



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 N° 01

NOMBRE COMPLETO: Hernández Vázquez Daniela

N° de Cuenta: 318092867

GRUPO DE LABORATORIO: 01

GRUPO DE TEORÍA: 02

SEMESTRE 2024-2

FECHA DE ENTREGA LÍMITE: 17 de febrero de 2024

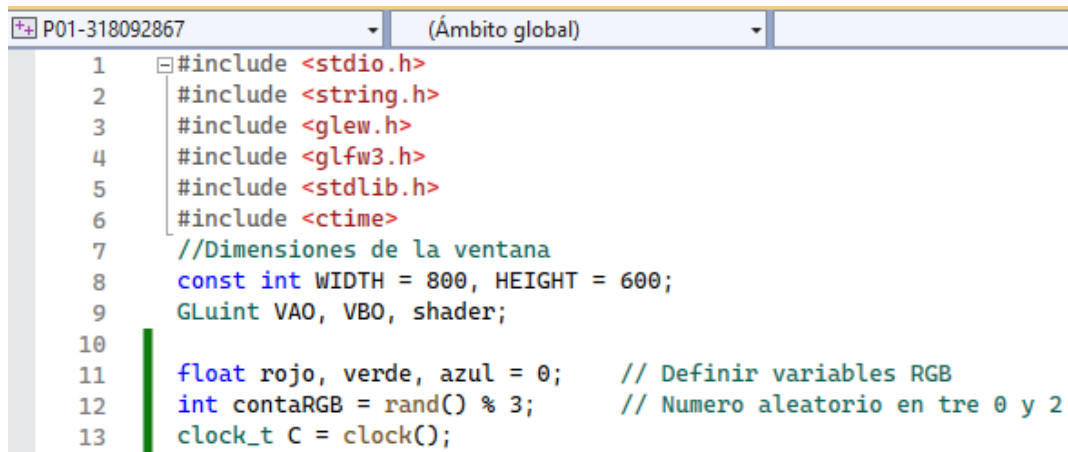
CALIFICACIÓN: _____

REPORTE DE PRÁCTICA:

Actividad 1.

Como primera actividad se realizó una ventana que cambia el color de fondo de forma random tomando rango de colores RGB y con una periodicidad de 2 segundos. Para ello se tomaron como base los ejercicios realizados en clase.

Lo primero que se realizó fue la declaración de las nuevas variables así como las bibliotecas necesarias que se usarán a lo largo del programa. Ahora procedemos a modificar la función principal.

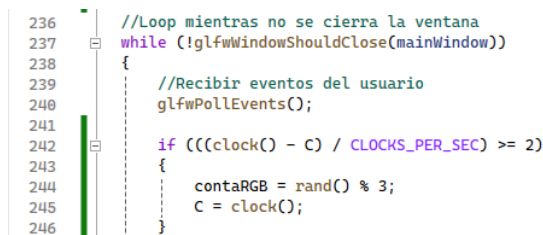


```
P01-318092867 (Ámbito global)
1  #include <stdio.h>
2  #include <string.h>
3  #include <glew.h>
4  #include <glfw3.h>
5  #include <stdlib.h>
6  #include <ctime>
7  //Dimensiones de la ventana
8  const int WIDTH = 800, HEIGHT = 600;
9  GLuint VAO, VBO, shader;
10
11 float rojo, verde, azul = 0;    // Definir variables RGB
12 int contaRGB = rand() % 3;     // Numero aleatorio en tre 0 y 2
13 clock_t C = clock();
```

Figura 1.1 Declaración de bibliotecas y variables

Dentro del bucle *while*, el cual ejecuta mientras la ventana GLFW mainWindow no se cierre, tenemos la función `glfwPollEvents()` la cual mantiene le programa en espera a las acciones del usuario.

