# Información de la práctica

|  |  |  |  |
| --- | --- | --- | --- |
| **Nombre del alumno:** | González Casanova Gallegos Renato Alfonso | | |
| **Número de Cuenta:** | 311158894 | **Práctica número:** | 1 |
| **Título de la práctica:** | Introducción a OpenGL | | |
| **Fecha de entrega:** | 21 febrero 2025 | | |

**Introducción:** (descripción inicial sobre la práctica)

En esta práctica, se trabajará con las primitivas para una primera aproximación a la tecnología de OpenGL y al uso de las bibliotecas adicionales GLEW y GLFW. El objetivo es dar los primeros pasos en la computación gráfica, al igual que un primer acercamiento al pipeline de renderizado, con sus diferentes fases.

Como es sabido, existen cuatro primitivas básicas en OpenGL: los vértices, las líneas y los triángulos, cada una con un comando específico en el lenguaje de programación C++. A lo largo de esta práctica, se realizarán diversos ejercicios para dominar las diferentes implementaciones de estos comandos.

**Resumen Teórico:** (introducción sobre los fundamentos teóricos en los que se basa la práctica y cuáles son los objetivos teóricos que persigue.)

Los fundamentos teóricos en los que se basa esta práctica están principalmente relacionados con el *pipeline* de renderizado, el cual consta de varias etapas que se abordarán a lo largo del curso. En esta práctica, se hará énfasis en la fase inicial de este proceso, trabajando con los vértices y las primitivas. Posteriormente, el uso de los distintos métodos disponibles en las bibliotecas mencionadas permitirá orquestar correctamente todo el flujo, desde la manipulación de los vértices hasta la visualización bidimensional en pantalla.

Dependiendo del autor, sabemos que el pipeline de renderizado tiene diferentes etapas, pero todas comparten el mismo objetivo final. En este momento, estamos trabajando con la fase inicial, que es el envío de datos, donde se definen los vértices que se almacenan en los buffers de la GPU. Por ahora, no abordaremos un procesamiento de vértices más complejo, pero sí hemos llegado a la tercera etapa, que es el ensamblado de primitivas, así como a las siguientes fases necesarias para alcanzar el resultado final.

**Descripción de la práctica:** (En esta sección se deben describir todos los pasos ejecutados durante la sesión práctica realizada en el laboratorio.)

A lo largo de la práctica realizamos diversas actividades, siendo la más importante la configuración inicial del entorno de desarrollo para utilizar las bibliotecas y documentos anexados en Visual Studio. Una vez superada esta etapa, exploramos las distintas funciones del código, identificando cómo interactúan entre sí. Desde los “*shaders”* hasta la verificación del tamaño de la pantalla, revisamos posibles errores en la ventana o fallos en la inicialización de bibliotecas.

En el núcleo del código principal, encontramos un arreglo de números flotantes que define los vértices y sus respectivos índices. Este enfoque nos permite manipular la construcción de figuras sin seguir estrictamente el orden original de declaración. Posteriormente, estos datos se almacenan en los buffers y son procesados por OpenGL mediante las bibliotecas GLEW y GLFW.

Uno de los aspectos clave comprendidos en esta práctica fue el uso de un bucle infinito (while), diseñado para ejecutar la renderización 60 veces por segundo, permitiéndonos visualizar el resultado en tiempo real. Dentro de este ciclo, empleamos diferentes comandos para dibujar vértices y primitivas gráficas.

A continuación, se listan algunas de las primitivas básicas utilizadas junto con una breve descripción de su función:

* glPointSize(10);
  + Define el tamaño de los píxeles en 10 unidades.
* glDrawArrays(GL\_POINTS, 4, 4);
  + Dibuja los vértices como puntos en la pantalla.
* glDrawArrays(GL\_LINES, 8, 4);
  + Dibuja líneas abiertas, sin formar figuras cerradas.
* glDrawArrays(GL\_LINE\_LOOP, 12, 4);
  + Dibuja líneas conectadas en forma de un bucle cerrado.
* glDrawArrays(GL\_TRIANGLES, 4, 3);
  + Renderiza triángulos utilizando los vértices definidos.
* glDrawElements(GL\_TRIANGLES, 3, GL\_UNSIGNED\_INT, (void\*)(24));
  + Dibuja un triángulo basado en índices predefinidos en el *buffer*, permitiendo un mayor control sobre la geometría.

