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: Diego Aldair García Hernández

Nº de Cuenta: 31902404-5

GRUPO DE LABORATORIO: 02

GRUPO DE TEORÍA: 04

SEMESTRE 2025-2

FECHA DE ENTREGA LÍMITE: 15 de febrero de 2025

CALIFICACIÓN: _____

Practica 1: Introducción a OpenGL

Introducción

En esta práctica se desarrollarán dos ejercicios utilizando OpenGL. El primero consiste en cambiar el color del fondo de la ventana de manera aleatoria cada dos segundos, donde empleare funciones para la generación de números aleatorios y gestión del tiempo con GLFW. El segundo ejercicio se enfocará en la representación de letras a partir de triángulos, lo que permitirá entender la manipulación de vértices y buffers en OpenGL. A través de estos ejercicios, se reforzará el conocimiento sobre la estructura básica de un programa en OpenGL, el uso de shaders y manejo de eventos de renderizado aprendido durante la primera clase de laboratorio.

Ejecución de los ejercicios

Ejercicio 1: Cambio de color aleatorio cada 2 segundos.

En este ejercicio se implementó un programa en OpenGL que cambia el color de fondo aleatoriamente cada 2 segundos.

Primero, se declararon tres variables de tipo float (rojo, verde, azul) que representa los valores de los colores en formato RGB, todos con valor inicializado en 0.0. También se declara la variable de tipo float llamada *duracion*, que tiene el valor 2.0, es la variable que define el tiempo en segundos para cambiar de color.

Posteriormente, se creó la función *generarColorAleatorio()* que genera valores aleatorios para los componentes rojo, verde y azul utilizando *rand()* que es una función de la biblioteca estándar de C (stdlib.h) donde RAND_MAX es un valor constante máximo definido que al operar el numero por esta constante, normaliza el valor dentro del rango [0.0, 1.0]. Esto asegura que rojo, verde y azul tengan valores flotantes entre 0.0 y 1.0.

```
// Variables de color
float rojo, verde, azul = 0.0f; // Variables
float duracion = 2.0f;

void generarColorAleatorio() {
    rojo = (float)rand() / RAND_MAX;
    verde = (float)rand() / RAND_MAX;
    azul = (float)rand() / RAND_MAX;
}
```

Dentro del main, se inicializa la aleatoriedad con *srand()*, que también es una función de la biblioteca stdlib.h que establece la semilla para el generador de números aleatorios. Como argumento tiene *time(NULL)* que devuelve el tiempo actual en segundos. Pasar como argumento *time(NULL)* a *srand()* garantiza que los números aleatorios generados por *rand()* sean diferentes en cada ejecución del programa.

La línea donde se llama a la función *generarColorAleatorio()* define los valores aleatorios para las variables de color, asegurando que la ventana inicie con un color aleatorio en el rango RGB.

La línea siguiente con *glfwSetTime(0)* establece el temporizador de GLFW en 0 segundos para controlar la frecuencia con la que se ejecutan ciertas acciones en el programa, incluyendo el cambiar de color cada dos segundos.

```
// Iniciar aleatoridad
srand(time(NULL));
generarColorAleatorio();
glfwSetTime(0); // Para reiniciar el temporizador de GLFW
```

Por consiguiente, se tiene una condicional if que tiene como condición a *glfwGetTime()* que obtiene el tiempo transcurrido desde la última vez que se reinició el temporizador con *glfwSetTime(0)*, y se compara con la variable *duracion*. Si han pasado al menos 2 segundos, se llama a *generarColorAleatorio()*, lo que cambia los valores de las variables de colores y generando un nuevo color aleatorio. Luego *glfwSetTime(0)* reinicia el temporizador para que comience un nuevo ciclo de dos segundos.

```
// Cambio de estado
if (glfwGetTime() >= duracion) {
    generarColorAleatorio();
    glfwSetTime(0); // Reiniciar el temporizador a 0

}

//Limpiar la ventana
glClearColor(rojo, verde, azul, 1.0f);
glClear(GL_COLOR_BUFFER_BIT);
```

Finalmente, se define el color de fondo de la ventana utilizando las variables rojo, verde y azul con los valores actualizados previamente, y se limpia la ventana borrando el buffer de color de la ventana y se reemplaza con el nuevo color especificado.

