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: Bueno Hernández Héctor Daniel

Nº de Cuenta: 319233148

GRUPO DE LABORATORIO: 02

GRUPO DE TEORÍA: 04

SEMESTRE 2025-2

FECHA DE ENTREGA LÍMITE: 15/02/2025

CALIFICACIÓN: _____

Cambio de color del fondo de pantalla

Se solicitó cambiar el color del fondo de pantalla de tal forma que este mostrara colores aleatorios del rango RGB sin algún orden específico. Este cambio se debe dar cada dos segundos y cada vez se debe mostrar un color distinto. Para poder dar solución a este ejercicio utilicé nuevamente las variables `red`, `green` y `blue` que declaré previamente en los ejercicios de clase para asignarle un valor diferente al rojo, verde y azul de los pixeles de la pantalla.

Primero, fue necesario agregar la biblioteca estándar `<random>` para poder utilizar las funciones del siguiente fragmento de código en C++:

```
//Creando el objeto para generar el número aleatorio
random_device rd;       //Semilla del generador
mt19937 gen(rd());      //Generador de números aleatorios
uniform_real_distribution<> distrib(0.0, 1.0);      //Distribución uniforme de números reales entre 0.0 y 1.0

//Valores iniciales de las variables del color del fondo
red = distrib(gen);
green = distrib(gen);
blue = distrib(gen);
```

Primero, se declara un objeto de la clase `random_device` llamado `rd`, el cual actuara como la semilla del generador de números aleatorios. Así mismo, se declara un objeto de `mt19937`, una clase para generar números pseudoaleatorios, y se le pasa la semilla `rd` como argumento para que en cada ejecución se obtenga un número aleatorio distinto. Posteriormente, se declara un objeto de la clase `uniform_real_distribution<>` para poder generar números reales aleatorios en un rango dado, en este caso se busca que los números generados sean entre 0.0 y 1.0, por lo que dichos números se pasan como argumento. Con esto ya tenemos todo lo necesario para poder generar los números aleatorios.

En la segunda parte del fragmento se les asigna un valor inicial aleatorio a las variables `red`, `green` y `blue` haciendo uso de los objetos antes creados. Se utiliza el objeto de la clase `uniform_real_distribution<>` antes declarado y como argumento recibe a `gen`, el objeto de la clase `mt19937`, creando así tres números aleatorios distintos.

Este fragmento de código se encuentra fuera del ciclo `while` ya que solamente es necesario crear el generador de números aleatorios una vez, de lo contrario, es posible que se pudieran llegar a generar números aleatorios menos consistentes debido que se tendría que inicializar la semilla múltiples veces a lo largo de la ejecución.

Para lograr el que cambio de color del fondo se diese cada dos segundos se utilizaron distintas funciones de la biblioteca `<chronos>`, la cual contiene elementos para trabajar directamente con el reloj de la computadora.

```
high_resolution_clock::time_point tiempo_base = high_resolution_clock::now();   //Tiempo de referencia para empezar a contar

//Loop mientras no se cierra la ventana
while (!glfwWindowShouldClose(mainWindow))
{
    high_resolution_clock::time_point tiempo_actual = high_resolution_clock::now();   //Se almacena el tiempo actual de ejecución
    auto inter = duration_cast<milliseconds>(tiempo_actual - tiempo_base);            //Se hace la diferencia entre los tiempos
```

Primero se declara un objeto de tipo `high_resolution_clock::time_point` llamado `tiempo_base`, al cual se le asigna el valor de retorno de la función `high_resolution_clock::now()`. Lo que hace la función es regresar el tiempo actual con una alta precisión, por lo que `tiempo_base` servirá como una variable auxiliar para determinar cuánto tiempo ha transcurrido y en este caso se le está asignando un tiempo inicial.

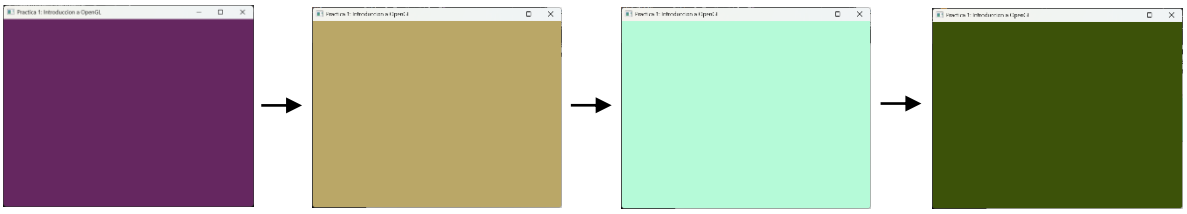
Posteriormente, ya dentro del ciclo `while`, se declara otro objeto de la clase `high_resolution_clock::time_point` llamado `tiempo_actual`. Del mismo modo que con `tiempo_base`, se vuelve a hacer uso de la función para almacenar en el objeto el tiempo actual, sin embargo, este valor se actualizará cada que se ejecute el bucle, es decir, cada ciclo de reloj.

