Project Milestone

Rose Liu (roseliu), Elly Zheng (ellyz)

April 15, 2025

Webpage: https://github.com/ellyzh/crowd-simulation

Schedule:

Week	Plan	Person
April 2 - April 9	1. Revised project goals and implementation a. Redefined project statement from crowd control to crowd collision 2. Complete sequential implementation of crowd collision simulation 3. Create visualization tool to debug unexpected behavior of collisions 4. Write small input files (grid size of 128x128) to test the validity of the code	1. Rose/Elly 2. Elly 3. Rose 4. Rose
April 9 - April 15	1. Integrate quadtree-based spatial partitioning 2. Parallelize algorithm with OpenMP 3. Finish project milestone report	1. Rose 2. Elly 3. Rose/Elly
April 16 - April 19	1. Add functions to verify that the code is working as expected 2. Analyse bottlenecks and explore optimization techniques a. Parallelize building of quadtree and collision check algorithm	1. Rose 2. Elly
April 20 - April 22	1. Analyse bottlenecks and explore optimization techniques a. Find ways to reduce the number of collision checks 2. Test code with larger input files (ex. grid size of 4096x4096)	1. Rose 2. Elly
April 23 - April 26	1. Experiment with PSC to test scalability 2. Gather results and examine key metrics	1. Rose/Elly 2. Rose/Elly
April 27 - April 29	1. Finish our final report and prepare for presentation 2. Finalize Github code and documentation	1. Rose/Elly 2. Rose/Elly

Summary of Work Completed:

As of our project milestone report, we have finalized the serial implementation (serving as the base that we are parallelizing) and have completed a functional parallelized implementation of our crowd movement algorithm. For each iteration, all agents take a step in their given direction. We check for collisions by comparing a given agent's next step to the next step of all the other agents on the grid. If any agents collide (try to move onto the same grid cell), the agents will return to their original position for that iteration. Our paralleized version utilizes a quadtree and OpenMP for agent collision handling. The quadtree spatially partitions the grid, and the agent only checks nearby agents to see if they will collide rather than all agents on the grid, greatly reducing computation time. Through iteration and testing, we have also determined and enacted design choices to ensure that our project is a good candidate for parallelization.

Additionally, we have developed a visualization program using SDL that plots agent positions as they move and a quadtree printing function to aid in debugging and detecting unexpected behaviours. In our implementation of the quadtree itself, we also included cases and wrote our quadrant checks such that we ensure completeness in the return of all necessary agent values (collidable agents).

Completion of Goals:

In our proposal, we had planned to start experimenting with non-blocking communication and other load-balancing strategies at this point. However, we have only accomplished up to the previous week's goals. This is a result of several roadblocks we ran into, as after we implemented our serial implementation according to our proposal's specification, where agents (defined in our proposal as independent moving entities on a grid) have fixed goals that they were moving toward, we realized that the algorithm was not very parallelizable and required a large amount of atomicity and locking to ensure correctness. Additionally, there was no intuitive/broad way to parallelize it (only granular), as the agents are meant to operate independently, but the grid occupancy limit we proposed prevented that and forced our program to constantly fetch and lock the most current occupancy value of the grid.

Thus, we decided to transition away from focusing on determining an agent's next movement and, instead, focus on interactions between agents. Instead of having fixed goal coordinates, our agents are now only given a starting position and a starting direction, and their interactions with the grid bounds and each other determine the direction in which they are moving. Then, using our original idea of a quadtree, we parallelize the method in which we check the agents that can potentially collide. This worked well in terms of making our idea fit the scope of the 15-418 final project better and provided us with clearer ideas of how to do it. As noted in a later section, we have also seen positive speedup results with this project idea alteration.

We believe that we should still be able to achieve all the goals that we have initially outlined in our proposal. Although the area for parallelization has changed, we still want to achieve the same goals, including having a high-performing parallel implementation as well as ensuring that it is scalable for high thread counts and large grid sizes. Additionally, we have already achieved one of our "hope to achieve" goals with our visualization tool. We are optimistic about our program's ability to perform well on large grids, as that is an inherent trait of quadtrees, and are looking to find ways to further optimize our algorithm before testing on PSC, as planned.

Poster Session Presentation Plans:

As we did in our past programming assignments, we plan to provide graphs to demonstrate the amount of speedup we are achieving with varying thread counts, as we are using OpenMP to parallelize our code. This will allow us to clearly present positive results and the scalability of our algorithm. We would test these with a handful of different inputs (grids and agent counts of varying sizes) with varying thread counts.

The visualization program we have developed will be a valuable addition to our poster presentation, whether that be through a live or a pre-recorded demonstration. This will help us intuitively explain our project to our classmates and point out agent behaviour as necessary. Additionally, we may include a visual of our quadtree as it is grown and utilized, as we have special casing to deal with edge cases.

Preliminary Results:

Compared to our serial implementation, our parallel implementation shows some computational speed-up. We ran both our implementations on two input files with varying grid sizes and numbers of agents for 500 iterations, experimenting with different numbers of threads on our parallel version. We plan to expand the size of our grid and the number of agents to take advantage of a quadtree's strengths in grid locality and find other ways to optimize and parallelize our code so we can observe improved performance.

Computation Time (seconds)	**small_test.txt (16x16 grid with 7 agents)**	**medium_test.txt (128x128 grid with 64 agents)**
Sequential	0.0274432080	0.3100250830
Quadtree with OpenMP (2 threads)	0.0282889930	0.2045353680
Quadtree with OpenMP (4 threads)	0.0194684320	0.1485483640
Quadtree with OpenMP (8 threads)	0.0174304310	0.1289310930

Concerns:

A primary concern we have is ensuring correct and efficient collision checks between agents in different quadrants. Since the agents on the grid are partitioned into different nodes of the quadtree based on their location, there is a possibility that agents who are on or near the edges of these quadtree boundaries will collide with an agent in an adjacent quadrant. To resolve this, we add the agents who are near boundaries into all adjacent nodes so these agents can be accounted for when checking for collisions. This has occasionally resulted in infinitely growing trees with small grid sizes. Therefore, we are concerned about the completeness of our algorithm and ensuring correctness in the placement of agents and collision handling. We have added functions to check that all agents are moving within the bounds of the grid, but we want to implement additional functions to check that agents are colliding properly.

Another concern we have is the inherently serial creation of our quadtree itself. With our current implementation, our quadtree is mainly sequential, it is built node-by-node to ensure correctness and avoid unexpected behaviors. However, we want to investigate ways in which we can perform the this building or insertion of agents into nodes in parallel to better optimize our code. We currently do not have any promising ideas on how to do this, and this is something that we will tackle in the near future.

Sources

https://pvigier.github.io/2019/08/04/quadtree-collision-detection.html
https://examples.libsdl.org/SDL3/renderer/04-points/