ACM SIGGRAPH / Eurographics Symposium on Computer Animation 2020
J. Bender and T. Popa
(Guest Editors)

Volume 39 (2020), Number 8

Multi-Agent Evolution Using Boids

C. Hollier

BC, Victoria. Canada
University of Victoria, Computer Science 473 - Introduction to Computer Animation

1. Abstract

In this project, a simple simulator was created to model multi-agent evolution. Evolution is typically very slow and difficult to observe on the timescale of a human life. However, computer animation provides an excellent framework for modelling evolution with very rapid generations and a high mutation rate to observe the effects quickly. A flocking method called Boids was utilized to create interesting looking flocks that could work together to overcome a predator. Boids provide a simple set of rules that produce realistic looking flocking with emergent behaviour.

The Boids were extended by creating multiple distinct flocks, with each having members created with random initial properties. Each of these attributes influence the survival rate of the Boids, the interactions with other Boids in their flock, and the predators. A food system was also added to force the Boids to eat in order to survive. Each simulation step increases the Boids hunger and their desire to find food.

If a Boid dies, either from starvation or being consumed by a predator, then a new Boid is created by randomly selecting attributes from two other Boids. These attributes also have a chance to change marginally through mutations.

This simulator displayed evolution by increasing or decreasing the value of each attribute over time until the optimal value was found for each. Overall, the goal was achieved of having novel and unpredictable evolution occur based on varied initial parameters.

2. Introduction

Evolution has played a huge part in moving humanity to where it is today. This force is omnipresent, but difficult to observe on a human time-scale. Evolution is typically slow, especially for creatures with low reproductive rates. Life is also very complicated which makes it difficult to accurately model. Evolution is influenced by survival fitness, which encompasses all factors that influence death and reproduction chance. Some of these include the ability to find food, survive the climate in the environment, survive disease, find a mate, and protect offspring, in addition to many more.

Survival chance for social species is also highly dependant on communication and the ability to work together. Humanity became much more successful once people started specializing in skills allowing people to stop worrying about growing their own food and performing research and tool manufacturing. This multi-agent collaboration is much harder to simulate than individuals evolving on their own.

This project seeks to create an animation that considers both of these problems by creating a multi-agent system in which the individuals' properties affect other members of the species. Due to the limited time and resources allotted, it will be impossible to model realistic human evolution, so a simplified scenario will be created.

3. Related Work

The majority of this project was created from scratch with inspiration drawn from real life. Several sources were used to create the initial Boid behaviour and perform performance optimizations to increase the number of Boids the application could support.

Craig W. Reynolds' paper entitled "Flocks, Herds, and School: A Distributed Behavioral Model" provided the groundwork for how Boid flocking forces work and the inspiration for the project [Rey87]. Craig Reynolds' paper described three simple rules that each agent can follow to create emergent flocking behaviour.

Boids have become a well-known method of producing flocking behaviour since Craig Reynolds released his paper in 1987. Many other authors have built on his work since then, such as the paper by Conrad Parker that describes pseudo-code for the Boid flocking [Par07]. This paper was used to get an initial Boid animation running.

Once the Boids were created they needed to evolve. To gain some insight into how evolution should work in code, a paper by *Towards Data Science* was used that described the process of implementing a Genetic Algorithm [Mal17]. It describes these as a search heuristic that mimics natural selection using survival of the fittest. It also describes the five key components of a genetic algorithm such as the initial population, fitness function, selection, crossover and mutation. Each of these key components were built in some capacity in this project.

In addition, a spacial subdivision optimization data structure called Quadtrees was implemented in this project to improve the performance. This data structure was implemented with the help of a Wikipedia article that provided pseudo-code to get started with the basic functions [Wik].

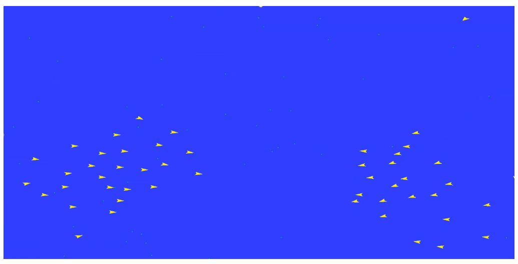
4. Overview

In this project, an evolution simulator was created using Boids to model flocking behaviour for the agents. The project will be described as several different subsystems that work together in the completed product. A main goal of this project was to create a realistic evolution simulation that didn't have a pre-defined target. Different input parameters should produce organic results, instead of prescribing an optimal strategy for the Boids. As such, every advantage given to the Boids has a drawback which makes it difficult to guess the outcome of a given set of parameters without running the simulation.

