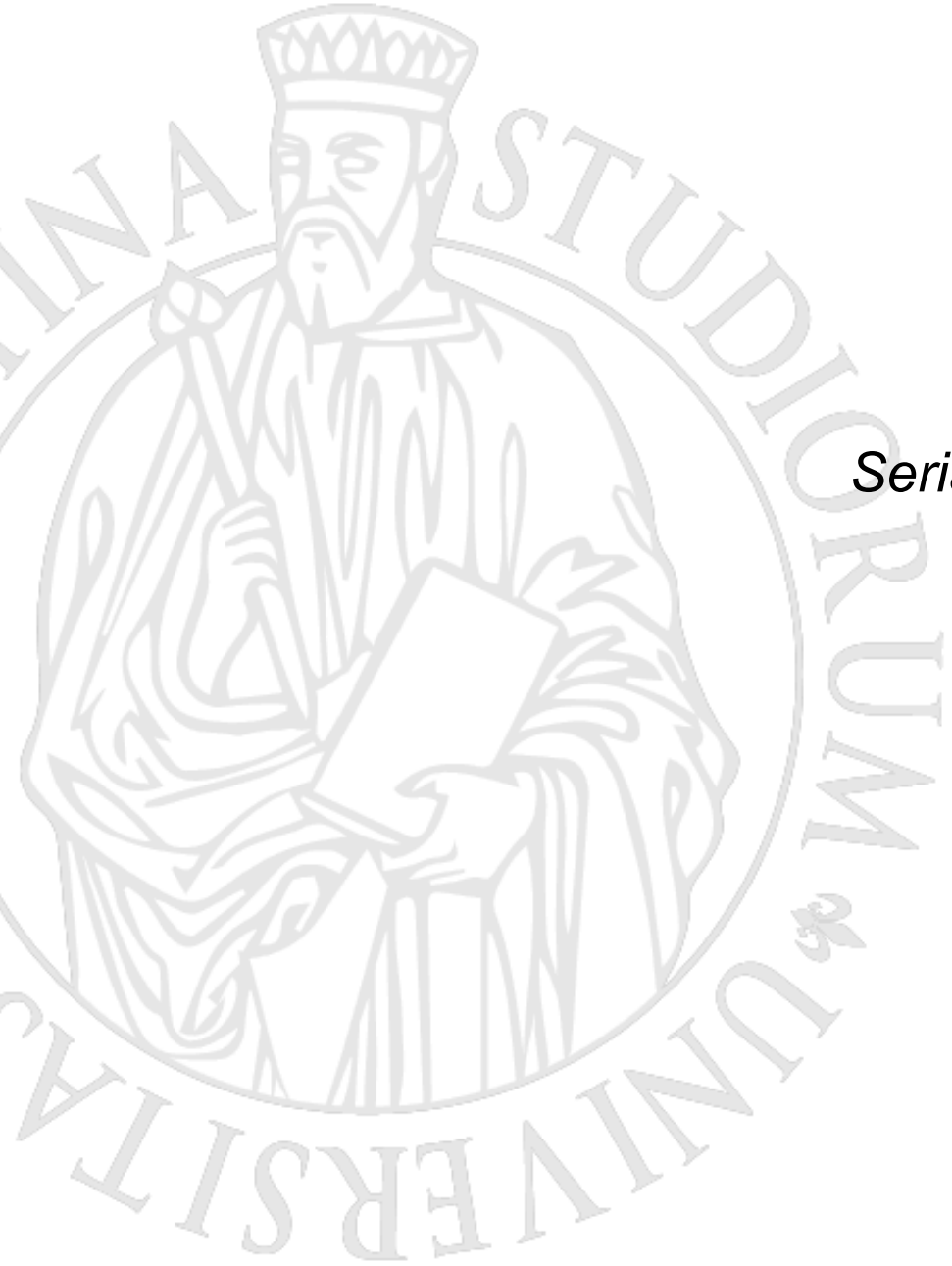


Rendering Circles

Serial vs Parallel Approach with OpenMP for Circle Rendering

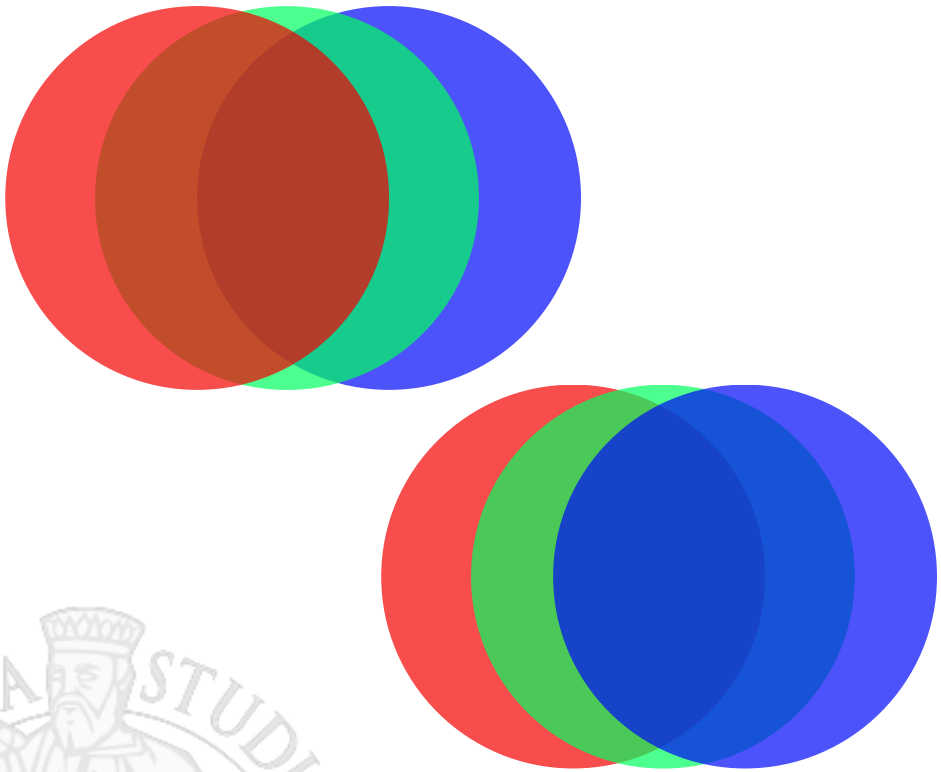
Mattia Marilli

A.A. 2024/2025

Table of Content

INTRODUCTION

Circle Rendering and OpenMP Optimization



Rendering Goal

Develop a C++ system to display overlapping circles with properties like position, size, color, transparency, and depth (z-coordinate).

Depth Perception

Ensure the correct layering of circles, with those closer to the viewer rendered above those farther away, achieving a realistic visual effect.

Sequential and Parallel Versions

Implement both a sequential and a parallel version using OpenMP. The performance of both approaches will be analyzed to assess the impact of parallelization on efficiency while maintaining visual accuracy.

OBJECTIVES

Project Goals and Key Tasks

O1

Develop a rendering system in C++ to display overlapping circles with accurate depthbased layering

O2

Implement a sequential version of the rendering system to establish a baseline for correctness and performance.

O3

Implement a parallel version using OpenMP to improve rendering speed while maintaining the same output as the sequential version.

O4

Evaluate and compare with different kinds of metrics the performance of both versions to understand the impact of parallelization.

SEQUENTIAL IMPLEMENTATION

The program structure

Circle Structure

Defines the attributes of each circle, including its position (x, y, z), size (radius), and color (r, g, b, a).

Random Circle Generation

Creates a specified number of circles with randomized attributes (circle's center, depth (z), radius, color, and alpha transparency using uniform distributions).

Helper Function

Checks if a point lies within a circle's radius. Essential for determining which pixels belong to which circles when rendering the image.

The program structure

Rendering Function

- Initializes a blank width x height image with a white background (RGB = 255, 255, 255).
- Sorts circles based on the z value, so closer circles render on top of those farther away.
- Loops through each pixel in the image to determine if given one lies within the circle's radius. Uses the circle's alpha transparency to blend the circle's color with the background color of each pixel.

Image Saving

The rendered image is saved in the PPM (Portable Pixmap) format. The function writes the image file pixel by pixel.

