UNIVERSIDAD DEL VALLE DE GUATEMALA

CC3069 - Computación Paralela y Distribuida

Sección 21

Ing. Juan Luis García Zarceño

Proyecto 1 - OpenMP

Screensaver

Diego Alberto Leiva	21752
José Pablo Orellana	21970

Guatemala, 30 de agosto del 2024

Índice

Introducción

En este informe, detallamos el desarrollo y la implementación de nuestro proyecto, cuyo objetivo principal es la creación de un screensaver utilizando la librería SDL, que simula la generación y movimiento de curvas de Rosa Polar. Nuestro trabajo se centró en diseñar e implementar un algoritmo secuencial eficiente que más tarde paralelizamos usando OpenMP para mejorar su rendimiento.

La razón detrás de este proyecto radica en la necesidad de aprovechar los múltiples núcleos de procesamiento disponibles en los sistemas modernos, con el fin de reducir los tiempos de ejecución y mejorar la eficiencia de los programas. A través de este proyecto, no solo buscamos crear un software funcional, sino también optimizarlo mediante técnicas de paralelización que explotan al máximo los recursos del hardware.

Nuestro enfoque se basó en una comparación directa entre las versiones secuenciales y paralelas del código, analizando métricas clave como los frames por segundo (FPS) para evaluar la eficacia de nuestra implementación paralela. A medida que avancemos en este informe, discutiremos los detalles técnicos del proyecto, las decisiones de diseño que tomamos y los desafíos que encontramos durante el proceso.

Antecedentes

El uso de OpenMP como herramienta para la paralelización de procesos en programas con memoria compartida es una práctica que ha ganado relevancia en el campo de la computación paralela. OpenMP facilita la transformación de programas secuenciales en paralelos mediante la adición de directivas que distribuyen el trabajo entre múltiples hilos de procesamiento. Esta técnica es especialmente útil en aplicaciones donde existen bucles independientes que pueden ejecutarse en paralelo, como es el caso de nuestro proyecto (OpenMP Architecture Review Board, 2023).

La elección de OpenMP se fundamenta en su simplicidad de uso y en la capacidad que ofrece para realizar cambios iterativos en un programa secuencial. A través de OpenMP, podemos aprovechar las capacidades de múltiples núcleos de procesamiento sin necesidad de rediseñar completamente el software. Además, OpenMP es compatible con C y C++, lo que lo convierte en una opción ideal para nuestro proyecto (Barney, 2023).

Por otro lado, la librería SDL (Simple DirectMedia Layer) es ampliamente utilizada en el desarrollo de aplicaciones gráficas y videojuegos debido a su versatilidad y portabilidad. SDL nos permitió gestionar la creación de ventanas, la captura de eventos, y, lo más importante para este proyecto, el renderizado de gráficos en tiempo real. La combinación de OpenMP y SDL en este proyecto nos proporcionó una base sólida para implementar y paralelizar el screensaver de curvas de Rosa Polar (SDL, 2023).

Screensaver - “Rosa Polar”

Descripción del Proyecto

El objetivo del proyecto fue desarrollar un screensaver que simulara la generación y movimiento de curvas de Rosa Polar, las cuales son figuras matemáticas interesantes por su simetría y complejidad visual. La idea principal era que el screensaver pudiera mostrar múltiples curvas en movimiento, con colores generados de manera pseudoaleatoria y con un comportamiento dinámico, como el rebote en los bordes de la pantalla.

Inicialmente, implementamos una versión secuencial del programa que renderizaba estas curvas en una ventana de 640x480 píxeles, utilizando SDL para el manejo de gráficos. Posteriormente, nuestro trabajo se enfocó en paralelizar el cálculo de los puntos de estas curvas utilizando OpenMP, con la intención de aumentar el rendimiento del programa al aprovechar el paralelismo inherente en el proceso de cálculo de los puntos de cada curva.

Implementación del Código Secuencial

La implementación secuencial del screensaver se realizó utilizando C++ junto con la librería SDL para el manejo de gráficos. El código se estructuró en varias funciones clave, cada una de las cuales tenía una responsabilidad bien definida.

Generación de Curvas de Rosa Polar:

- La estructura RosaPolar se diseñó para almacenar los parámetros de cada curva, como el número de pétalos, la escala, el color y las coordenadas del origen. Esto permitió un manejo ordenado y eficiente de los datos necesarios para renderizar cada curva.

Cálculo de Puntos en la Curva:

- El cálculo de los puntos que conforman la curva se realizó mediante funciones trigonométricas que determinaron las coordenadas x e y de cada punto en función del ángulo y el número de pétalos. Este cálculo se realizó en un bucle que iteraba a través de todos los puntos necesarios para dibujar la curva.

Renderizado de la Curva:

- Utilizando SDL, los puntos calculados se renderizaron en la pantalla, y se actualizó la ventana en cada ciclo para reflejar el movimiento de las curvas. Se implementó una función de generación de colores aleatorios para darle dinamismo visual al screensaver.

Manejo de FPS:

- Para asegurar que la experiencia visual fuera fluida, implementamos una medición de FPS (frames per second) y ajustamos dinámicamente el título de la ventana para mostrar esta métrica en tiempo real. Al final de la ejecución, generamos un reporte que incluía el FPS promedio, mínimo y máximo, lo que nos permitió evaluar el rendimiento de la versión secuencial.

Implementación del Código Paralelo

La paralelización del código se centró en dos áreas principales: la generación de las rosas polares y el cálculo de los puntos de cada curva de Rosa Polar en cada fotograma. Ambas áreas fueron identificadas como ideales para la paralelización debido a la independencia entre las iteraciones en sus respectivos bucles.

Paralelización con OpenMP

Generación de las Rosas Polares

- Utilizamos la directiva “#pragma omp parallel for” para paralelizar el bucle que genera las rosas polares. Cada rosa polar es independiente de las demás, lo que permitió que múltiples hilos trabajaran simultáneamente para generar varias rosas en paralelo. La estructura “RosaPolar” almacena las propiedades de cada rosa (número de pétalos, escala, color, etc.), y este proceso se benefició enormemente de la paralelización, ya que cada hilo podía encargarse de generar una rosa de manera independiente.