4.1. Boids

The groundwork of the project uses Boid simulation to create interesting flocks. The standard Boid simulation uses three forces which work together to create flocks of birds with interesting emergent behaviour from simple rules. The first of these rules is separation which provides a repulsive force to Boids that get too close to each other. The force is only applied when the Boids are within a specified radius of each other and gets stronger as the Boids crowd each other more. The purpose of this force is to prevent Boids from colliding with each other. The second force is an attractive force that pulls all Boids to the center of the flock. This force keeps the flocks in a group and will be known as the cohesion force. The final rule causes each Boid to try to align their velocity with the average velocity of Boids in their flock. This causes the Boids to move together in the same direction instead of circling each other. This final force is called the alignment force.

Every 0.01 seconds (simulation step) the three fundamental forces are recalculated for each Boid and added to the velocity of each Boid. This created two problems that had to be addressed: the velocity would increase without bound which created unnatural movement and the Boids would instantly change their velocity to the new direction. The first problem was solved by implementing a maximum velocity for the Boids. After the velocity change was calculated and added to the current velocity of the Boid, the total velocity vector is normalized to create a unit vector in the direction of the Boids movement. This unit vector is then scaled by the maximum allowed velocity to create a vector in the direct direction and with a fixed maximum magnitude.



This image shows a basic flocking pattern created by the fundamental forces.

Figure 1: *Basic Flocking.*

To correct the Boids turning too rapidly, a solution called "turn rate" was created that calculates a portion of the desired velocity update. Turn rate starts with the current velocity of the Boid and the velocity of the Boid that was calculated after adding the three fundamental forces (the desired velocity). The difference between these vectors is computed to get the total change in velocity, which is then scaled by a fraction such as 1/10 to get a portion of the desired velocity change. This change is then added to the current velocity of the Boid to create a final result. This system is ideal because it is simple and computationally inexpensive, but it suffers from creating a variable turn rate depending on the the magnitude of the velocity change. If the Boids desired velocity is significantly different than its current velocity than it will turn faster compared to a Boid that is travelling at a velocity that is close to its desired velocity. If a constant turn rate is desired then the velocity change could be normalized and scaled by a constant using the method described for velocity limiting. For this project, the computation cost of normalizing the turn rate was deemed too expensive so a dynamic turn rate was used.

4.1.1. Multiple Flocks

With the basic Boid implementation complete, this system needed to be further extended with the ability to add multiple flocks of Boids. This feature is important for working towards an evolution simulator and provides several other benefits to the overall system. Creating multiple flocks is a relatively simple procedure that involved splitting the desired number of Boids into several equal sized groups. Each flock then updates its velocity every simulation step based on nearby members of their own flock. This provides a huge advantage of reducing the number of nearby Boids that each individual Boid needs to consider. The nature of the Boid algorithm is $O(n^2)$ which means that if there are n Boids created then each Boid needs to perform n checks. This means that a system with 1000 Boids would need to perform $1000 * 1000 = 1,000,000$ checks per step. Using an example of 5 flocks we can see how having multiple independent flocks can reduce the number of checks per step. If the 1000 Boids were divided between 5 flocks then each Boid would only need to check 199 other Boids. This would result in $200 * 200 * 5 = 200,000$ checks per step. This simple division creates a 5x improvement in the number of checks that are needed per step in this case.

It is worth noting that the multiple independent flock approach loses some granular control that is available when all Boids are contained in a single flock. For example, the Boids are not capable of checking separation forces between themselves and members of other flocks which means that collisions can occur. This problem is due to the multi-threading implementation and could be fixed with the use of mutexes for thread safety. However, due to the large performance impact of this feature, it was deemed beyond the scope of the project.

