



UNIVERSIDAD NACIONAL AUTÓNOMA DE MÉXICO

FACULTAD DE INGENIERÍA

DIVISIÓN DE INGENIERÍA ELÉCTRICA

INGENIERÍA EN COMPUTACIÓN

LABORATORIO DE COMPUTACIÓN GRÁFICA e
INTERACCIÓN HUMANO COMPUTADORA



EJERCICIOS DE CLASE N° 01

NOMBRE COMPLETO: Hernández Vázquez Daniela

N° de Cuenta: 318092867

GRUPO DE LABORATORIO: 01

GRUPO DE TEORÍA: 02

SEMESTRE 2024-2

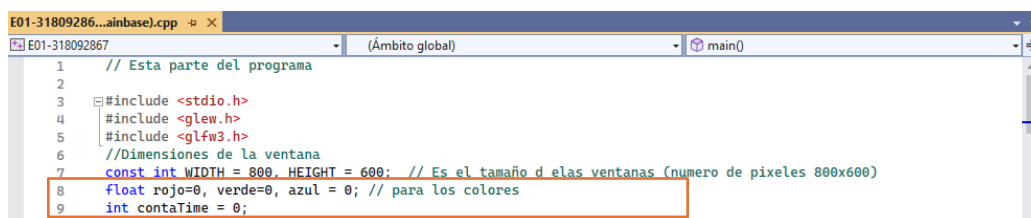
FECHA DE ENTREGA LÍMITE: 13 de febrero de 2024

CALIFICACIÓN: _____

EJERCICIOS DE SESIÓN:

Durante la sesión fueron revisados los diferentes archivos del proyecto proporcionado previamente en clase, centrándose principalmente en los dos *main*s principales y las funciones más importantes de cada uno línea por línea.

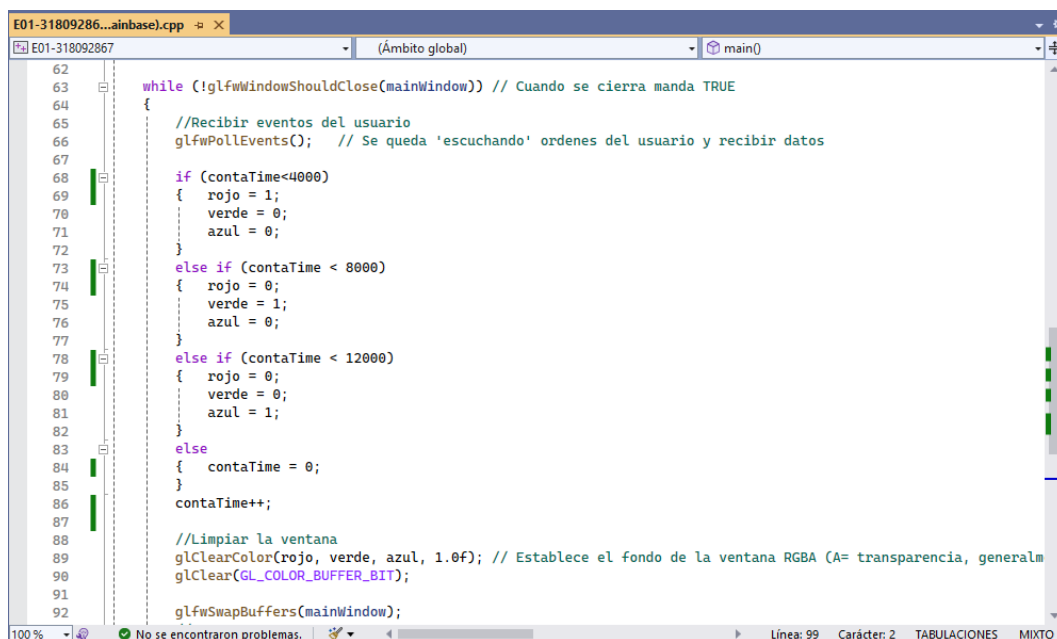
Como primer ejercicio y utilizando como base el archivo “mainbase.cpp”, el cual nos da un color base en la pantalla, se agregaron una serie de variables para manejar los parámetros de color RGBA, en este caso *rojo*, *verde* y *azul* inicializados en 0 con lo cual en un inicio se muestra la pantalla negra. Al momento que es cambiado un parámetro, la pantalla se muestra como una combinación de los colores activados.



```
1 // Esta parte del programa
2
3 #include <stdio.h>
4 #include <glw.h>
5 #include <glfw3.h>
6 //Dimensiones de la ventana
7 const int WIDTH = 800, HEIGHT = 600; // Es el tamaño d elas ventanas (numero de pixeles 800x600)
8 float rojo=0, verde=0, azul = 0; // para los colores
9 int contaTime = 0;
```