Cálculo de los Puntos de la Curva de Rosa Polar

- El cálculo de los puntos que componen la curva de Rosa Polar también fue paralelizado. Este proceso es computacionalmente costoso, ya que implica cálculos matemáticos (senos, cosenos) para cada punto en función de la estructura de la rosa y su ángulo de rotación. Nuevamente, aplicamos la directiva “#pragma omp parallel for” para distribuir este trabajo entre múltiples hilos. Cada hilo calculaba los puntos para una rosa diferente, lo que permitió que los puntos de múltiples rosas se generaran en paralelo.

Sincronización del Renderizado:

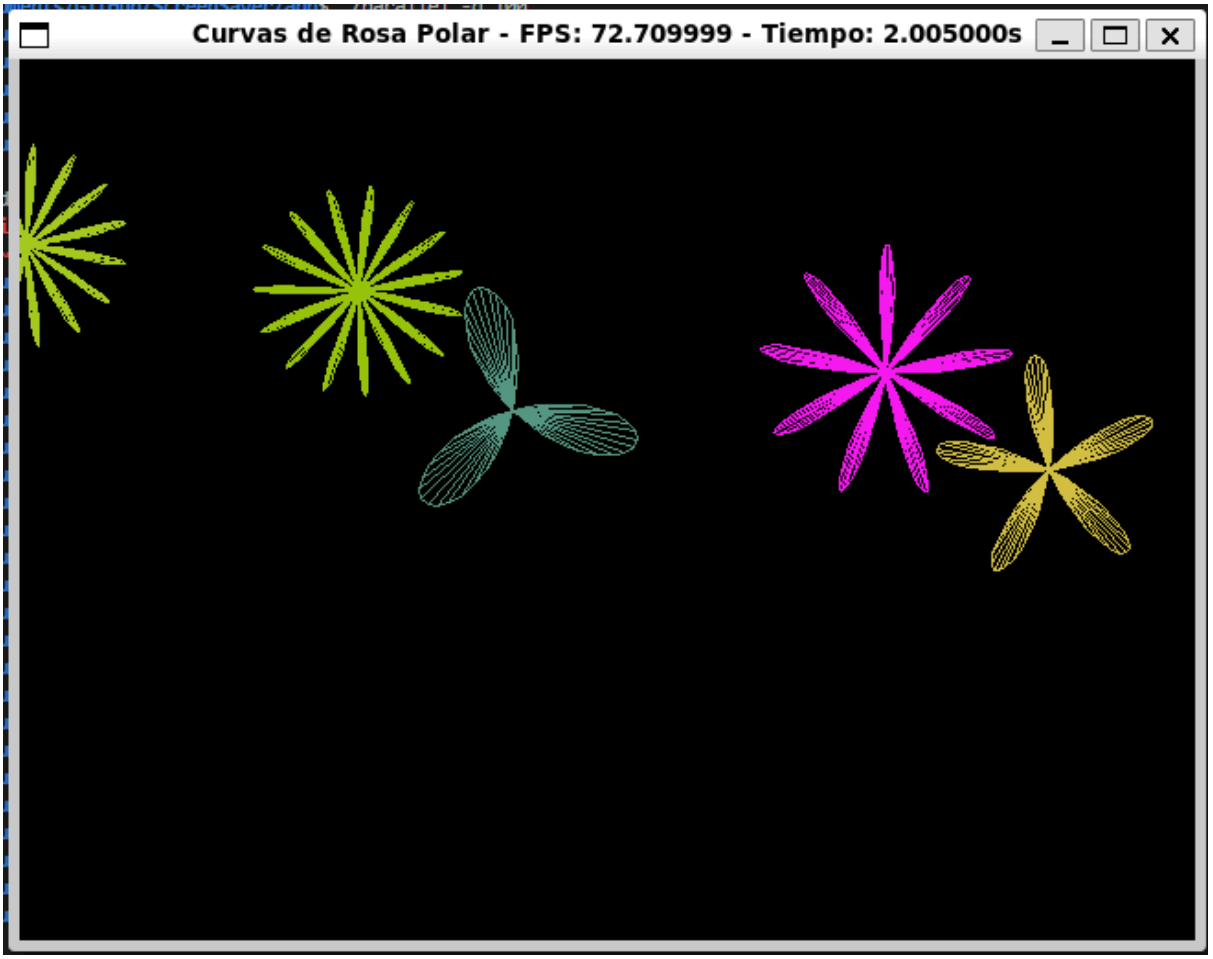
- Debido a que SDL2 no es "thread-safe", el proceso de renderizado no fue paralelizado. SDL2 requiere que todas las operaciones de renderizado (como dibujar las líneas y los puntos en la ventana) se ejecuten en un solo hilo, el hilo principal.

Generación de Reportes:

- Al final de la ejecución, se generó un reporte de los **FPS** de la misma forma que en la versión secuencial. Este reporte incluye métricas como FPS promedio, máximo, mínimo y el valor de FPS en el percentil más bajo (1% Low FPS).

Resultados

Después de implementar y ejecutar las versiones secuencial y paralela del programa de curvas de Rosa Polar, obtuvimos los siguientes resultados en términos de rendimiento medido en FPS (frames por segundo):