4.1.2. Predators

In order for evolution to occur, the Boids needed a mechanism to determine which Boids were the weakest. This was accomplished by introducing predators into the system that actively hunt and kill Boids. The predator isn't a member of a flock and is, therefore, not influenced by the flocking forces described in section 4.1. Instead, the predator travels in a straight line until its hunger increases above 20% of its maximum hunger amount. Hunger is described in greater detail in section 4.2.2. Once this threshold is reached, the predator switches into a hunting state and will try to catch food. On each step, every predator calculates the closest Boid to its current location and determines the vector between itself and the prey. A portion of that vector is added to the predator's velocity similar to the "turning rate" method described previously. If a predator is within a small range of a Boid, then the Boid is killed and the predators hunger bar is reset to 0.

The Boids were also given a mechanism to escape the predator. During each simulation step each predator generates a list of Boids that are within its perception radius. A force is then applied to each Boid in the direction of the vector between the predator and the Boid. At first glance this approach seems counter-intuitive since the predator is looking for Boids instead of each Boid checking for a list of predators. However, due to the optimization method that will be described in section 4.5, it was vastly more efficient to check predator interactions in this way and it doesn't change the overall outcome of the simulation.

4.2. Working as a Flock

Everything described so far in this project has discussed how flocking behaviour is created, but doesn't provide a motivation for why the Boids should create flocks. This project aimed to provide motivation for the Boids to flock instead of enforcing arbitrary rules on the agents. This section discusses how the Boid behaviour was modified to provide realistic incentives for Boids to group together.

4.2.1. Herd Safety

In nature, the main reason for birds to flock together is to outnumber predators which increases their survival chance. This project emulated this behaviour by implementing a deterrent for predators hunting habits based on the density of Boids.

An extension of the predator hunter behaviour was created that prevented the predator from pursuing Boids that were part of a large local group. The predator looks at Boids one at a time from each flock, starting at the closest and working outwards until a target is found. For each Boid, a radius is drawn around the potential target and a list of Boids that are also members of that flock is returned. The total mass of these Boids is added up and compared to a threshold level. If the mass is below the threshold then the predator stops looking for prey and uses that target to calculate a change in velocity as described in section 4.1.2. If the total mass is above the threshold then it continues on to the next potential target. The target is recalculated each step of the simulation which means the predator can search for new prey if its first target is too fast to catch or joins together with a larger group of its flock before the predator can catch it.

4.2.2. Hunger and Food

The safety in numbers strategy improves the realism of the animation but violates the goal of not prescribing an optimal strategy. At this point there is no disadvantage to creating large groups and no incentive to leave the group. To fix this problem, a hunger system was introduced that forces Boids to seek food in order to survive. This system works by adding a hunger level to each Boid that is tracked internally along with randomly generated food for the Boids to eat. On every step of the simulation, the Boid's hunger increases proportionally to its speed. That means that fast Boids will starve quicker but are also more likely to be able to outrun a predator.

Once the Boid's hunger exceeds 20% of it's maximum hunger, it starts looking for food in its immediate surroundings. This mechanism works similarly to the hunting algorithm of the predators described in section 4.1.2. On each simulation step, each Boid that has hunger above the threshold looks for the nearest food to itself. A force is then applied to the Boid in the direction of the food proportional to how hungry the Boid is. The force starts small but grows exponentially as the Boid gets hungrier. This force is added to the fundamental forces from section 4.1 before the turning rate and speed limiting are applied.

4.3. Algorithm Overview

This section shows pseudo-code for the algorithms described in the previous sections. The first algorithm shows a very broad overview of the velocity update on each Boid for each time-step. Note that this algorithm has been simplified and is missing the force calculation for avoiding predators.

```
for Each Boid do
    $Boid.velocity \leftarrow randomVelocity$
end for
while true do
    for Each Flock do
        for Each Boid b in Flock do
            $b.hunger \leftarrow b.hunger + b.speed$
            $desiredVel \leftarrow b.velocity$
            $desiredVel \leftarrow desiredVel + cohesion$
            $desiredVel \leftarrow desiredVel + separation$
            $desiredVel \leftarrow desiredVel + alignment$
            if b.hunger > 20 then
                $desiredVel \leftarrow desiredVel + foodForce$
            end if
            $velChange \leftarrow desiredVel - b.velocity$
            $b.velocity \leftarrow b.velocity + velChange * turnrate$
            $b.velocity \leftarrow \frac{b.velocity}{|b.velocity|} * b.turnRate$
        end for
    end for
end while
```