Posteriormente es agregada una condicional *if*, dentro de esta se verifica si han pasado al menos 2 segundos desde la última vez que se actualizó el estado de los colores, esto se controla mediante la variable C que almacena el tiempo actual en cada iteración. Si han pasado 2 segundos, se actualiza el estado de los colores (rojo, verde y azul) de acuerdo con el valor de *contaRGB*, que toma un valor aleatorio entre 0, 1 y 2, se pueden repetir, por lo cual un color puede parecer estar más tiempo en la pantalla.



```
236 //Loop mientras no se cierra la ventana
237 while (!glfwWindowShouldClose(mainWindow))
238 {
239     //Recibir eventos del usuario
240     glfwPollEvents();
241
242     if (((clock() - C) / CLOCKS_PER_SEC) >= 2)
243     {
244         contaRGB = rand() % 3;
245         C = clock();
246     }
```

Figura 1.2 Asignar a la variable *contaRGB* un valor aleatorio

Con una segunda condicional *if*, se evalúa para cada caso el color que tendrán las variables de acuerdo con el valor de *contaRGB*.

En cada iteración de limpia el color del buffer de color con el color definido por los valores de rojo, verde y azul utilizando `glClearColor()`, así como se limpia el buffer de color con el color definido previamente.

```
248     if (contaRGB == 0)
249     {
250         rojo = 1;
251         verde = 0;
252         azul = 0;
253     }
254     else if (contaRGB == 1)
255     {
256         rojo = 0;
257         verde = 1;
258         azul = 0;
259     }
260     else if (contaRGB == 2)
261     {
262         rojo = 0;
263         verde = 0;
264         azul = 1;
265     }
266     else
267     {
268         rojo = 1;
269         verde = 1;
270         azul = 1;
271     }
272
273     //Limpiar la ventana
274     glClearColor(rojo, verde, azul, 1.0f);
275     glClear(GL_COLOR_BUFFER_BIT);
```

Figura 1.3 Asignar color RGB de acuerdo a la variable *contaRGB*

Actividad 2.

Dentro del mismo main como segunda actividad se dibujó, a través de primitivas de triángulos, las 3 letras iniciales de nuestros nombres, siendo todas del mismo color. Para esto se utilizó como base el segundo main de las actividades realizadas en clase.

Se declaran tanto el Vertex Shader como el Fragment Shader, cada uno con sus entradas y salidas predefinidas.

```
15  //Vertex Shader
16  //recibir color, salida Vcolor
17  static const char* vShader = "
18  #version 330
19  layout (location =0) in vec3 pos;
20  void main()
21  {
22      gl_Position=vec4(pos.x,pos.y,pos.z,1.0f);
23  }";
24
25  //Fragment Shader
26  //recibir Vcolor y dar de salida color
27  static const char* fShader = "
28  #version 330
29  out vec4 color;
30  void main()
31  {
32      color = vec4(0.0f,0.0f,0.0f,1.0f);
33  }";
```

Figura 1.4 Vertex Shader y Fragment Shader

Dentro de la función CrearTriangulo() se define el arreglo de vértices que se utilizarán, tomaremos en cuenta 9 datos en bloques de 3, al ser leídos 3 datos se formará un vértice con coordenadas x , y y z (VAO) , el conjunto de 9 datos formará un triángulo.

Con estos triángulos procedemos a dibujar nuestras iniciales en la pantalla, usando para cada letra un número determinado de vértices y triángulos:

- Letra D: 27 vértices (9 triángulos)
- Letra H: 18 vértices (6 triángulos)
- Letra V: 12 vértices (4 triángulos)

```

37 void CreateTriangulo()
38 {
39     GLfloat vertices[] = { // so619
40         // D // Consid70
41         -0.90f, 0.4f, 0.0f, //P-Q-R 71
42         -0.90f, -0.4f, 0.0f, 72
43         -0.70f, -0.4f, 0.0f, 73
44         -0.90f, 0.4f, 0.0f, //P-S-R 74
45         -0.70f, 0.4f, 0.0f, 75
46         -0.70f, -0.4f, 0.0f, 76
47         // H
48         -0.30f, 0.10f, 0.0f, //T-U-V 78
49         -0.30f, -0.10f, 0.0f, 79
50         -0.40f, -0.30f, 0.0f, 80
51         -0.30f, 0.10f, 0.0f, //T-W-V 81
52         -0.60f, -0.40f, 0.0f, 82
53         -0.40f, -0.30f, 0.0f, 83
54         -0.30f, -0.10f, 0.0f, //P-Q-R+ 84
55         -0.90f, -0.40f, 0.0f, 85
56         -0.60f, -0.40f, 0.0f, 86
57         // H
58         -0.30f, 0.00f, 0.0f, //T-U-V 88
59         -0.40f, 0.30f, 0.0f, 89
60         -0.40f, -0.30f, 0.0f, 90
61         // H
62         -0.30f, -0.10f, 0.0f, //T-U-V 92
63         -0.30f, 0.10f, 0.0f, 93
64         -0.40f, 0.30f, 0.0f, 94
65         -0.30f, -0.10f, 0.0f, //T-W-V 95
66         -0.60f, -0.40f, 0.0f, 96
67         -0.40f, 0.30f, 0.0f, 95
68         -0.30f, 0.10f, 0.0f, //P-Q-R+ 96
69         -0.90f, 0.40f, 0.0f, 97
70         -0.60f, 0.40f, 0.0f, 98
71         // H
72         0.25f, 0.4f, 0.0f, //1-2-3 102
73         0.25f, -0.4f, 0.0f, 103
74         0.10f, -0.4f, 0.0f, 104
75         0.25f, 0.4f, 0.0f, //1-4-3 105
76         0.10f, 0.4f, 0.0f, 106
77         0.10f, -0.4f, 0.0f, 107
78         // H
79         0.10f, 0.10f, 0.0f, //5-6-7 109
80         0.10f, -0.10f, 0.0f, 110
81         -0.10f, -0.10f, 0.0f, 111
82         0.10f, 0.10f, 0.0f, //5-8-7 112
83         -0.10f, 0.10f, 0.0f, 113
84         -0.10f, -0.10f, 0.0f, 114
85         // H
86         -0.25f, 0.4f, 0.0f, //9-10-11 116
87         -0.25f, -0.4f, 0.0f, 117
88         -0.10f, -0.4f, 0.0f, 118
89         -0.25f, 0.4f, 0.0f, //9-12-11 119
90         -0.10f, 0.4f, 0.0f, 120
91         -0.10f, -0.4f, 0.0f, 121
92         // V
93         0.30f, 0.4f, 0.0f, //1-A-B 123
94         0.30f, 0.4f, 0.0f, //1-A-B 96
95         0.50f, 0.4f, 0.0f, 97
96         0.50f, -0.4f, 0.0f, 98
97         0.70f, -0.4f, 0.0f, //C-A-B
98         0.50f, 0.4f, 0.0f, 99
99         0.50f, -0.4f, 0.0f, 100
100        0.70f, -0.4f, 0.0f, //C-D-B
101        0.70f, 0.4f, 0.0f, 102
102        0.50f, -0.4f, 0.0f, 103
103        0.70f, -0.4f, 0.0f, //C-D-E
104        0.70f, 0.4f, 0.0f, 105
105        0.90f, 0.4f, 0.0f, 106
106    };
107    glGenVertexArrays(1, &VAO); //generar 1 VAO
108    glBindVertexArray(VAO); //asignar VAO
109
110    glGenBuffers(1, &VBO); // Asigna en memoria la cantidad necesaria para almas
111    glBindBuffer(GL_ARRAY_BUFFER, VBO);
112    glBindBuffer(GL_ARRAY_BUFFER, sizeof(vertices), vertices, GL_STATIC_DRAW);
113
114    glVertexAttribPointer(0, 3, GL_FLOAT, GL_FALSE, 3 * sizeof(GL_FLOAT), (GLvoid*)
115    ((Arrriba) cantidad de datos iniciales, flotante, no normaliza, cuantos valo
116    //agregar valores a vertices y luego declarar un nuevo vertexAttribPointer
117    glBindBuffer(GL_ARRAY_BUFFER, 0);
118    glBindVertexArray(C0);

