Previo 2: Dibujo de primitivas en 2D
Fecha de entrega: 19 de agosto del 2025

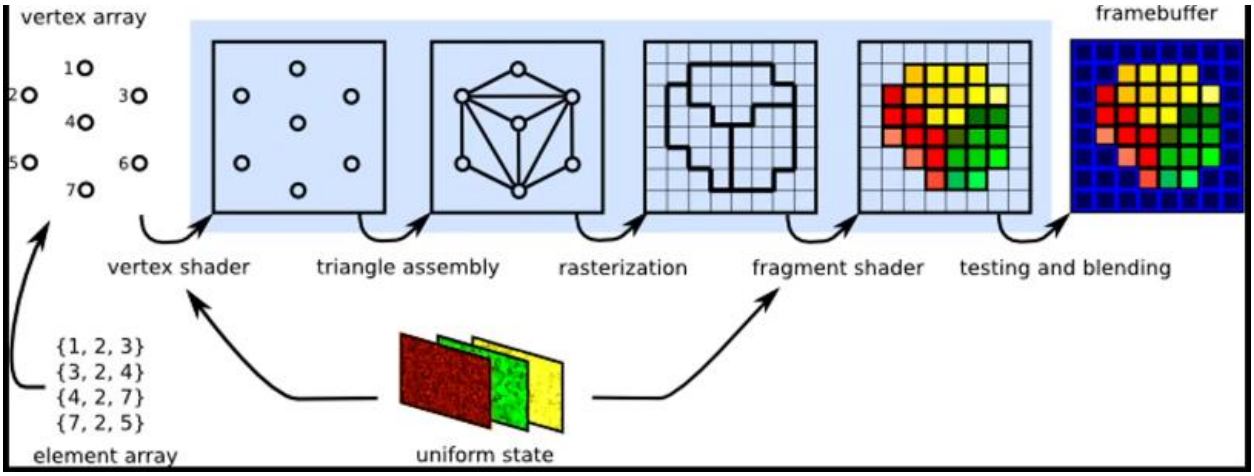
Alumna: Plata Alvarez Dayana
No. Cuenta: 318300764

Introducción

En este material vamos a aprender a como dibujar primitivas en 2D utilizando shaders en Open GL, exploraremos el código base y entenderemos como cada parte contribuye a la renderización final en pantalla y de igual manera vamos a ver como modificar el proyecto de la sesión anterior, para poder integrar estos nuevos archivos.

Desarrollo

Es importante repasar el Pipeline de renderizado en Open GL



Los datos provienen de los vértices, que pasan primero al shader de vértices y luego se ensamblan para pasar a la etapa de rastirizacion y posteriormente al shader de fragmentos posteriormente estos entran en el shade bufer y de ahí llegamos a renderizar las primitivas en 2D.

Agregamos el nuevo código base y una librería que es el shader en nuestro proyecto de Visual Studio.

Ahora agregamos los shaders como elementos por separado para tener un mejor orden y estructura al momento de generar nuestros códigos.

Tenemos las bibliotecas que vamos a estar ocupando

```
#include <GL/glew.h>

#include <GLFW/glfw3.h>

// Shaders
```

Alumna: Plata Alvarez Dayana
No. Cuenta: 318300764

Tenemos aquí la creación de la ventana, el tamaño que vamos a ocupar

```
void resize(GLFWwindow* window, int width, int height);

const GLint WIDTH = 800, HEIGHT = 600;
```

Y el nombre que va a tener la ventana dibujo de primitivas 2D, posteriormente inicializamos la parte de las librerías para la creación de la ventana

```
//Verificación de errores de creacion  ventana
if (window== NULL)
{
    std::cout << "Failed to create GLFW window" << std::endl;
    glfwTerminate();

    return EXIT_FAILURE;
}

glfwMakeContextCurrent(window);
glewExperimental = GL_TRUE;
```

Aquí tenemos información referente al software grafico que estamos ocupando.

Uno de los cambios significativos es el siguiente:

```
Shader ourShader("Shader/core.vs", "Shader/core.frag");
```

Aquí lo que hacemos es cargar la información de los archivos que tenemos por separado.

Colocamos una variable tipo shader una variable que va a tener el shader y la ruta en donde se encuentran los archivos.

Posteriormente tenemos la definición de los vértices que estaremos ocupando, esto nos va a permitir trabajar con la información que vamos a utilizar.

Aquí tenemos la posición en 3 coordenadas x,y,z.

```
// Set up vertex data (and buffer(s)) and attribute pointers
float vertices[] = {
    0.5f,  0.5f, 0.0f,    1.0f,0.0f,0.0f,  // top right
    0.5f, -0.5f, 0.0f,    1.0f,1.0f,0.0f,  // bottom right
   -0.5f, -0.5f, 0.0f,    1.0f,0.0f,1.0f,  // bottom left
   -0.5f,  0.5f, 0.0f,    1.0f,1.0f,0.0f, // top left
};
```

Pero en este caso, como solo vamos a trabajar en 2 dimensiones dejamos en 0 la componente z que representa la componente de la profundidad.

A agregamos el color que van a tener cada una de la información extra o atributos/vértices y están en el modelo de color RGB, cada una de las componentes que van de 0 a 1.