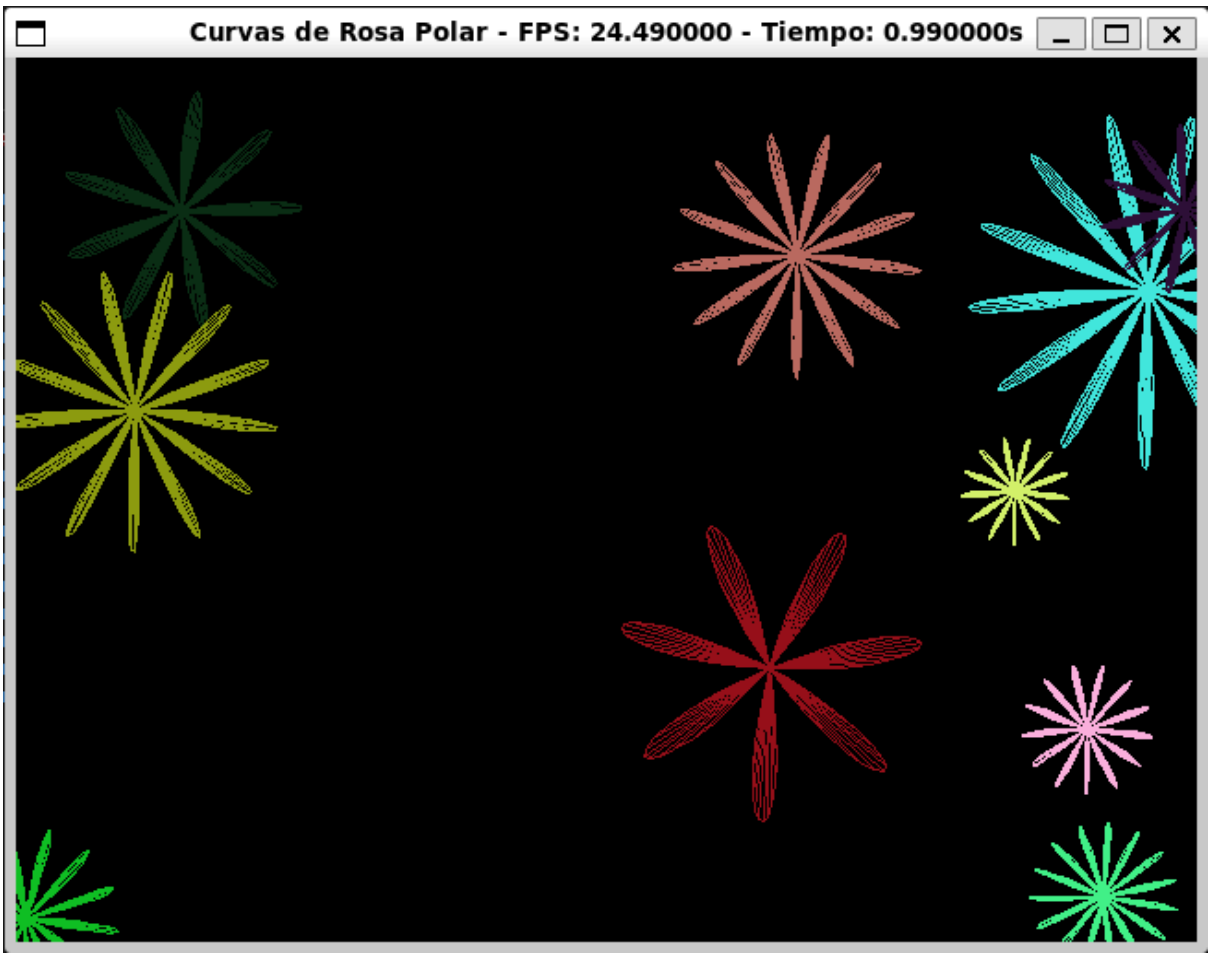


Curvas de Rosa Polar Secuencial
Cantidad de Rosas: 5

Metrics Report:
Average FPS: 58.5836
Minimum FPS: 52.22
Maximum FPS: 60.7
1% Low FPS: 52.22

Curvas de Rosa Polar Paralelizado
Cantidad de Rosas: 5

Metrics Report:
Average FPS: 167.711
Minimum FPS: 109.56
Maximum FPS: 214
1% Low FPS: 109.56