```

Figura 1.5 Vértices de letras

Finalmente, en el main principal despues se activa el uso del shader y se enlazan el búfer de vértices (VAO) para dibujar las primitivas (triángulos en este caso) utilizando el shader y los datos de vértices del VAO actual.

```

273 //Limpiar la ventana
274 glClearColor(rojo, verde, azul, 1.0f);
275 glClear(GL_COLOR_BUFFER_BIT);
276
277 glUseProgram(shader);
278
279 glBindVertexArray(VAO);
280 glDrawArrays(GL_TRIANGLES, 0, 1200);
281 // (ARRIBA) Primitiva se puede modificar a LINES, TRIANGLES , POINTS
282 // // se modifica el indice para ver el numero de vertices
283 glBindVertexArray(0);
284
285 glUseProgram(0);
286
287 glfwSwapBuffers(mainWindow);

```

Figura 1.6 Índice de vértices

Como ya se mencionó con anterioridad, mientras la ventana no sea cerrada o que el usuario interactúe con ella los elementos continuaran limpiándose y sustituyéndose por los colores predefinidos en cada iteración.



Figura 1.7 Ejecución del programa

Problemas:

En la primera actividad la mayor complejidad se tuvo a la hora de asignar un número aleatorio y que este continuara siendo el mismo sin importar que el ciclo while se repitiera con cada ciclo, esto provocaba en un principio que los colores cambiaran demasiado rápido por lo que no se podían apreciar correctamente. Para resolver esto se tenía la opción de utilizar un contador que aumentara con cada ciclo y poner un numero cercano a los 2 segundos o, en este caso utilizar una variable ya relacionada a los ciclos de reloj, en este caso Ctime, esto permitió tener un control más preciso pues sin importar cuantos ciclos transcurrieran era mas factible controlar la condición mediante un tiempo ya establecido.

Mientras la segunda actividad su mayor dificultad era calcular el número de triángulos en el espacio, así que se realizó un boceto en papel ubicando con coordenadas cada uno de los vértices para posteriormente dibujarlos en el plano.

Al haber centrado las letras fue relativamente sencillo ya que en el caso de la D y H se reflejan en la parte de abajo.

En la función `glDrawArrays(GL_TRIANGLES, 0, 1200)`, el número de vértices a mostrar se modifica para poder observar todos los vértices definidos, si es colocado un número menor de vértices a los necesarios los datos faltantes no se dibujaran en pantalla, por este motivo se puso un número arbitrario que los cubriera.

Conclusión:

A mi parecer los ejercicios tuvieron un buen nivel, resulta algo compleja la manipulación del tiempo en el programa y mantener la consistencia en la selección aleatoria de colores a lo largo del tiempo, lo cual se solucionó utilizando una variable relacionada con el tiempo para controlar el cambio de color de manera más precisa.

En la segunda actividad, se crearon las tres primeras letras de los nombres a partir de triángulos, todas del mismo color, y se mostraron simultáneamente con la primera actividad en el mismo main. La mayor dificultad fue calcular el número de triángulos necesarios para formar las letras en el espacio, pero esto no siempre puede ser aplicado. La explicación de clase me pareció suficiente para poder tener los elementos para modificar el programa, aunque algo apresurada. A pesar de ello las actividades se realizaron exitosamente.

Referencias:

Kessenich, J. M., Sellers, G., & Shreiner, D. (2016). *OpenGL Programming Guide: The Official Guide to Learning OpenGL, Version 4.5 with SPIR-V*. OpenGL.

Méndez Servín, M. (2022). *NOTAS PARA EL CURSO DE GRAFICACIÓN POR COMPUTADORA* [Libro electrónico].

https://prometeo.matem.unam.mx/recursos/VariosNiveles/iCartesiLibri/recursos/Notas_Graficacion_por_Computadora/index.html

Moreno V., J. (2023). *Tema 3 y Tema 4* [Diapositivas; Presentación electrónica]. www.uhu.es. https://www.uhu.es/francisco.moreno/gii_rv/

Ramírez, E. (2014). Despliegue básico en OpenGL moderno. *Lecturas En Ciencias de la Computación*, ISSN 1316-6239.

Sellers, G., Wright, R. S., & Haemel, N. (2015). *OpenGL superbible: Comprehensive Tutorial and Reference*. Addison-Wesley Professional.