Y aquí están definidos cada uno de los vértices que vamos a tener, en este caso que seria definida como superior derecha/izquierda, la inferior derecha/izquierda.

Posteriormente tenemos los índices que tienen cada uno de los vértices los vamos a ocupar para dibujar cada uno de estos vértices.

```
};
unsigned int indices[] = {  // note that we start from 0!
    3,2,1,// second Triangle
    0,1,3,
```

Cargamos la información de los vértices a los bufers, para poderlos enlazar y posteriormente enviarlos al shader.

Frames por segundo, son los ciclos o velocidades es lo que vamos a estar haciendo para estas primitivas básicas

Las primitivas básicas

1.-Limpiar el bufer de color

2.-Dibujar cada uno de los elementos

```
while (!glfwWindowShouldClose(window))
{
    // Check if any events have been activiated (key pressed, mouse moved etc.) and call corresponding respons
    glfwPollEvents();

    // Render
    // Clear the colorbuffer
    glClearColor(0.0f, 0.0f, 0.0f, 1.0f);
    glClear(GL_COLOR_BUFFER_BIT);


    // Draw our first triangle
    ourShader.Use();
    glBindVertexArray(VAO);


    glPointSize(1);
    glDrawArrays(GL_POINTS,0,1);

    //glDrawArrays(GL_LINES,0,2);
    //glDrawArrays(GL_LINE_LOOP,0,4);

    //glDrawArrays(GL_TRIANGLES,0,3);
    //glDrawElements(GL_TRIANGLES, 3,GL_UNSIGNED_INT,0);
```

Alumna: Plata Alvarez Dayana
No. Cuenta: 318300764

¿Cómo funciona la comunicación entre shaders?

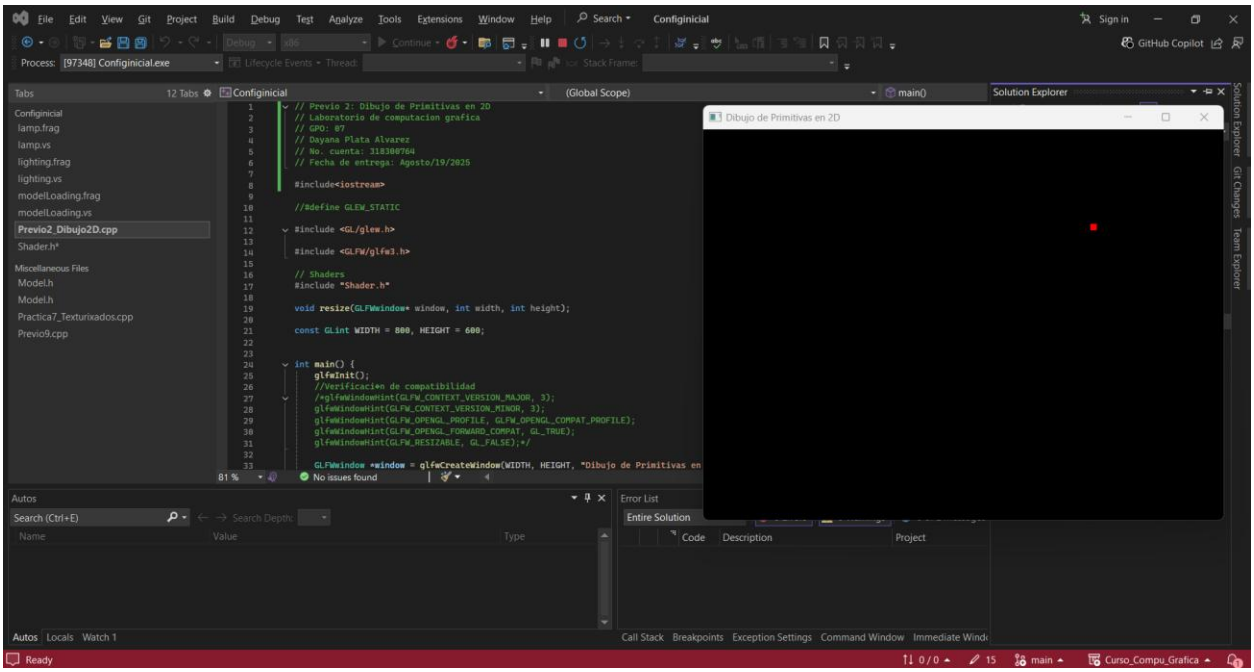
Primero se manda a llamar el shader de vértices, lo que nosotros hacemos es tener definido la localidad de memoria en donde se van a estar almacenando cada uno de estos elementos.

Dibujo

Lo más básico que vamos a poder dibujar es un punto.

Si ejecutamos este código, podemos observar que tenemos una pantalla con fondo negro, y hay un pequeño vertice de color rojo.

Si modificamos el tamaño ya podemos observar el punto rojo.



