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: Casillo Martinez Diego Leonardo

N.º de Cuenta: 319041538

GRUPO DE LABORATORIO: 11

GRUPO DE TEORÍA: 06

SEMESTRE 2024-2

FECHA DE ENTREGA LÍMITE: 21/08/2024

CALIFICACIÓN: _____

REPORTE DE PRÁCTICA:

1. Ejecución y código:

Ejercicio 1: Cambiar el color del fondo de la ventana de forma random en las gamas de colores RGB cada 3 segundos

Para este ejercicio, se planteó que el fondo de pantalla de nuestra ventana cambiara cada 3 segundos. Esto se logró utilizando la función rand () de la biblioteca stdlib y controlando el tiempo con la función glfwGetTime, de manera que se capturaran los intervalos de 3 segundos dentro de una condicional. Esto se logró de la siguiente forma:

Primero, fuera del while que detecta si la ventana ha sido cerrada, declaramos una variable para comenzar a contar el tiempo.

```
double primerTiempo = glfwGetTime(); // Para medir el tiempo transcurrido
```

Luego, dentro del while, generamos valores aleatorios y los imprimimos en pantalla para asegurarnos de que no sigan un patrón.

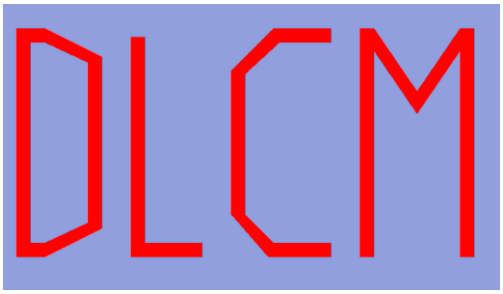
```
double tiempoActual = glfwGetTime();

// Si han pasado 3 segundos, cambiar los colores de fondo de manera aleatoria
if (tiempoActual - primerTiempo >= 3.0)
{
    primerTiempo = tiempoActual;
    // recordemos que rand devuelve un numero aleatorio entre 0
    // y un numero muy, alto, para poder hacer que siempre de un valor de maximo 1 se div
    // que es una constante definidia en la biblioteca stdlib
    Rojo = (float)rand() / RAND_MAX;
    Verde = (float)rand() / RAND_MAX;
    Azul = (float)rand() / RAND_MAX;

    printf("el valor de \n Rojo : %f \n Verde: %f  \n Azul: %f \n\n",Rojo,Verde,Azul);
}
```

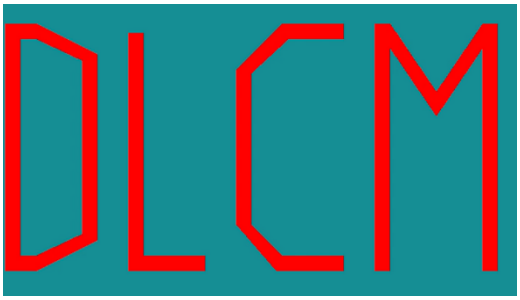
Nota importante: En la función main, es necesario definir la semilla para srand usando el tiempo actual como parámetro, de la siguiente manera:

```
srand(time(NULL));
```

Si no se realiza esta configuración, el comportamiento no será aleatorio, ya que siempre se usará la misma semilla.

De esta forma la ejecución de nuestro código es así:





En terminal podemos comparar los colores para revisar si hay un patrón de comportamiento:

```
el valor de
 Rojo : 0.954283
 Verde: 0.484298
 Azul: 0.998016

el valor de
 Rojo : 0.477157
 Verde: 0.484207
 Azul: 0.113285

el valor de
 Rojo : 0.823573
 Verde: 0.755150
 Azul: 0.708518

el valor de
 Rojo : 0.298898
 Verde: 0.591601
 Azul: 0.775292

el valor de
 Rojo : 0.778710
 Verde: 0.385815
 Azul: 0.370006
```

