**STARLING MURMURATION**

**MODEL & SIMULATION**

By:

Harshit Goel 2016CS10319

Soumya Sharma 2016CS10311

**Contents**

1 Introduction 3

2 The model 4

2.1 Boids . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 4

2.2 Forces . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 4

2.2.1 Alignment . . . . . . . . . . . . . . . . . . . . . . . . . 5

2.2.2 Cohesion . . . . . . . . . . . . . . . . . . . . . . . . . . 5

2.2.3 Separation . . . . . . . . . . . . . . . . . . . . . . . . . 5

2.3 Predators . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 6

2.4 The world . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 6

3 Serial implementation 7

3.1 Runtime behaviour . . . . . . . . . . . . . . . . . . . . . . . . 8

4 References 9

2

* **Introduction**

Swarm behaviour is something humans have long admired for its elegance and beauty, eﬃciency and seemingly perfect self-organisation. The emergent behaviour that results from each individual animal following some very ba-sic rules is simply astonishing. Amongst the most prominent examples of swarm behaviour (also often simply called "swarming") are "flocking" for birds or "schooling" for fish. Often essential for survival, swarms form to guard against attackers, to keep warm in rough conditions and to harness aerodynamics eﬀectively.

Biologists are not the only people professionally invested in understand-ing, modelling and simulating swarms. Swarm simulations are also used for entertainment purposes, in animated movies, TV shows and even computer games.

Swarm simulations for us are a great problem to work on for lots of rea-sons: The results produced can be visualised in a multitude of ways leaving room for unique and uncommon approaches. The simulation, while computationally intensive, can be broken down into smaller pieces and is thus a prime candidate for parallelisation.

The foundation of our work was laid by Craig Reynolds in his paper titled "Flocks, Herds, and Schools: A Distributed Behavioural Model" in 1987. In this report, we will follow the terminology he introduced closely.

With our project, we set out to create a functioning simulation of the boid flocking algorithm.

3

* **The model**

In this section we are going to present our model, including our world, boids, predators and forces that govern the interactions between boids, boids and predators and predators among themselves.

**2.1 Boids**

In our model, a Boid is a massless particle representing one organism in a swarm. Its two important properties are position and velocity, but there is also a third field, force, which is used to store the force calculated in each step.

The actual structure definition reads as follows:

struct Boid {

V e c t o r 3 f p o s i t i o n ;

V e c t o r 3 f v e l o c i t y ;

V e c t o r 3 f f o r c e ;

// . . .

} ;

typedef Boid P r e d a t o r ;

Each boid‘s position is confined to our world boundaries. Velocities are also bounded by a maximum velocity.

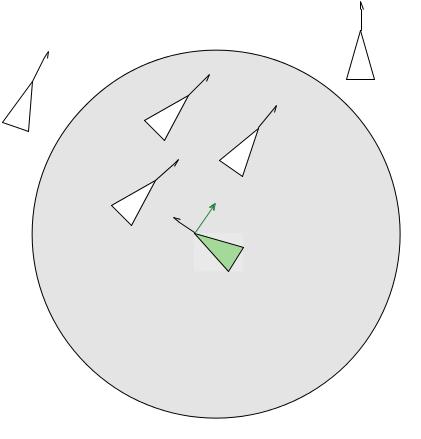
**2.2 Forces**

Each boid follows three simple steering behaviours called alignment, cohesion and separation.

For every force we present both a short text describing the basics and a graphic for visualisation: A green triangle represents the boid that is currently being observed, every other triangle represent neighbours of this boid. Each triangle inside the grey circle is a direct neighbour. It influences the boid that is being observed. The arrows point in the direction each boid is heading. The green arrow symbolises the force of the next discrete time step