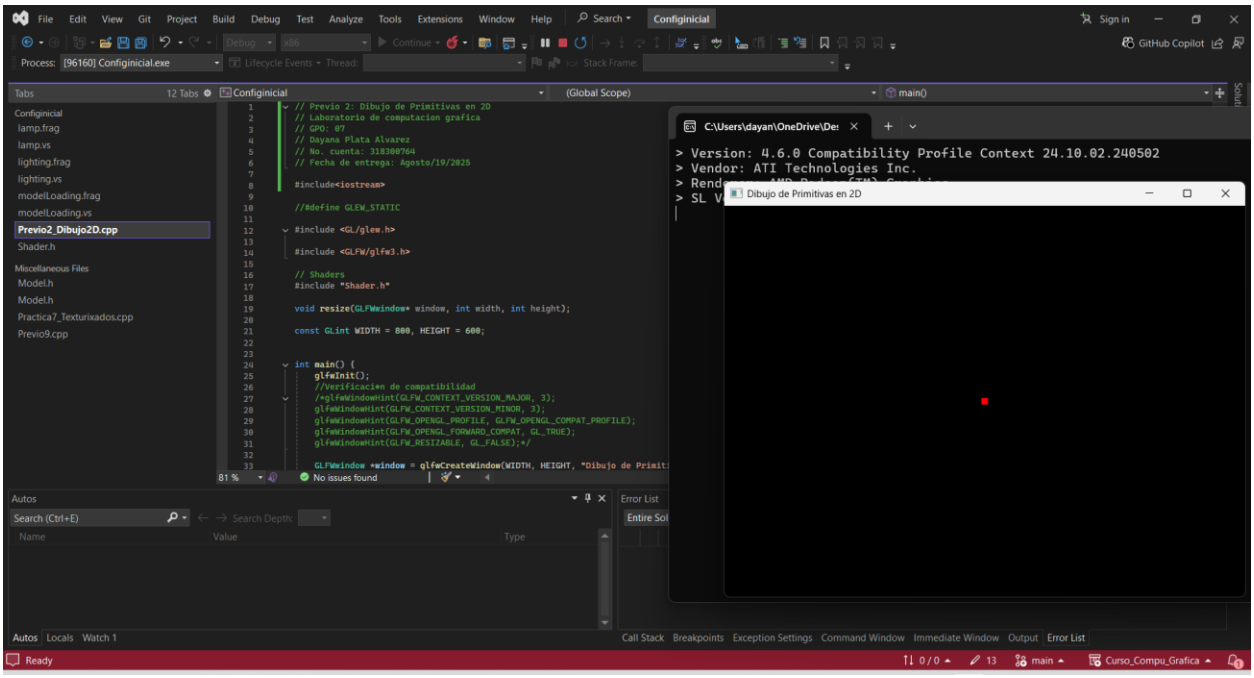
La ventana va de -1 a 1 en el caso de la componente horizontal/vertical.

Si quisiéramos tener el vertice en el origen, cambiamos los parámetros del vertice:

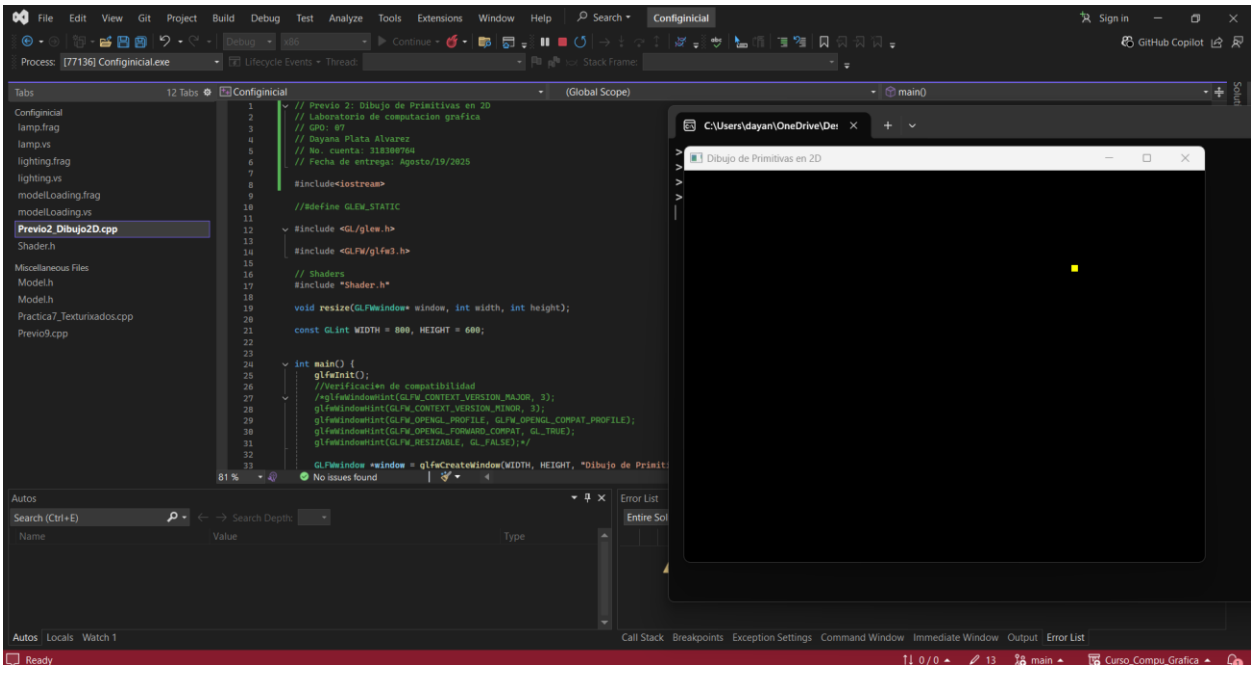
```
// Set up vertex data (and buffer(s)) and attribute pointers
float vertices[] = {
    0.0f,  0.0f, 0.0f,    1.0f,0.0f,0.0f,  // top right
    0.5f, -0.5f, 0.0f,    1.0f,1.0f,0.0f,  // bottom right
   -0.5f, -0.5f, 0.0f,    1.0f,0.0f,1.0f,  // bottom left
   -0.5f,  0.5f, 0.0f,    1.0f,1.0f,0.0f, // top left
```

