EFFICIENT METHOD TO FIND NEAREST NEIGHBOURS IN FLOCKING BEHAVIOURS

Omar Adwan

Computer Information Systems Department, The University of Jordan, Amman – Jordan

ABSTRACT

Flocking is a behaviour in which objects move or work together as a group. This behaviour is very common in nature think of a flock of flying geese or a school of fish in the sea. Flocking behaviours have been simulated in different areas such as computer animation, graphics and games. However, the simulation of the flocking behaviours of large number of objects in real time is computationally intensive task. This intensity is due to the n-squared complexity of the nearest neighbour (NN) algorithm used to separate objects, where n is the number of objects. This paper proposes an efficient NN method based on the partial distance approach to enhance the performance of the flocking algorithm and its application to flocking behaviour. The proposed method was implemented and the experimental results showed that the proposed method outperformed conventional NN methods when applied to flocking fish.

KEYWORDS

flocking behaviours, nearest neighbours, partial distance approach, computer graphics and games

1. INTRODUCTION

Flocking is a behaviour in which objects move or work together as a group [1]. Flocking can be defined as the behaviour of a group of objects that usually has their members (agents) are locally controlled by a small set of rules. These members are fewer than particle systems. We find many examples in life and nature of this behaviour: birds, fish, sheep, etc. [2-4]: birds fly in swarms, fish swim in fish schools and sheep move as herd steering by a dog. They behave in such a way that they appear as a single coherent entity, in spite of their shape and direction [1-2]. Bird flocks and fish schools can be modelled and simulated to mimic the flocking behaviour of birds on computer [3]. Flocking simulations have been widely studied in computer animation, graphics, games and other areas such as road designs in order to simulate pedestrian's behaviour [4].

One of the first algorithms to for producing flocking behaviour in groups of computer characters (objects) was presented by Reynolds [1], with the motivation to simulate flocks of birds for computer graphics. The algorithm has three rules. Each flock agent makes steering decisions based on the following behaviours: Cohesion, Alignment and Separation. Flock individuals would follow simple local rules to avoid collisions (separation), match velocities to their neighbours (alignment), and center themselves among their neighbours (cohesion). These rules are applied to each individual object in a group with the result of very convincing flocking behaviour [1,4-6]. These rules (illustrated in Fig. 1).

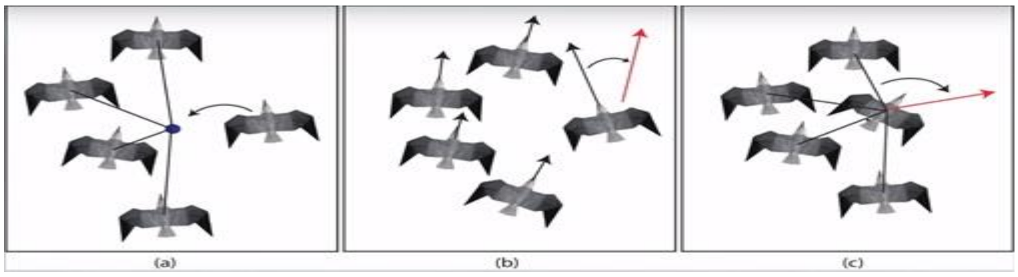

Fig 1. Flocking rules: (a) move toward average group position, (b) align heading with average group heading, and (c) avoid others [6]

Cohesion implies that all the objects in the flock stay together in a group; we don't want each object breaking from the group and going its separate way. To satisfy this rule, each object should steer toward the average position of its neighbours. Alignment implies that all the object in a flock to head in generally in the same direction. To satisfy this rule, each object should steer so as to try to assume a heading equal to the average heading of its neighbours. Separation implies that we want the objects to maintain some minimum distance away from each other, even though they might be trying to get closer to each other as a result of the cohesion and alignment rules that is to avoid crowding neighbours. We don't want the objects running into each other or coalescing at a coincident position. Therefore, we'll enforce separation by requiring the objects to steer away from any neighbour that is within view and within a prescribed minimum separation distance. To maintain a distance with other neighbours in the flock to avoid collision, the middle moving entity sometimes called flockmates or (boid) is shown moving in a direction away from the rest of the boids, without changing its heading as shown in Fig. 2.

