Trabajo Práctico Final

Asignatura:
Programación Eficiente
Profesor:
Gómez, Pablo
Eschoyez, Maximiliano

Integrantes del grupo:
Lo Preiato, Lucas
Oliva, Benjamín
Ventura, Gino

Fecha: 01/08/2024

UBP UNIVERSIDAD Blas Pascal

INTRODUCCIÓN

En el campo de la programación y la informática gráfica, la eficiencia y el rendimiento son aspectos cruciales que determinan la calidad y la experiencia del usuario en aplicaciones y videojuegos. La elección de un algoritmo de renderizado adecuado desempeña un papel fundamental en este contexto. Este informe tiene como objetivo presentar los resultados de un estudio exhaustivo en el que se seleccionó un algoritmo de renderizado y se evaluó su rendimiento en diferentes computadoras.

En este informe, presentaremos el desarrollo de una aplicación grafica utilizando la biblioteca OpenGL. Esta aplicación tiene como objetivo graficar diferentes funciones coseno en ejes de coordenadas x e y.

HERRAMIENTAS UTILIZADAS

- ***OpenGL:***

Es un estándar para renderizar gráficos en 2D y 3D en aplicaciones de software. Fue en la década de 1990 y desde entonces ha sido ampliamente adoptada en la industria de la informática gráfica debido a su versatilidad y capacidad para trabajar en una variedad de sistemas operativos, incluyendo Windows, macOS y Linux.
En nuestro proyecto, se utiliza OpenGL para crear y gestionar la ventana gráfica, así como para dibujar los ejes de coordenadas y las respectivas funciones.

- ***WSL2:***

WSL2, o Windows Subsystem for Linux 2, es una tecnología desarrollada por Microsoft que permite ejecutar sistemas operativos basados en Linux, como Ubuntu, Debian y otras distribuciones populares, directamente en Windows 10 o versiones posteriores.
En nuestro proyecto lo utilizamos para habilitar un entorno de desarrollo Linux dentro de Windows, lo que permitió compilar y ejecutar programas que dependen de Linux.

- ***GLFW:***

Es una biblioteca de código abierto diseñada para simplificar el proceso de creación y gestión de ventanas y contextos OpenGL en aplicaciones gráficas.

- ***XLaunch:***

Es una herramienta que forma parte del sistema X Window System (también conocido como X11), que es un sistema de ventanas utilizado en sistemas operativos Unix y Linux para la gestión de interfaces gráficas de usuario. La función principal de XLaunch es facilitar la configuración y el inicio de un servidor de visualización X en sistemas Windows.

- ***Visual Studio Code:***

Visual Studio Code, a menudo abreviado como VS Code, es un entorno de desarrollo integrado (IDE) de código abierto desarrollado por Microsoft. Fue utilizado para escribir y depurar el código.

DESARROLLO DEL PROGRAMA

El programa desarrollado tiene como función principal generar valores de x e y para graficar y visualizar las funciones coseno en una ventana gráfica. La cantidad de estas funciones las ingresa el usuario y los puntos se generan con una variación determinada previamente.

- ***Función coseno:***

El coseno es una función matemática fundamental en trigonometría que relaciona un ángulo en un triángulo rectángulo con la relación entre dos de sus lados.
En nuestro programa, realizamos nuestra propia función coseno, en vez de utilizar la función ya conocida de la librería math.h.
La función llamada customCos(x), es una implementación personalizada del cálculo del coseno de un valor x utilizando una serie de Taylor. Esta serie es una aproximación matemática que se utiliza para calcular funciones trigonométricas como el coseno.

1. ***Parámetros:*** La función toma un argumento **x**, que es el valor para el cual deseas calcular el coseno.
2. ***Constante terms:*** Se define una constante llamada **terms** con un valor de 1,000. Esta constante determina cuántos términos de la serie de Taylor se utilizarán en el cálculo. Cuantos más términos se utilicen, mayor será la precisión de la aproximación al coseno.
3. ***Variables locales:***
    - **result**: Inicializada en 1.0, esta variable almacenará el resultado acumulado de la serie de Taylor a medida que se calculan más términos.
    - **term**: Inicializada en 1.0, esta variable almacena el valor del término actual de la serie de Taylor y se multiplica por los términos sucesivos en cada iteración del bucle.
4. ***Bucle for:*** La función utiliza un bucle **for** que itera desde **n = 1** hasta **terms - 1**. En cada iteración, se calcula un nuevo término de la serie de Taylor y se agrega al resultado acumulado **result**.
5. ***Cálculo del Término:*** El cálculo del término se realiza dentro del bucle utilizando la fórmula de la serie de Taylor para el coseno. Esta fórmula utiliza potencias alternas de **x** y números factoriales. El término se multiplica por **-x * x / ((2 * n) * (2 * n - 1))**, y luego se multiplica por el término anterior en la siguiente iteración. Esto agrega sucesivamente más términos de la serie a medida que avanza el bucle.
6. ***Resultado Final:*** Una vez que se han calculado todos los términos dentro del bucle, la función devuelve el valor acumulado en la variable **result**, que es una aproximación al coseno de **x** basada en la serie de Maclaurin.

