Boids: A Case Study

Michael Stewart

22 May 2015

Title: Boids: A Case Study

Author: Michael Stewart

Abstract

"Boids" is a computational model that simulates the flocking behaviour of birds. This paper aims to explore the background of the boids algorithm and discusses my Python implementation of the model. Each component of the boids algorithm is explained with the aid of Python-like pseudo-code. The behaviour and performance yielded by my implementation is discussed in the results section. Potentials improvements to my code are also discussed. Ultimately it is concluded that Boids is a strong example of a complex system and provides an effective means of modelling the behaviour of flocking animals such as birds.

1 Introduction

"Boids" (bird-like androids) is a model developed by Craig W Reynolds [3] in 1988 that aims to simulate the flocking behaviour of birds. The model was inspired by the fascinating behaviour exhibited by groups of animals such as fish and land animals. According to Moere [2], the flocking behaviour of animals could be explained by the fact that animals on the outskirts of a group are more likely to be targeted by predators. Reynolds' approach allows for the simulation of this behaviour by treating the flock as a group of agents, which each choose their own path based on three distinct rules. Each of these rules is applied to the boids in the form of a steering vector. Using these rules, interesting behaviour emerges such as boids grouping together into flocks as they fly. The only reason a flock arises is purely because of the rules set upon each individual agent, and because of this, the Boids model can be seen as an example of emergence.

2 Previous and Related Work

Boids has been covered by a number of researchers in the past. Hartman and Benes [1] extended the boids algorithm by introducting the concept of "change of leadership", which defines the chance of any given boid to become a leader and try to escape from the flock. Their system was designed to be used in real-time simulations as well as games and crowd simulations. According to Hartman and Benes, real birds have a tendency to randomly shoot-off from the flock, causing other birds to follow. This behaviour was simulated in Hartman and Benes' extension by first determining whether a particular boid is situated on the outside of the flock. If it is, the boid is given a random chance to shoot-off that is dependent on its promimity to the front of the flock it is situated in. The focus of Hartman and Benes' extension was to add more realism to the Boids algorithm.

Moere [2] utilised Reynolds' research in order to represent dynamic data evolution. The concepts behind the Boids algorithm were used in order to visualise time-varying data. Moere accomplished this by developing a working software prototype that is capable of handling up to 500 boids. Each boid within this prototype was treated as a stock market company, and continously received updated data values. This process was then visualised using a 3D plot and it was discovered that boids are able to emulate changes in a stock market. For example, it was found that sudden significant events (such as a stock market crash) are depicted by boids suddenly leaving the flock. Moere's research shows that the boids algorithm is useful beyond simply simulating bird behaviour.

3 Methodology

The boids model aims to visualise the path of many different agents. At first glance it seems difficult to program "flocking" behaviour for a large number of agents, however Reynolds' algorithm is in reality quite simple. The most basic form of Boids features three steering vectors, which govern the movement of each boid. These vectors are determined by three simple rules for boids to follow. They are:

1. Collision Avoidance: boids aim to avoid collisions with nearby flockmates
2. Velocity Matching: boids attempt to match velocity with nearby flockmates
3. Flock Centering: boids attempt to stay close to nearby flockmates

Mathematically, a boid's velocity at any given timestep can be determined as follows:

$$\mathbf{v} = \mathbf{v_{current}} + \mathbf{v_{ca}} + \mathbf{v_{vm}} + \mathbf{v_{fc}}$$

Where:

$\mathbf{v_{final}}$ is the final velocity

$\mathbf{v_{current}}$ is the current velocity

$\mathbf{v_{ca}}$ is the velocity obtained by the collision avoidance calculation

$\mathbf{v_{vm}}$ is the velocity obtained by the velocity matching calculation

$\mathbf{v_{fc}}$ is the velocity obtained by the flock centering calculation

3.1 Collision Avoidance

The collision avoidance vector, $\mathbf{v_{ca}}$, is present so that boids attempt to avoid collisions with their nearby flockmates. The pseudocode for the calculation of this vector is as follows:

```
def CollisionAvoidance(boid)
    v_ca = 0;
    for other_boid in boid.neighbours:
        if |other_boid.position - boid.position| < CLOSE_THRESHOLD:
            v_ca = v_ca - (other_boid.position - boid.position)
    return v_ca
```

In order to calculate $\mathbf{v_{ca}}$ for each boid, we iterate through each boid's neighbours (other boids within a certain radius) and find any neighbours that are situated within the CLOSE_THRESHOLD. If an other_boid is within this distance, $\mathbf{v_{ca}}$ is updated to steer the boid away from the other_boid.

3.2 Velocity Matching

The velocity matching vector, $\mathbf{v_{vm}}$, aims to keep boids at the same velocity as their neighbours. The pseudocode is as follows:

```
def VelocityMatching(boid)
    v_vm = 0
    for other_boid in boid.neighbours:
        v_vm = v_vm + other_boid.velocity
    if boid.neighbours.count > 0:
        v_vm = v_vm / boid.neighbours.count
    v_vm = (v_vm - boid.velocity) / 10
    return v_vm
```

Once again, each of the boid's neighbours are iterated through and their velocities are added to the total. The total is then divided by the number of neighbours in order to obtain the average velocity of the neighbouring boids. The boid's velocity is then subtracted from the vector and the vector is divided by some arbritary number (10 in this case) in order to curb the rate at which the boid's velocity increases.