**Resultados:** (explicar lo que se logró o aquello que no lograron realizar o comprender)

A different colored symbols on a black background

AI-generated content may be incorrect.

Figura 1. Todos los comandos utilizados al mismo tiempo.

A group of colorful symbols

AI-generated content may be incorrect.

Figura 2. Es el resultado de OpenGL del programa, con numeración para explicar los resultados.

Como se puede observar en esta pantalla, se muestran cuatro figuras. Estas figuras se generan a partir de los cuatro vértices originales proporcionados por el profesor, y a partir de ellos se añadieron cuatro vértices adicionales por cada uno, sumando un total de 16 vértices adicionales. El objetivo de este procedimiento fue practicar y visualizar todo el trabajo en un mismo espacio.

Además de la adición de vértices, también se incorporaron índices para experimentar con diferentes configuraciones de dibujo y poder presentar el trabajo de manera más estructurada.

**Elemento 1**

**glDrawElements(GL\_TRIANGLES, 3, GL\_UNSIGNED\_INT, (void\*)(12));**

**glDrawElements(GL\_TRIANGLES, 3, GL\_UNSIGNED\_INT, 0);**

En esta figura, se dibujaron dos triángulos con el propósito de practicar estos comandos. En particular, el primer comando presentó una situación compleja, ya que fue necesario utilizar un cast a void para especificar un desplazamiento de 12 bytes. Esto permitió acceder a la dirección de memoria correcta dentro de los índices y así renderizar el triángulo correspondiente.

**Elemento 2**

**glPointSize(10);**

**glDrawArrays(GL\_POINTS, 4, 4);**

**glDrawArrays(GL\_TRIANGLES, 4, 3);**

**glDrawElements(GL\_TRIANGLES, 3, GL\_UNSIGNED\_INT, (void\*)(24));**

En esta figura, se enfatiza el uso de puntos en la representación gráfica. Aunque los triángulos se incluyeron por motivos estéticos, el enfoque principal está en los puntos. Se observa que se dibujan cuatro puntos, comenzando desde el índice 4, con un tamaño de 10 píxeles.

**Elemento 3**

**glDrawArrays(GL\_LINE\_LOOP, 12, 4);**

Este código dibuja líneas de manera continua en un solo comando. Aunque podría parecer que se debería haber generado un polígono, en realidad el orden original de los vértices en la declaración influye en el resultado. Aunque se podría haber modificado dicho orden, se decidió mantenerlo para ilustrar con mayor precisión el uso de GL\_LINE\_LOOP en la representación de líneas conectadas.

**Elemento 4**

**glDrawArrays(GL\_LINES, 8, 4);**

En esta última configuración de dibujo, se utiliza el comando GL\_LINES, que, a diferencia de GL\_LINE\_LOOP, no cierra automáticamente las líneas. Como resultado, las líneas dibujadas son independientes entre sí y no se conectan unas con otras.

**Conclusiones:**

Lo más desafiante de esta práctica fue comprender el uso del comando que permite trabajar con índices. Inicialmente, se intentó asignar un valor entero, lo que generó un error de compilación, ya que la biblioteca requiere un tipo de dato específico. Esto llevó a investigar el uso correcto de los índices, donde se descubrió que deben definirse como un arreglo de tipo const void\* indices[1], lo que implica un desplazamiento en bytes.

Otro aspecto clave fue entender el orden de los vértices en la declaración y determinar en qué casos se debe utilizar cada elemento para dibujar correctamente las primitivas. Además, hubo dificultades al agregar los 16 vértices de forma adecuada, ya que cualquier error podía causar discontinuidades o figuras desproporcionadas, afectando la estética del trabajo.

**Bibliografía consultada:**

***[1] Registry.khronos.org*. (s. f.). registry.khronos.org/. https://registry.khronos.org/OpenGL-Refpages/es3.1/html/glDrawElements.xhtml?utm\_source=chatgpt.com**