Optimizing the Rendering Process

Naive Approach

The naive method calculates the distance from each circle to every pixel, resulting in redundant checks as many circles do not affect certain pixels. This approach becomes increasingly inefficient as the number of circles and pixels grows.

Grid-Based Method

The image is divided into a grid of cells, with each cell containing only the circles that intersect it. Pixels in a cell are checked against only these relevant circles, significantly reducing computational effort.

Advantages of the Grid

The grid-based method improves efficiency by narrowing the scope of checks, enhances memory locality by grouping nearby circles, and scales better for large images and datasets, offering optimized performance.

PARALLEL IMPLEMENTATION

Parallelizing the Rendering Process

OpenMP for Parallelization

OpenMP was introduced to parallelize the pixel-processing loop, leveraging multi-core systems to perform computations simultaneously.

Nested Loop Optimization

Using **#pragma omp parallel for collapse(2)**, the height and width loops were merged for better load balancing, ensuring efficient thread utilization.

Independent Pixel Processing

Each pixel is processed independently, eliminating the need for thread synchronization. This ensures accurate color blending and proper handling of overlapping circles.

Parallelizing the Rendering Process

Conditional Multi-Threading

Parallelization is applied only when multiple threads are available (if(numThreads > 1)), ensuring optimal performance without unnecessary overhead on single-threaded systems.

Combined Benefits

The combination of grid-based optimization and OpenMP parallelization significantly accelerates rendering times, achieving scalability and efficiency on modern multi-core systems.

EXPERIMENTATION AND RESULTS

Test Setup

- The experiment assessed the impact of parallelization on **rendering performance** by comparing a sequential version (single-threaded, OpenMP disabled) as a **baseline** with a parallelized implementation.
- Tests were conducted on an **Apple M1 chip system featuring 8 cores (4 high-performance and 4 high-efficiency)**. The number of threads was varied (1, 2, 4, 8, 16) to evaluate scalability and efficiency on this heterogeneous architecture.
- The test images were fixed at 2000x2000 pixels, with the number of rendered circles ranging from 10 to over 100,000. This range demonstrated how **performance scales** with increasing workloads.
- Rendering time, excluding file writing, was measured to calculate performance metrics like speedup and efficiency. **Multiple runs were averaged** to ensure consistent and reliable results.

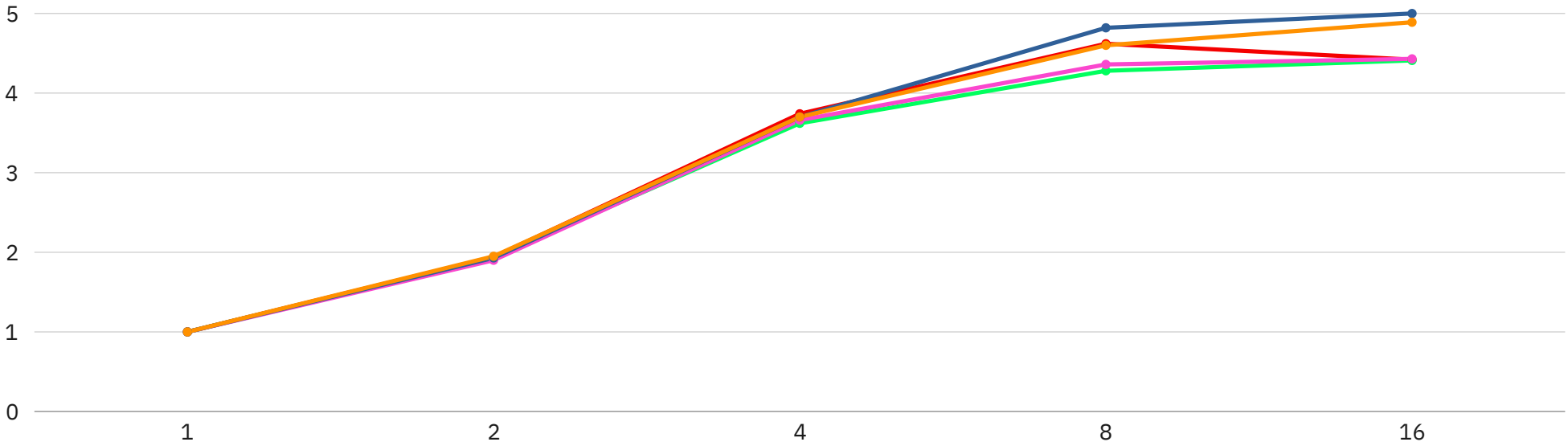
Speedup

The following graph illustrates the speedup achieved with increasing numbers of threads for different circle counts.