El objeto `inter` será usado para almacenar la diferencia entre los dos tiempos obtenidos. Para ello, se le asigna el resultado de la resta entre `tiempo_actual` y `tiempo_base` y ese resultado se pasa a milisegundos usando `duration_cast<miliseconds>`.

```
//Cambiar el color del fondo
if (inter.count() >= 2000)      //Si han pasado 2000 milisegundos, es decir, 2 segundos
{
    red = distrib(gen);
    green = distrib(gen);
    blue = distrib(gen);
    tiempo_base = tiempo_actual;    //Se actualiza el tiempo de referencia
}
```

Una vez que se tiene la diferencia entre el tiempo de referencia y el tiempo actual, se evalúa si esta es mayor o igual a 2000. Se utiliza este número como condición ya que representa 2000 milisegundos, lo que es igual a 2 segundos. Si la condición se cumple, se utiliza nuevamente el objeto de la clase `uniform_real_distribution<>` para asignarle nuevos valores aleatorios `red`, `green` y `blue` para cambiar el color del fondo. Finalmente, el tiempo actual es almacenado en `tiempo_base` para que este sea el nuevo tiempo de referencia y se cuenten a partir de ahí otros 2000 milisegundos para hacer el siguiente cambio.

La ejecución se ve de la siguiente forma:

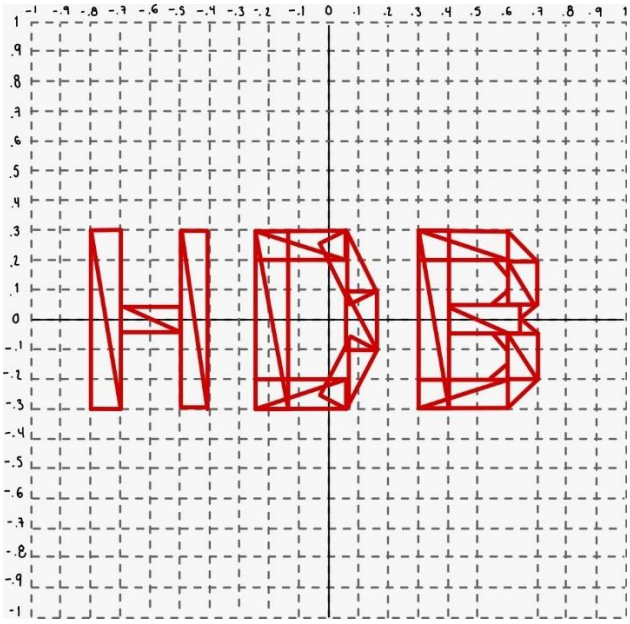


Las capturas de pantalla muestran cuatro de los colores generados pero el programa sigue cambiando de color el fondo de pantalla a colores aleatorios hasta que se cierre la ventana.

Construcción de las iniciales

De forma simultánea al cambio de color del fondo, se solicitó crear tres letras utilizando triángulos. Estas letras deben ser las iniciales del nombre del alumno, en mi caso H, D y B.

Antes de comenzar a establecer las coordenadas de los vértices, realicé un dibujo en mi tablet usando una cuadrícula para poder diseñar las letras sin tener que estar modificando el código. El diseño final obtenido fue el siguiente:



Los valores de cada eje van de -1.0 a 1.0, por lo que cada cuadro del dibujo vale 0.1. Ya teniendo este diseño, me fue mucho más sencillo el poder establecer las coordenadas finales de las letras.

La H está compuesta de 3 rectángulos, por lo contiene 18 vértices como se muestra a continuación:

```
GLfloat vertices[] = {
    //H
    -0.8f,0.3f,0.0f,
    -0.8f,-0.3f,0.0f,
    -0.7f,-0.3f,0.0f,
    -0.8f,0.3f,0.0f,
    -0.7f,0.3f,0.0f,
    -0.7f,-0.3f,0.0f,

    -0.5f,0.3f,0.0f,
    -0.5f,-0.3f,0.0f,
    -0.4f,-0.3f,0.0f,
    -0.5f,0.3f,0.0f,
    -0.4f,0.3f,0.0f,
    -0.4f,-0.3f,0.0f,

    -0.7f,0.05f,0.0f,
    -0.5f,-0.05f,0.0f,
    -0.5f,0.05f,0.0f,
    -0.7f,0.05f,0.0f,
    -0.5f,-0.05f,0.0f,
    -0.7f,-0.05f,0.0f,
```