Es importante mencionar que esta implementación utiliza una cantidad fija de términos (1,000 en este caso) para la aproximación, lo que puede afectar la precisión y el rendimiento dependiendo del valor de x.
Las implementaciones del coseno en bibliotecas matemáticas estándar suelen ser más eficientes y precisas, pero esta función muestra cómo se puede realizar una aproximación del coseno mediante una serie matemática. Para el caso de la función seno, es una aplicación similar.

ESTRUCTURA DEL PROGRAMA

1. ***Inclusión de bibliotecas***: en esta parte, se incluyen las bibliotecas necesarias para el proyecto, estas bibliotecas son utilizadas para trabajar con OpenGL (gráficos), GLFW (manejo de ventanas) y otras para medir el tiempo.
2. ***Variables globales***: estas variables definen el ancho y alto de la ventana de visualización.
3. ***Funciones trigonométricas personalizadas:*** se definen dos funciones personalizadas para calcular el coseno y el seno utilizando aproximaciones matemáticas basadas en series de Taylor.
4. ***Dibujo de la función:*** se dibujan las funciones en el contexto OpenGL. Toma dos valores x e y, así como valores de color r, g, y b, para especificar el color de la línea de la función.
5. ***Función principal:*** en la función principal "**graficarFunciones()**" se inicializa GLFW, se crea una ventana de visualización, y luego entra en un bucle principal que se ejecuta hasta que se cierra la ventana, donde se establece que funciones va a graficar, según la elección del usuario. "True" para funciones "custom" y "False" para las funciones de la biblioteca "Math".

OPTIMIZACIONES Y CORRECCIONES CON RESPECTO A LA VERSIÓN ANTERIOR

Ambos códigos son iguales en términos de la funcionalidad que realizan, pero existen algunas diferencias clave:

- ***Número de Términos en las Series:*** en la nueva versión, utilizamos 1,000 términos en las series de Taylor para calcular las funciones trigonométricas (seno y coseno), mientras que en la primera y segunda versión utilizábamos 100,000 y 10,000 términos respectivamente. Para versiones anteriores, se utilizaba una mayor cantidad de términos, lo que puede resultar en una mayor precisión en los cálculos, pero también en un mayor tiempo de ejecución. Luego de distintas pruebas, descubrimos que, disminuyendo la cantidad de términos, el resultado final no se veía afectado, caso contrario con el tiempo de ejecución, que para graficar 10 funciones disminuyó de 3 segundos a casi 3 milésimas de segundos.
  Esta disminución en la cantidad de términos de las series de Taylor también nos permitió poder graficar mas funciones sin que el programa se rompa o se quede clavado.
- ***Medición del Tiempo:*** otro de los cambios, es la utilización de una función "measure" para medir el tiempo que lleva ejecutar la función "iterateAndDraw" con un número especificado de líneas. Ya que antes lo realizábamos dentro del bucle principal.
- ***Implementación de funciones de la librería math.h:*** una mejora importante con respecto a las versiones anteriores fue la implementación de las funciones para calcular seno y coseno, utilizando la librería math.h. Dichas funciones están altamente optimizadas y son extremadamente eficientes en términos de tiempo de ejecución, además están diseñadas para ofrecer resultados muy precisos.
- ***Encapsulamiento de funciones:*** otro de los cambios con respecto a la segunda versión, fue el encapsulamiento de funcionalidades en entidades separadas, el uso

de clases y de archivos headers. Esto ayuda a facilitar el mantenimiento del código, la reutilización y el entendimiento de este.