Complete Data

Click Here

It shows the relationship between the number of threads (from 1 to 16) and the corresponding speedup for various numbers of circles (10, 100, 1000, 10000, 100000).



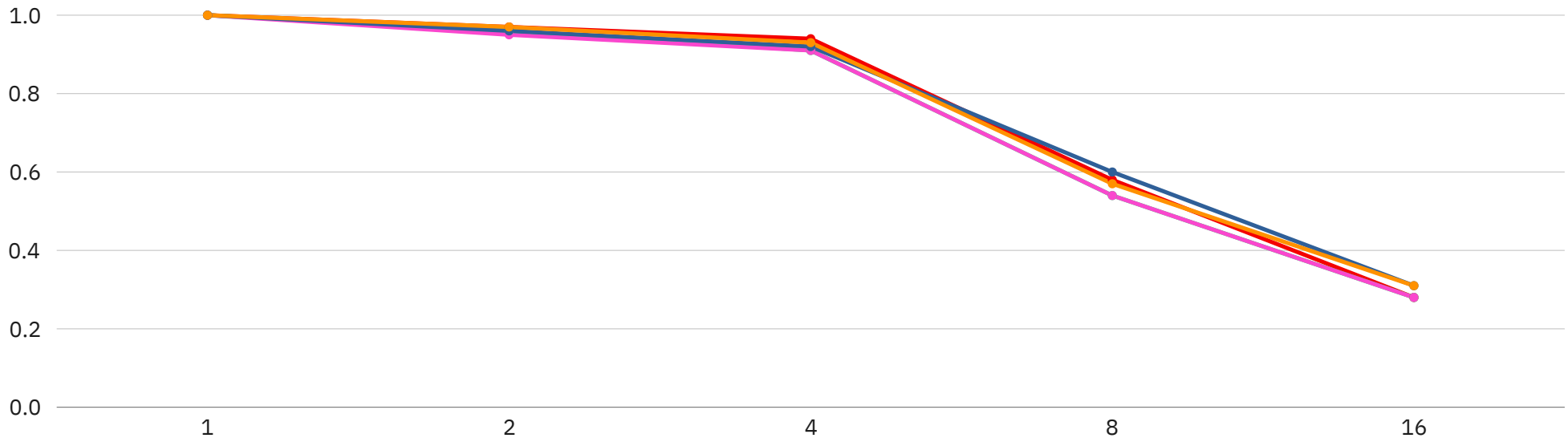
Efficiency

The following graph illustrates the efficiency achieved with increasing numbers of threads for different circle counts.

Complete Data

Click Here

It shows the relationship between the number of threads (from 1 to 16) and the corresponding efficiency for various numbers of circles (10, 100, 1000, 10000, 100000).



Results Analysis

Rendering Time Improvement

As the number of threads increases, the rendering time generally decreases, showing that parallelization improves performance.

Speedup from 1 to 4 Threads

The speedup is most significant when moving from 1 to 4 threads. Beyond 4 threads, the speedup continues to improve but at a slower rate.

Efficiency Decline with More Threads

Efficiency decreases as the number of threads increases, especially when the thread count exceeds the number of available cores, indicating diminishing returns on parallelization.

Results Analysis

High-Efficiency Core Impact

Between 4 and 8 threads, the speedup continues to increase but at a slower rate. This is due to the involvement of high-efficiency cores, which are less performant than high-performance cores.

No Improvement Beyond 8 Threads

For 16 threads, the performance remains comparable to 8 threads because the M1 chip does not support hyper-threading, and additional threads do not provide further performance improvements.

Workload Dependency on Speedup

The best speedup occurs when rendering a high number of circles, as the overhead of thread creation is outweighed by the benefits of parallelization. For smaller workloads, thread management overhead limits the speedup.

Thank you!

Project Repository

Click Here