



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: Barragán Pilar Diana

N° de Cuenta: 318147981

GRUPO DE LABORATORIO: 03

GRUPO DE TEORÍA: 04

SEMESTRE 2025-1

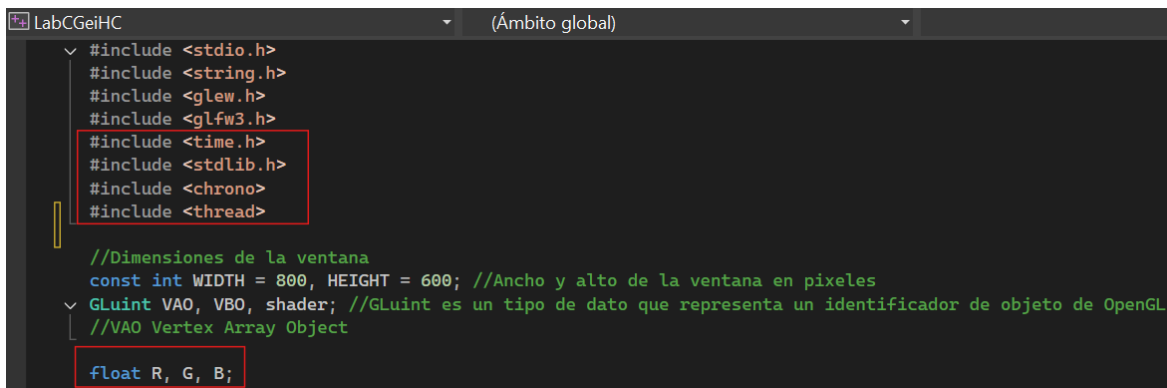
FECHA DE ENTREGA LÍMITE: 13 de agosto de 2023

CALIFICACIÓN: _____

Ejercicio de Clase 1: Introducción a OpenGL

1. **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.**

Para lograr que el fondo de la pantalla cambiara entre los colores rojo, verde y azul primero tuve que importar nuevas librerías para las funciones que iba a estar ocupando pertenecientes a estas. Las nuevas librerías que utilice fueron; `time.h`, `stdlib.h`, `chrono` y `thread`.



```
LabCGeiHC (Ámbito global)
#include <stdio.h>
#include <string.h>
#include <glw.h>
#include <glfw3.h>
#include <time.h>
#include <stdlib.h>
#include <chrono>
#include <thread>

//Dimensiones de la ventana
const int WIDTH = 800, HEIGHT = 600; //Ancho y alto de la ventana en pixeles
GLuint VAO, VBO, shader; //GLuint es un tipo de dato que representa un identificador de objeto de OpenGL
//VAO Vertex Array Object

float R, G, B;
```

Figura 1. Agregación de librerías y variables tipo `float`.

La librería `time.h` la utilice a la hora de generar un numero aleatorio para el uso de la función `time()`, la librería `stdlib.h` para `rand()` y `chrono` y `thread` para utilizar el `thread sleep`, con lo que detuve la ejecución por cierto tiempo en milisegundos.

Posteriormente inicialice las variables con el tipo de dato `float` para cambiar el color en este caso las nombre como `R`, `G`, `B`.

Para lograr que la pantalla cambiara de color utilice la función `srand(time(NULL));` para que me diera un número aleatorio, luego asigne a las variables `R`, `G`, `B` que había declarado anteriormente un valor que fuera de 1 o de 0, para ello al número que me daba `rand()` le aplique el módulo de dos.

Dado que me daba más combinaciones que no eran solo verde, azul y rojo puse una condición de `if` para que solo cambiará de color la pantalla cuando el código RGB cumpliera para estos colores.

Finalmente, con `sleep_for` detuve la ejecución para que el cambio de colores fuera perceptible para el ojo humano.

```
LabCGeiHC (Ámbito global) main()

//Crear triangulo
CrearTriangulo();
CompileShaders();

srand(time(NULL));

//Loop mientras no se cierra la ventana
while (!glfwWindowShouldClose(mainWindow))
{
    //Recibir eventos del usuario
    glfwPollEvents();

    R = rand() % 2;
    G = rand() % 2;
    B = rand() % 2;

    //Limpiar la ventana
    //glClearColor(0.0f,0.0f,0.0f,1.0f);
    if((R == 1.0f && G == 0.0f && B == 0.0f) || (R == 0.0f && G == 1.0f && B == 0.0f) ||
        (R == 0.0f && G == 0.0f && B == 1.0f)) {

        glClearColor(R, G, B, 1.0f);
    }

    std::this_thread::sleep_for(std::chrono::milliseconds(500));
    glClear(GL_COLOR_BUFFER_BIT);
}
```

Figura 2. Obtención del número aleatorio con valor 1 o 0. Asignación de valores en variables float. Filtro de colores y uso del sleep_for.