Alumna: Plata Alvarez Dayana
No. Cuenta: 318300764

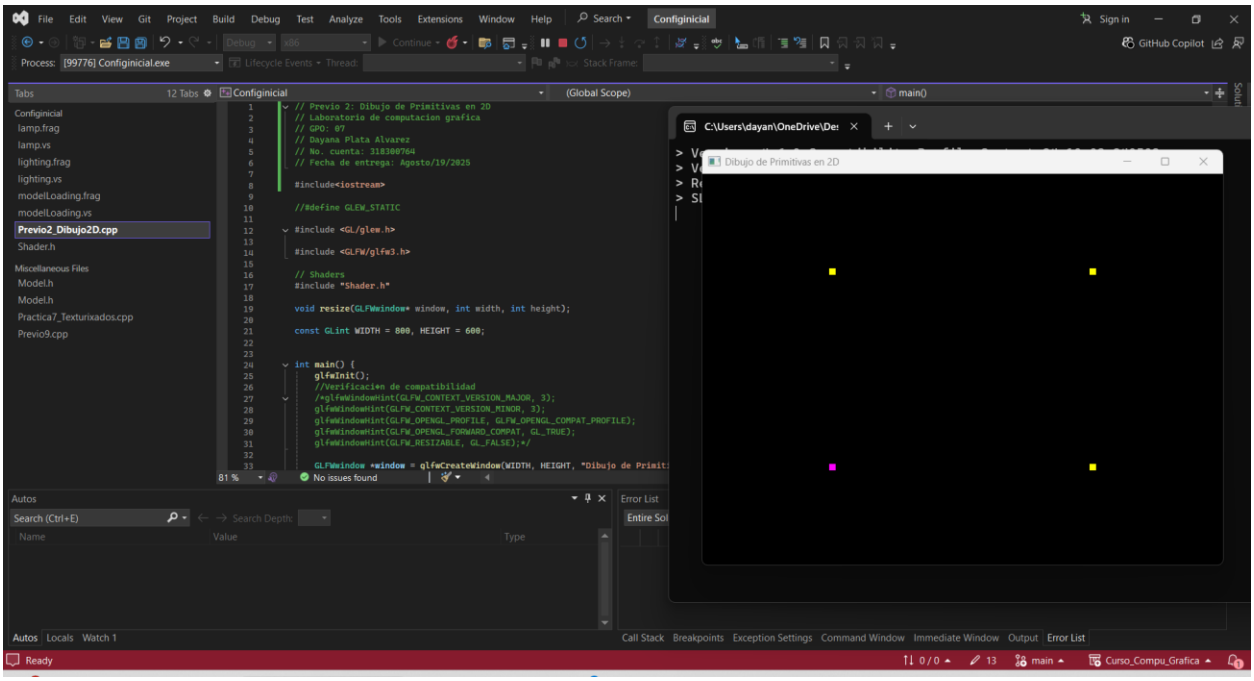
Aquí podemos observar que lo estamos dibujando en el centro de la ventana