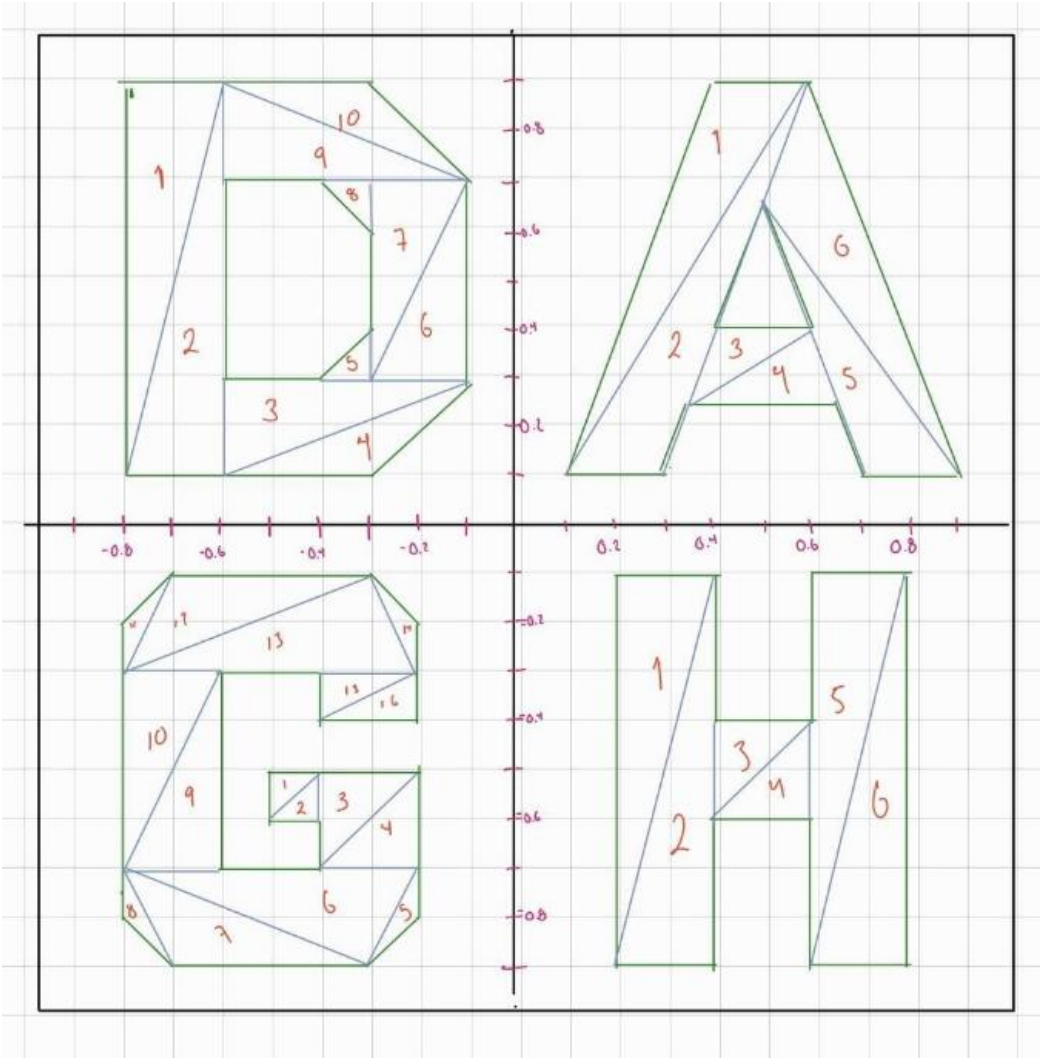
Ejercicio 2: Creación de letras usando triángulos

Este ejercicio consiste en representar las letras utilizando triángulos en OpenGL. Para ello, se diseñaron las letras con coordenadas en un espacio cartesiano.

