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 Nº 1

NOMBRE COMPLETO: Uriarte Ortiz Enrique Yahir

Nº de Cuenta: 318234757

GRUPO DE LABORATORIO: 02

GRUPO DE TEORÍA: 04

SEMESTRE 2025-1

FECHA DE ENTREGA LÍMITE: Martes 13 de Agosto del 2024

CALIFICACIÓN: _____

EJERCICIOS DE SESIÓN:

1. Actividades realizadas.
   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 esta primera actividad cree 3 variables, “Color” es una variable para representar el color actual, ya sea rojo 0, verde 1 o azul 2; “Cuenta” es un contador de frames y “lapso_tiempo” es para controlar el tiempo que debe pasar para que se ejecute el cambio de color, en este caso se asigno que pasaran los 5000 frames. A la variable “Cuenta” haremos que se incremente en cada iteración del bucle.
   La condición “if (Cuenta >= lapso_tiempo)” comprueba si se alcanzó o supero el número de frames necesarios para cambiar el color, si es así, la línea de código “Color = (Color + 1) % 3” hará que el color se cambia incrementando en 1 a la variable “Color” y para asegurarse que el valor se quede entre 0 y 2, usamos el operador módulo (%). Al final se reinicia “Cuenta” en 0 para comenzar el conteo para el próximo cambio de color.
   Por último, para seleccionar los colores, empleamos un switch, que selecciona el color basado en el valor de “Color”, en cada caso empleamos “glClearColor” para seleccionar el color.

   Capturas de pantalla del código:

```
//Variables para el cambio de color
int Color = 0;
int Cuenta = 0;
const int lapso_tiempo = 5000;

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

    Cuenta++;

    //Cambiar el color si ha pasado el intervalo
    if (Cuenta >= lapso_tiempo){
        Color = (Color + 1)%3;
        Cuenta = 0; //Reiniciar el contador
    }
```

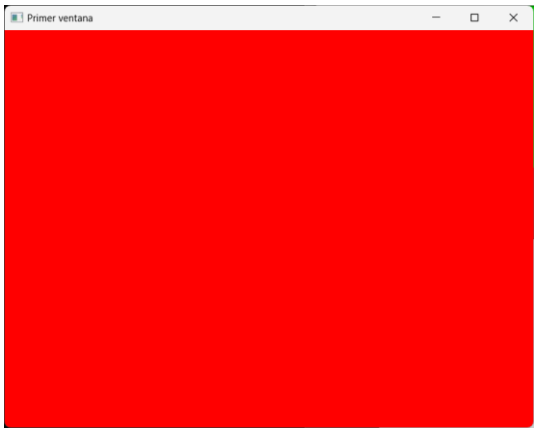
Fig. 1. Código para cambiar el color de fondo.

```
//Establecer el color de fondo según el color actual
switch (Color){
case 0: glClearColor(1.0f, 0.0f, 0.0f, 1.0f); break;//Rojo
case 1: glClearColor(0.0f, 1.0f, 0.0f, 1.0f); break;//Verde
case 2: glClearColor(0.0f, 0.0f, 1.0f, 1.0f); break;//Azul
}

//Limpiar la ventana
glClear(GL_COLOR_BUFFER_BIT);

glUseProgram(shader);
```

Fig. 2. Código para cambiar el color de fondo.



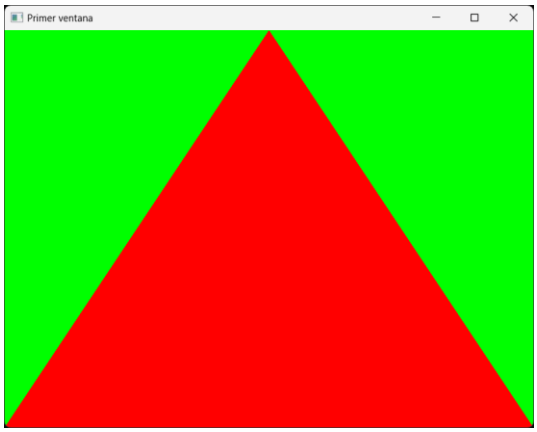


Fig. 3. y Fig. 4. Ejecución del programa.