EJECUCIÓN DEL PROGRAMA:

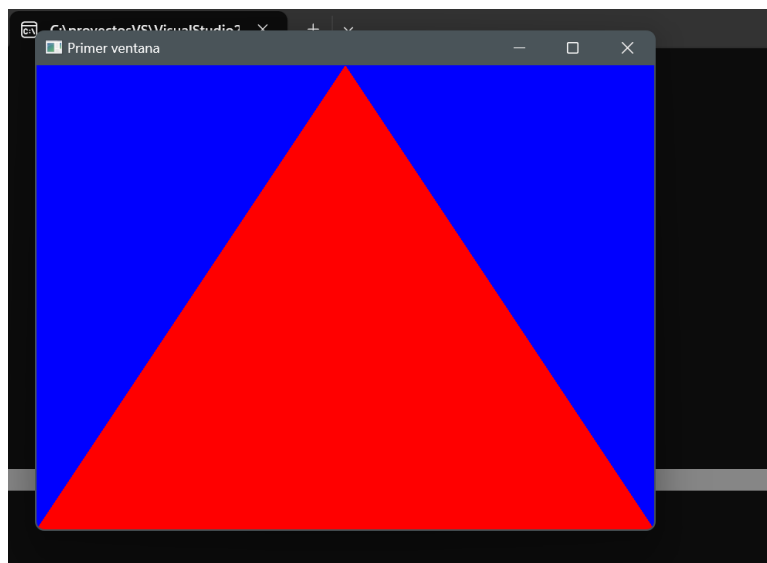


Figura 3. Ejecución pantalla color azul.

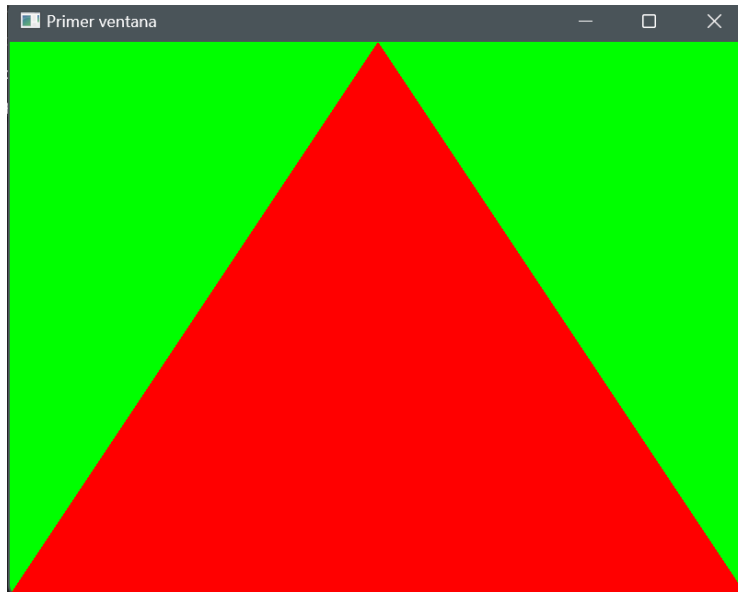


Figura 4. Ejecución pantalla color verde.

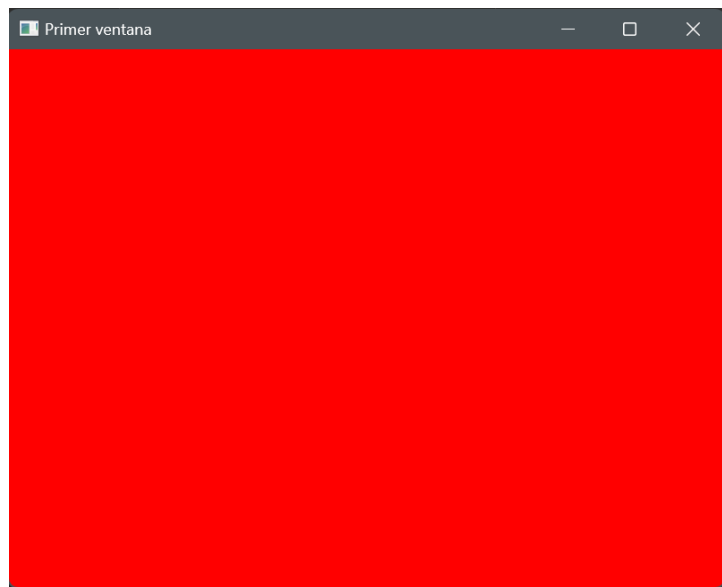


Figura 5. Ejecución pantalla color rojo (al ser rojo el objeto no se distingue).