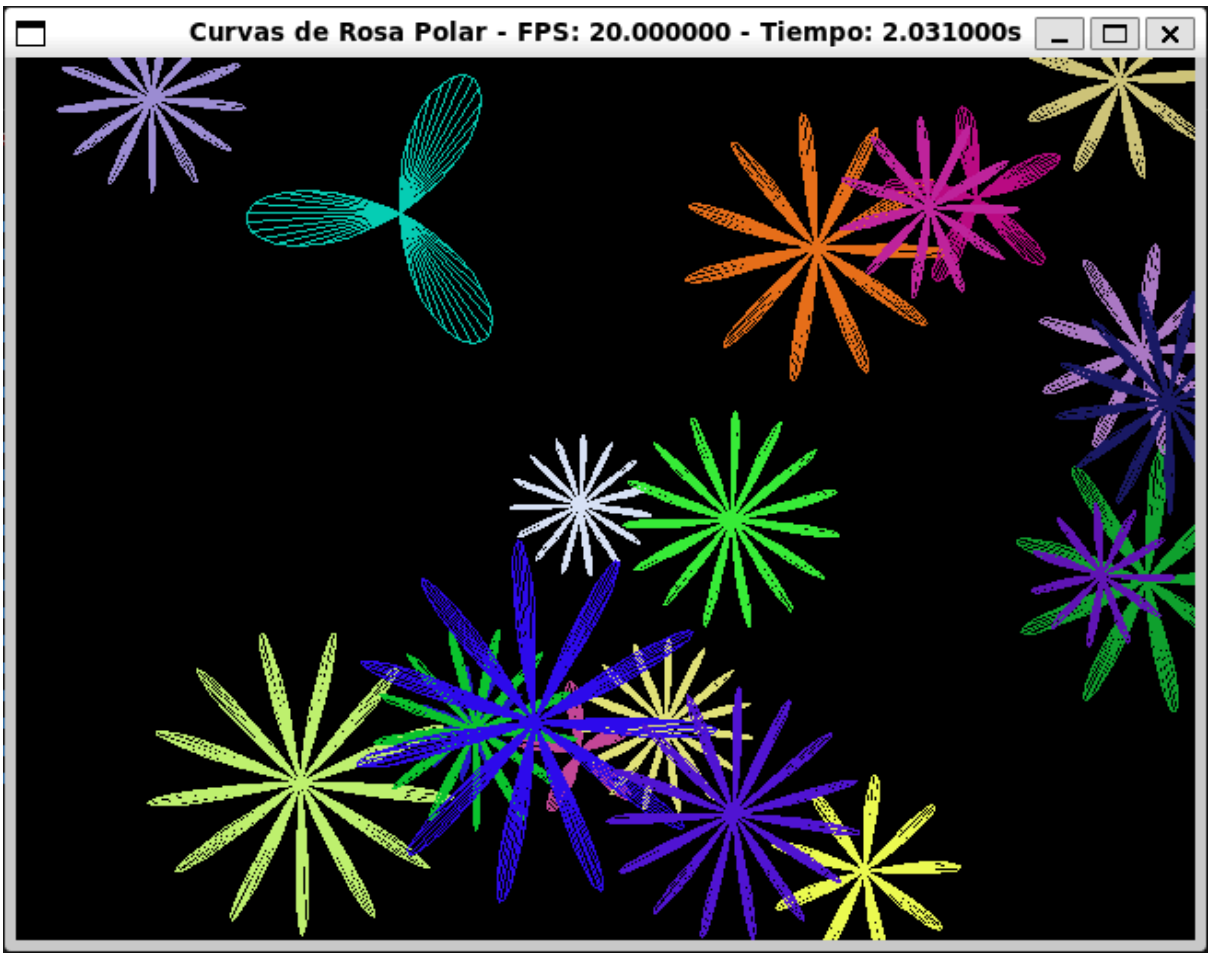


Curvas de Rosa Polar Secuencial
Cantidad de Rosas: 10

Metrics Report:
Average FPS: 41.35
Minimum FPS: 35.47
Maximum FPS: 42.96
1% Low FPS: 35.47

Curvas de Rosa Polar Paralelizado
Cantidad de Rosas: 10

Metrics Report:
Average FPS: 117.888
Minimum FPS: 106.26
Maximum FPS: 120.52
1% Low FPS: 106.26

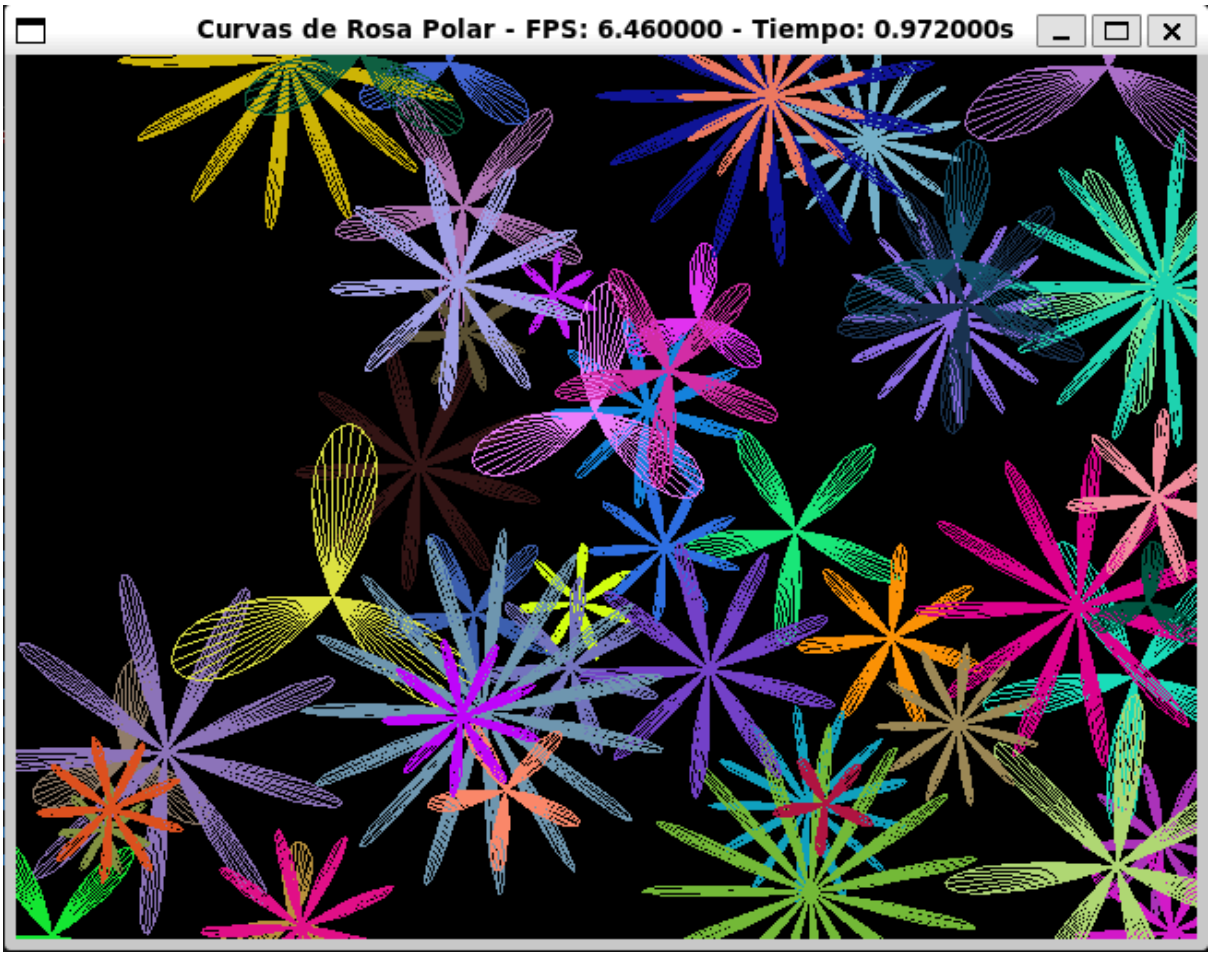


Curvas de Rosa Polar Secuencial
Cantidad de Rosas: 20

Metrics Report:
Average FPS: 21.1757
Minimum FPS: 17.26
Maximum FPS: 21.78
1% Low FPS: 17.26

Curvas de Rosa Polar Paralelizado
Cantidad de Rosas: 20

Metrics Report:
Average FPS: 58.0043
Minimum FPS: 47.67
Maximum FPS: 60.04
1% Low FPS: 47.67