4.4. Evolution

With all the features in place to facilitate the evolution, the only remaining step was to create a system to spawn new offspring. In nature, populations of predators and prey are kept in balance. When the predator population increases too much then they begin to starve due to lack of food which leads to an explosion in the population of the prey species as the predators die off. This increase in food bolsters the predator population in a cycle until an equilibrium is reached. This system works in nature because habitats are very large which means that finding prey becomes more difficult as the population of prey species dwindles. However, the simulation in this project is in a small space which means that locating prey is simple. There are no burrows for the prey to hide in or camouflage of any kind.

Due to the small size of the environment, a straightforward approach was taken. Each time a Boid died from starvation or from a predator, a new Boid would be created on the next simulation step. This isn't very realistic, but provides a good enough model for demonstrating evolution without worrying about the entire predator or prey population dying from starvation. This addresses the question of when to create new Boids but not how they should be created. Before discussing that topic, the Boids needed properties that could evolve. These properties were created as follows:

1. Scalar multiplier for cohesion. This multiplies the force that the Boid experiences from cohesion towards other Boids. The cohesion calculation is still computed as discussed in section 4.1, but the total cohesive force is multiplied by the scalar before it is applied to the Boid.
2. Scalar multiplier for separation. This works identically to the scalar for cohesion, but it is applied to the separation force experiences by the Boid.
3. Scalar multiplier for alignment. This works identically to previous two multipliers but it is applied to the alignment force applied to the Boid.
4. Speed. This has been referenced in previous sections but in this section it is turned into a dynamic property of the Boid instead of a constant. The speed of the Boid is added to its hunger every step, resulting in fast Boids starving quicker. However, Boids with higher speed have the advantage of escaping predators more easily, locating food and other flock members.
5. Mass. The mass of a Boid doesn't do anything by itself but it can alter the behaviour of the Boids interacting with a flock. As described in section 4.2.1, the predator cannot hunt groups of Boids that are above a certain mass. This mass is calculated by summing the individual masses of each of the Boids in the group. However, the turn rate of the Boids is divided by their mass which results in large Boids turning at a slower rate. This makes it more difficult for them to avoid predators.

The reason for creating these trade-offs was to try to create several viable strategies for the Boids. For example, fast Boids could survive by outrunning the predators with their superior speed and turning rate, while slow Boids with large mass could flock tightly together to prevent the predator from selecting them as a valid target. There is also a trade-off in separation and cohesion. Tightly grouped Boids are more likely to reach the threshold mass required to scare away the predator, but cohesive Boids are also more likely to be eaten in rapid succession by a predator if the group splits up.

In the initial prototype, no disadvantage was given for speed which caused the Boid speed to increase forever until the simulation looked very unnatural. The trade-offs provide a more interesting scenario by creating an unpredictable result.

With all the properties of the Boids created, it is finally possible to create evolution. To start, a modification was made to the algorithm that initialized each of the flocks with random attributes. Each flock generated a unique set of values in the range from 0.5 to 1.5 for each of the five attributes and then assigned those values to each of its members. If a simulation was initialized with 5 flocks, then 5 flocks would be created in which the members in each flock have the same attributes but each have different values than the members of the other flocks.

The simulation is then started and run until a Boid dies. At this time, two parents need to be selected to create a new Boid. The first parent is chosen by selecting a random Boid from the flock that the Boid died from. The second parent is chosen as a random Boid from any of the other flocks.

Once the parents are selected, a process called crossover occurs [Mal17]. This process involves creating a new Boid with uninitialized attributes and then filling in the values from the parents. In this project, the child has a uniform chance of inheriting each attribute from either parent. This means that on average, the Boid will inherit 50% of its attributes from one parent and 50% from the other. However, since there are 5 attributes with each having a 50% chance of being inherited, there is a $\frac{1}{2^5} = 3.125\%$ chance of the new Boid being a duplicate of parent 1 and a 3.125% change of it being a duplicate of parent 2.

