Final Project Proposal

Rose Liu (roseliu), Elly Zheng (ellyz)

March 26, 2025

Title: Parallel Quadtree for Large-Scale Crowd Simulation

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

Summary: Inspired by the Barnes-Hut simulation, we want to create a parallel quadtree algorithm to simulate a large crowd.

Background: Simulating large crowds efficiently is a computationally complex problem, as each individual must navigate toward a goal while interacting with their environment and other individuals. Traditional grid-based approaches suffer from high computational overhead due to frequent collision checks and movement updates, making them inefficient for large-scale simulations. Thus, inspired by the Barnes-Hut algorithm, we propose using a quadtree-based approach to optimize crowd movement. A quadtree would allow for efficient spatial partitioning and reduce unnecessary computations, which are essential in reducing computation time.

In our simulation, each individual has an occupancy value (e.g., 1 for a baby, 2 for a child, 3 for a teenager, and 4 for an adult), and the grid has a maximum allowable occupancy value per cell. This means that multiple people can share the same space only if their combined occupancy values do not exceed the constraint. For example, a baby and a teenager may share a grid cell (occupancy value = 4), but an adult and a teenager cannot. This constraint influences movement decisions, thus requiring a balance between individual movement goals and grid occupancy limits.

Our method is outlined as follows:

1. Determine movement per timestep
    a. The selection is based on a combination of distance to the goal and ensuring the movement does not break occupancy constraints
2. Recursively construct a quadtree

    a. A region is subdivided until it contains at most four people, ensuring efficient local processing, and that region contains only nearby people
3. Region assignment
    a. Use the quadtree to assign threads to each partitioned region
    b. Calculate the next position of the people inside based on the occupancy grid
        i. Threads operate independently on non-overlapping regions, avoiding contention
4. Locking and unlocking for grid control
    a. A thread locks a cell before attempting to move an individual into it and unlocks once the move is finalized to prevent conflicts
    b. If a person's move would exceed the max occupancy value, they are forced to choose an alternate path
    c. This ensures that movement remains valid while allowing parallel execution
5. Multi-threading quadtree updates
    a. All partitions of the quadtree are updated simultaneously across different threads
    b. After each timestep, the quadtree is reconstructed only where necessary, avoiding full recomputation

This problem is computationally expensive due to frequent updates to grid occupancy, movement selection based on individual goals and occupancy constraints, and large-scale interactions that require efficient spatial querying and collision avoidance. Our parallelism plan significantly enhances performance by leveraging quadtree partitioning to reduce search complexity, limiting interactions to nearby groups rather than checking against all others. Additionally, efficient load balancing is achieved through dynamic updates to the quadTree structure, ensuring that computational resources are allocated appropriately to crowded areas, preventing bottlenecks, and optimizing overall performance.

The Challenges: One challenge of our program is load balancing. Similar to a real-life crowd, where people may be more densely packed in one area, we want to ensure that all threads receive a balanced workload. This way, each thread can efficiently route the paths of the people, and no threads are idle. Furthermore, there is a high communication-to-computation ratio. As each thread takes on a subsection of the crowd, managing synchronization and communication is important to ensure that each processor is

aware of potential collisions in paths and maximum occupancies in a cell. For example, some people may be moving to the same cell, and depending on the occupancy value, the people may or may not be able to exist in the same cell. Similarly, as a person decides on their next step, the locations of other people in the crowd need to be communicated to make the best decision. In the case where multiple threads are writing to the same cell, race conditions may occur if not properly handled.

Thus, we plan to resolve these issues by exploring the possibilities of a quadtree. With a quadtree, we can split the grid that simulates the crowd recursively until each node has a certain number of people while also preserving locality. Then, each processor receives a more balanced workload. One consideration for the quadtree would be how often to rebuild the tree as people move across the grid to preserve correctness and efficiency.

Resources: We will work off our code from Assignment 3, which employs similar grid-updating strategies. Additionally, we have the beginnings of a quadTree function from the same assignment that we will finish and parallelize. We will also be looking into online resources for Barnes-Hut algorithms to help us get started.

Goals and Deliverables:

<u>Plan to Achieve</u>:

1. Serial implementation of the crowd simulator
    a. Develop a basic crowd simulation using a serial approach to establish a functional baseline
    b. Focus on the core logic of movement selection, ensuring that it respects occupancy constraints
    c. This stage will ensure that the fundamental algorithms work correctly before introducing parallelism
2. Functional parallel crowd simulator
    a. Implement the quadtree-based parallel algorithm using OpenMP for multi-threading
    b. Ensure each thread handles a non-overlapping region of the grid, allowing for independent movement updates
    c. Develop locking mechanisms to handle contention when multiple threads attempt to update the same grid cells, maintaining synchronization

Hope to Achieve:

1. Grid size expansion
    a. Scale the simulator to support larger grids with higher population densities
    b. Investigate the limitations and performance characteristics of the system when operating with large crowds and complex environments
2. Visualization tool
    a. Create a basic visualization tool to depict the crowd movement and quadtree subdivisions.
    b. This could help us identify performance bottlenecks, visually validate correctness, and would be a great tool to have for the final report and poster session.

Platform Choice: We plan to develop the code in C++ and run it on the GHC machines. This feels like the most logical choice, as we are familiar with the language and technology. We may also experiment with the PSC machines to analyse the impact of a higher number of threads and see how our program scales.

Schedule:

Week	Plan
March 26 - April 2nd	- Complete a serial implementation of the Crowd Simulator - Create pseudocode for parallel implementation
April 2nd - April 9th	- Complete a crowd generator function and implement the Crowd Simulator algorithm that correctly uses OpenMP - Integrate quadtree-based spatial partitioning
April 9th - April 16th	- Experiment with non-blocking communication and implement other strategies to balance workload (task queue, task stealing) - Test the scalability of the program across various grid sizes, people counts, and threads - Finish project milestone report
April 16th - April 23th	- Analyse bottlenecks and explore optimization techniques (ie, less quadtree rebuilding, per-thread profile analysis) - Dynamically rebuild quadtree when necessary
April 23th - April 30th	- Experiment with PSC to test scalability - Stretch goals (grid size expansion, visualization tool)

April 30th - May 5th	- Finish our final report, examine key metrics, prepare for presentation - Finalize Github repository code and documentation
May 6th	Poster session