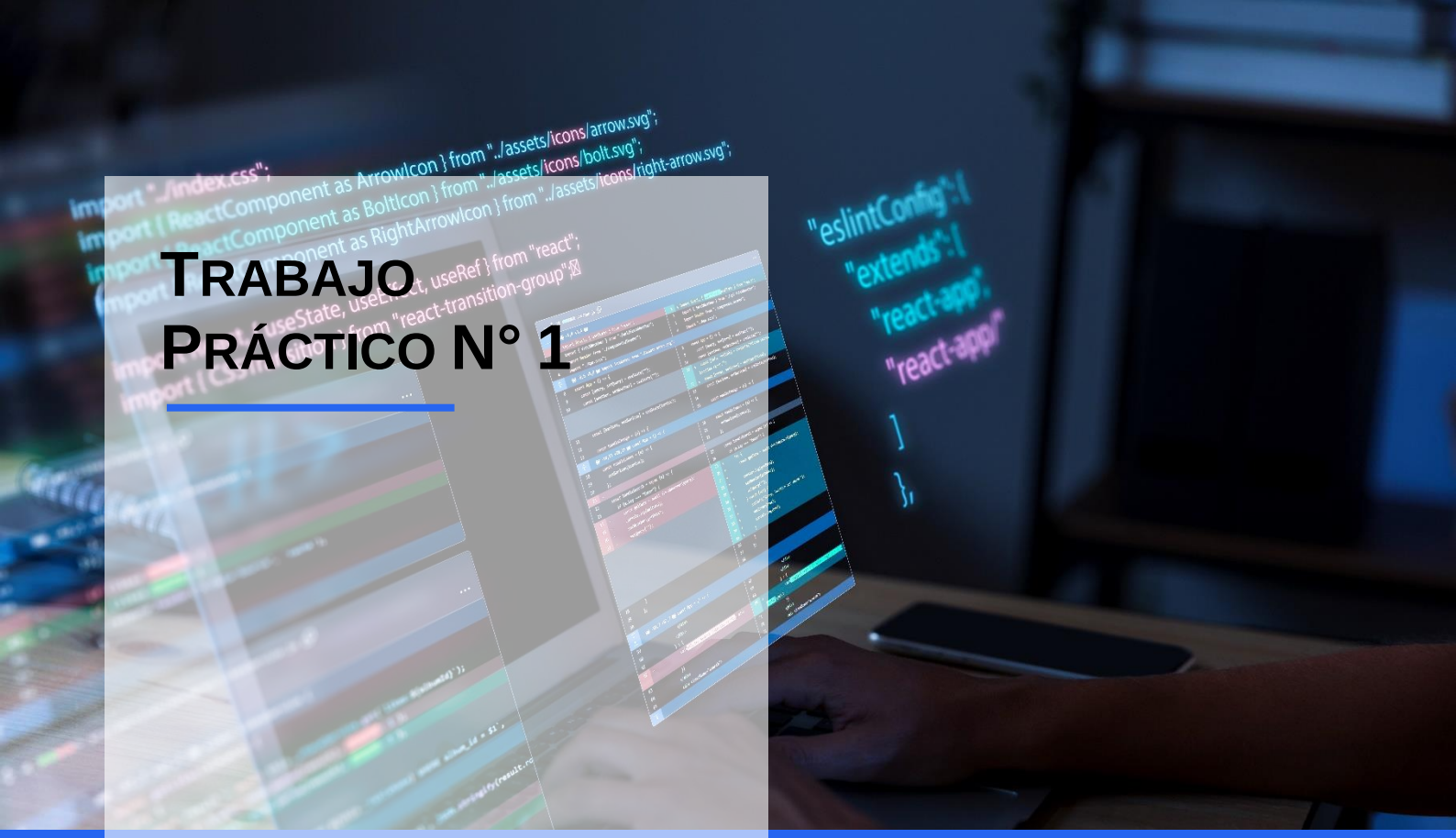


Trabajo Práctico N° 1

Asignatura:
Programación Eficiente
Profesor:
Gómez, Pablo

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

Fecha: 28/09/2023

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 grafica. 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 100,000. 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 (100,000 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.

