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 01

NOMBRE COMPLETO: CARBALLO RAMÍREZ HANNY

Nº de Cuenta: 319236761

GRUPO DE LABORATORIO: 03

GRUPO DE TEORÍA: 03

SEMESTRE 2026-2

FECHA DE ENTREGA LÍMITE: DOMINGO 22 DE FEBRERO 2026

CALIFICACIÓN: _____

PRÁCTICA 01

1. ACTIVIDADES REALIZADAS. UNA DESCRIPCIÓN DE LOS EJERCICIOS Y CAPTURAS DE PANTALLA DE BLOQUES DE CÓDIGO GENERADOS Y DE EJECUCIÓN DEL PROGRAMA

1.- Ventana cambia el color de fondo de forma random tomando rango de colores RGB y con una periodicidad de 2 segundos. (Verificar que al ejecutar el programa varias veces el orden de los colores si lo vean aleatorio y no siempre los mismos)

En este ejercicio programé el fondo para que cambiara de color automáticamente cada cierto tiempo. Usé rand() para generar valores entre 0.0 y 1.0 en RGB, y esos valores se los paso a glClearColor() antes de limpiar la pantalla.

Además, incluí un sleep_for para que el color no cambiara demasiado rápido. De esta forma, cada vez que se actualiza el buffer aparece un color diferente, como si las letras estuvieran flotando en un fondo que cambia constantemente. Esto lo hice dentro del ciclo principal del programa.

```
float r, g, b;

while (!glfwWindowShouldClose(mainWindow))
{
    glfwPollEvents();

    // Fondo aleatorio
    r = (rand() % 100) / 100.0f;
    g = (rand() % 100) / 100.0f;
    b = (rand() % 100) / 100.0f;

    glClearColor(r, g, b, 1.0f);
    glClear(GL_COLOR_BUFFER_BIT);

    glUseProgram(shader);
    glBindVertexArray(VAO);

    glDrawArrays(GL_TRIANGLES, 0, 54); // 54 vértices totales
```

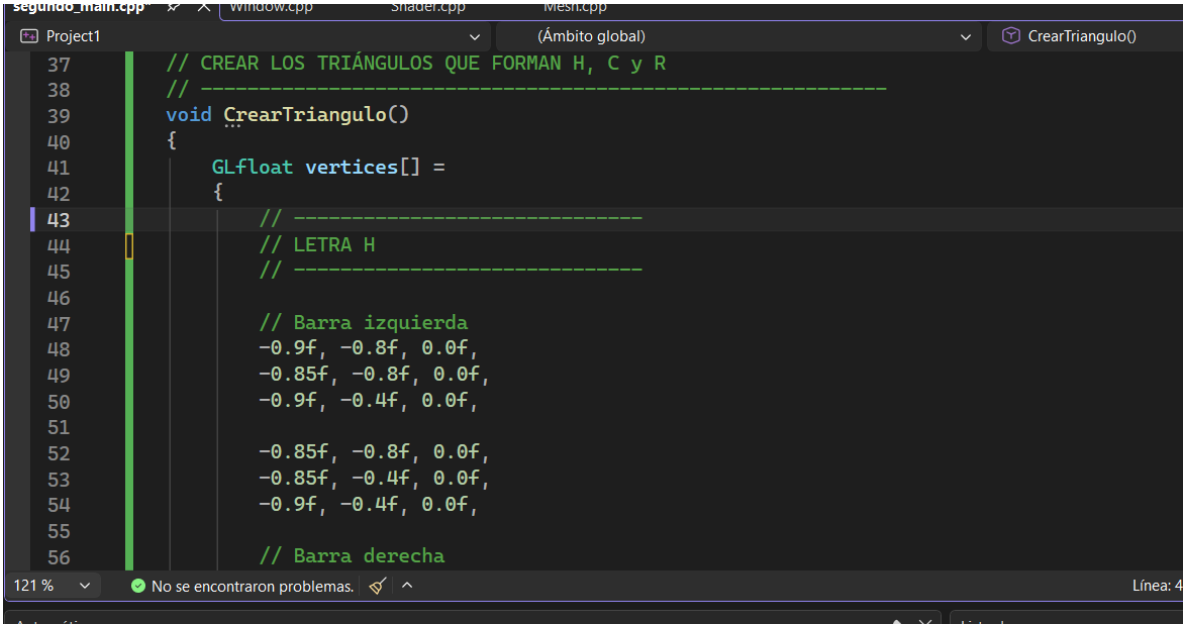
No se encontraron problemas.