Diseño de las letras.

El diseño de las letras se administró de forma uniforme en donde cada letra estaría representada en un cuadrante la ventana.

- D lado izquierdo superior
- A lado derecho superior
- G lado izquierdo inferior
- H lado izquierdo inferior



Explicación

Primero, se definen los shaders en GLSL, uno para los vértices y otro para los fragmentos. El Vertex Shader se encarga de establecer las posiciones de los vértices en la pantalla. El Fragment Shader define el color de los fragmentos (píxeles) que componen las letras, en este caso un color blanco. Estos shaders se compilan y se asocian al programa con la función *CompileShaders()*, la cual utiliza *glCreateProgram()*, *glAttachShader()* y *glLinkProgram()* para crear el programa de sombreado.

```
//Vertex Shader
//recibir color, salida Vcolor
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\
}";
//recibir Vcolor y dar de salida color
static const char* fShader = "                    \n\
#version 330                                      \n\
out vec4 color;                                   \n\
void main()                                       \n\
{                                                 \n\
    color = vec4(1.0f,1.0f,1.0f,1.0f);            \n\
}";
```

Posteriormente, el código define una función llamada *CrearTriangulo()* que genera los vértices necesarios para representar las letras iniciales de un nombre utilizando triángulos. Los vértices se almacenan en un arreglo de *GLfloat*, donde cada grupo de tres valores representa un punto en el espacio 3D con coordenadas (x, y, z).

Por ejemplo, la letra "D" está compuesta por múltiples triángulos, cada uno definido por tres vértices:

```
void CrearTriangulo()
{
    GLfloat vertices[] = {
        // D
        -0.8f, 0.9f, 0.0f, //1
        -0.6f, 0.9f, 0.0f,
        -0.8f, 0.1f, 0.0f,

        -0.6f, 0.9f, 0.0f, //2
        -0.8f,0.1f, 0.0f,
        -0.6f, 0.1f, 0.0f,

        -0.6f, 0.1f, 0.0f, //3
        -0.6f, 0.3f, 0.0f,
        -0.1f, 0.3f, 0.0f,

        -0.6f, 0.1f, 0.0f, //4
        -0.3f, 0.1f, 0.0f,
        -0.1f, 0.3f, 0.0f,

        -0.4f, 0.3f, 0.0f, //5
        -0.3f, 0.3f, 0.0f,
        -0.3f, 0.4f, 0.0f,

        -0.3f, 0.3f, 0.0f, //6
        -0.1f, 0.3f, 0.0f,
        -0.1f, 0.7f, 0.0f,
```

Luego, estos datos se cargan en un Vertex Buffer Object (VBO) y se asocian a un Vertex Array Object (VAO), permitiendo que sean utilizados en la renderización.

```
glGenVertexArrays(1, &VAO); //generar 1 VAO
glBindVertexArray(VAO);//asignar VAO

glGenBuffers(1, &VBO);
glBindBuffer(GL_ARRAY_BUFFER, VBO);
glBufferData(GL_ARRAY_BUFFER, sizeof(vertices), vertices, GL_STATIC_DRAW); //pasar

glVertexAttribPointer(0, 3, GL_FLOAT, GL_FALSE, 3 * sizeof(GL_FLOAT), (GLvoid*)0);
glEnableVertexAttribArray(0);
//agregar valores a vèrtices y luego declarar un nuevo vertexAttribPointer
glBindBuffer(GL_ARRAY_BUFFER, 0);
glBindVertexArray(0);
```