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 manipular vectores y 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 (no utilizada) utilizando aproximaciones matemáticas basadas en series de Taylor.
4. ***Dibujo de la función:*** esta función se encarga de dibujar una función en el contexto OpenGL. Toma dos vectores de 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:*** la función principal es el punto de entrada del programa. Aquí 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.
6. ***Bucle principal:*** Este es el bucle principal del programa que se ejecuta mientras la ventana no se haya cerrado. Aquí se realizan las siguientes tareas:
    a. Se borra el búfer de color actual.
    b. Se generan los valores x para la función.
    c. Se calculan y dibujan las funciones coseno con diferentes desplazamientos verticales para crear múltiples líneas en la gráfica.
    d. Se intercambian los búferes (para mostrar la gráfica en pantalla) y se manejan los eventos de la ventana.
    e. La sección final de este bucle mide el tiempo que tarda en calcular y dibujar las funciones.

OPTIMIZACIONES ENCONTRADAS

Las diferencias en el código assembler entre las versiones con O0 (sin optimización) y O1 (optimización nivel 1) son las siguientes:

- ***Función original:***

```
13  double customCos(double x) {
14      constexpr int terms = 100000;
15      double result = 1.0;
16      double term = 1.0;
17
18      for (int n = 1; n < terms; n++) {
19          term *= (-x * x) / ((2 * n) * (2 * n - 1));
20          result += term;
21      }
22
23      return result;
24  }
```

- ***Función con nivel de optimización -O0 (assembler):***

```
customCos(double):
        push    rbp
        mov     rbp, rsp
        movsd   QWORD PTR [rbp-40], xmm0
        mov     DWORD PTR [rbp-24], 100000
        movsd   xmm0, QWORD PTR .LC0[rip]
        movsd   QWORD PTR [rbp-8], xmm0
        movsd   xmm0, QWORD PTR .LC0[rip]
        movsd   QWORD PTR [rbp-16], xmm0
        mov     DWORD PTR [rbp-20], 1
        jmp     .L2
.L3:
        movsd   xmm0, QWORD PTR [rbp-40]
        movq    xmm1, QWORD PTR .LC1[rip]
        xorpd   xmm0, xmm1
        mulsd   xmm0, QWORD PTR [rbp-40]
        mov     eax, DWORD PTR [rbp-20]
        add     eax, eax
        sub     eax, 1
        imul    eax, DWORD PTR [rbp-20]
        add     eax, eax
        pxor    xmm1, xmm1
        cvtsi2sd        xmm1, eax
        divsd   xmm0, xmm1
        movsd   xmm1, QWORD PTR [rbp-16]
        mulsd   xmm0, xmm1
        movsd   QWORD PTR [rbp-16], xmm0
        movsd   xmm0, QWORD PTR [rbp-8]
        addsd   xmm0, QWORD PTR [rbp-16]
        movsd   QWORD PTR [rbp-8], xmm0
        add     DWORD PTR [rbp-20], 1
.L2:
        cmp     DWORD PTR [rbp-20], 99999
        jle     .L3
        movsd   xmm0, QWORD PTR [rbp-8]
        movq    rax, xmm0
        movq    xmm0, rax
        pop     rbp
        ret
```

- ***Función con nivel de optimización -O1 (assembler):***

```
customCos(double):
        movapd  xmm4, xmm0
        xorpd   xmm4, XMMWORD PTR .LC1[rip]
        mulsd   xmm4, xmm0
        mov     ecx, 1
        mov     eax, 1
        movsd   xmm1, QWORD PTR .LC0[rip]
        movapd  xmm0, xmm1
.L2:
        mov     edx, ecx
        imul    edx, eax
        add     edx, edx
        pxor    xmm3, xmm3
        cvtsi2sd        xmm3, edx
        movapd  xmm2, xmm4
        divsd   xmm2, xmm3
        mulsd   xmm1, xmm2
        addsd   xmm0, xmm1
        add     eax, 1
        add     ecx, 2
        cmp     eax, 100000
        jne     .L2
        ret
.LC0:
        .long   0
        .long   1072693248
.LC1:
        .long   0
        .long   -2147483648
        .long   0
        .long   0
```

Como podemos observar en ambos códigos, las optimizaciones realizadas entre las versiones con O0 (sin optimización) y O1 (optimización nivel 1) son las siguientes:

1. ***Instrucciones simplificadas:*** En la versión con O1, se han eliminado instrucciones redundantes y se han reorganizado algunas operaciones para reducir la cantidad de código. Por ejemplo, se ha eliminado el uso de registros de propósito general como **rbp** y se han utilizado registros xmm para realizar operaciones de punto flotante de manera más eficiente.
2. ***Uso de movapd y xorpd:*** En la versión con O1, se utilizan las instrucciones **movapd** y **xorpd** para cargar y realizar operaciones de exclusión OR en registros xmm, respectivamente, en lugar de las múltiples instrucciones **movsd** y **xorpd** en la versión sin optimización. Esto simplifica el código y mejora la eficiencia.
3. ***Reducción de operaciones de carga de memoria:*** En O1, se ha reducido la cantidad de veces que se accede a la memoria para cargar datos, lo que reduce la latencia de memoria y mejora el rendimiento general del código.

4. ***Optimización de bucle:*** En la versión con O1, el bucle se ha optimizado para eliminar operaciones redundantes y reducir la carga de registros. Se utilizan registros xmm adicionales para almacenar valores temporales, lo que reduce la necesidad de acceder a la memoria en cada iteración del bucle.
5. ***Eliminación de saltos condicionales:*** En O1, se ha eliminado el salto condicional (**jle**) y se utiliza una comparación simple para verificar si se ha alcanzado la condición de finalización del bucle. Esto puede mejorar el rendimiento al reducir la penalización de salto condicional.