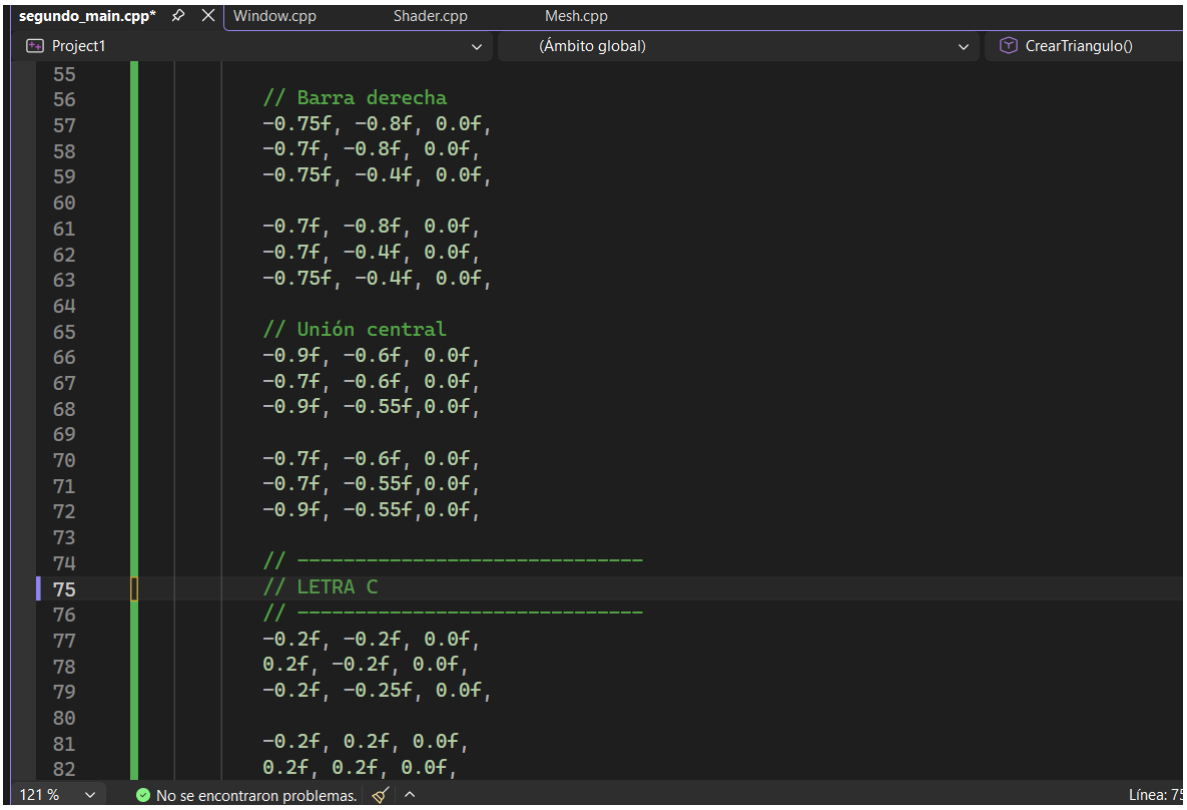
```
    glBindVertexArray(0);
    glfwSwapBuffers(mainWindow);

    std::this_thread::sleep_for(std::chrono::milliseconds(2000));
}
```