Para cambiar los colores

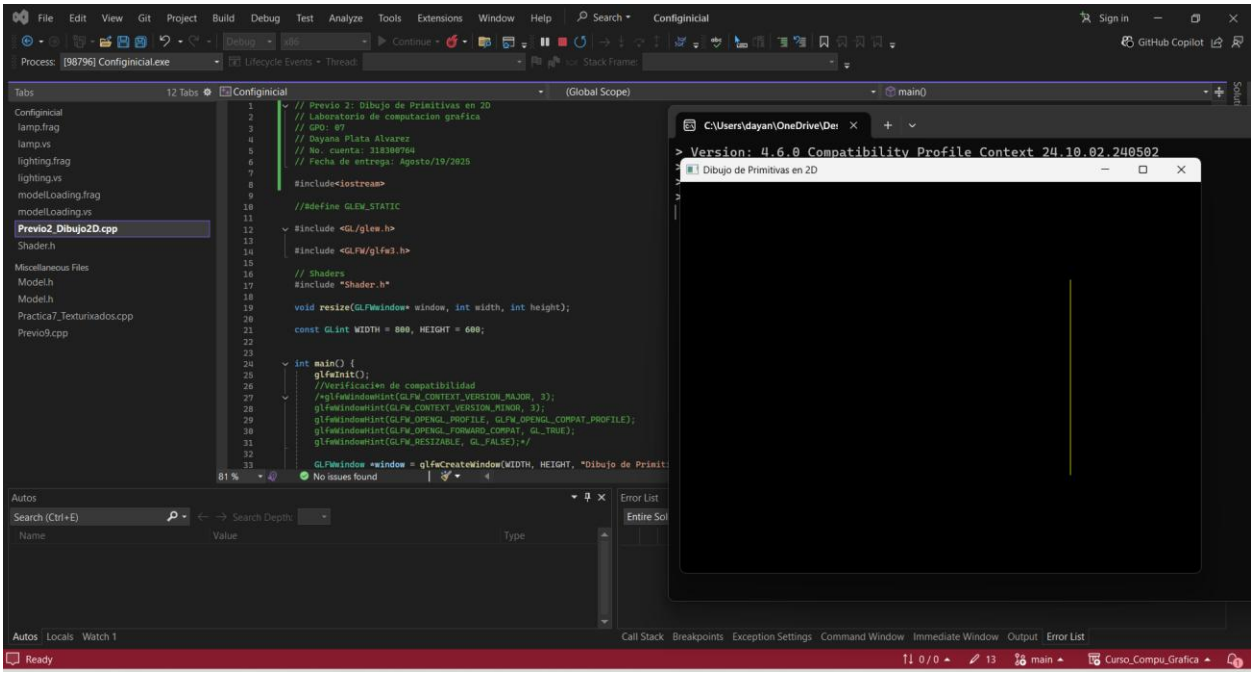


Ahora para tomar los 4 elementos modificamos los parámetros y obtenemos lo siguiente:

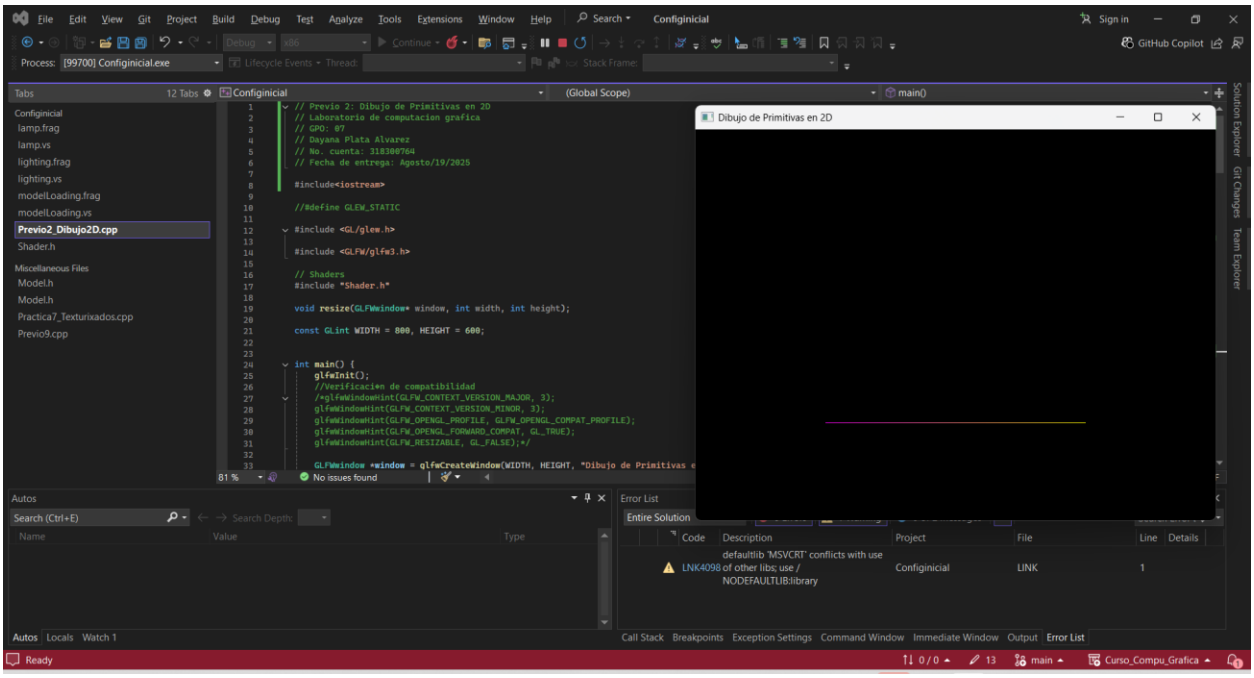


Y ya obtenemos los 4 vértices de la ventana, Posteriormente lo que podemos dibujar como primitivas básicas son las líneas.