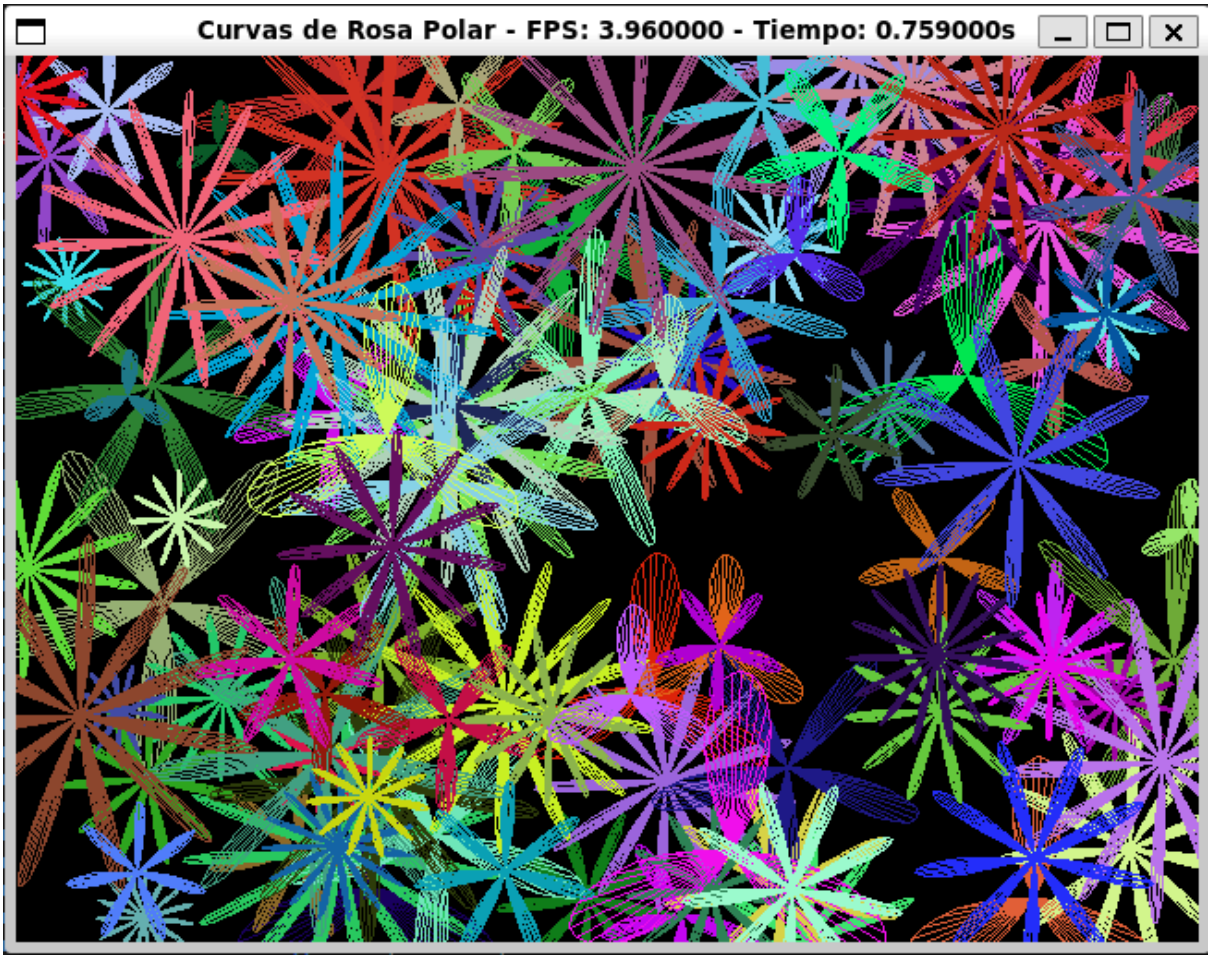


Curvas de Rosa Polar Secuencial
Cantidad de Rosas: 50

Metrics Report:
Average FPS: 7.555
Minimum FPS: 5.26
Maximum FPS: 8.88
1% Low FPS: 5.26

Curvas de Rosa Polar Paralelizado
Cantidad de Rosas: 50

Metrics Report:
Average FPS: 28.9057
Minimum FPS: 25.74
Maximum FPS: 29.76
1% Low FPS: 25.74



Curvas de Rosa Polar Secuencial
Cantidad de Rosas: 100

Metrics Report:
Average FPS: 4.615
Minimum FPS: 3.81
Maximum FPS: 4.86
1% Low FPS: 3.81

Curvas de Rosa Polar Paralelizado
Cantidad de Rosas: 100

Metrics Report:
Average FPS: 15.125
Minimum FPS: 12.96
Maximum FPS: 15.97
1% Low FPS: 12.96

Interpretación de los Resultados

Los resultados muestran que la versión paralelizada del programa ofrece un rendimiento significativamente superior en comparación con la versión secuencial. En cada uno de los casos, el FPS promedio de la versión paralelizada es al menos tres veces mayor que el de la versión secuencial, alcanzando una mejora especialmente notable en casos con menor cantidad de rosas, donde la versión paralelizada llega a triplicar los FPS obtenidos en la versión secuencial.

Conclusiones

Los resultados obtenidos sugieren que la paralelización del programa utilizando OpenMP resultó en una mejora de rendimiento significativa en este caso específico. A diferencia de otros escenarios donde la sincronización en el acceso al renderizador podría ser un cuello de botella, los cálculos paralelizados de los puntos de las curvas de Rosa Polar han permitido aumentar considerablemente los FPS. En algunos casos, la versión paralelizada muestra un rendimiento casi tres veces superior al de la versión secuencial, especialmente cuando se maneja un mayor número de rosas. Aunque el renderizado sigue siendo secuencial debido a las limitaciones de SDL2, la distribución del cálculo entre múltiples núcleos ha compensado este factor, haciendo que la paralelización sea una solución efectiva para este proyecto en particular.

Recomendaciones

Reducción de la Frecuencia de Actualización del Renderizado:

- Para evitar la sincronización constante entre los cálculos y el renderizado, se podría reducir la frecuencia de actualización del renderizado, dibujando los resultados menos veces por segundo, mientras los cálculos se siguen realizando en paralelo a alta velocidad. Esto podría reducir la sobrecarga del renderizado sin afectar demasiado la percepción de la fluidez visual.