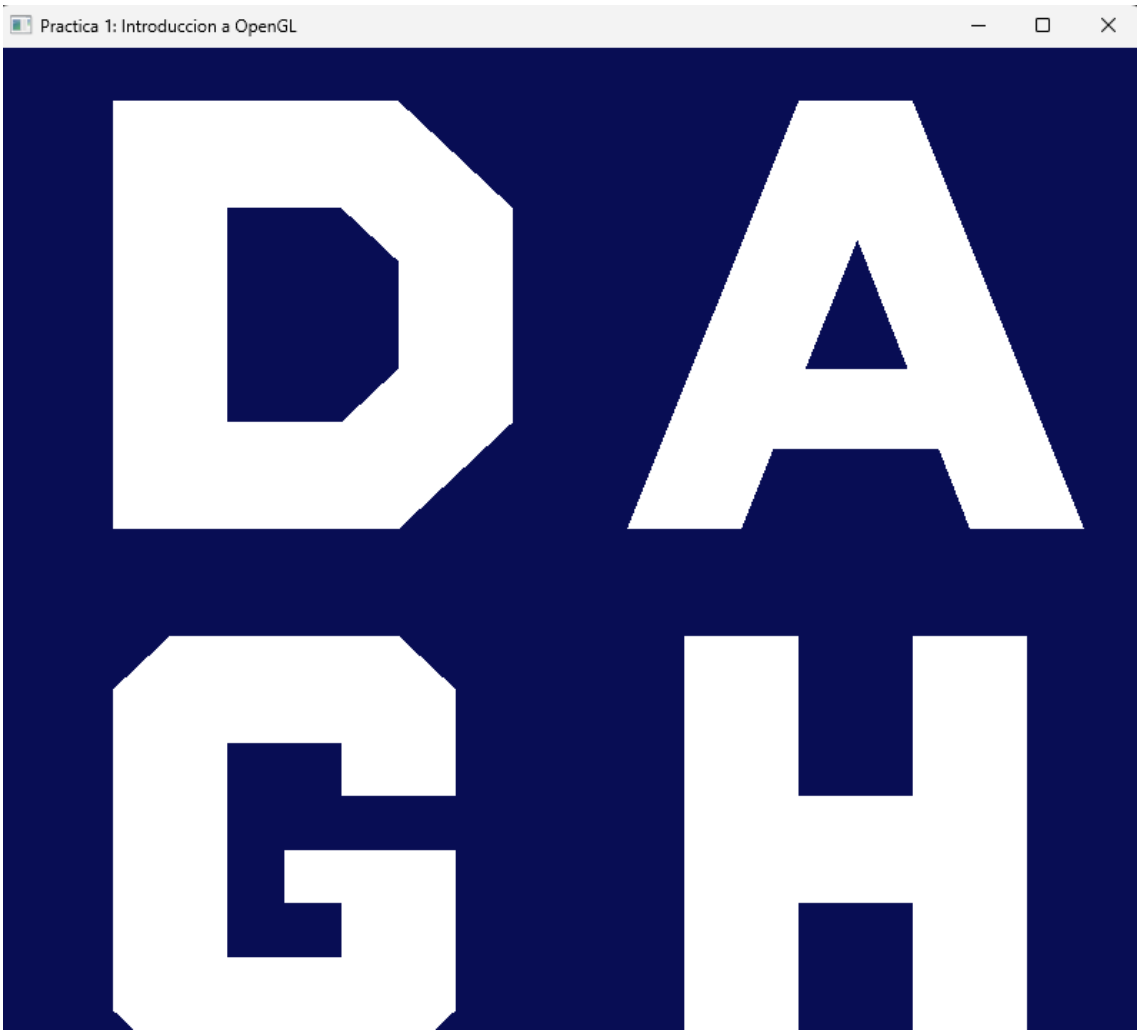
Dentro del *main()* se ejecuta un bucle que mantiene la ventana abierta mientras no se solicite su cierre. Dentro del bucle se activan los shaders y se dibujan las letras usando *glDrawArrays(GL_TRIANGULES, 0, 114);* El numero 114 es la cantidad total de vértices que se dibujaron dentro de la ventana.