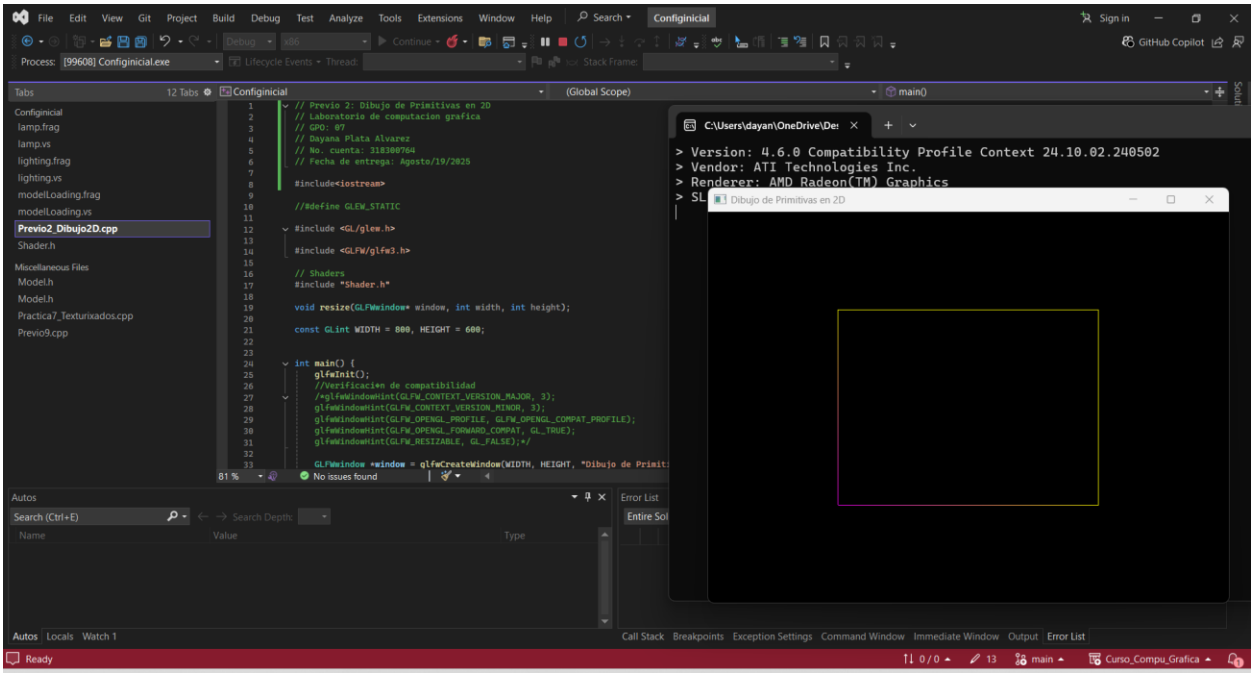
Aquí ya estamos uniendo las 2 aristas



Agregamos otra línea, que seria el siguiente elemento de mi arreglo.



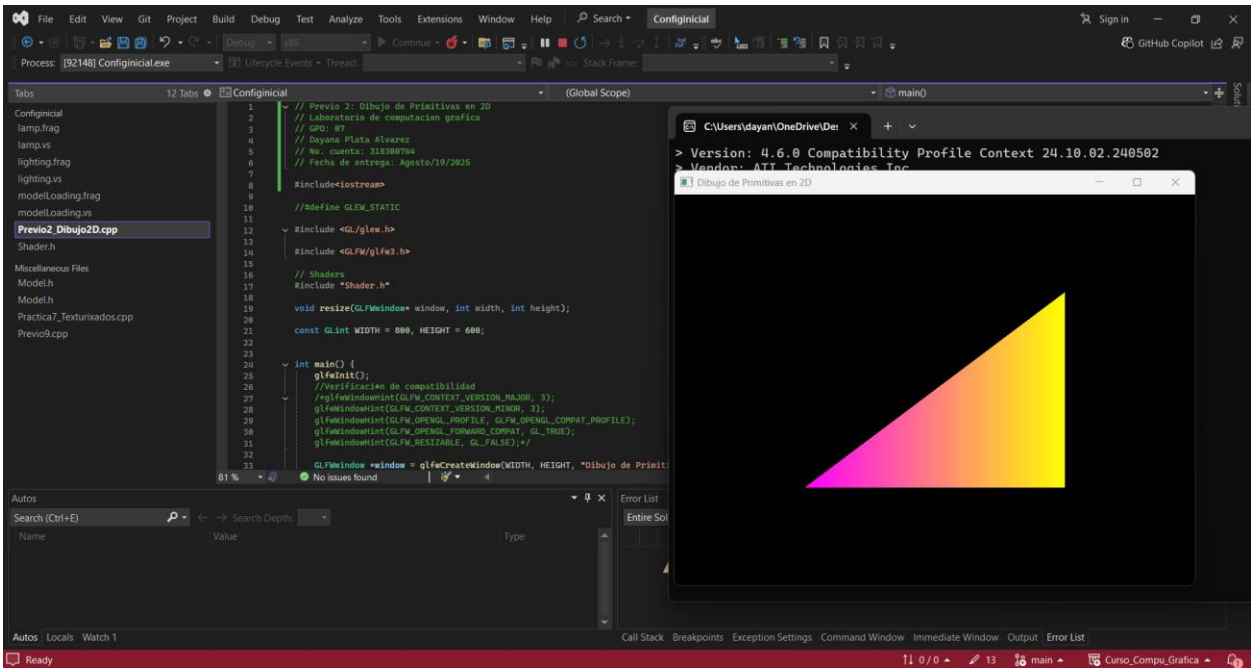
Podemos ir manipulando parte por parte, pero si ahora queremos dibujar el cuadrado completo, tenemos otra instrucción.