2. Dibujar de forma simultánea en la ventana 1 cuadrado y 1 rombo separados.

En este segundo ejercicio para dibujar en la pantalla un cuadrado junto a un rombo lo primero que hice fue realizar un pequeño boceto de como debía ser el dibujo y que coordenadas serían las mejores para los vértices.

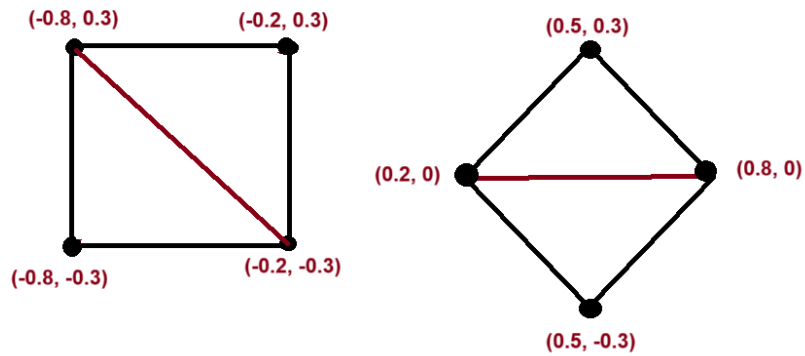


Figura 6. Boceto de figuras y coordenadas.

Para la modificación del programa, después de haber planteado como debería de quedar el dibujado, lo primero que modifique fue la matriz de vértices de los triángulos, dado que la pantalla es de 1x1 todos los valores que asigne son menores a 1 y mayores que -1.

Las primeras seis coordenadas en la matriz de vértices pertenecen al cuadrado, puesto que para formarlo utilice dos triángulos con tres coordenadas cada uno de ellos correspondientes a sus vértices. Para el rombo repetí el proceso que utilicé en el cuadrado solo que, rotando los vértices, las ultimas seis coordenadas son de los dos triángulos que conforman al rombo.

```

LabCGeIH C (Ámbito global)

void CrearTriangulo()
{
    GLfloat vertices[] = {
        -0.8f, 0.3f, 0.0f,
        -0.2f, 0.3f, 0.0f,
        -0.2f, -0.3f, 0.0f,

        -0.8f, 0.3f, 0.0f,
        -0.8f, -0.3f, 0.0f,
        -0.2f, -0.3f, 0.0f,

        0.2f, 0.0f, 0.0f,
        0.5f, 0.3f, 0.0f,
        0.8f, 0.0f, 0.0f,

        0.2f, 0.0f, 0.0f,
        0.5f, -0.3f, 0.0f,
        0.8f, 0.0f, 0.0f
    };

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

```

Figura 7. Modificación de los vértices para los 4 triángulos.

Por último, para que se dibujará correctamente cambie la función `glDrawArrays` en el ultimo campo puesto que al ser mayor cantidad de triángulos hay mayor cantidad

de vértices y en este caso utilice 12 vértices para generar las figuras del cuadrado y el rombo.

```
LabCGeiHC (Ámbito global)

    glClearColor(R, G, B, 1.0f);
}

std::this_thread::sleep_for(std::chrono::milliseconds(500));

glClear(GL_COLOR_BUFFER_BIT);

glUseProgram(shader);

glBindVertexArray(VAO);
glDrawArrays(GL_TRIANGLES, 0, 12);
glBindVertexArray(0);

glUseProgram(0);

glfwSwapBuffers(mainWindow);
```

Figura 8. Modificación del numero de vértices en la función `glDrawArrays`.

EJECUCIÓN DEL PROGRAMA:

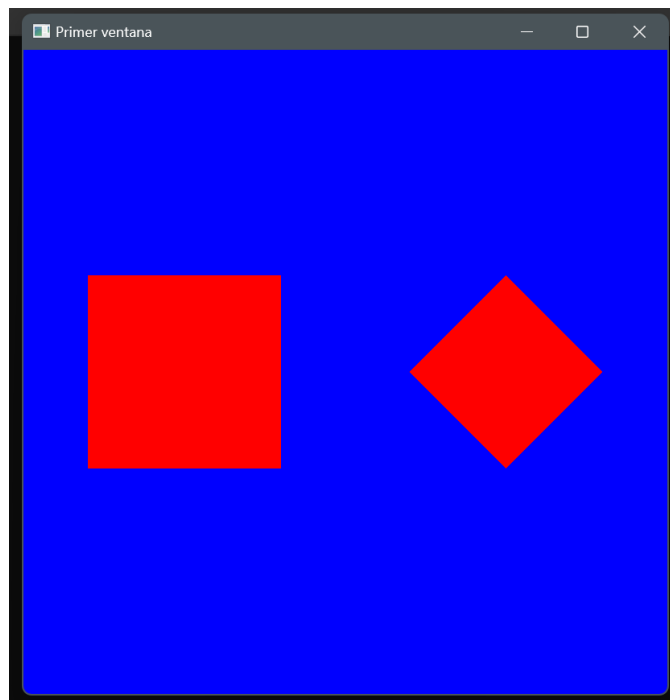


Figura 9. Pantalla azul con las figuras de cuadrado y rombo.