Evaluar la Paralelización en Otras Áreas:

- A pesar de que el cálculo de puntos de las rosas fue eficazmente paralelizado, se podría explorar la paralelización de otras partes del proceso, como la actualización de los estados o rotaciones de las rosas, o la posibilidad de dividir la ventana en diferentes regiones que se rendericen de manera más independiente. Esto podría balancear la carga de trabajo y aprovechar mejor los recursos de procesamiento disponibles en el sistema.

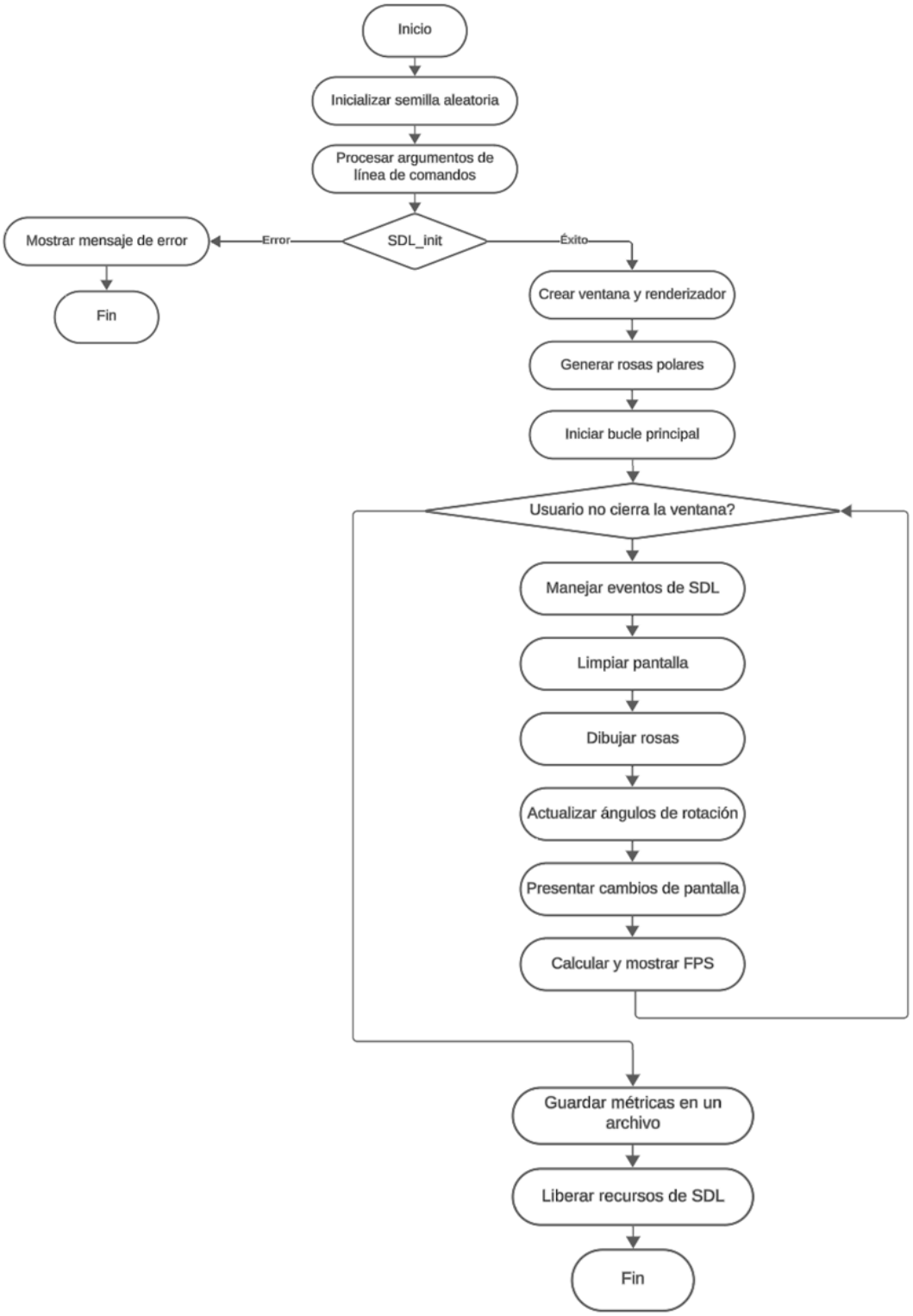
Aprendizajes

Este proyecto ha sido un excelente aprendizaje sobre las oportunidades y desafíos de la paralelización en aplicaciones gráficas. Hemos aprendido que no todas las tareas son fácilmente paralelizables, como es el caso del renderizado con SDL2, que no está diseñado para ser "thread-safe". Este proyecto demuestra la importancia de evaluar cuidadosamente qué partes del código pueden beneficiarse de la paralelización, y también nos ha enseñado que la paralelización no es una solución universal, especialmente cuando se trabaja con bibliotecas que imponen restricciones en el acceso concurrente.

Anexos

Enlace Repositorio Github: https://github.com/LeivaDiego/ScreenSaver

Anexo 1 (Secuencial) - Diagrama de Flujo del Programa



Anexo 2 (Secuencial) − Catálogo de funciones

Funciones Principales del Código Secuencial:

- **parseArguments(int argc, char* argv[])**
    - Entrada: argc (int), argv[] (char*)
    - Salida: int (cantidad de rosas)
    - Descripción: Procesa los argumentos del programa, verificando si se proporcionó el argumento -q para definir la cantidad de rosas. Implementa manejo de errores para valores inválidos.

- **generateRandomColor()**
    - Entrada: Ninguna
    - Salida: SDL_Color (color aleatorio)
    - Descripción: Genera un color aleatorio que se utiliza para pintar las curvas de Rosa Polar.

- **generateRosaPolar()**
    - Entrada: Ninguna
    - Salida: RosaPolar (estructura con los parámetros de la rosa)
    - Descripción: Genera una rosa con parámetros aleatorios, incluyendo el número de pétalos, escala, y color.

