Assignment 11: Parallel Raycasting
Zander Schatzberg, 12/11/2024
CS 3410

Implementation Summary
This project had me implement three strategies for rendering light illumination effects in a 2D scene. Below is a quick breakdown of each approach.

1. Sequential Implementation (raycast_sequential)

The sequential implementation computes the illumination for each pixel in the scene by sequentially iterating over all lights for that pixel.

2. Light-Parallel Implementation (raycast_parallel_lights)

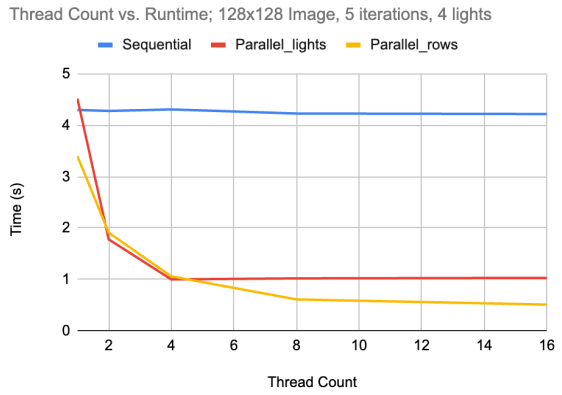
The light-parallel implementation divides the computation by assigning subsets of lights to different threads, which compute the illumination for all pixels in parallel. This was accomplished by defining a helper function ("parallel_lights_worker(void* arg)"), which is passed a struct containing relevant information for computing a certain subset of lights.

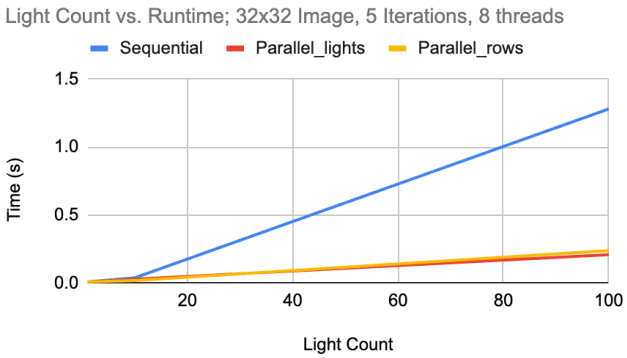
3. Row-Parallel Implementation (raycast_parallel_rows)

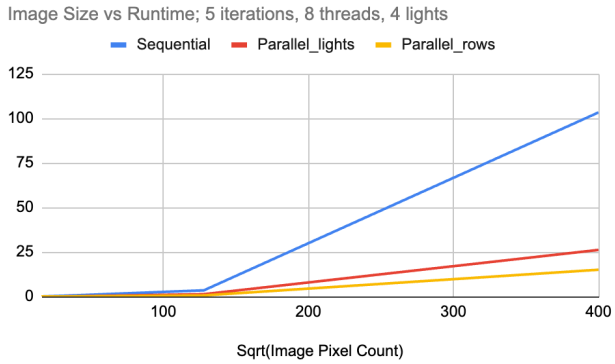
The row-parallel implementation divides the computation by assigning subsets of image rows to different threads. Each thread processes its assigned rows independently, computing illumination for all pixels in those rows. This was accomplished similarly to parallelizing the lights, just with rows instead of lights.

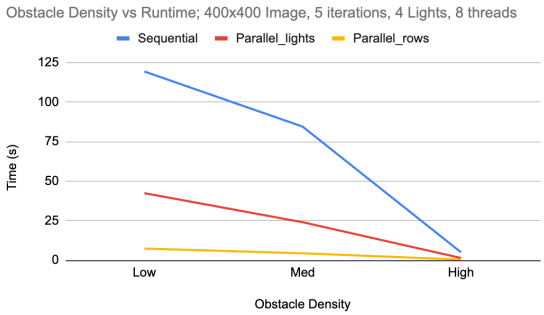
Reflecting on my implementation, I could have designed helper functions that made use of the already written sequential implementation. Instead, I adapted large potions of its code into new functions. This led to quite a bit of redundancy.

Experimental Design
The parameters for each experiment are listed in the title. When choosing these parameters (such as light count, image size, and number of iterations), I played around with simple cases and struck a rough balance between accuracy and efficiency. When measuring the impacts of image density, I simply used the provided files large_blank.png, large.png, and large_dense.pn









Link to data:
https://docs.google.com/spreadsheets/d/1KrIT9YHh5ZpjpfpWHihAr5jXNJDAa8A2jc_O2a7Pxy0/edit?gid=0#gid=0

Analysis

None of my findings were surprising. In short: sequentially performed poorly when tracing images larger than 128x128, while the parallelized implementations performed better. The light-parallelized raycaster could only take advantage of the processor's threads when the scene had many lights while breaking the image down and processing it by rows allowed the program to run more quickly, regardless of the number of lights. The performance of the light-parallelized and row-parallelized implementations does converge when processing a scene with a light count equal to or greater than the number of threads. This is expected, as both implementations fully utilize each thread under these circumstances.

If I were deciding between an application's parallel-lights and parallel-rows implementations, I would elect to parallelize the rows. The parallel-rows implementation can take full advantage of the CPU's cores in a broader range of circumstances - not just when the light count is ≥ the number of threads available.