Figura 1. Variables agregadas

El primer ejercicio consistió en cambiar el color de fondo de la pantalla entre rojo, verde y azul de forma cíclica y solamente mostrando esos 3 colores con un periodo de lapso adecuado para el ojo humano. Esto se logró a través de las variables para manejar color anteriormente creadas, adicionalmente se coloca la variable *contaTime*, la cual es un contador que nos ayudara a contabilizar el número de ciclos de reloj más adelante.



```
62
63 while (!glfwWindowShouldClose(mainWindow)) // Cuando se cierra manda TRUE
64 {
65     //Recibir eventos del usuario
66     glfwPollEvents(); // Se queda 'escuchando' ordenes del usuario y recibir datos
67
68     if (contaTime<4000)
69     {
70         rojo = 1;
71         verde = 0;
72         azul = 0;
73     }
74     else if (contaTime < 8000)
75     {
76         rojo = 0;
77         verde = 1;
78         azul = 0;
79     }
80     else if (contaTime < 12000)
81     {
82         rojo = 0;
83         verde = 0;
84         azul = 1;
85     }
86     else
87     {
88         contaTime = 0;
89     }
90     contaTime++;
91
92     //Limpiar la ventana
93     glClearColor(rojo, verde, azul, 1.0f); // Establece el fondo de la ventana RGBA (A= transparencia, generalm
94     glClear(GL_COLOR_BUFFER_BIT);
95
96     glfwSwapBuffers(mainWindow);
```

Figura 2. Modificación al bloque main

Para lograr una secuencia cíclica y considerando que las instrucciones del *while* se ejecutaba cada ciclo de reloj, dentro de este se colocó una condición *if* que dependiendo del valor de *contaTime* reasignaba un nuevo valor a las variables *rojo*, *verde* y *azul* que determinaban el color de la pantalla. Con cada una de las opciones de cada caso la instrucción *glClearColor* recibía nuevos parámetros y así cambiaba el color.

Para hacer que cada cambio de color fuera más perceptible, se utilizó la variable *contaTime*, la cuál aumentaba con cada ciclo de reloj, y con este valor se evaluaba la condición *if*, y mientras se encontrara dentro de uno de los casos especificados para cada rango se modificaban las condiciones de color recibidas por *glClearColor*.