4.4.1. Mutation

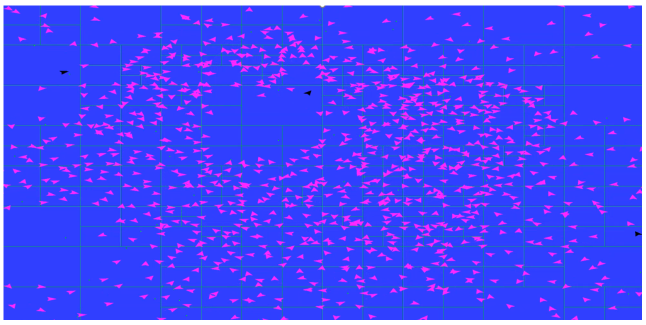
With crossover complete, the Boids technically evolve, but each time the simulation runs there are only as many possible values for each attribute as there are flocks. This means that simulations with 3 flocks only have $5^3 = 125$ possible Boid permutations. It's possible that none of these permutations is optimal for the provided environment which means a more dynamic system is required. To remedy this, a mutation chance is introduced into the system that gives each attribute a chance to change slightly when it is inherited from a parent. In this implementation a mutation chance of 10% was used. After the child randomly selects an attribute from a parent there is a 5% chance that the inherited value will be increased by 0.1 and a 5% chance that it will be decreased by 0.1. This system relies heavily on randomness to produce the best result over time through trial and error instead of prescribing an optimal strategy.

4.5. QuadTrees

All the main mechanics of the system have been discussed at this point, but the performance of the system is lacking. The main problem is the number of entities that each Boid needs to consider to calculate its forces. At present, each Boid needs to calculate the distance to each other Boid to calculate cohesion, separation and alignment forces. The predator also needs to consider the distance to every Boid from every flock to find the closest target and apply repulsive forces to the Boids.

A solution used for this project involved the introduction of a spacial division data structure called QuadTrees. QuadTrees are a tree-based data structure in which each node can contain references to four other nodes, called children nodes. Additionally, the node can contain a list of Boids within it, a center point of the node, and the width/height. Each of these nodes represents an Axis-Aligned-Bounding-Box (AABB) which occupies a region of the screen. When each tree is initialized, it just has a single root node that contains the entire screen. When the maximum capacity of the root node is exceeded by inserting more than five Boids into it, then the root creates 4 children. Each of these children will occupy 1/4 of the screen in the top-right, top-left, bottom-left, and bottom-right regions and have their own capacity of 5.

The image below shows a visualization of a QuadTree for a single flock. The densely populated regions are subdivided into more sections than the areas around the predators due to the low quantity of Boids in those regions.



Quad Trees Example.

Figure 2: *QuadTrees.*

When a new Boid is inserted into the tree, the top level node calls the insertion function on each of its children until one of them returns that it was successfully inserted. Each of the children operates identically to the root node in a recursive fashion. When the insert function is called on the child, it checks if the Boid being inserted is contained in its region. This can be done very easily since the Boid's center is a point mass and the regions of the QuadTrees are AABB. If the Boid is not inside the child then the parent can immediately move on to trying the insert function on the next child. When a child node becomes saturated with Boids then it will create children of its own, splitting into four sub-regions.

Once all the Boids are inserted into the QuadTree, the tree can be queried to find all the Boids in a specific region. For this application, the regions are circular areas around each of the Boids that represent their perception radius. The query functions works with the addition of an intersection function that checks if a given region overlaps with a QuadTree node. If it does, all the Boids contained by the QuadTree are appended into a list of nearby Boids and the query function is recursively called on the QuadTree's children. Each Boid then checks the distance to each of the Boids in the list returned by the query instead of checking distance to every Boid.

This query function converts the algorithm from a $O(n^2)$ into a $O(n * log_2 n)$. That means that a system using 1000 Boids would go from computing $1000 * 1000 = 1,000,000$ checks to using $1000 * log_2 1000 \approx 10,000$ which is a 100x performance improvement. The QuadTree was also made to work with generic objects that have a position property. This allowed the food system to use QuadTrees to save on Boids needing to calculate the distance to every item of food.