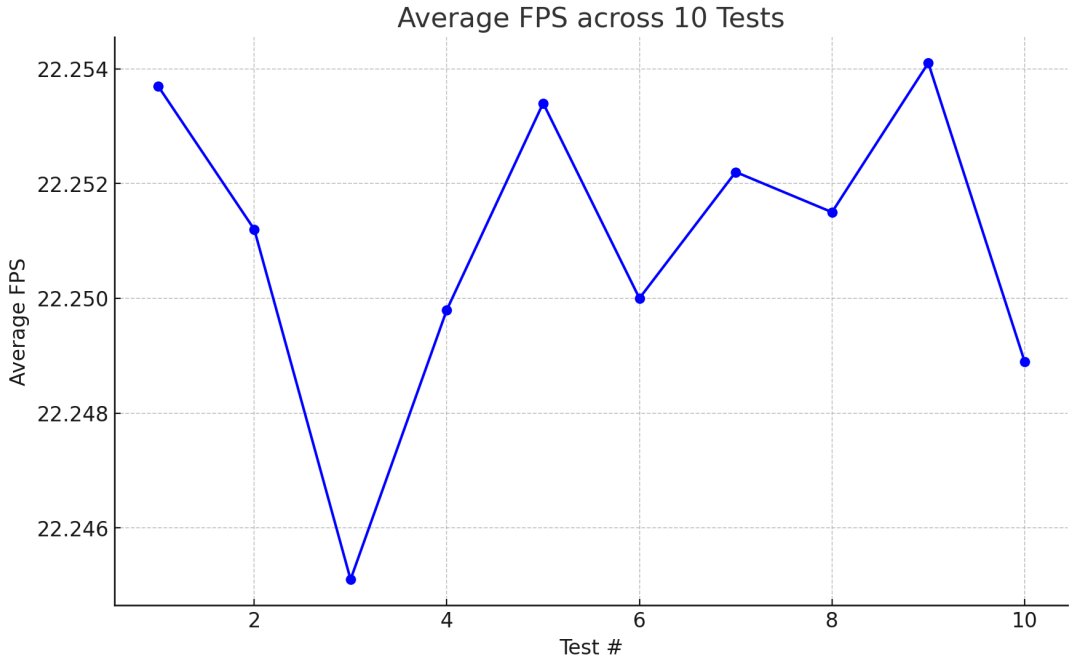
- **drawMovingPoints(SDL_Renderer* renderer, const RosaPolar& rosa, float rotation_angle)**
    - Entrada: renderer (SDL_Renderer*), rosa (const RosaPolar&), rotation_angle (float)
    - Salida: Ninguna
    - Descripción: Dibuja los puntos de la curva de Rosa Polar en la pantalla, considerando la rotación.

- **main(int argc, char* argv[])**
    - Entrada: argc (int), argv[] (char*)
    - Salida: int (código de salida)
    - Descripción: Función principal que coordina la ejecución del programa, desde la captura de argumentos hasta la generación de rosas, cálculo de puntos, renderizado, y generación de reportes.

Anexo 3 (Secuencial) - Bitácora de Pruebas

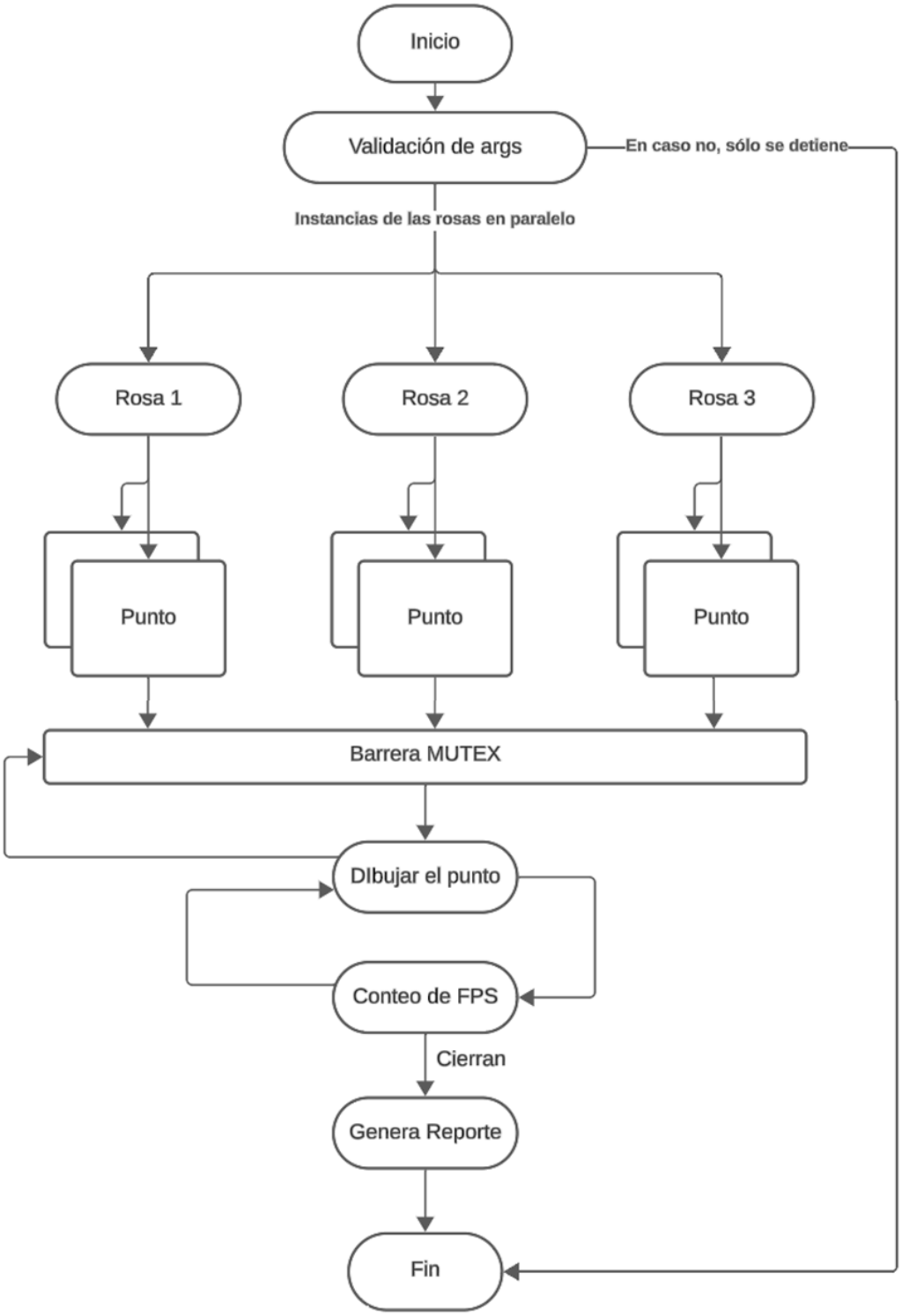
Utilizando los datos generados en seq_report.txt se generó esta tabla y la gráfica