Para la D, al tener rectángulos inclinados, fue un poco más complicado determinar las coordenadas de los vértices. Sin embargo, fui posible de determinar las coordenadas adecuadamente por medio de prueba y error, logrando que la letra se pudiese distinguir bien. La D está compuesta de 6 rectángulos, por lo que se establecieron 36 vértices en total para su construcción:

```
//D
-0.25f,0.3f,0.0f,
-0.15f,-0.3f,0.0f,
-0.25f,-0.3f,0.0f,
-0.25f,0.3f,0.0f,
-0.15f,-0.3f,0.0f,
-0.15f,0.3f,0.0f,

-0.25f,0.3f,0.0f,
0.05f,0.3f,0.0f,
0.05f,0.2f,0.0f,
-0.25f,0.3f,0.0f,
0.05f,0.2f,0.0f,
-0.25f,0.2f,0.0f,

-0.25f,-0.3f,0.0f,
0.05f,-0.3f,0.0f,
0.05f,-0.2f,0.0f,
-0.25f,-0.2f,0.0f,
0.05f,-0.2f,0.0f,
-0.25f,-0.3f,0.0f,
```

```
0.05f,0.1f,0.0f,
0.15f,0.1f,0.0f,
0.15f,-0.1f,0.0f,
0.05f,0.1f,0.0f,
0.15f,-0.1f,0.0f,
0.05f,-0.1f,0.0f,

0.05f,0.3f,0.0f,
0.15f,0.1,0.0f,
0.06f,0.07,0.0f,
0.05f,0.3f,0.0f,
0.06f,0.07,0.0f,
-0.04f,0.27f,0.0f,

0.15f,-0.1f,0.0f,
0.05f,-0.3f,0.0f,
0.06f,-0.07f,0.0f,
0.05f,-0.3f,0.0f,
0.06f,-0.07f,0.0f,
-0.04f,-0.27f,0.0f,
```

El diseño de la B fue el más complicado de las tres iniciales debido a la forma de la letra. Al contar con dos curvas, tuve que encontrar la forma de representarla sin que esta se viese muy puntiaguda o muy saturada en el centro. Para ello, no solo hice uso de rectángulos, sino también de triángulos individuales para curvear un poco los bordes, logrando así una forma adecuada. El diseño final está conformado por 6 rectángulos y 8 triángulos, por lo que se establecieron 60 vértices en total:

```
//B
0.3f,0.3f,0.0f,
0.4f,-0.3f,0.0f,
0.3f,-0.3f,0.0f,
0.3f,0.3f,0.0f,
0.4f,-0.3f,0.0f,
0.4f,0.3f,0.0f,

0.3f,0.3f,0.0f,
0.6f,0.3f,0.0f,
0.6f,0.2f,0.0f,
0.3f,0.3f,0.0f,
0.6f,0.2f,0.0f,
0.3f,0.2f,0.0f,

0.3f,-0.3f,0.0f,
0.6f,-0.3f,0.0f,
0.6f,-0.2f,0.0f,
0.3f,-0.2f,0.0f,
0.6f,-0.2f,0.0f,
0.3f,-0.3f,0.0f,
```

```
0.4f,0.05f,0.0f,
0.63f,0.05f,0.0f,
0.63f,-0.05f,0.0f,
0.4f,0.05f,0.0f,
0.63f,-0.05f,0.0f,
0.4f,-0.05f,0.0f,

0.6f,0.2f,0.0f,
0.7f,0.2f,0.0f,
0.7f,0.05f,0.0f,
0.6f,0.2f,0.0f,
0.7f,0.05f,0.0f,
0.6f,0.05f,0.0f,

0.6f,-0.2f,0.0f,
0.7f,-0.2f,0.0f,
0.7f,-0.05f,0.0f,
0.6f,-0.2f,0.0f,
0.7f,-0.05f,0.0f,
0.6f,-0.05f,0.0f,
```

```
0.6f,0.3f,0.0f,
0.7f,0.2f,0.0f,
0.6f,0.2f,0.0f,

0.7f,0.05f,0.0f,
0.63f,0.0f,0.0f,
0.63f,0.05f,0.0f,
0.7f,-0.05f,0.0f,
0.63f,0.0f,0.0f,
0.63f,-0.05f,0.0f,

0.7f,-0.2f,0.0f,
0.6f,-0.2f,0.0f,
0.6f,-0.3f,0.0f,

0.6f,0.2f,0.0f,
0.6f,0.15f,0.0f,
0.55f,0.2f,0.0f,
0.6f,0.05f,0.0f,
0.6f,0.1f,0.0f,
0.55f,0.05f,0.0f,
```

```
0.6f,-0.05f,0.0f,
0.55f,-0.05f,0.0f,
0.6f,-0.1f,0.0f,
0.6f,-0.2f,0.0f,
0.6f,-0.15f,0.0f,
0.55f,-0.2f,0.0f
```