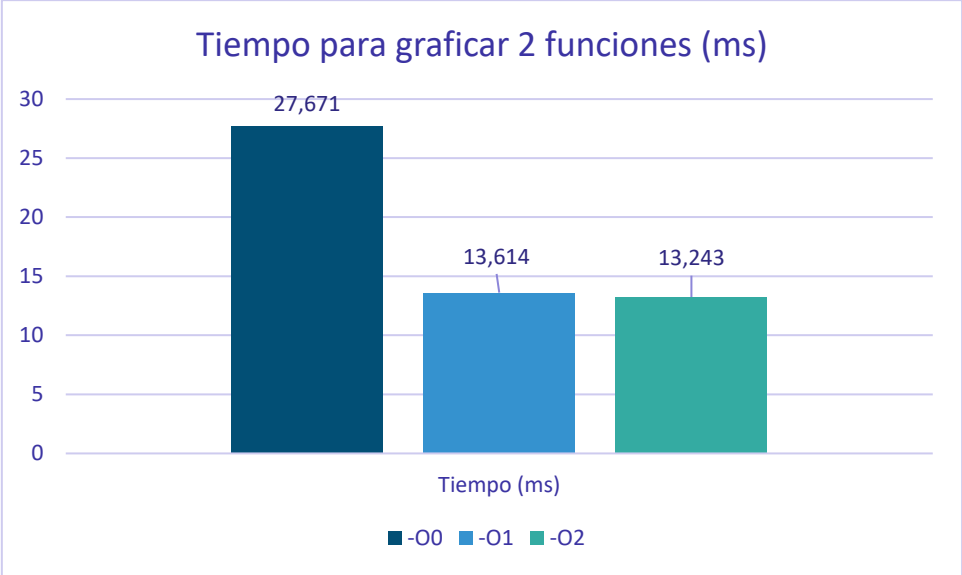
- ***Utilización de paralelismo o asincronismo:*** se implementaron funciones con paralelismo para calcular los valores de seno y coseno, sin embargo, esto no ayudó a reducir el tiempo de ejecución con respecto a las versiones secuenciales. Esto se puede deber al overhead que supone la generación de hilos y las operaciones relacionadas a el paralelismo, debido a que no son cálculos tan grandes.

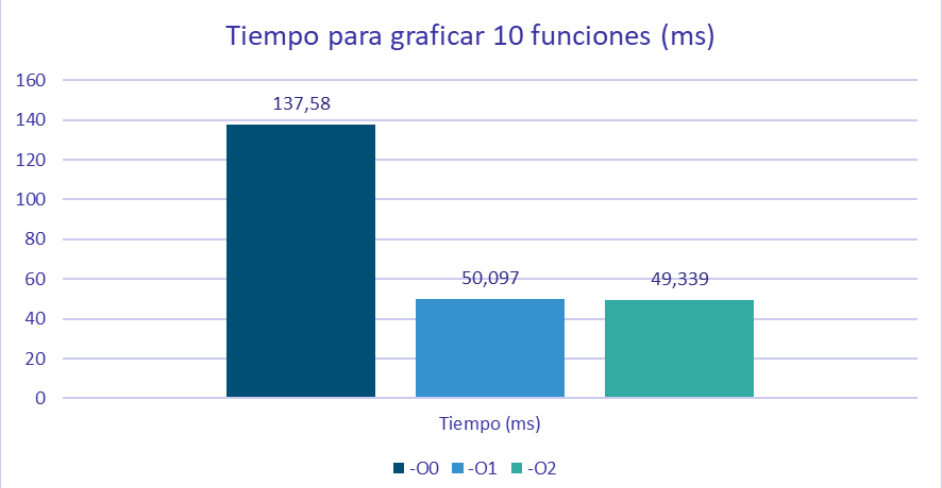
COMPARACIÓN DE TIEMPOS ENTRE VERSIONES

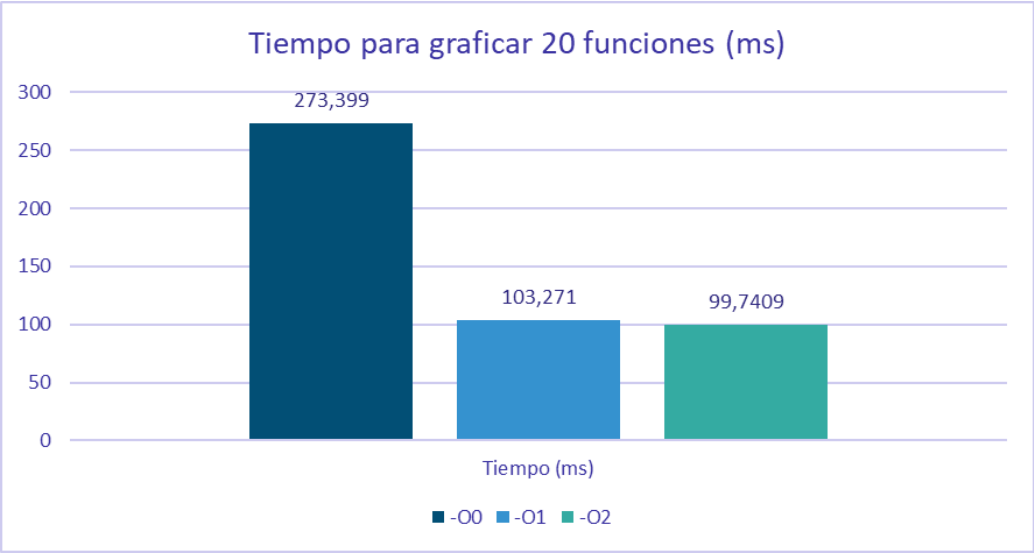
Pruebas de tiempo de ejecución para graficar funciones seno y coseno.