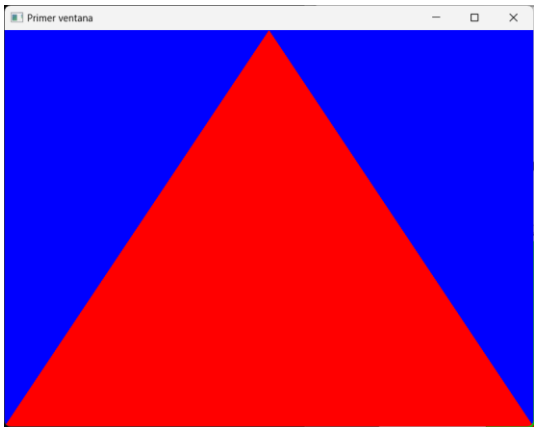


Fig. 5. Ejecución del programa.

2) *Dibujar de forma simultánea en la ventana 1 cuadrado y 1 rombo separados.* Para realizar las figuras utilice dos triángulos en cada uno, tomando como ejemplo las coordenadas que utilizamos para el triángulo original, imagine dividir en 20 coordenadas la ventana y realizando unos dibujos en boceto de cómo se vería la ventana, hice un cálculo aproximado de dónde colocar las coordenadas.

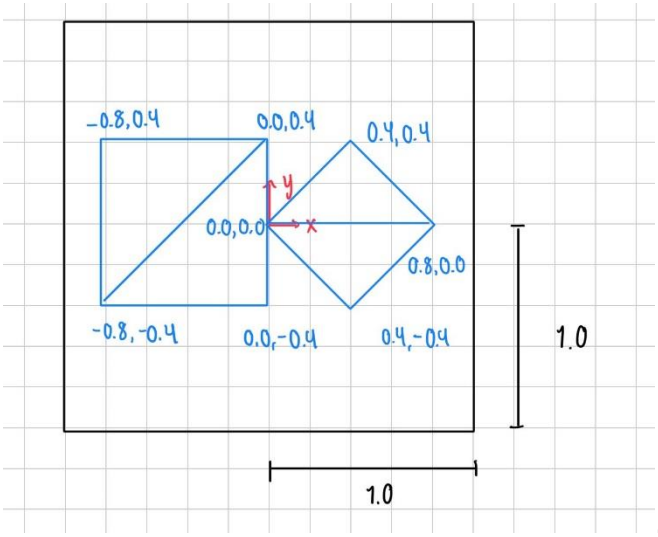


Fig. 6. Boceto para las coordenadas de los triángulos.

Sin embargo, al momento de ejecutarlo me percaté que no tenía espacio entre el cuadrado y el rombo, para que se viera mejor decidí agregarle un espacio de separación. Aparte a pesar de que según las coordenadas que coloque la distancia entre los lados del cuadrado eran los iguales, al momento de ejecutarlo la figura más bien parecía un rectángulo por lo que también agregue una décima en el eje y del cuadrado para que pareciese mas al momento de ejecutarlo.

Capturas de pantalla del código:

```
void CrearTriangulo()
{
    GLfloat vertices[] = {
        -0.9f,-0.5f,0.0f,
        -0.1f,-0.5f,0.0f,
        -0.1f,0.5f,0.0f,

        -0.9f,-0.5f,0.0f,
        -0.9f,0.5f,0.0f,
        -0.1f,0.5f,0.0f,
```

Fig. 7. Coordenadas de triángulos para formar el cuadrado.

```
        0.0f,0.0f,0.0f,
        0.4f,0.5f,0.0f,
        0.8f,0.0f,0.0f,

        0.0f,0.0f,0.0f,
        0.4f,-0.5f,0.0f,
        0.8f,0.0f,0.0f
    };
    glGenVertexArrays(1, &VAO); //generar 1 VAO
    glBindVertexArray(VAO);//asignar VAO
```

Fig. 8. Coordenadas de triángulos para formar el rombo.

Otro cambio que realice fue cambiar el número de vértices que se van a dibujar, ya que en total tendremos 4 triángulos.

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

Fig. 9. Coordenadas de triángulos para formar el cuadrado.