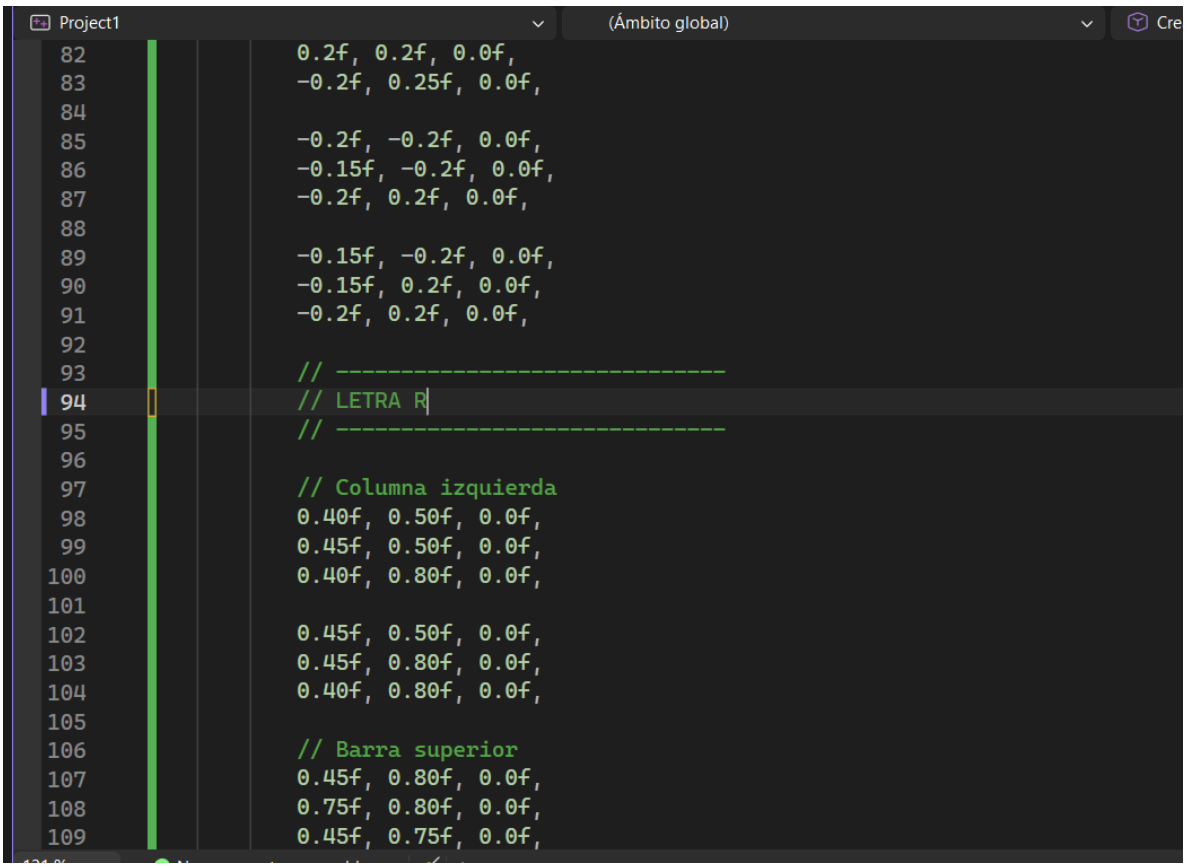
2.- 3 letras iniciales de sus nombres creadas a partir de triángulos, acomodadas en forma diagonal de abajo hacia arriba, todas las letras son del mismo color.

En este ejercicio lo que hice fue construir mis iniciales *HCR* usando únicamente triángulos en OpenGL. Literalmente fui definiendo vértice por vértice en un arreglo GLfloat vertices[], acomodándolos a mano dentro del plano 2D normalizado (de –1 a 1). Para cada letra dibujé varias partes: columnas, barras y diagonales, y cada una la formé con dos triángulos para crear rectángulos o formas más complejas.