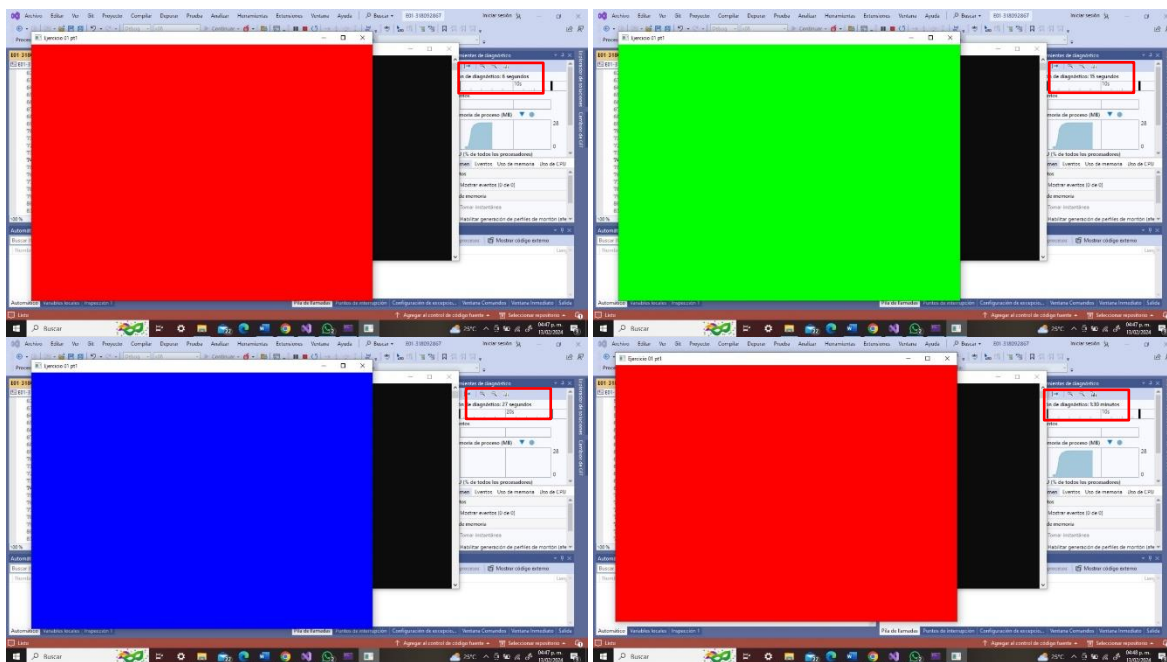


Figura 3. Ejecución del programa

En las imágenes anteriores podemos notar como la pantalla cambia de color cada cierto tiempo pasando de rojo a verde, de verde a azul y nuevamente a rojo.

Como segundo ejercicio se buscó dibujar de forma simultánea en la ventana un cuadrado y un rombo separados. Como base para este ejercicio se utilizó el *segundo_main.cpp* el cual nos mostraba un triángulo rojo dentro de toda la pantalla. La dimensiones de la pantalla en 2D se consideraban como de $x[-1,1]$ y $y[-1,1]$ lo cual dibuja nuestra superficie completa, es en este espacio dentro del cual se deben de colocar los vértices de los triángulos para realizar cada figura.

Dentro de la función *CrearTriangulo* es donde se crean las figuras en base a triángulos. Cada triángulo está formado por 9 datos agrupados en grupos de 3, que al ser leídos son considerados como coordenadas en el espacio x , y y z , por tanto

para crear tanto el cuadrado como el rombo se necesitan 2 triángulos por figura, cada triángulo conformado por sus vértices individuales. Se debió de tomar en cuenta que dos de los vértices de cada figura debían de coincidir para mostrar los triángulos sin separación.

```

31 void CrearTriangulo()
32 {
33     GLfloat vertices[] = { // son 9 datos en bloques de 3 // lee los datos y los transforma en un vertice x,y,
34         // Cuadrado
35         -0.75f, 0.75f, 0.0f,
36         -0.75f, 0.25f, 0.0f,
37         -0.25f, 0.75f, 0.0f,
38
39         -0.25f, 0.25f, 0.0f,
40         -0.75f, 0.25f, 0.0f,
41         -0.25f, 0.75f, 0.0f,
42
43         // Rombo //especificar coordenadas
44         0.5f, 0.75f, 0.0f,
45         0.25f, 0.5f, 0.0f,
46         0.75f, 0.5f, 0.0f,
47
48         0.5f, 0.25f, 0.0f,
49         0.25f, 0.5f, 0.0f,
50         0.75f, 0.5f, 0.0f,
51     };
52     glGenVertexArrays(1, &VAO); //generar 1 VAO
53     glBindVertexArray(VAO); //asignar VAO
54 }

```

Figura 4. Vértices para cada figura, se considera $z=0$.

