el flujo de OpenGL construyendo un escenario con figuras básicas y colorearlas con shaders separados. Más que aprender comandos nuevos, entendí la idea general de trabajar con un conjunto de vértices que se pueden reutilizar y de posicionarlos con transformaciones simples hasta formar una composición. Usar un shader por color me ayudó a ver de manera muy directa cómo el color se decide antes de dibujar cada parte y el por qué conviene organizar el proyecto en archivos de shader desde el inicio. También, me quedó más claro cómo pequeños detalles, como mover ligeramente un objeto hacia adelante para evitar la sobreposición, o definir un “suelo” para que las piezas no floten, marcan la diferencia en el resultado final. En conjunto, fue una buena introducción práctica al uso de shaders y a la composición de escenas 3D.

REFERENCIAS

- LearnOpenGL. Shaders. (n.d.). https://learnopengl.com/Getting-started/Shaders