Boids Simulation with QuadTree Optimization

Bella Liu (yuchenli)

Project Goal

The project aims to simulate flocking behavior using the Boids model by Craig Reynolds with C++ and SFML.

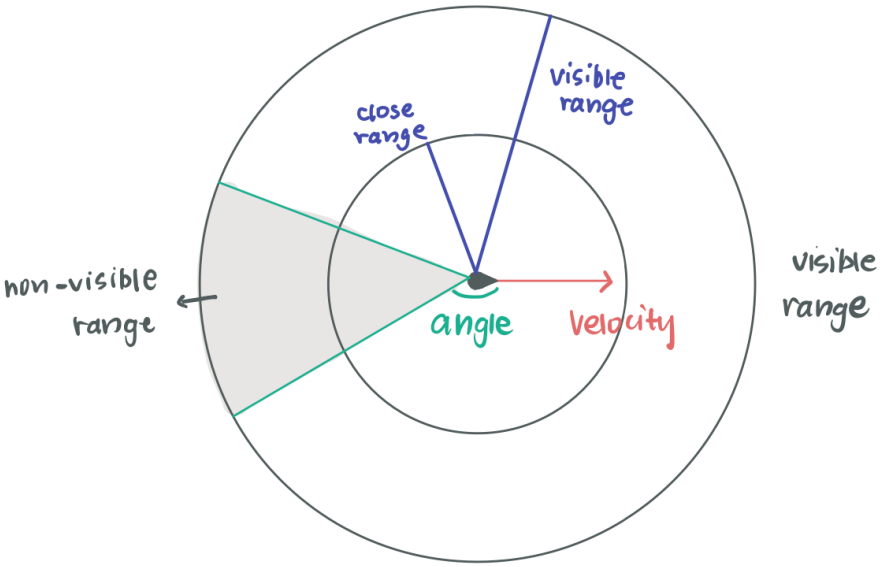
The basic goal of the project is to implement flock simulation in 2D space using the three principles of Boids model: separation, alignment, and cohesion. The project would also aim to achieve following goals:

1) Target and Predator
   The flocks would change their behavior in the existence of either a target or a predator. In the case of target, the flocks would move towards the target; in the case of predator, the flocks would attempt to avoid the predator.
2) Quad Tree Optimization
   Parallelize the code by using quadtree structure, which would speed up the program significantly.

Algorithm and Techniques

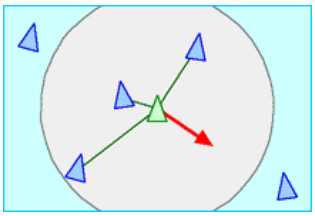
Basic Implementation:

The program would display an infinite animation of flocking simulation based on the Boids model. For each boid, we consider both a close range and a visual range. The boid will attempt to avoid other boids in the close range, and will attempt to match and align with other boids in the visual range.



The flocking behavior would follow the three fundamental principles:

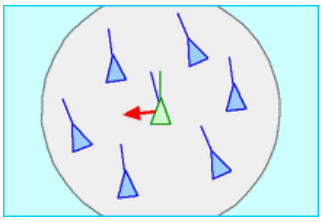
1) **Separation**: every boid will steer to avoid crowding local flockmates



Separation algorithm:

```
steer = (0,0)
For every boid:
      If the dist between boid and otherboid < close_range:
            steer += (boid.position – otherboid.position)
boid.acceleration += steer * separation_factor
```

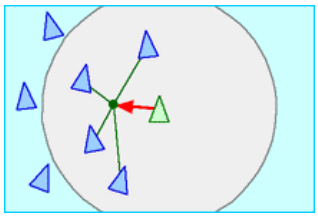
2) **Alignment**: every boid will steer towards the average heading of local flockmates



Alignment algorithm:

```
steer = (0,0), total = 0
For every boid:
      If the dist between boid and otherboid < visual_range:
            steer += otherboid.velocity
            total += 1
steer /= total
steer -= boid.velocity
boid.acceleration += steer * alignement_factor
```

3) **Cohesion**: every boid will steer to move toward the average position of local flockmates



Cohesion algorithm:

```
steer = (0,0), total = 0
For every boid:
      If the dist between boid and otherboid < visual_range:
            steer += otherboid.position
            total += 1
steer /= total
steer -= boid.position
boid.acceleration += steer * cohesion_factor
```

Additional Implementation:

1) Further optimization

a) Visible range angle

To simulate actual visual perception of a boid, an angle value defined to limit its close and visual range. If another boid is outside of its perception, the boid is not considered as a flockmate since it would not be "seen" by the current boid. This perception implementation also allows flocks to form triangle-like groups, which is common in nature.

To achieve this, we add the check-angle requirement for every boid:

```
angle = get_angle(boid.velocity, otherboid.position – boid.position)
if angle < angle_limit:
   update steer
```

b) Randomness

To allow certain level of randomness for the simulation, multiple methods were implementated:

- Change from checking every boid to every other boid to add randomness and increase computation speed.
- Set random acceleration and velocity within a certain range for each boid

c) Parameter tweaking

Once the basic simulation is achieved, the parameters are tested extensively to achieve best result. The final parameters used for flock simulations are as follows:

```
#define BOID_SIZE 6
#define FLOCK_NUM 1000

#define CLOSE_RANGE 50
#define VISUAL_RANGE 150
#define ANGLE_LIMIT 120

#define SEPARATION_FACTOR 0.002f
#define ALIGNMENT_FACTOR 0.09f
#define COHESION_FACTOR 0.0009f
```