Finalmente, el programa finaliza cuando se cierra la ventana.

```
C:\compu_Grafica\CGIH_GHDA_P1\CGIH_GHDA_P1_E1\Debug\CGIH_GHDA_P1_E1.exe (process 15364) exited with code 0 (0x0).
To automatically close the console when debugging stops, enable Tools->Options->Debugging->Automatically close the conso
le when debugging stops.
Press any key to close this window . . .
```

Salida.

La salida de las letras en la ventana se representa tal cual se menciona en el diseño. Las letras fueron representadas de color blanco, mientras que el fondo de la ventana cambia de color cada 2 segundos.



Problemas

Durante la práctica, tuve algunos errores comunes, como olvidar puntos y comas en el código, lo que causó problemas de compilación. Estos fueron fáciles de corregir con una revisión cuidadosa. El problema más notable ocurrió al dibujar las letras con *glDrawArrays(GL_TRIANGLES, 0, #VERTICES);* ya que algunos triángulos no aparecían en pantalla. Después de revisar el código, me di cuenta de que había contado mal la cantidad de vértices necesarios. La solución fue verificar y ajustar el número de vértices en el VBO para asegurarme de que coincidiera con los datos reales. Una vez corregido todas las letras se dibujaron correctamente.

También enfrenté una complicación al diseñar las letras, ya que al principio no lograba que las formas tuvieran la apariencia deseada. La distribución de los triángulos no siempre encajaba correctamente, lo que hacía que algunas partes de las letras se vieran desproporcionadas o fuera de alineación. Para solucionar esto recurrí a conceptos de geometría analítica dividiendo el espacio en coordenadas y ajustando los puntos estratégicamente para asegurar una representación más precisa como se muestra en la parte de diseño.

Conclusión

En esta práctica se logró implementar dos ejercicios que me permitieron reforzar el uso de OpenGL para la manipulación de gráficos y renderización en tiempo real. A través del primer ejercicio, se comprendió como generar colores aleatorios y aplicarlos al fondo de la ventana con una periodicidad definida, utilizando la función *rand()* junto con la gestión de tiempo de GLFW. Este proceso ayudo a familiarizarme con la manipulación del color en OpenGL y el uso de eventos para actualizar la ventana.

El segundo ejercicio me permitió profundizar en la representación de figuras geométricas mediante vértices y triángulos, lo que facilito la comprensión del uso de VBO y VAO para la administración de datos en la GPU. A pesar de los errores iniciales en la cantidad de vértices y la activación de los shaders, la práctica permitió aprender la importancia de verificar la estructura de los datos y ajustar la cantidad de elementos renderizados correctamente.

Finalmente, esta experiencia proporciono un conocimiento más solido sobre los fundamentos de OpenGL, desde la configuración inicial hasta la renderización y actualización en tiempo real.

Bibliografía

- cplusplus.com. (s.f.). rand. Recuperado de https://cplusplus.com/reference/cstdlib/rand/
- Stack Overflow. (2015, 27 de junio). Using rand()/(RAND_MAX +1). Recuperado de https://stackoverflow.com/questions/31089952/using-rand-rand-max-1
- aprenderaprogramar.com. (s.f.). Generar números o secuencias aleatorios en C. Intervalos. srand y rand. Time null. RAND_MAX. Recuperado de https://www.aprenderaprogramar.com/index.php?option=com_content&view=article&id=917:generar-numeros-o-secuencias-aleatorios-en-c-intervalos-srand-y-rand-time-null-randmax-cu00525f&catid=82&Itemid=210
- Aghajari, A. (2019, 2 de octubre). Shading the Canvas: A Beginner's Guide to Vertex and Fragment Shaders. Medium. Recuperado de https://medium.com/@aghajari/shading-the-canvas-a-beginners-guide-to-vertex-and-fragment-shaders-f2a0b446294f
- opengl-tutorial.org. (s.f.). Tutorial 2: El primer triángulo. Recuperado de https://www.opengl-tutorial.org/es/beginners-tutorials/tutorial-2-the-first-triangle/
- GLFW. (s.f.). Time input. Recuperado de https://www.glfw.org/docs/3.0/group__time.html