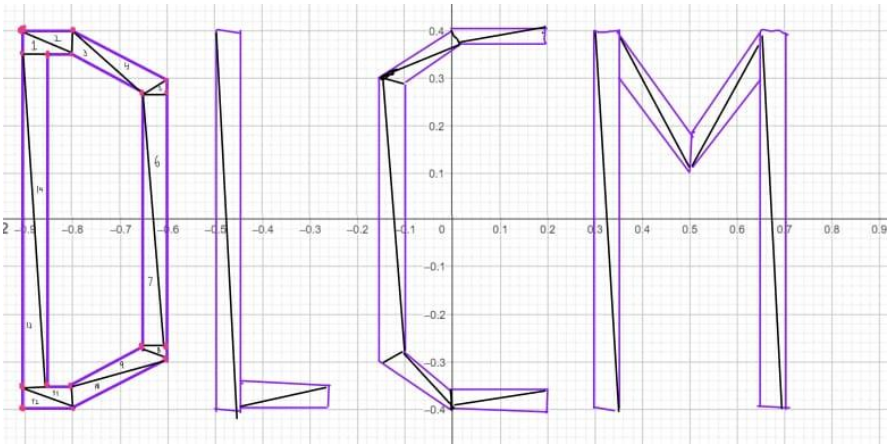
```
el valor de
 Rojo : 0.887204
 Verde: 0.729972
 Azul: 0.097720

el valor de
 Rojo : 0.256874
 Verde: 0.640065
 Azul: 0.464827

el valor de
 Rojo : 0.899014
 Verde: 0.876492
 Azul: 0.588031

el valor de
 Rojo : 0.569964
 Verde: 0.629353
 Azul: 0.870632

el valor de
 Rojo : 0.083163
 Verde: 0.551897
 Azul: 0.580035
```

```
el valor de
 Rojo : 0.970031
 Verde: 0.310709
 Azul: 0.135044

el valor de
 Rojo : 0.505112
 Verde: 0.218787
 Azul: 0.047121

el valor de
 Rojo : 0.831111
 Verde: 0.640461
 Azul: 0.323710

el valor de
 Rojo : 0.654286
 Verde: 0.878445
 Azul: 0.987671

el valor de
 Rojo : 0.527451
 Verde: 0.168493
 Azul: 0.375652
```

Como podemos ver no hay un patrón de comportamiento.

Ejercicio 2: Letras iniciales de sus nombres (3 o 4 letras diferentes) dibujadas con triángulos del mismo color y acomodadas en la ventana que va en un rango de (-1,-1.0) hasta (1,1,0)

Para este ejercicio se planteó dibujar con triángulos las iniciales de nuestros nombres, esto se hizo agregando uno a uno los vértices, y haciendo un diseño en una plantilla cuadriculada de la siguiente forma:

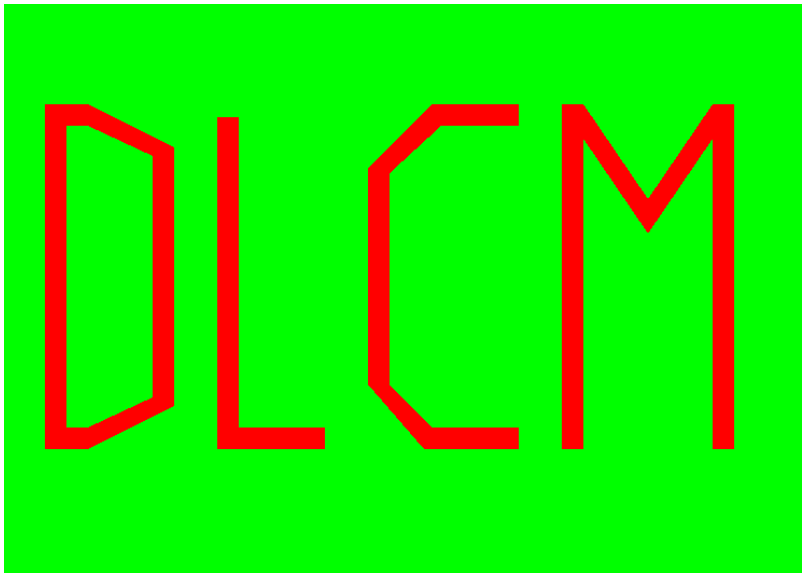


Al pasarlo a arreglo de vértices tenemos el siguiente código (para no llenar de capturas de código solo se usará una parte ya que se hicieron 108 vértices)

```
// Triángulo 1
-0.9f,  0.4f, 0.0f,
-0.8f,  0.4f, 0.0f,
-0.9f,  0.35f, 0.0f,

// Triángulo 2
-0.8f,  0.4f, 0.0f,
-0.8f,  0.35f, 0.0f,
-0.9f,  0.35f, 0.0f,

//triangulo 12
-0.9f, -0.35f, 0.0f,
-0.8f, -0.35f, 0.0f,
-0.9f, -0.4f, 0.0f,

//triangulo 11
-0.8f, -0.35f, 0.0f,
-0.8f, -0.4f, 0.0f,
-0.9f, -0.4f, 0.0f,

// palo izquierdo de la d
// Triángulo 14
```