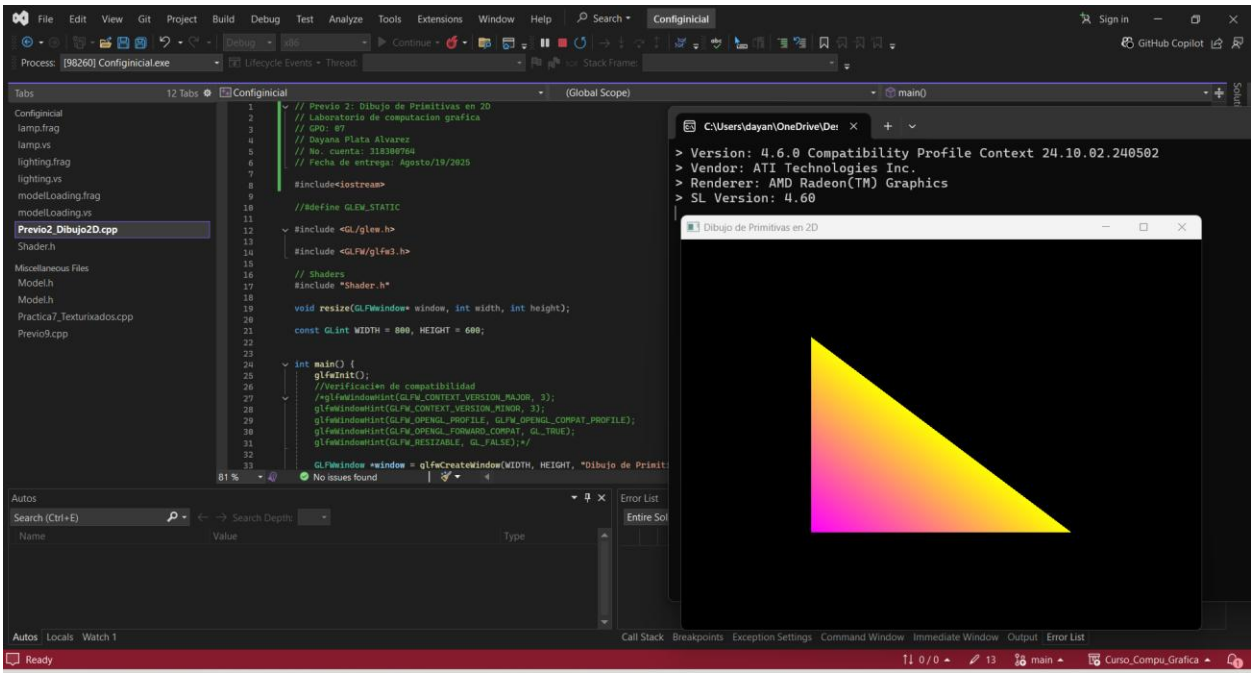
Esta instrucción, permite poder cerrar cada uno de estos vértices por medio de las líneas.

Alumna: Plata Alvarez Dayana
No. Cuenta: 318300764

La siguiente instrucción, es para dibujar triángulos.



Esta instrucción lo que hace es tomar del arreglo el primer elemento y los 3 vértices posteriores a ese arreglo y podemos ir modificando cada uno de los argumentos para obtener las variantes que pueda ir creando.