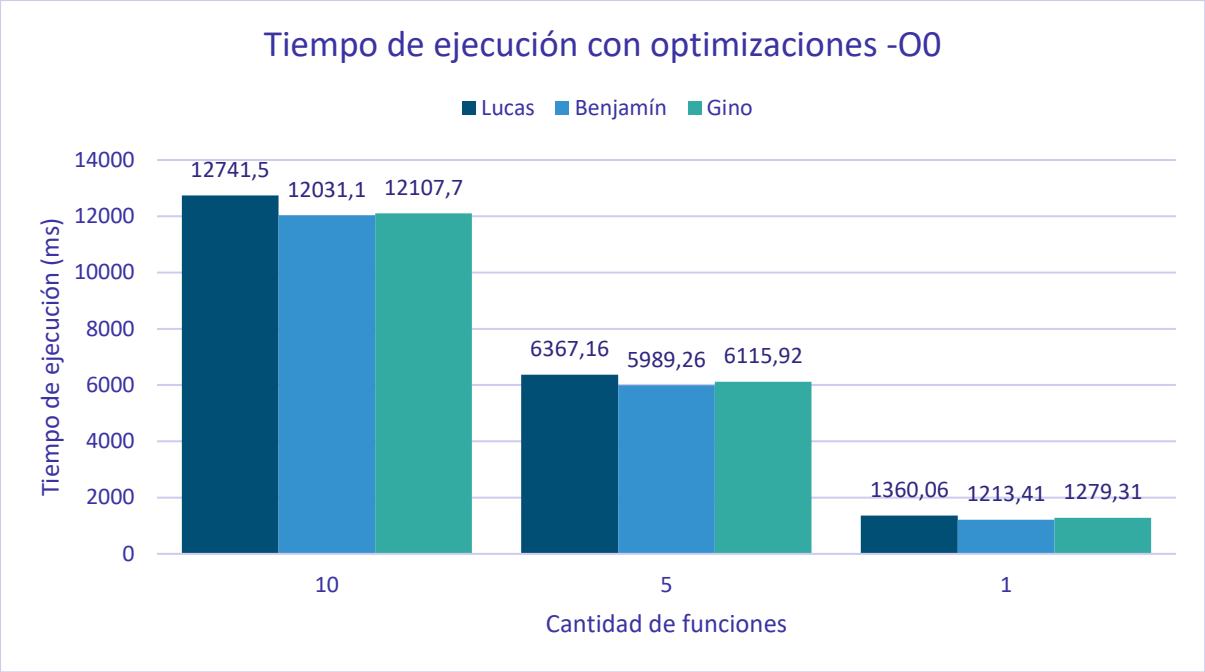
RESULTADOS

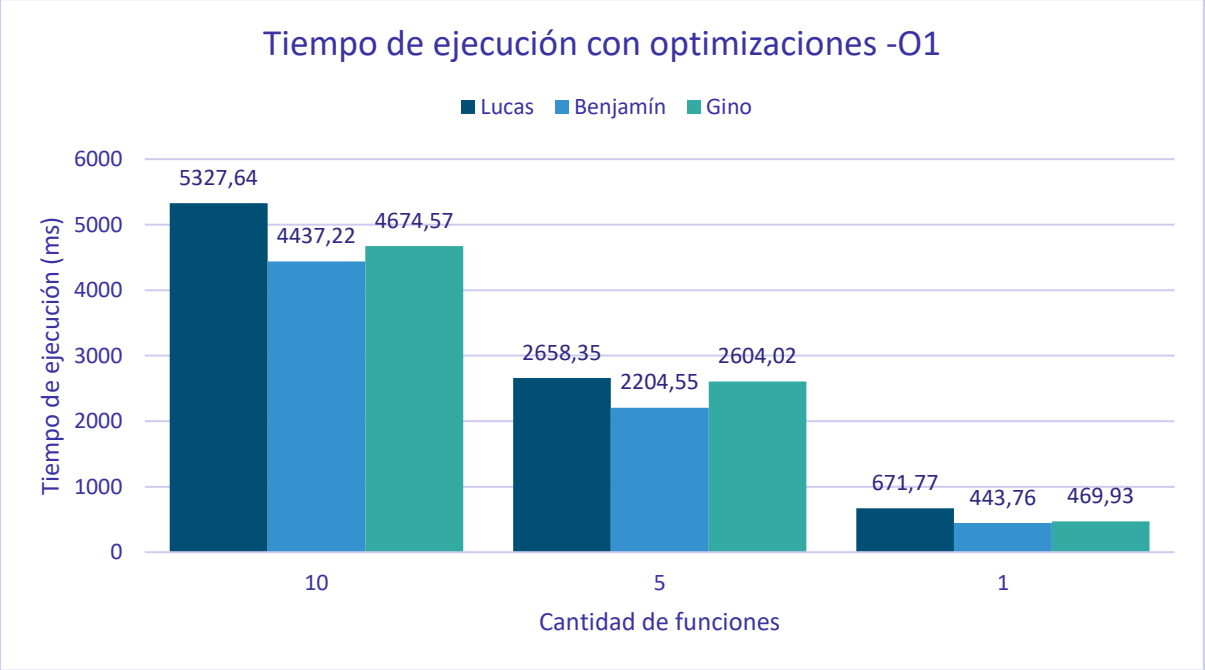
Pruebas de tiempo de ejecución para graficar funciones coseno
Valores de entrada:

1. Arreglo de valores de x (tipo vector y double).
2. Arreglo de valores de y (tipo vector y double).
3. Cantidad de funciones a graficar (entero).

	Lucas	**Benjamín**	**Gino**
Nivel de optimización -O0			
1 funciones	1360.06 ms	1213.41 ms	1279.31 ms
5 funciones	6367.16 ms	5989.26 ms	6115.92 ms
10 funciones	12741.50 ms	12031.1 ms	12107.7 ms
Nivel de optimización -O1			
1 funciones	671.77 ms	443.76 ms	469.938 ms
5 funciones	2658.35 ms	2204.55 ms	2604.02 ms
10 funciones	5327.64 ms	4437.22 ms	4674.57 ms
Hardware			
Procesador	Intel Core i5-8350U	AMD Ryzen 5 5600H	AMD Ryzen 5 3500U
Frecuencia	1.70 GHz	3.30 GHz	2.1 GHz
Memoria principal	8 GB	8 GB (5.96 GB utiliz.)	7.8 GB
Sistema operativo	Windows 11 Pro	Windows 10 Home	Ubuntu 20.04.6 LTS
Caches			
Nivel 1	4 x 32 KB 4 x 32 KB	6 x 32 KB 6 x 32 KB	4 x 64 KB 4 x 32 KB
Nivel 2	4 x 256 KB	6 x 512 KB	4 x 512 KB
Nivel 3	6 MB	16 MB	4 MB

GRÁFICOS COMPARATIVOS DE TIEMPO DE EJECUCIÓN EN DIFERENTES COMPUTADORAS:





ENLACE AL REPOSITORIO

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