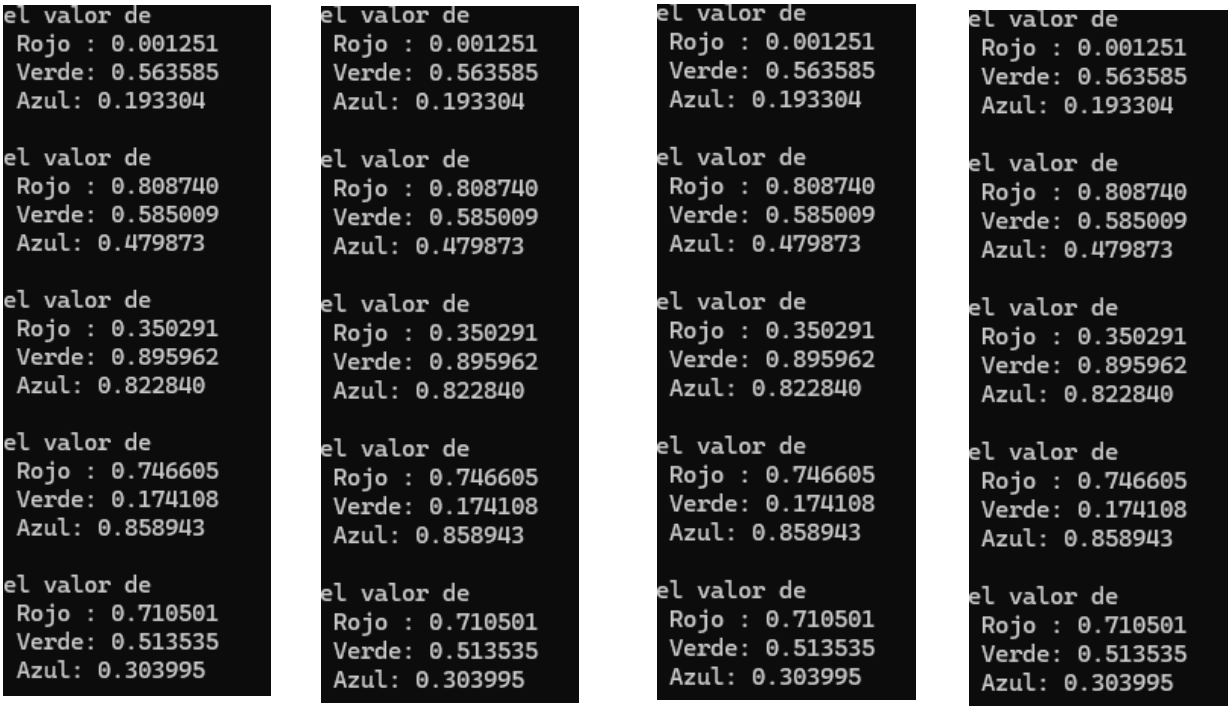
```
// palo izquierdo de la d
// Triángulo 14
-0.9f,  0.35f, 0.0f,
-0.85f, 0.35f, 0.0f,
-0.85f, -0.35f, 0.0f,

// Triángulo 13
-0.9f,  0.35f, 0.0f,
-0.85f, -0.35f, 0.0f,
-0.9f, -0.35f, 0.0f,

// con esto ya tenemos toda la part
// Triángulo 3
-0.8f,  0.4f, 0.0f,
-0.8f,  0.35f, 0.0f,
-0.65f, 0.28f, 0.0f,

// Triángulo 4
-0.8f,  0.4f, 0.0f,
-0.65f, 0.28f, 0.0f,
-0.6f,  0.3f, 0.0f,


// Triángulo 10
-0.8f, -0.4f, 0.0f,
-0.8f, -0.35f, 0.0f,
-0.6f, -0.3f, 0.0f,
```

La ejecución de este código queda de la siguiente forma:



2.- problemas que surgieron:

Para esta práctica surgió un problema al realizar el ejercicio 1. Aunque todo parecía comportarse correctamente, no se estaban controlando los números aleatorios por segundos, lo que hizo que el comportamiento del programa no fuera realmente aleatorio. Además, al no inicializar srand usando el tiempo actual como parámetro, el programa se comportaba de manera predecible. A pesar de que parecía aleatorio, después de tres ejecuciones se notaba que los valores de RGB se repetían constantemente, como se muestra a continuación:



Como podemos observar después de 4 ejecuciones, los colores se repiten siguiendo un patrón.

Esto se debía a dos cosas: en primer lugar, al inicio no se trataba el número aleatorio de manera correcta en relación con los segundos; y en segundo lugar, no se añadió la siguiente línea de código:

```
srand(time(NULL));
```

3.- Conclusión:

Esta práctica presentó una complejidad significativa al trabajar con números aleatorios, ya que, como se explicó en el laboratorio, abordar los problemas relacionados con el ciclo de reloj y los segundos requería un conocimiento adicional sobre algunas funciones integradas en GLW. Esto me llevó a buscar documentación sobre estas funciones.

Por otro lado, el segundo ejercicio no fue particularmente difícil, pero tuvo cierta complejidad debido a que realizar el proceso paso a paso para todos los vértices resultó ser algo tedioso y, en algunos momentos, complicado.

En general, el ejercicio fue entretenido, aunque algo tedioso. Me gustaría sugerir que proporcionen un poco de documentación útil sobre GLFW para facilitar el proceso.

4.- Bibliografía en formato APA

C++ - using glfwGetTime() for a fixed time step. (s. f.). Stack Overflow. https://stackoverflow.com/questions/20390028/c-using-glfwgettime-for-a-fixed-time-step