4

**2.2.1 Alignment**



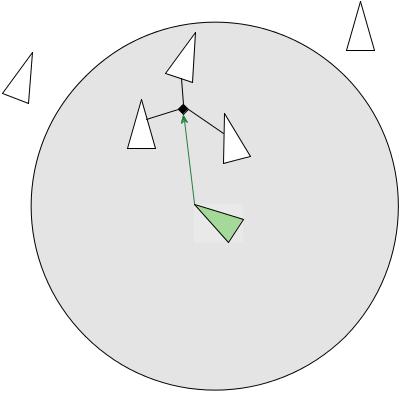
The first force is alignment, which

results in each boid flying towards

the average heading of its neigh-

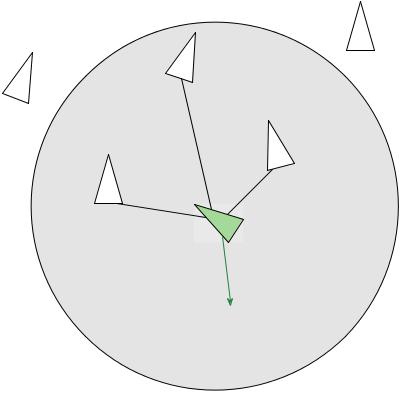
bours.

**2.2.2 Cohesion**



Cohesion results in each boid flying towards the centroid of all surrounding boids. This helps keep the swarm close.

**2.2.3 Separation**



In our model, just like in the

real world, organisms in swarms

rarely, if ever, collide. To model

this, Reynolds introduced Cohesion

which makes sure a boid steers

away from other boids that get too

close.

5

Each of these forces can be weighted diﬀerently and, by carefully choosing parameters, the resulting sum of those three forces results in swarm behaviour. If these forces are assigned indiﬀerent weights, for example if alignment and cohesion are not considered at all, the resulting model does not resemble swarm behaviour, but instead it looks like a static particle grid. In this case, since all boids try to steer away from all the other boids, we expect to see a uniform distribution of boids in our world.

In Reynolds original proposal, the direct neighbourhood was defined in terms of a specific distance and a specific angle to model the fact, that any organism’s perspective is limited. In our implementation we decided only to consider the distance and disregard the angle component. This was mainly done to keep the implementation as simple as possible. Nonetheless, it should be easy to take the angle under consideration as well.

**2.3 Predators**

Predators were added to make the simulation more dynamic. They are modelled the same way boids are, but interact diﬀerently with each other and with boids: A predator tends to avoid other predators and tries to get close to boids, which, on the other hand, are trained to evade predators and, if a predator gets too close, steer in the opposite direction to maximize the distance between the attacker and themselves. In our model, predators never actually catch boids, since there is no collision detection.

**2.4 The world**

We use a three-dimensional cube to model the world. All endings are connected: Left wraps around to Right, Up to Down, Front to Back. Because no boid leaves the world, we do not have to manage spawning new ones.

6

* **Serial implementation**

Our earliest prototype, written in C, made it clear we needed operator over-loading, mainly to work with three-dimensional vectors, so we chose C++ for our programming language. C++11 allows us to use many advanced language features like extended for-loops, enum classes for better readabil-ity and anonymous lambda functions to encapsulate common tasks inside of functions. As for libraries, we use Boost C++ Libraries for parsing of command line parameters and SFML for visualising the data.

We built seperate programs for producing the data and for consuming the data. This was done because we knew we only had to parallelise the actual simulation, not the visualisation. For initialisation, we populated our world with boids generated with pseudo random position and velocity. To allow easier testing and to ensure each process in the parallel version would always generate the same data during initialisation, we seed srand with a constant.

After initialisation the main simulation loop begins: In each iteration, forces for all boids and all predators are calculated, new positions are de-termined - based on the forces calculated previously - and the new data is written to disk.