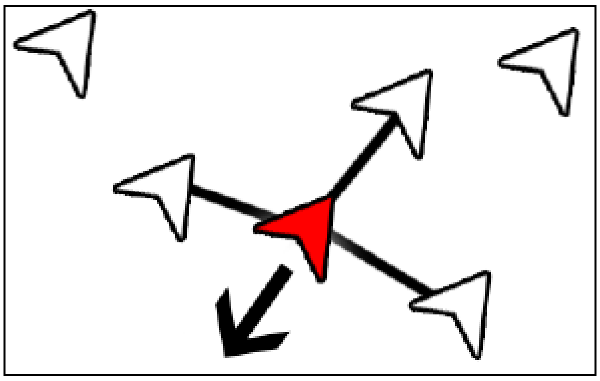

Fig 2. maintaining a distance with other neighbours in the flock

There are two conflicting propensities when working in flocking: collision avoidance and flock centering. It is a huge challenge is to simulate tens of thousands of objects in real-time where they realistically separate or avoid collisions with each other. In this paper, we are focusing on the separation rule. A conventional approach for achieving the separation rule in flocking behaviour is to perform nearest neighbour approach. Many implementations of the basic flocking algorithm grow in complexity. For a flock of n members, calculation of their instantaneous influences on each other, i.e. finding the nearest neighbour approach has the complexity of $O(n^2)$, because every object has to calculate its distance from every other object in the entire flock, in order to ensure the separation rule. For example, if we double the number of boids, it quadruples the amount of time taken. This high complexity makes it difficult to the game to be efficient, and therefore, techniques to reduce the high complexity is acquired [5].

The objective of this paper is to propose an efficient NN method based on the partial distance approach for simulation of flocking behaviours.

2. RELATED WORK

Flocking behaviour is a combination of three rules behaviours cohesion, separation, and alignment. To find the nearest neighbours efficiently, first a search is made to find other boids within the simulated world. This might be an exhaustive search of all boids in the simulated world. One way to find the nearest neighbours efficiently is to perform a three-dimensional bucket sort and then check adjacent buckets for neighbours. Such a bucket sort can be updated incrementally by adjusting bucket positions of any members that deviate too much from the bucket center as the buckets are transformed along with the flock. There is, of course, a time-space trade-off involved in bucket size—the smaller the buckets, the more buckets needed but the fewer members per bucket on average. However, this does not completely eliminate the n-squared problem because of worst-case distributions [9-13]

In [10], the authors improved the conventional flocking algorithm by using the characteristic of flocking behaviour which two objects may share many common neighbours if they are spatially close to each other. They proposed the condition which can check whether two objects share their neighbours for a given set of agents or not.

In [14], the authors introduced a procedure that identifies the influential neighbours of fish moving in a group, and tested it along a series of experiments in groups of two and five individuals of the freshwater tropical fish swimming in a ring-shaped tank. Four parameters were used to identify influential neighbours: the time-delay τ, the window size w, the correlation threshold Cmin above which individuals are supposed to be interacting, and the threshold ε for selecting more than one influential fish.

In [15], the Partial Distance (PD) algorithm has been proposed. The algorithm allows early termination of the distance calculation by introducing a premature exit condition in the search process. The Partial Distance (PD) algorithm is proposed to reduce the computation complexity of the exhaustive search. The PD algorithm allows early termination of the distance calculation between an object and all other objects in the flock by introducing a premature exit condition in the search process.

Let C = {ci, i = 1,…,N} be a set of cluster centers of size N, where (cij, j = 1,…, K) is a K dimensional data point (vector). For a given data point X = (xj, j = 1,…, K), it is required to find the vector with the minimum distance from the set C under the squared-error distance measure defined as follows:

.

$$d(X, c_i) = \sum_{j=1}^{K} (x_j - c_{ij})^2$$

The basic structure of the PD algorithm is illustrated below:

```
  Loop A: For i = 1, …, N
             d = 0
Loop B:   For j = 1, …, K
             d = d + (xj – cij)2
             if (d > dmin) Next i // (exit condition)
             Next j
             dmin = d
             min = i
             Next i
```

It can be observed that the partial distance search algorithm gains computation saving over the full search algorithm because of the provision for a premature exit from Loop B, on satisfying the condition d > dmin (called the exit condition) before the completion of the distance computation d(X, ci). The problem with the PD algorithm is that its efficiency is dependent on the current (initial) distance dmin found so far. The larger the distance is, the less useful this method becomes [16].

In [16], the authors proposed efficient initial distances to the PD algorithm. The proposed strategy avoids many unnecessary distance calculations by applying efficient PD strategy (EPD). Let x be any data point, let c be a cluster center and let cprev be the previous location of the same object. Suppose that in the previous iteration we know that dprev is the distance between x and cprev, then we can use dprev as an initial distance for the PD algorithm. In other words, if cprev has moved a small distance, then dprev is a good initial distance. Having these observations, the EPD algorithm works as follows:

Step 0: assign each data point, x, to its closest cluster center, c, using the PD algorithm. Use each resulting distance (dprev) as an initial distance in the next step.

```
Step 1:   dmin = dprev   (from step 0)
Loop A:   For i = 1, …, N
          d = 0
Loop B:   For j = 1, …, K
          d = d + (xj – cij)2
          if (d > dmin) Next i // (exit condition)
          Next j
          dmin = d
          min = i
          Next i
```

It can be noticed that the new proposed strategy would gain more computation time saving than the conventional PD algorithm. This is because the initial distance, d > dmin produced from the proposed strategy is very small, as shown in Step1 (first line) in the algorithm above. Note that step 0 is applied only once to find previous distances for the next steps. If the PD algorithm is used in step 0, then we still gain some CPU time savings.