5. Evaluation

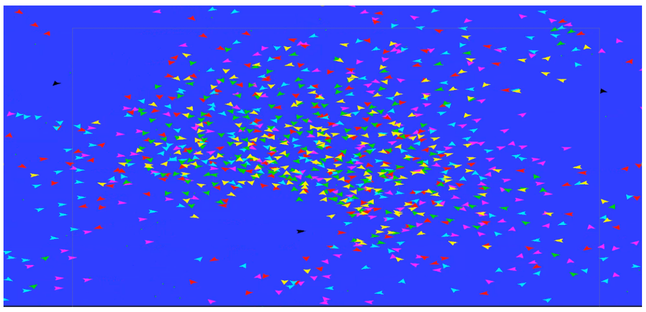
In order to evaluate the project, the execution environment must first be understood. For this project, all simulations were compiled and run on a Windows 10 desktop computer with a Ryzen 5 3600 CPU, a RX 5700XT GPU made by AMD, and 16GB of 3600Ghz RAM.

This project contained a large number of input parameters that can be tuned to produce different results. Some of these include the following

- The range of default values provided to the Boids for mass, separation, alignment, cohesion and speed.
- The perception radius of the predator looking for Boids to eat and the perception radius of the Boids looking for food.
- The range in which Boids have their fundamental forces altered by other Boids.
- The quantity of food in the environment.
- The rate at which predators and Boids get hungry.
- The threshold value for mass that prevents a predator from targeting a Boid in a flock.
- The strength of the force that pushes Boids away from the predator.
- The mutation rate used when creating offspring.
- The number of Boids and predators at the start.

Each of these parameters will affect how the simulation looks and how the properties of the Boids evolve over time. Due to the number of changeable parameters and the number of values each of these parameters can have, it is impossible to demonstrate the results for each of them. However, this section will show the results of several hand-picked examples that show general trends to get an idea of how the system works.

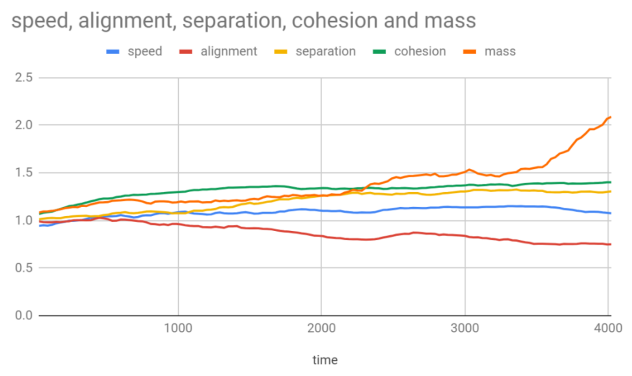
With the introduction of QuadTrees and multi-threading, the system can handle around 1000 Boids split into 5 flocks. An example frame of this simulation can be seen in the figure below. It can be observed that Boids are generally flocking with members of their own colour and avoiding the predator (black) Boids. In this run of the program, the yellow Boids had a low initial speed which can be seen by the yellow Boid that is very close to the predator in the middle of the screen.



This image shows an example of a simulation step in the final product.

Figure 3: *Final Result.*

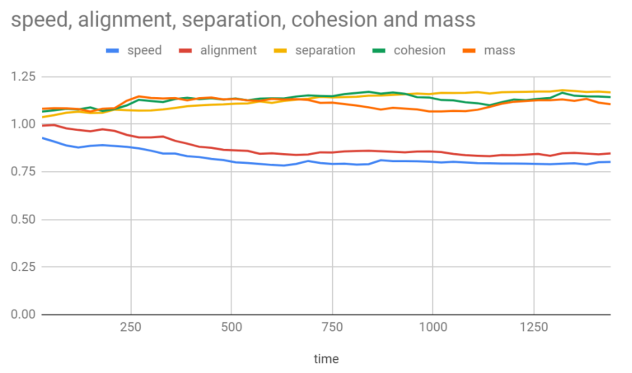
To see more clear results from the evolution, a CSV file was created that contained the average Boid attributes over time. These attributes were logged every 30 seconds and loaded into spreadsheet software to generate a graph over time. The next figure shows the attributes over time with the configuration variables considered "normal" for the system. The simulation was run with 5 flocks of 50 Boids each for 45 minutes. After this point the attributes leveled off.