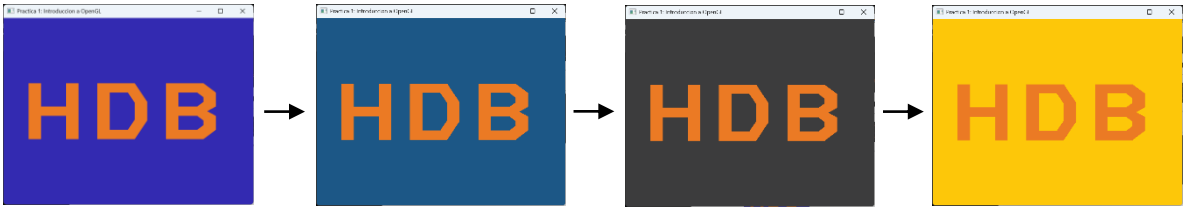
Además, modifiqué el código del shader para que la letras se mostraran de color naranja. Para ello le asigné un valor de 0.92 al rojo, 0.48 al verde y 0.14 al azul:

```
#version 330                                    \n\
out vec4 color;                                 \n\
void main()                                     \n\
{                                               \n\
    color = vec4(0.92f,0.48f,0.14f,1.0f);       \n\
}";
```

Finalmente, para poder ejecutar el programa fue necesario cambiar el arguemento de la función `glDrawArrays` para que se dibujaran los 114 vértices en total:

```
glBindVertexArray(VAO);
glDrawArrays(GL_TRIANGLES, 0, 114);
glBindVertexArray(0);
```

La ejecución del programa ya con el cambio de color y las letras se ve de la siguiente forma:



Problemas presentados

Realmente no tuve mayores problemas al momento de realizar la práctica ya que lo que se pedía pudo ser fácilmente resuelto con los conocimientos adquiridos durante la sesión de laboratorio y con lo visto en los ejercicios de clase. Si bien la parte más tardada fue el diseñar las letras para que se viesen agradables a la vista y después obtener las coordenadas de cada vértice, considero que el hacer el diseño de estas antes de empezar a programar me facilitó bastante el proceso.

Así mismo, no tuve problemas al momento de ejecutar el código ya que tomé como base el código que usé en los ejercicios de clase y solamente cambié unos fragmentos de código.

Conclusiones

La complejidad de los ejercicios de la practica me pareció bastante bien para ser la primera práctica en la que apenas nos estamos familiarizando con OpenGL y aprendiendo a identificar las distintas partes del código.

El buscar funciones para poder generar números aleatorios me ayudó a darme cuenta que en C++ existen mucha funciones que me pudieron haber resuelto el problema a pesar de que haya escogido una solución en específico. Además, el tener que diseñar letras con triángulos fue bastante útil para hacer notar la versatilidad que tiene esta figura.

La explicación dada por el profesor durante la clase de laboratorio me pareció bastante acertada ya que nos explicó las bases de las herramientas a utilizar. Esto nos permitió buscar formas de explotar los triángulos para poder hacer distintas figuras sin que el profesor nos diese la solución a los ejercicios.

Bibliografía

<chrono>. (s. f.). cplusplus.com. Recuperado 14 de febrero de 2025, de https://cplusplus.com/reference/chrono/

GeeksforGeeks. (2021, 30 marzo). *std::mt19937 Class in C++*. GeeksforGeeks. Recuperado 12 de febrero de 2025, de https://www.geeksforgeeks.org/stdmt19937-class-in-cpp/

GeeksforGeeks. (2022, 20 abril). *random header in C++ | Set 1(Generators)*. GeeksforGeeks. Recuperado 14 de febrero de 2025, de https://www.geeksforgeeks.org/random-header-c-set-1generators/

GeeksforGeeks. (2024, 23 mayo). *How to Generate Random Number in Range in C++?* GeeksforGeeks. Recuperado 12 de febrero de 2025, de https://www.geeksforgeeks.org/how-to-generate-random-number-in-range-in-cpp/?ref=ml_lbp

std::chrono::high_resolution_clock::now. (s. f.). cpluspluc.com. Recuperado 12 de febrero de 2025, de https://cplusplus.com/reference/chrono/high_resolution_clock/now/#google_vignette