In this paper, we apply both the partial distance (PD) and its enhanced version (EPD) to the flocking fish project

3. EXPERIMENTAL WORK

It is more practical to understand flocks and herds by relating them to the real-life behaviours they model. These concepts describe a group of objects, or boids, as they are called in artificial intelligence terms, moving together as a group [17-18]. The flocking algorithm gets its name from the behaviour fish flocking exhibit in nature, where a group of fish follow one another toward a common destination, keeping a mostly fixed distance from each other. The emphasis here is on the group.

In [6], de Byl, Penny developed a computer game to simulate the behaviour of fish flocks using Unity-3D game engine [Unity3D.com] with C# programming language. Fig. 3 shows a flock of 50 fish simulated by [6]. The figure shows a clear separation between the fish.

In this paper, we implement the work of [6] using Unity3D game engine. And then implement our method to enhance the algorithm in [4].

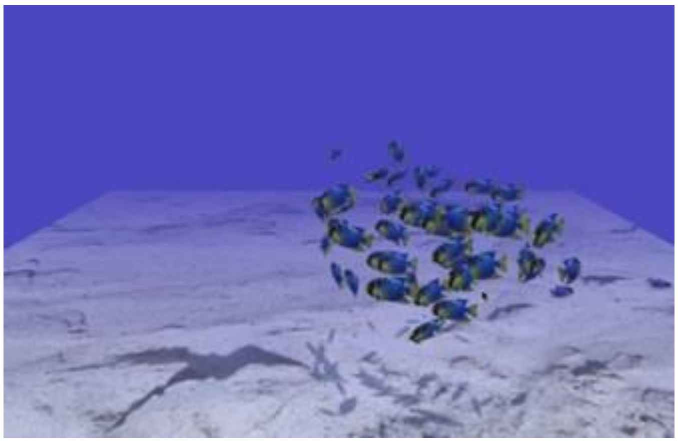

Fig. 3. A flock of 50 fish simulated by [de Byl, Penny]

In order to test the efficiency of PD strategy, the fish flock of [6] has been implemented. The frame per second (fps) is measured. The results are show in Table 1. The Table shows that the PD method outperformed the conventional approach. Table 1 also shows the performance of the EPD methods for the flock of fish, and the number of objects (No. Objects) in the flock. It can be noticed from Table 1 that the Partial Distance (PD) method outperformed the conventional approach. The Table also shows that the EPD method gave the best results in all cases. The frame-per-second (fps) of the EPD is always higher.

Table 1: fps for exhaustive, the PD and EPD methods for the fish flock

No Objects	Ex	PD	EPD
100	100	100	100
200	80	80	80
300	75	80	85
400	65	75	80
500	54	65	70
600	30	35	45
700	22	30	38
800	17	30	35
900	13	25	32
1000	11	25	32

It can be noticed from the table above that the EPD method outperformed the exhaustive and the PD methods. Fig.4 show the graphical results.

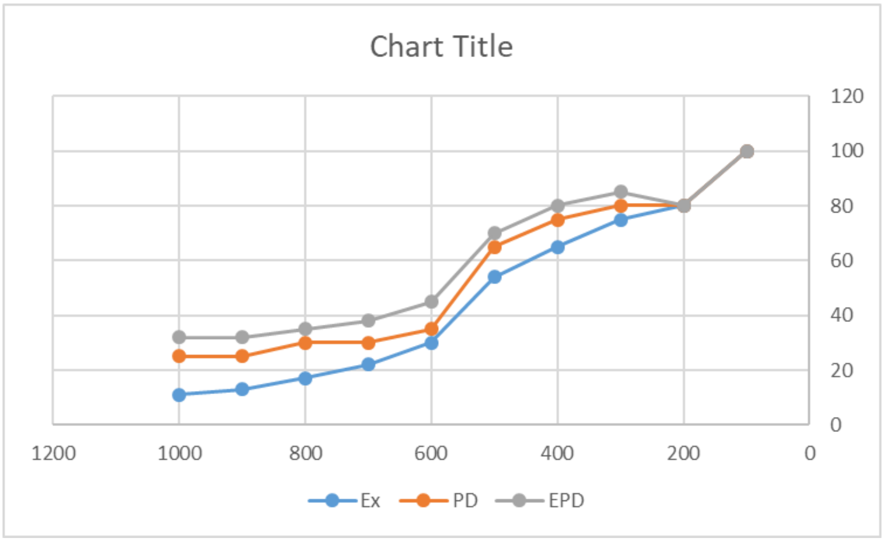

Fig.4 Graphical results (fps) of the fish flock.