An example of the evolution process.

Figure 4: *First Evolution.*

This figure shows that this configuration of the parameters favours a higher average mass, cohesion and separation while discouraging alignment between the Boids. In the next example, the rate of starvation was increased so that Boids starve three times faster. This forces Boids to leave the flock more often in order to find food. In this run, speed become severely detrimental because of the starvation multiplier that speed causes. It was better for the Boids to move slowly as a group instead of increasing their speed to find food more quickly.



An example of the evolution process with faster starvation.

Figure 5: *Fast Starvation Evolution.*

Frame-rate is not the best characteristic to determine performance in this project. With 5 Boids in the system, the simulation ran at around 12,000 frames per second (FPS). With the introduction of 100 Boids, the system ran at 8000 FPS, 2500 FPS for 500 boids, 1400FPS with 1000 boids, and 150FPS with 10,000 boids. Running on a 60Hzh monitor, 150 FPS is still enough to produce a smooth animation, but in actuality the 10,000 boids simulation looks choppy. This is because the time in the simulator is falling substantially behind the real time.

To measure the true performance of the system, the ratio between real time and simulation time was used. This measures how many time-steps the simulator can process per second of real time. It was found that 1200 boids in the point at which the simulator and real time match up. That is, the simulator produces 1 second worth of simulation steps for every 1 second of real time that passes.

6. Future Work

In the future, this system can be greatly expanded and is continuing to see active development. The biggest area of future work lies in the process of creating offspring in the evolution process. At this point the child creation process isn't very realistic since the parents are chosen at random. It could be improved by considering Boid proximity when creating children and also by creating a fitness function to decide which Boids get to reproduce. This could work by considering the quantity of food that Boids have eaten recently and increasing the likelihood of Boids reproducing based on their ability to find food. Additional parameters could also be added that give Boids a tune-able value for their breeding priority. Lastly, the Boids could be given a family structure and a sex that prevents Boids from breeding with other Boids in their immediate family or with members of the same sex. This would increase the genetic diversity and be more realistic to how reproduction works in real life.

In addition to these features, the simulation could be made 3-dimensional and increase the visual distinctness of the Boids based on their attributes. For example, the Boids could be scaled in size based on their mass, increased in sleekness based on their speed or otherwise visually change to see the attributes.

7. Conclusion

In this project, a multi-agent evolution simulator was created using Boids. Given a set of initial parameters, novel and unpredictable results are created through an evolution process. This process worked by creating offspring that inherit attributes from their parents in conjunction to a mutation mechanism.

While it has its limitations, the overall number of Boids in the system is greater than originally expected due to the effectiveness of the optimization techniques employed. These techniques included the spatial subdivision algorithm called QuadTrees that increased the performance by around 30x and the multi-threading which boosted the performance after the QuadTrees by an additional 2x.

The performance optimizations came at the cost of increased complexity involving communication between threads and interactions between flocks.

In the future, this project could be improved by adding a greater depth of communication between the flocks. This would require rethinking the multi-threading performance optimizations. Additional parameters could also be added to the Boids that increase the realism of the reproduction process such as breeding cool-down, and a mate finding system that is based on proximity and attractiveness.

References

[Mal17] MALLAWAARACHCHI, VIJINI. *Introduction to Genetic Algorithms*. 2017. URL: https://towardsdatascience.com/introduction-to-genetic-algorithms-including-example-code-e396e98d8bf3 1, 4.

[Par07] PARKER, CONRAD. *Boids Pseudocode*. 2007. URL: http://www.kfish.org/boids/pseudocode.html 1.

[Rey87] REYNOLDS, CRAIG W. "Flocks, Herds, and Schools: A Distributed Behavioral Model". *Symbolics Graphics Division* ACM SIGGRAPH 87 (1987), 25–34 1.

[Wik] WIKIPEDIA. *Quadtree*. URL: https://en.wikipedia.org/wiki/Quadtree 1.