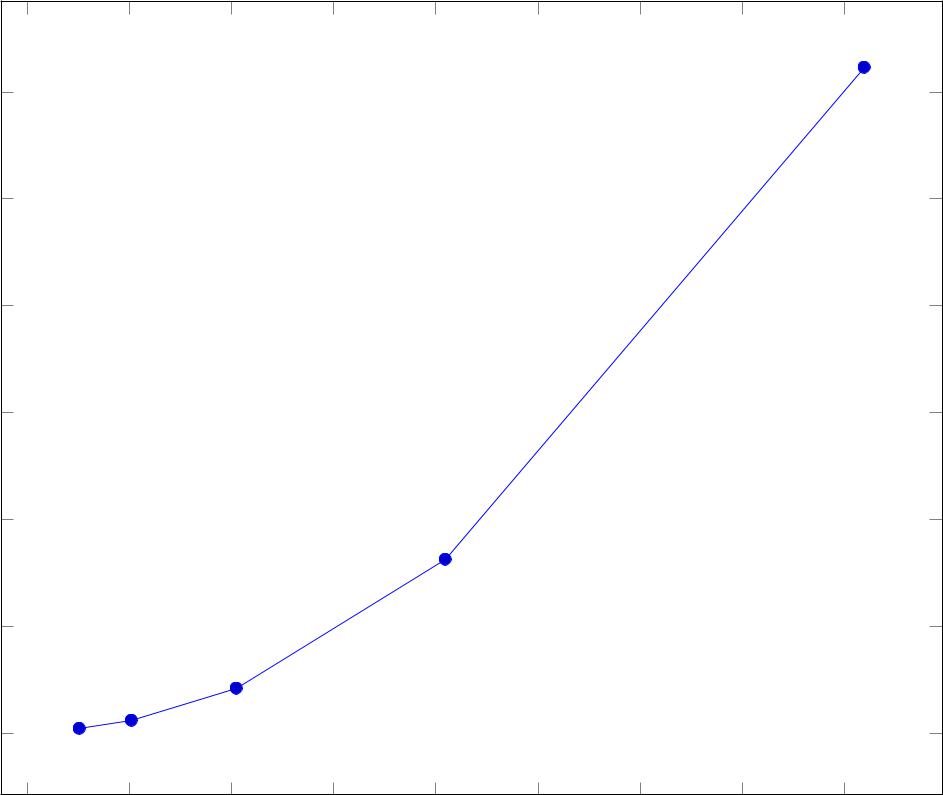
To fully implement and create our idea, we had to change our calculate Diﬀerence() function that is used to calculate a diﬀerence vector between two boids’ positions. Starting oﬀ, we used a simple vector subtraction to calculate the distance between two boids. Obviously this behaviour is diﬀerent from the behaviour we originally wanted to create in our model, so we had to redo this function. Our new, more complicated version fixed this discrepancy, but introduced a performance loss by more than factor two.

7

**3.1 Runtime behaviour**

The underlying algorithm is of complexity O(n2). Because we also added predators, our new complexity for n boids and m predators is O((n + m)2).

Runtime behaviour



|  |
| --- |
| Runtime in seconds |

120

100

80

60

40

20

0

* 500 1;000 1;500 2;000 2;500 3;000 3;500 4;000 Number of boids

The graph above is a diagram showing the runtime of our serial program. Plotted on the x-axis is the number of boids, plotted on the y-axis is the runtime in seconds. We simulated 1000 steps - no predators were included. The diagram shows the general runtime behaviour as n gets large: For twice as many boids, there are roughly four times as many calculations to be done. This fits perfectly with the projected complexity of O((n + m)2).

8

**REFERENCES**

* <https://en.wikipedia.org/wiki/Flocking_(behavior)>
* <https://en.wikipedia.org/wiki/Starling>
* <https://en.wikipedia.org/wiki/Boids>
* <https://www.youtube.com/watch?v=U4TvNqijfC4>
* <https://www.youtube.com/watch?v=QbUPfMXXQIY>
* <http://cp3-origins.dk/a/14187>
* <https://www.wired.com/2011/11/starling-flock/>
* <https://www.technologyreview.com/s/415022/first-simulation-of-the-flocking-behavior-of-starlings/>
* <https://wr.informatik.uni-hamburg.de/_media/teaching/sommersemester_2014/papo14-besner-swarmflocking-report.pdf>
* <https://arxiv.org/abs/0908.2677>
* <http://www.pnas.org/content/111/29/10422>