4. CONCLUSIONS

Flocking behaviours are used in computer animation, games and graphics for realistic simulation of massive crowds. In real life, simulation of massive crowd is complex and intensive. This paper describes a faster flocking algorithm and its application to the fish flocking problem. We propose a nearest neighbour's method based on the partial distance (PD) approach. The proposed algorithms avoid many unnecessary distance calculations by applying an efficient partial distance strategy. Experimental results show that the proposed algorithm gave better results than the conventional algorithms when applied to flocking fish.

REFERENCES

[1] Reynolds, C. W., Flocks, Herds, and Schools: A Distributed Behavioural Model, In Proceedings of SIGGRAPH 87, 21(4), 1987, pp. 25-34.

[2] Farine, D.R., Garroway, C.J. & Sheldon, B.C. (2012) Social network analysis of mixed-species flocks: exploring the structure and evolution of interspecific social behaviour. Animal Behaviour, 84, 1271–1277.

[3] Levent Bayındır, A review of swarm robotics tasks, Neurocomputing, 172(8), 2015, pp292-321

[4] Jae Moon Lee, Seongdong Kim, A simulation of multiple grouping movements for pedestrians, International Journal of Computational Vision and Robotics, 7(3), 2017.

[5] Mohit Sajwan, Devashish Gosain, Sagarkumar Surani, Flocking Behaviour Simulation: Explanation and Enhancements in Boid Algorithm, International Journal of Computer Science and Information Technologies, Vol. 5 (4), 2014, 5539-5544

[6] de Byl, Penny. Holistic Game Development with Unity: An All-in-One Guide to Implementing Game Mechanics, Art, Design and Programming, CRC Press. Kindle Edition, 2018.

[7] D. Shiffman, The Nature of Code, Processing Foundation, available: https://natureofcode.com/, last visited, May, 2019.

[8] E. Adams, Fundamentals of Game Design Third Edition, Pearson, 2014.

[9] R. Parent, Computer Animation, Algorithms and Techniques, Third Edition, Morgan Kaufmann, 2012.

[10] Jae Moo Lee, An Efficient Algorithm to Find k-Nearest Neighbors in Flocking Behavior, Information Processing Letters, 110, 2010, pp. 576-579.

[11] M. Sajwan, D. Gosain, S. Surani, Flocking Behaviour Simulation: Explanation and Enhancements in Boid Algorithm, International Journal of Computer Science and Information Technologies, 5 (4), 2014, 5539-5544.

[12] M. Moussaıd, D. Helbing, and G. Theraulaz, "How simple rules determine pedestrian behavior and crowd disasters," in Proceedings of the National Academy of Science, vol. 108, 2011, pp. 6884–6888.

[13] B. Chazelle, The Convergence of Bird Flocking, J. ACM 61(4), 2014, pp. 21-35.

[14] Li Jiang, L. Giuggioli, A. Perna, R. Escobedo, V. Lecheval, C. Sire, Z. Han, and G. Theraulaz, Identifying influential neighbors in animal flocking. PLOS Computational Biology 13(12), 2017.

[15] S. Chen and W. Hsieh, Fast Algorithm for VQ Codebook Design. IEE Proc., 138 (5), 1991. pp. 357-362.

[16] M. B. Al- Zoubi, A. Hudaib, A. Huneiti and B. Hammo , New Efficient Strategy to Accelerate K-Means Clustering Algorithm , American Journal of Applied Sciences, 5(9), 2008, pp. 1247-1250.

[17] Ray Barrera, Aung Sithu Kyaw, Clifford Peters and Thet Naing Swe, Unity AI Game Programming, Second Edition , Packt Publishing, 2015.

[18] Aung Sithu Kyaw. Clifford Peters and Thet Naing Swe, Unity 4.x Game AI Programming, Packt Publishing, 2013.

AUTHOR

Omar Adwan is an Associate Professor in the Department of Computer Information System at the University of Jordan, where he has been since 2010. From 2012 to 2016 he served as Department Chair. He received a B.S. in Computer Science from Eastern Michigan University in 1987. Dr Adwan received his M.S. and Ph.D. in Computer Science majoring in Software Engineering from The George Washington University. Currently Dr. Adwan is the Deputy Dean of Student Affairs at the University of Jordan. Dr Adwan research interests are in Image Processing, software engineering, with focus on software testing, software analytics, software security, intelligent software engineering, and both data mining and machine learning (adwanoy@ju.edu.jo)