Después de definir todos los vértices, generé un VBO y un VAO para mandarlos a la GPU, y activé el atributo de posición con glVertexAttribPointer. Finalmente, en el while simplemente hice glDrawArrays() para que OpenGL renderizara mis triángulos en pantalla. Sinceramente tardé un buen rato ajustando coordenadas, pero al final pude formar las letras de manera limpia.

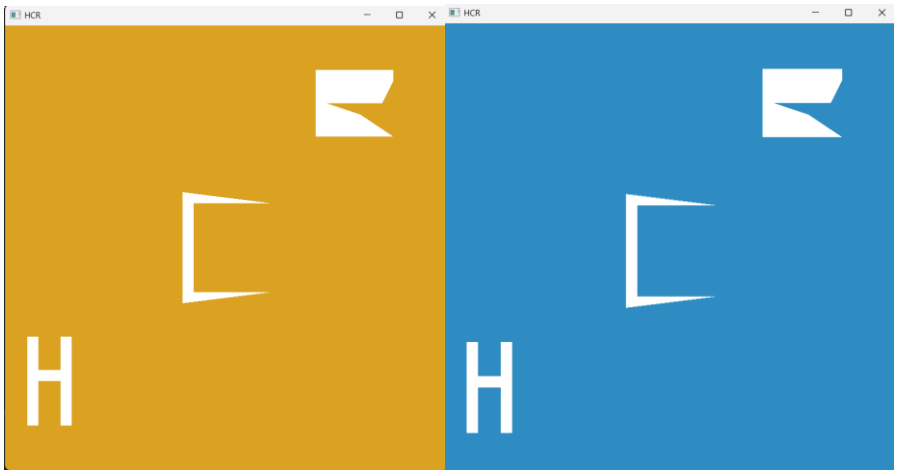



segundo_main.cpp*
Window.cpp
Shader.cpp
Mesh.cpp
Project1
(Ámbito global)
CrearTriangulo()
// Barra derecha
-0.75f, -0.8f, 0.0f,
-0.7f, -0.8f, 0.0f,
-0.75f, -0.4f, 0.0f,
-0.7f, -0.8f, 0.0f,
-0.7f, -0.4f, 0.0f,
-0.75f, -0.4f, 0.0f,
// Unión central
-0.9f, -0.6f, 0.0f,
-0.7f, -0.6f, 0.0f,
-0.9f, -0.55f,0.0f,
-0.7f, -0.6f, 0.0f,
-0.7f, -0.55f,0.0f,
-0.9f, -0.55f,0.0f,
// ------------------------------
// LETRA C
// ------------------------------
-0.2f, -0.2f, 0.0f,
0.2f, -0.2f, 0.0f,
-0.2f, -0.25f, 0.0f,
-0.2f, 0.2f, 0.0f,
0.2f, 0.2f, 0.0f,
121 %
No se encontraron problemas.
Línea: 75