Número de versión	Cantidad de funciones (Custom)		
	2 funciones	10 funciones	20 funciones
Versión 1	1279,31 ms	12107,70 ms	25325,28 ms
Versión 2	202,77 ms	1421,22 ms	2671,64 ms
Versión Final	27,67 ms	137,58 ms	273,40 ms

GRÁFICOS COMPARATIVOS DE TIEMPO DE EJECUCIÓN EN DIFERENTES NIVELES DE OPTIMIZACIÓN (VERSIÓN FINAL, FUNCIONES CUSTOM):

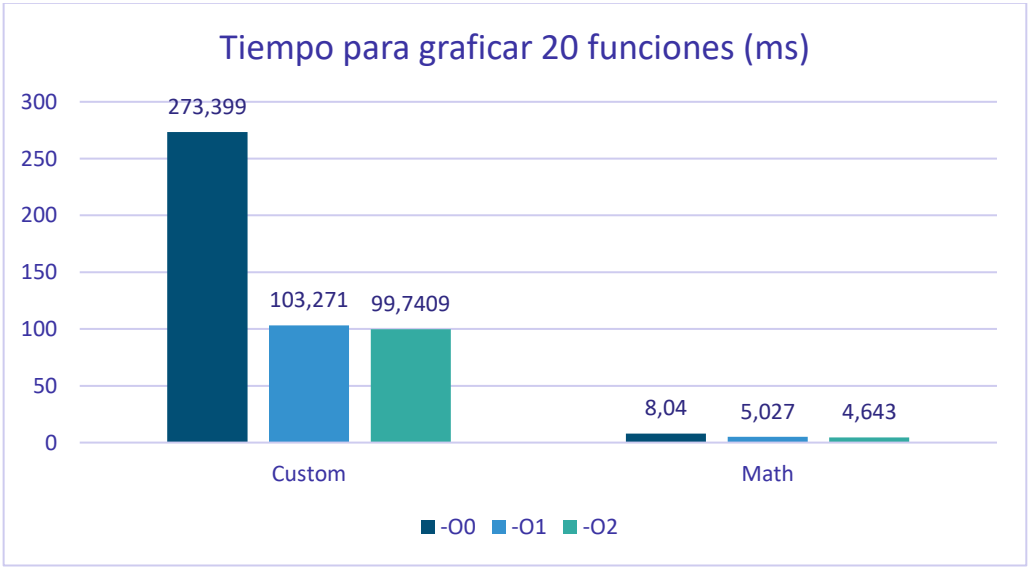






GRÁFICOS COMPARATIVOS DE TIEMPO DE EJECUCIÓN CON FUNCIONES CUSTOM Y MATH:

Cantidad de funciones:	20 funciones	
Nivel de optimización	**Custom**	**Math**
-O0	273,399 ms	8,04 ms
-O1	103,271 ms	5,027 ms
-O2	99,7409 ms	4,643 ms



CONCLUSIONES

La diferencia principal entre las versiones generadas con diferentes niveles de optimización (O0 y O2) radica en la eficiencia y la estrategia de optimización utilizada. Aquí hay algunas diferencias clave:

- ***Directivas de Salto:***

- En la versión de O0, se usan las instrucciones call cos y call sin para llamar a las funciones de la biblioteca estándar cos y sin respectivamente.
- En la versión de O2, se utilizan instrucciones de salto (jmp cos y jmp sin) para saltar directamente a las implementaciones de las funciones cos y sin en la biblioteca estándar. Esto evita la sobrecarga de las llamadas de función.

- ***Optimización de Bucle:***

- La versión de O0 utiliza un bucle explícito para calcular la aproximación de las funciones trigonométricas personalizadas (customCos y customSen). El bucle se ejecuta 999 veces en ambos casos.
En la versión de O2, las funciones personalizadas se implementan con un enfoque diferente. Se utilizan técnicas de optimización para reducir la cantidad de cálculos repetidos y se logra el mismo resultado sin necesidad de un bucle explícito.

- ***Registro de Propósito General:***

- En la versión de O0, se utilizan registros de propósito general como eax, edx, y ecx para algunas operaciones aritméticas y de control de bucle.
En la versión de O2, el uso de registros de propósito general se minimiza, lo que puede mejorar el rendimiento al reducir el número de transferencias entre la memoria y los registros.

- ***Uso de Instrucciones SIMD:***

- En la versión de O2, se utilizan instrucciones SIMD (Single Instruction, Multiple Data) como mulsd, addsd, y divsd para realizar cálculos en paralelo en datos de doble precisión, lo que puede mejorar significativamente el rendimiento en procesadores modernos que admiten estas instrucciones.

ENLACE AL REPOSITORIO

https://github.com/BenjaOliva/PEF-2023