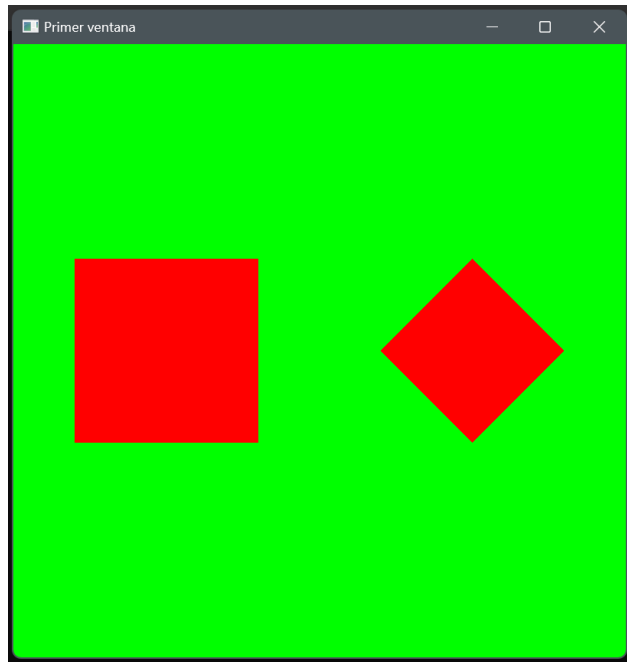


Figura 10. Pantalla verde con las figuras de cuadrado y rombo.

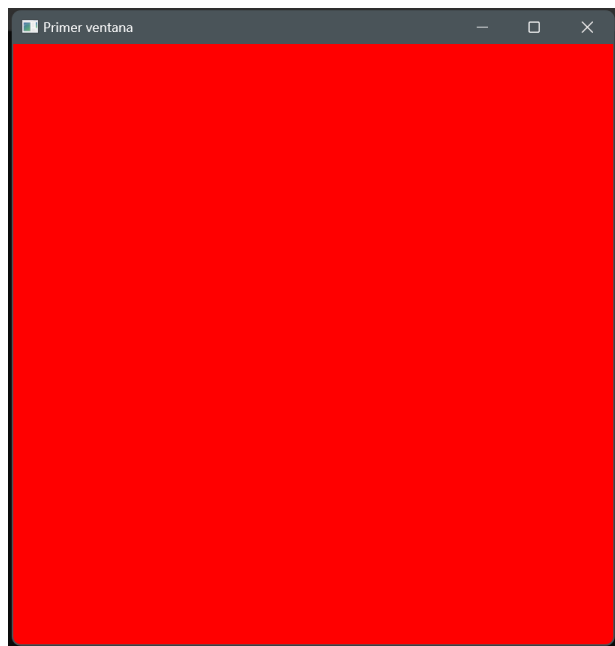


Figura 9. Pantalla roja con las figuras de cuadrado y rombo (al ser las figuras rojas no se distiguen).

En este primer ejercicio no tuve ningún inconveniente a la hora de compilar el código después de haberle realizado modificaciones, puesto que el programa se ejecutó como debía de ser.

El único inconveniente que se me presentó fue a la hora de intentar hacer el ejecutable pues este se crea, pero no me deja abrirlo porque no se encontró glew32.dll. Sin embargo, esto se solucionó al pasar el archivo glew32.dll que se encontraba en el zip a la carpeta donde está el ejecutable.

CONCLUSIONES:

Considero que los ejercicios que se dejaron para ir conociendo y familiarizándonos con el software, tanto de Visual Studio como de OpenGL son bastante buenos y al nivel de lo que vimos en la sesión, pues me ayudaron a entender mejor las primitivas especialmente la que se maneja en esta clase que fueron los triángulos. Si bien definitivamente entender la sintaxis y estructura de OpenGL es complicado y es necesario repasar el código para comprender todo lo que implican muchas de las funciones que son básicas y estarán presentes de forma inherente en todos nuestros futuros programas, la explicación con respecto a ellas fue clara y el tener que modificar el código facilita entenderlo con mayor profundidad.

De los ejercicios que realice sin duda alguna el que me causó más inconvenientes a la hora de escribir el código fue la primera actividad, ya que, él que solo debía cambiar a tres colores distintos la pantalla se me hizo algo difícil de implementar pues al ser al azar los números cero o uno, en algunas ocasiones la combinación de RGB se repetía, por lo que un color a veces tardaba más tiempo en cambiar.