Project1
(Ámbito global)
0.2f, 0.2f, 0.0f,
-0.2f, 0.25f, 0.0f,
-0.2f, -0.2f, 0.0f,
-0.15f, -0.2f, 0.0f,
-0.2f, 0.2f, 0.0f,
-0.15f, -0.2f, 0.0f,
-0.15f, 0.2f, 0.0f,
-0.2f, 0.2f, 0.0f,
// ------------------------------
// LETRA R
// ------------------------------
// Columna izquierda
0.40f, 0.50f, 0.0f,
0.45f, 0.50f, 0.0f,
0.40f, 0.80f, 0.0f,
0.45f, 0.50f, 0.0f,
0.45f, 0.80f, 0.0f,
0.40f, 0.80f, 0.0f,
// Barra superior
0.45f, 0.80f, 0.0f,
0.75f, 0.80f, 0.0f,
0.45f, 0.75f, 0.0f,

```
109             0.45f, 0.75f, 0.0f,
110
111             0.75f, 0.80f, 0.0f,
112             0.75f, 0.75f, 0.0f,
113             0.45f, 0.75f, 0.0f,
114
115             // Parte derecha vertical
116             0.75f, 0.75f, 0.0f,
117             0.70f, 0.65f, 0.0f,
118             0.45f, 0.75f, 0.0f,
119
120             0.70f, 0.65f, 0.0f,
121             0.45f, 0.65f, 0.0f,
122             0.45f, 0.75f, 0.0f,
123
124             // Unión interior
125             0.45f, 0.65f, 0.0f,
126             0.60f, 0.60f, 0.0f,
127             0.45f, 0.50f, 0.0f,
128
129             // Pierna diagonal
130             0.60f, 0.60f, 0.0f,
131             0.75f, 0.50f, 0.0f,
132             0.45f, 0.50f, 0.0f
133         };
134
```

Los dos ejercicios se muestran de forma simultanea y están en el mismo main



HAY MUCHOS COLORES MÁS!

2. PROBLEMAS PRESENTADOS. LISTAR SI SURGIERON PROBLEMAS A LA HORA DE EJECUTAR EL CÓDIGO

Durante la elaboración y ejecución del código sí me encontré con varios problemas. El más notable fue que las figuras, sobre todo la letra *R*, no tomaban la forma correcta porque algunos vértices estaban mal posicionados o generaban líneas no deseadas. También tuve dificultades al calcular la cantidad exacta de triángulos que debía dibujar y ajustar el tamaño de los segmentos para que las letras se vieran proporcionadas. Además, en algunos intentos surgieron errores al compilar los shaders y al cargar el arreglo de vértices, lo que me obligó a revisar el orden y la estructura del arreglo. Todos estos detalles los fui corrigiendo poco a poco hasta que el programa finalmente mostró las letras como esperaba.

3. CONCLUSIÓN:

A. LOS EJERCICIOS DE LA CLASE: COMPLEJIDAD, EXPLICACIÓN

B. COMENTARIOS GENERALES: FALTÓ EXPLICAR A DETALLE, IR MÁS LENTO EN ALGUNA EXPLICACIÓN, OTROS COMENTARIOS Y SUGERENCIAS.

En cuanto a la complejidad de los ejercicios de la clase, sentí que el reto principal fue entender cómo construir figuras completas únicamente con triángulos y cómo traducir formas visuales a coordenadas exactas dentro del plano de OpenGL. Aunque al principio me costó trabajo visualizar dónde colocar cada vértice, conforme avancé fui comprendiendo la lógica y pude ensamblar las letras de forma más precisa. La parte de los shaders y el buffer ya se me hizo más familiar, pero definitivamente la construcción geométrica fue donde más tuve que pensar y ajustar.

Como comentario general, sentí que en algunas partes de la explicación habría sido útil ir un poco más despacio o mostrar más ejemplos visuales, sobre todo al trabajar con coordenadas y proporciones. A veces los conceptos avanzaban muy rápido y tuve que regresar varias veces a revisar mis notas para poder seguir el ritmo. Aun así, la clase fue interesante y me permitió practicar tanto lógica espacial como programación, y creo que añadir explicaciones más detalladas y pausadas ayudaría mucho a reforzar los temas.

Fuentes de consulta:

Foley, J. D., van Dam, A., Feiner, S. K., & Hughes, J. F. (1995). *Computer graphics: Principles and practice* (2nd ed.). Addison-Wesley.

Stein, S. K., & Barcellos, A. (2011). *Calculus and analytic geometry*. McGraw-Hill.

Kessenich, J., Sellers, G., & Liceaga, A. (2016). *OpenGL® Programming Guide: The Official Guide to Learning OpenGL* (9th ed.). Addison-Wesley.