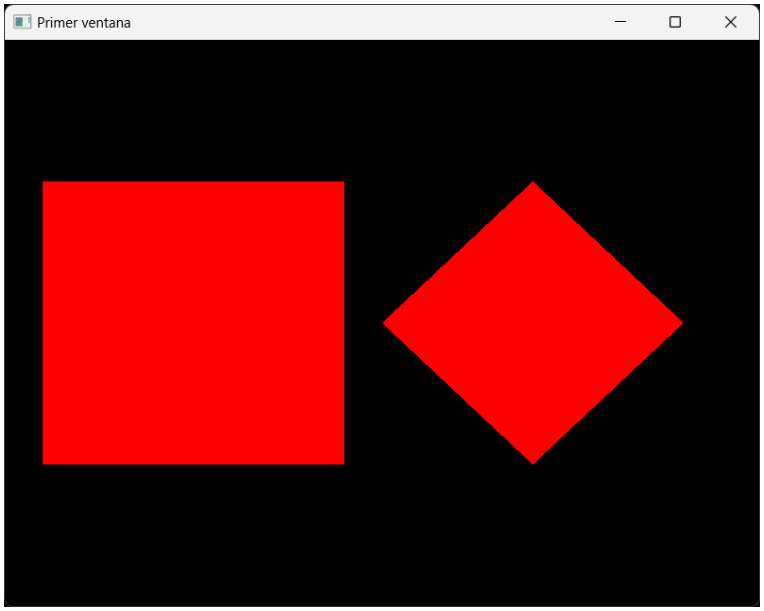
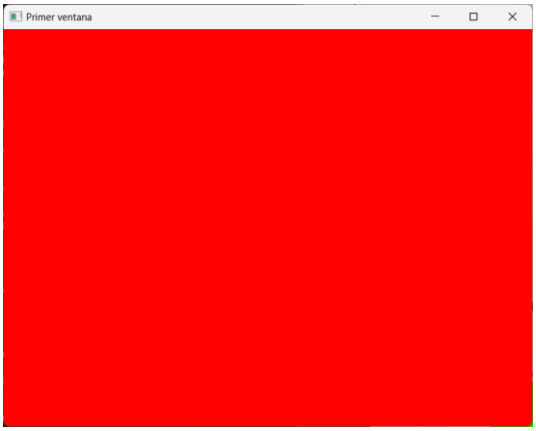


Fig. 10. Ejecución del programa.

Combinando las dos partes del ejercicio, la ejecución quedaría de la siguiente manera:



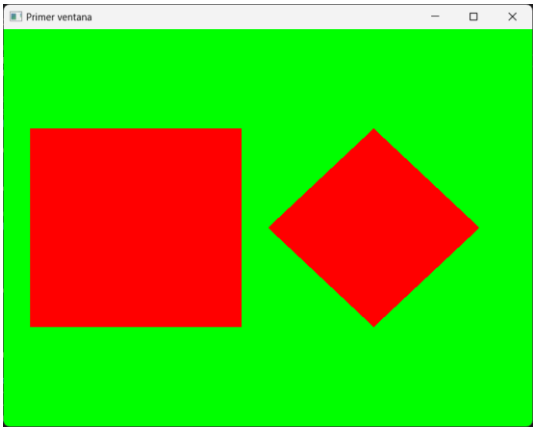


Fig. 11. y Fig. 12. Ejecución del programa.

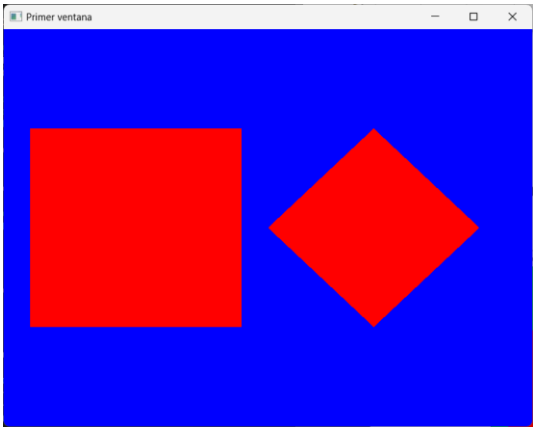


Fig. 13. Ejecución del programa.

2. *Problemas presentados.*
   En el primer ejercicio los problemas que tuve fueron encontrar la cantidad de frames adecuados para visualizar los colores bien, ya que en un inicio elegí darle 60 frames y la duración de los colores era mínima, por lo que después de dos ejecuciones mas ajuste mejor el tiempo.
   En el segundo ejercicio los problemas presentados fueron con los signos de los números para las coordenadas, ya que se me fueron algunos signos negativos.

3. *Conclusión:*
   Después de realizar los ejercicios propuestos, considero que la complejidad que tenían era normal y fueron una buena forma de recordar algunos conceptos de programación en el lenguaje C++ que había olvidado desde la ultima vez que los emplee en fundamentos de programación, lo mas complicado fue la asignación de las coordenadas de los puntos para las figuras, pero una vez plasmadas en un dibujo para analizar el espacio en la pantalla, la asignación fue más fácil.

Considero que al final falto explicar para estos primeros ejercicios de clase que ambos debían venir en un mismo archivo main, ya que había entendido que era cada ejercicio por archivo separado. Considero que la practica tuvo un buen nivel.