Test #	Average FPS	Minimum FPS	Maximum FPS	1% Low FPS
1	22.25	17.4	24.69	17.4
2	22.251	17.5	24.65	17.5
3	22.245	17.6	24.66	17.6
4	22.249	17.5	24.68	17.5
5	22.253	17.4	24.67	17.4
6	22.25	17.3	24.66	17.3
7	22.252	17.5	24.65	17.5
8	22.251	17.4	24.67	17.4
9	22.254	17.3	24.68	17.3
10	22.248	17.5	24.69	17.5



En base a estos resultados en las pruebas podemos decir que el programa secuencial de curvas de Rosa Polar nos ofrece un rendimiento estable con una ligera variabilidad en el FPS. Los valores de FPS promedio y máximo sugieren que el algoritmo funciona de manera eficiente, mientras que el FPS mínimo y el 1% Low FPS indican áreas donde se podrían aplicar optimizaciones adicionales para mejorar la fluidez en condiciones de mayor carga computacional.

Anexo 1 (Paralela) - Diagrama de Flujo del Programa



Anexo 2 (Paralela) − Catálogo de funciones

- **parseArguments(int argc, char* argv[])**
    - Entrada:argc (int): Número de argumentos. argv[] (char*): Lista de argumentos.
    - Salida: int: Cantidad de rosas a generar.
    - Descripción: Procesa los argumentos del programa, verificando si se proporcionó el argumento -q para definir la cantidad de rosas. Implementa manejo de errores para valores inválidos o fuera de rango.

- **generateRandomColor()**
    - Entrada: Ninguna.
    - Salida: SDL_Color: Estructura que contiene los valores RGB y de opacidad (alpha) para un color aleatorio.
    - Descripción: Genera un color aleatorio para pintar las curvas de Rosa Polar. Los valores de los componentes de color (rojo, verde y azul) se seleccionan aleatoriamente entre 0 y 255.

- **generateRosaPolar()**
    - Entrada: Ninguna.
    - Salida: RosaPolar: Estructura que contiene los parámetros de la rosa, incluyendo número de pétalos, escala, color, coordenadas del origen y velocidad de rotación.
    - Descripción: Genera una rosa con parámetros aleatorios, como el número de pétalos (k), la escala, el color, la posición en pantalla (origen), y la velocidad de rotación.

- **calculateRosePoints(const RosaPolar& rosa, float rotation_angle)**
    - Entrada: rosa (const RosaPolar&): Estructura que define los parámetros de la Rosa Polar. rotation_angle (float): Ángulo de rotación de la curva.
    - Salida: std::vector<Point>: Vector que contiene las coordenadas x e y para los puntos calculados.
    - Descripción: Calcula los puntos de la curva de Rosa Polar en paralelo utilizando OpenMP. La paralelización ocurre sobre el bucle que calcula cada punto de la curva, distribuyendo el trabajo entre varios hilos para mejorar el rendimiento.

- **fillPetalsWithLines(SDL_Renderer* renderer, const std::vector<Point>& points, int x_origin, int y_origin)**
    - Entrada: renderer (SDL_Renderer*): Renderizador de SDL utilizado para dibujar en la pantalla. points (const std::vector<Point>&): Vector de puntos calculados que representan la curva de la rosa. x_origin (int): Coordenada X del origen de la rosa. y_origin (int): Coordenada Y del origen de la rosa.
    - Salida: Ninguna.
    - Descripción: Dibuja los pétalos de la rosa conectando el centro de la rosa (x_origin, y_origin) con cada punto de la curva utilizando líneas. Este método es simple y eficiente, ideal para el renderizado de pétalos.

- **drawRoseContour(SDL_Renderer* renderer, const std::vector<Point>& points)**
    - Entrada: renderer (SDL_Renderer*): Renderizador de SDL utilizado para dibujar en la pantalla. points (const std::vector<Point>&): Vector de puntos que representa la curva de la rosa.
    - Salida: Ninguna.
    - Descripción: Dibuja el contorno de la curva de Rosa Polar conectando los puntos consecutivos de la rosa y cerrando el contorno al conectar el último punto con el primero.

- **main(int argc, char* argv[])**
    - Entrada: argc (int): Número de argumentos. argv[] (char*): Lista de argumentos.
    - Salida: int: Código de salida.

Anexo 3 (Paralela) - Bitácora de Pruebas

Test #	Average FPS	Minimum FPS	Maximum FPS	1% Low FPS
1	28.90	25.74	29.76	25.74
2	28.85	25.60	29.80	25.60
3	28.95	25.78	29.78	25.78
4	28.92	25.72	29.74	25.72
5	28.91	25.70	29.79	25.70
6	28.89	25.68	29.77	25.68
7	28.87	25.65	29.75	25.65
8	28.93	25.71	29.73	25.71
9	28.90	25.69	29.76	25.69
10	28.88	25.67	29.74	25.67

Los resultados demuestran que la paralelización en este caso ofrece beneficios en términos de rendimiento debido a la necesidad de sincronización al acceder al renderizador SDL2. Aunque el cálculo de los puntos de la curva se realiza en paralelo, el mutex utilizado para proteger el renderizado reduce la ventaja que podría haber ofrecido la paralelización.

Referencias

Barney, B. (2023). Introduction to OpenMP. Lawrence Livermore National Laboratory. https://computing.llnl.gov/tutorials/openMP/

OpenMP Architecture Review Board. (2023). OpenMP Application Program Interface. https://www.openmp.org/specifications/

SDL. (2023). Simple DirectMedia Layer. https://www.libsdl.org/