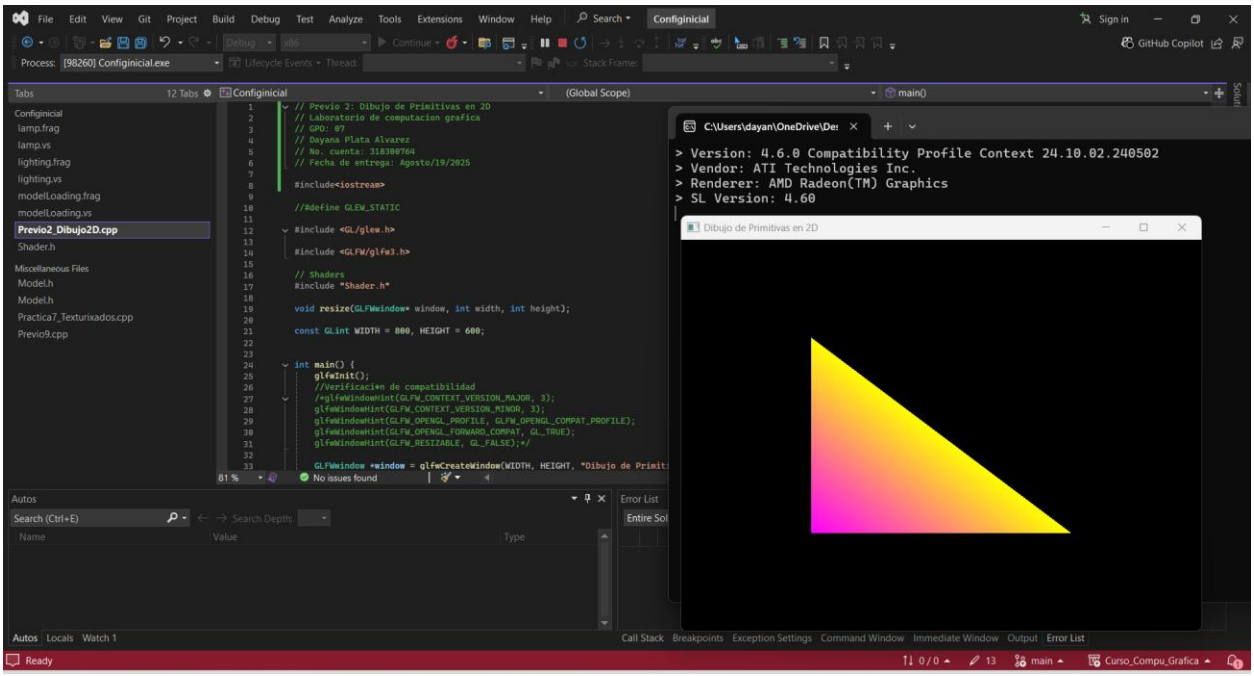


Alumna: Plata Alvarez Dayana
No. Cuenta: 318300764

Que pasa si no quiero utilizar precisamente el orden en como tengo definido el arreglo, si ya tenemos definidos unos índices, podemos invertir el orden.

Para eso podemos utilizar nuestra última función, que en lugar de dibujar el arreglo directamente, dibuja los elementos que se encuentran en el arreglo, por medio de los índices.

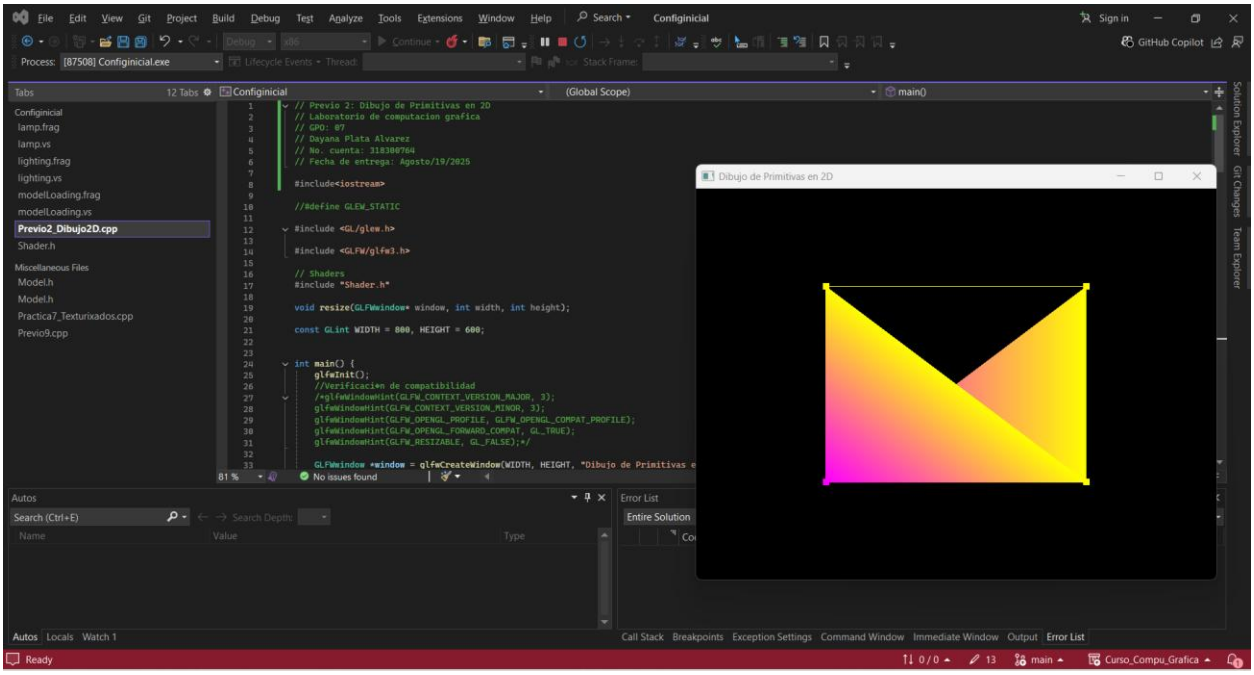
En este caso indicamos que queremos dibujar un triangulo y tomamos 3 elementos no asignados que son enteres y el ultimo digito nos indica a partir de donde vamos a comenzar a leer los índices.



Aquí obtenemos un orden diferente en el cual estamos dibujando nuestros triángulos.

Si descomentamos todas las funciones obtenemos:

Alumna: Plata Alvarez Dayana
No. Cuenta: 318300764



Conclusión

En esta práctica comprendimos cómo manipular los vértices y las líneas mediante la modificación de los valores en el arreglo, y cómo es posible alterar el orden de ejecución invirtiendo la secuencia de estos elementos.

Referencias:

NTecnologías Interactivas y Computación Gráfica. (2024, 9 de agosto). Dibujo de Primitivas en 2D con Shaders en OpenGL [Video]. YouTube.

https://www.youtube.com/watch?v=UWcnvhSDYYA

GitHub:

https://github.com/dayanapa45/Curso_Compu_Grafica