3.3 Flock Centering

Finally, the flock centering vector causes boids to stay close to nearby flockmates. The pseudocode is as follows:

```
def FlockCentering(boid)
  v_fc = 0
  for other_boid in boid.neighbours:
    v_fc = v_fc + other_boid.position
  if boid.neighbours.count > 0:
    v_fc = v_fc / boid.neighbours.count
  v_fc = (v_fc - boid.position) / 100
  return v_fc
```

This calculation works in a similar way to the velocity matching calculation. The average position of each neighbouring boid is calculated and divided by an arbitrary number so that the resulting steering vector causes each boid to fly towards the centre of its neighbours.

3.4 Predator Avoidance

My implementation contains one additional steering vector, called “predator avoidance”. A predator is a moving object that repels boids. To model this behaviour, the following pseudo-code is used:

```
def PredatorAvoidance(boid)
  v_pa = 0
  if |predator.position - boid.position| < PREDATOR_THRESHOLD:
    v_pa = v_pa - (predator.position - boid.position)
  return v_pa
```

This calculation closely resembles that of collision avoidance. By adding this function’s resulting vector to $\mathbf{v}_{\mathbf{final}}$ it is possible to cause boids to avoid every predator in the simulation.

4 Implementation

My implementation of the Boids model is written in Python 2.7. It uses a popular Python library, *pygame*, in order to draw each boid. *Numpy* is used to handle vectors. Other Python libraries in the code include *sys*, *math*, *division*, *time* and *random*.

Each boid is an object of the “Boid” class, which contains methods to position and draw the boids. The code within the `move` function loosely follows the pseudocode written by Conrad Parker[1]. I chose to implement a 2D representation of the algorithm for simplicity. With each timestep, the boids calculate their velocities, move accordingly, and are redrawn. The steps behind each timestep is as follows:

[1]Conrad Parker. http://www.kfish.org/boids/pseudocode.html

1. The neighbours of each boid are determined through the `determine_neighbours` function.

2. Each boid determines its velocity by calculating the aforementioned $\mathbf{v_{ca}}$, $\mathbf{v_{vm}}$, $\mathbf{v_{fc}}$ and $\mathbf{v_{pa}}$ vectors. It then redraws itself to the screen.

3. Each predator moves one step along its linear path, and redraws itboid.

The world is modeled in a similar way to a torus. Going past one edge of the world will cause a boid to "wrap around" the world to the opposite edge. The world is represented in this way so that each boid stays on the screen, and does not fly off indefinitely as they may if the world was an infinite plane. Another option would have been to steer the boids towards the centre of the screen once they travelled past the world edges[2].

4.1 Performance Considerations

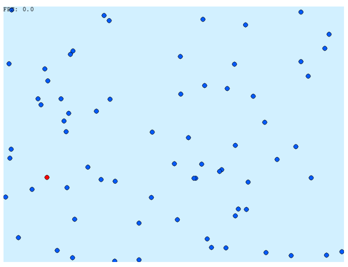
In order to boost the performance of the application, the `determine_neighbours` function does not determine the neighbours of every boid within each timestep. It is instead segmented so that a particular range of boids' neighbours are calculated within each timestep. The `CALCULATION_INTERVAL` variable determines the granularity of the segmentation - a high value will result in better performance. If, for example, the `CALCULATION_INTERVAL` is set to 15, and there are 75 boids in the simulation, 5 (75/15) boids will have their neighbours calculated per timestep. The downside to using a high value of `CALCULATION_INTERVAL` is that boids that see many changes in neighbours may not recognise their new neighbours for a number of timesteps, leading to slight inaccuracy.

5 Results and Discussion

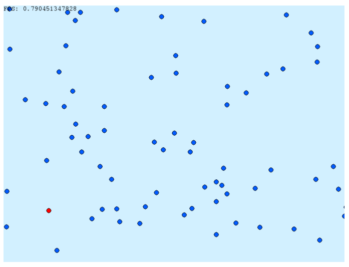
5.1 Behaviour

By implementing the three aforementioned rules, complex behaviour emerges. As the simulation runs, boids tend to group up into clusters, the size of which are dependent on the `NEIGHBOUR_THRESHOLD` variable. These clusters resemble flocks of birds, and simulating this behaviour is the main purpose of the boids algorithm. The addition of the fourth steering vector, $\mathbf{v_{pa}}$, creates potential for these flocks to separate as a predator approaches the flock. As predators move through the world, boids actively avoid them by steering themselves in the opposite direction of the predators. Screenshots from the application, which show this behaviour, are shown in Figure 1.