2) Target and Predator

a) Target Seeking

When clicking and dragging left mouse in the animation, the program would register the cursor as a target. The flocks within a certain range would move towards the target. This is achieved by updating the steer towards the target:

```
steer += target_position
steer -= boid.position
boid.acceleration += steer * drive_factor
```

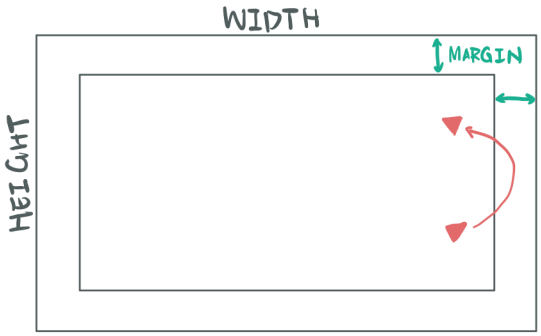
b) Predator Avoidance

Similarly, when clicking and dragging mouse mouse in the animation, the program would register the cursor as a predator. The flocks within a certain range would move away the predator. This is achieved by updating the steer away from the predator:

```
steer += target_position
steer -= boid.position
boid.acceleration -= steer * drive_factor
```

3) Edge Detection

The boids should turn-around at an organic-looking turn radius when they approach an edge of the screen. This is achieved by adding a turn_factor to velocity when they are near the margin.

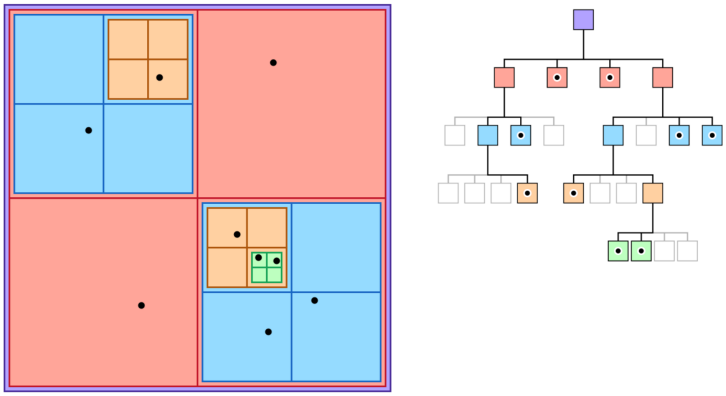



```
if (position.x > WIDTH - MARGIN):    velocity.x -= TURN_FACTOR
if (position.x < MARGIN):            velocity.x += TURN_FACTOR
if (position.y > HEIGHT - MARGIN):   velocity.y -= TURN_FACTOR
if (position.y < MARGIN):            velocity.y += TURN_FACTOR
```

4) Quadtree optimization

The long computation time in the previous implementation largely come from the need to check every other boid for each boid to check if it's in range. This causes $O(N^2)$ complexity.

To decrease computation time and achieve speedup to include more boids in the animation, quadtree structure was implemented.



We generate the quad tree with the following algorithm:

```
QuadTreeLeafSize = max number of particles per node
build_quad_tree(flock, range_min, range_max):
  if num(flock) <= QuadTreeLeafSize:
        return leaf node
  else:
        node = non-leaf node
        pivot = (range_min + range_max) * 0.5f
        assign particles to appropriate children of node based on
  position
        return result
```

When using a quad tree structure, each simulation iteration is separated into two stages: building the quad-tree, and using the quad-tree for simulation:

```
for each iteration
   1) build quad-tree
   2) for each boid b:
        find nearby boids using the quad-tree
        update steer force cause by flockmates in range
        acceleration += steer
        velocity += acceleration
        position += velocity
```

The quadtree implementation can improve the complexity from $O(N^2)$ to $O(\log N)$. This significantly speeds up the computation and improves performance, allowing more boids to be simulated in the animation.

Problems and Solution

The problem encountered during implementation include:

- Inorganic edge detection.

  In the initial implementation, the boids would hit edge, and then proceeds to appear on the other side of the screen. To get better edge avoiding result, the edge detection code was rewrote to the algorithm described above, which allows the boids to organically turn away when they approach an edge.

- Rigid movement

  The initial implementation did not allow randomness, so the boids would clump together and move rigidly after a short period of time. To fix this, random acceleration and velocity were assigned, allowing the simulation to appear more natural.

- Slow computation

  In the initial implemention, only 100-150 boids can be simulated without lagging, due to the inefficient looping algorithm to check for boids in range. The quadtree implementation successfully increased the boids number to around 1000-5000.

Result

The program can simulate flocking behavior based on principles of separation, alignement, cohesion, with implementations of visual perception range, organic edge detection, target seeking and predator avoidance.

The naïve implementation allows around 100-150 boids to be simulated, while the quadtree parallelization can allow around 1000-5000 boids to be simulated without animation lagging, achieving speed up of 10x.

The simulation result is shown in the video.

Further Improvement

The system currently does not have a user-friendly interface. All parameters, such as speed, boid number, etc, are set in the program, which limits user interaction with the simulation.

Another possible improvement is parallelization with OpenMP and GPU. With the quadtree structure implementation, the speedup can be further significantly improved by using OpenMP library and GPU processing. This would even further exponentially increase the number of boids that can be simulated, which would achieve an incredible result of storms formed by flocking.

Reference:

1) Craig Reynolds Boids model: http://www.red3d.com/cwr/boids/
2) Boids algorithm - augmented for distributed consensus: https://vanhunteradams.com/Pico/Animal_Movement/Boids-algorithm.html