Dentro de la función *main* y dentro del ciclo *while* se modificó la línea *glDrawArray* la cual recibe como parámetros el tipo de primitiva a dibujar, en este caso triángulos, el índice del vértice a dibujar y el número de vértices a dibujar, en un principio se mostraban solo 3, por lo cual se debe de modificar para que muestre 12 vértices.

```

178 //Loop mientras no se cierra la ventana
179 while (!glfwWindowShouldClose(mainWindow))
180 {
181     //Recibir eventos del usuario
182     glfwPollEvents();
183
184     //Limpiar la ventana
185     glClearColor(0.0f, 0.0f, 0.0f, 1.0f);
186     glClear(GL_COLOR_BUFFER_BIT);
187
188     glUseProgram(shader);
189
190     glBindVertexArray(VAO);
191     glDrawArrays(GL_TRIANGLES, 0, 12);
192     // (ARRIBA) Primitiva se puede modificar a LINES, TRIANGLES , POINTS
193     // // se modifica el índice para ver el numero de vertices
194     glBindVertexArray(0);
195
196     glUseProgram(0);
197
198     glfwSwapBuffers(mainWindow);
199 }
200

```

Figura 5. Modificación del programa.

Luego de especificar la posición de cada uno de los vértices de los triángulos a formar, podemos ver las figuras desplegadas en la pantalla.

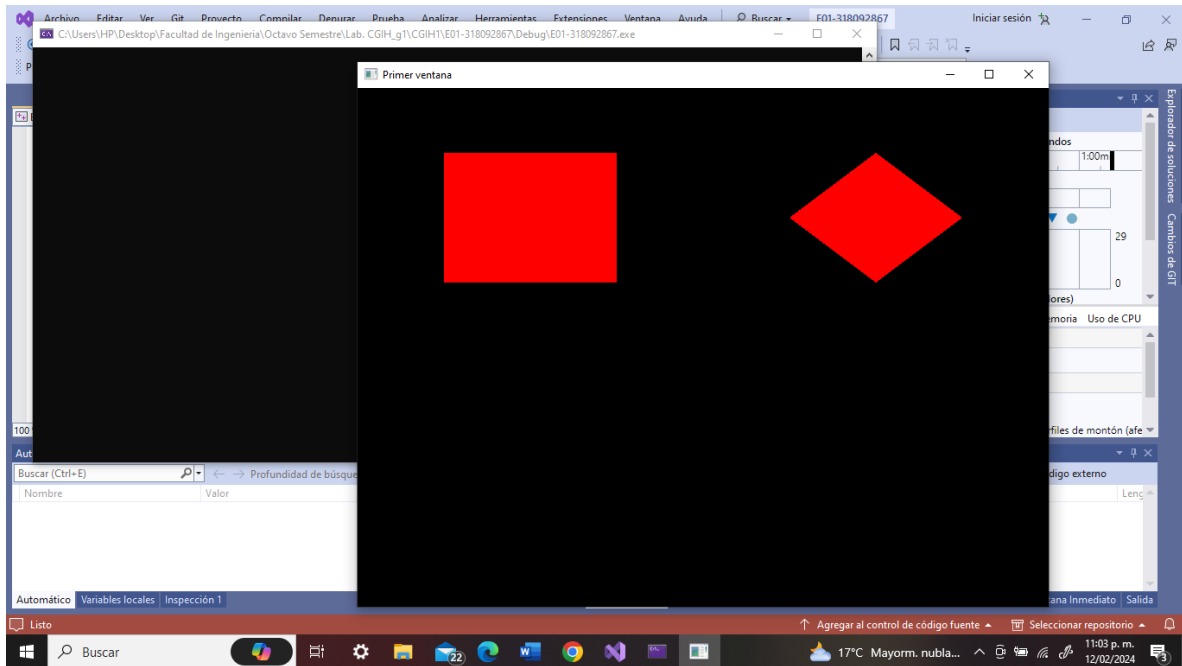


Figura 6. Ejecución del programa

CONCLUSIÓN:

En ninguno de los ejercicios se tuvo demasiada complejidad, sin embargo, el segundo ejercicio resultó, a mi parecer, menos complejo que el primero ya que no se necesitaba utilizar ninguna variable relacionada con el reloj. El mayor problema del primer ejercicio fue justamente integrar la variable de *contaTime* en cada ciclo de reloj, y en el segundo programa lo más complicado es descomponer las figuras en triángulos y ubicarlos en un plano cartesiano dentro de la pantalla como calores flotantes. La explicación fue rápida, a pesar de ello fue muy entendible.