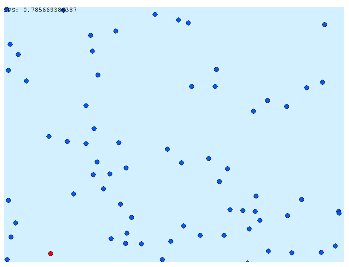
[2]Conrad Parker. http://www.kfish.org/boids/pseudocode.html



(a) Timestep 0.



(b) Timestep 10.



(c) Timestep 20.

Figure 1: A randomly-generated group of 70 boids and 1 predator.

There are a number of different variables in the application that govern the behaviour of the boids. These variables begin on line 147, and can be modified to test their effect on the simulation. Table 1 shows the effects of modifying each of these variables.

Variable	**Effect of increasing**
`CLOSE_THRESHOLD`	Boids group more tightly together.
`PREDATOR_THRESHOLD`	Boids avoid predators at a smaller distance.
`NEIGHBOUR_THRESHOLD`	Boids form smaller flocks.
`CA_WEIGHT`	Increases the speed at which boids move away from nearby flockmates.
`VM_WEIGHT`	Decreases the speed at which each boid matches its neighbours' average velocities.
`FC_WEIGHT`	Decreases the speed at which each boid moves towards the centre of its neighbours.
`PA_WEIGHT`	Increases the speed at which boids move away from nearby predators.
`CALCULATION_INTERVAL`	Increases framerate but decreases simulation accuracy.

Table 1: The effects of modifying each of the variables starting on line 147 of the application code.

5.2 Performance

All performance tests of the application code were conducted on an Asus N53SV laptop, which contains an Intel Core i7-2720QM processor. The application code draws the framerate at the top-left of the screen, so testing performance is fairly straightforward. The main difficulty when testing the performance of the application is that the framerate heavily depends on how clustered the boids are. Loosely-separated boids produce high framerates as the number of neighbours each boid must iterate through for the purposes of the vector calculations is low. On the other hand, highly clustered boids result in low framerates as each boid has a high number of neighbours. Framerates therefore fluctuate as boids disband and regroup.

In order to test performance, the code prints out the total time taken to reach timestep 1000 when the `TEST_TIME` variable is set to `True`. Table 2 shows the time taken to reach timestep 1000 for different settings of `CALCULATION_INTERVAL`, number of boids and number of predators. Each test was run three times and the average time is displayed. In Table 2, each test was run with `CALCULATION_INTERVAL` set to 15, `NEIGHBOUR_THRESHOLD` and `PREDATOR_THRESHOLD` set to 100, and `CLOSE_THRESHOLD` set to 30.

# Boids	# Predators	Average Total Time
70	0	22.415s
70	1	26.614s
70	2	27.126s
140	0	75.043s
140	1	84.273s
140	2	87.341s
210	0	170.448s
210	1	179.916s
210	2	184.962s

Table 2: Performance results for reaching timestep 1000 using varying amounts of boids and predators.

The worst-case complexity for this application is $O(N^2)$, where N is the number of boids, as each boid may have to iterate through every other boid to calculate the steering vectors. This is evident in the results, which show that doubling the number of boids more than doubles the time taken to reach the 1000th timestep. Tripling the number of boids multiplies the time taken to reach the 1000th timestep by over 7 times. Adding predator increases the time taken by an amount dependent on the number of boids. While it is not shown in this results section, the `CALCULATION_INTERVAL` variable helps to decrease processing time, but ultimately does not change the worst-case complexity of the code.

5.3 Improvements

It is clear that my implementation of the boids model is fairly slow when dealing with a large number of boids. It would be ideal to improve the program so that it can handle a larger number of boids without significant framerate loss. While the `CALCULATION_INTERVAL` variable helps to increase framerates, it causes inaccuracy due to all neighbours not being calculated for all boids at every timestep. One possible solution for this would be to expand the code to work on multiple cores using threading. The laptop the code was tested on has four cores but the code was only running on one core as Python does not support multithreading by default. Further optimisation could be conducted in order to boost the application's performance.

6 Conclusion

This paper aimed to explore the ideas behind the Boids model and discuss its background, methodology and uses. My implementation has been shown to handle approximately 100 boids without significant framerate loss and is also extended to handle the notion of “predators”, which boids try to avoid. By using four simple steering vector calculations, complex behaviour emerges in the simulation, and boids appear to flock together as birds do in real life. This emergent behaviour is a strong case for why Boids is a solid example of a complex system in action. It can be concluded that Boids is an important development in the area of computational modelling and that it provides a strong model for the behaviour of flocking animals.

References

[1] Christopher Hartman and Bedrich Benes. Autonomous boids. *Computer Animation and Virtual Worlds*, 17(3-4):199–206, 2006.

[2] Andrew Vande Moere. Time-varying data visualization using information flocking boids. In *Information Visualization, 2004. INFOVIS 2004. IEEE Symposium on*, pages 97–104. IEEE, 2004.

[3] Craig W Reynolds. Flocks, herds and schools: A distributed behavioral model. *ACM Siggraph Computer Graphics*, 21(4):25–34, 1987.