Technical Report CSTN-143

# Simulating Species Interactions and Complex Emergence in Multiple Flocks of Boids with GPUs

Alwyn V. Husselmann and Ken A. Hawick
Computer Science, Institute for Information and Mathematical Sciences,
Massey University, North Shore 102-904, Auckland, New Zealand
email: { a.v.husselmann, k.a.hawick }@massey.ac.nz
Tel: +64 9 414 0800 Fax: +64 9 441 8181

August 2011

**ABSTRACT**
The Boids model of spatial agents has proved a great source of insights into how localised microscopic agent-agent interactions can lead to emergent complexity at a macroscopic spatial scale. We explore the interactions between multiple separate species of boids where the flocks maintain a level of coherence amongst their own species but exhibit a preference to separate from other species. We have used data parallelism and Graphical Processing Units(GPUs) to explore large scale flock sizes in interactive simulation time. We also describe some space-covering histogram measurements and cluster component labelling analysis that characterise the flock separation emergent behaviour and scaling in terms of the species interaction parameters.

**KEY WORDS**
boids; multi-species; flocking; agent-based modelling; data parallelism; GPU; component labelling; emergence; spatial agents; complexity

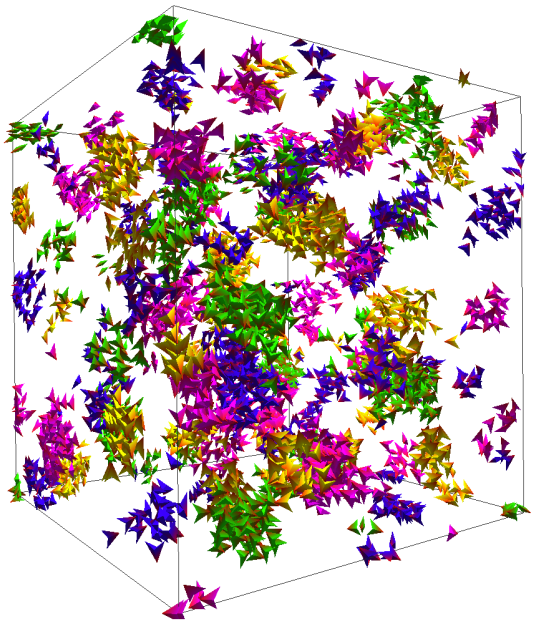

Figure 1: Several different flocks or shoals of agents, attracting one another locally within tribe but repelling one another globally.

## 1 Introduction

Agent based modelling (ABM) and simulation is now quite a mature research field with a body of practical [1, 2] and theoretical [3] agent modelling apparatus available. ABM also has similarities with discrete event simulation [4].

There are a number of agent based models reported in the literature, for simulating phenomena such as social interaction scenarios [5], crowds [6, 7], vehicles [8, 9], robots [10], predator-prey models [11–13] as well as flocking agents such as the "boids" model and its variants [14, 15]. Agent-based approaches like the boids spatial model have also found important uses in path planning [16] and in steering autonomous animated characters [17]. There are also interesting uses of genetic programming and evolutionary methods to adapt boids behaviours reported in the literature [18].

Rendering and hence visualising agents in a boids-like model is important to convey a sense of physical realism but also potentially to help debug and verify sim-

ulation code. Various shaders and occlusion rendering algorithms can be usefully deployed to do this using modern graphics processing unit hardware [19]. However, much recent work has been done using GPUs to speed up computation and not just graphics and rendering.

A number of authors have reported using GPUs and data parallel systems like CUDA to successfully to speed up agent-based simulations [20, 21]. Two recent and notable uses of large scale GPU accelerated supercomputers are: work by Aaby and co-workers at Oak Ridge National Laboratory to simulate very large numbers of boid-like agents [22] for military and socio-economic behaviour models; and reported work by Permulla, also at ORNL, on agent based simulations on GPUs using a range of ABM techniques with large numbers of agents [23].

In many of the phenomena studied it is important to be able to model systems of agents that exhibit logarithmic scaling in the numbers of agents. To obtain useful simulation results over several decades in the system size it is necessary to employ parallel processing technology. The general purpose graphical processing unit (GP-GPU) approach to data-parallelism lends itself especially well to agent based simulation models which generally require $O(n^2)$ computation time. Many agent-based models have spatially constant computation requirements, but this is often a special case.

The "Boids" algorithm was first devised by Craig Reynolds to model flocking behaviour of birds [14, 15] and in it, communication between agents is often far less trivial. In these cases, spatial data structures and sorting algorithms are a common approach to increase performance beyond that of a simple all-pairs implementation. As is often the case, space discretisation results in a system that is less precise. However, in doing so, the performance increase is often worth the loss of precision.

In this work, we endeavour to work with continuous space for precision, and employ other techniques in order to increase performance and analyse the resulting behaviour. For simulations such as Conway's Game of Life, Leadership and Mood Diffusion, space discretisation is an intrinsic attribute, and is readily exploited. Using the "B+2R" latency hiding scheme, these simulations can be scaled to more than a billion agents [22]. It is also worth noting that using special techniques, the simulation can be kept in continuous space, but still radically reduce communication time by using spatial indexing of some kind which can be created just before computing the next time step. This causes another increase in computation time, however, but it is often the source of a global increase in performance [19] by reducing the overall computation.

In this article we explore the use of GPU data parallelism to simulate large numbers of boid-like agents semi-interactively so that we can explore some new parameters introduced to simulate multiple and competing distinct flocks of agents. Our article is structured as follows. We summarise the multiple flocking boids model and its parameters in Section 2. In Section 3 we present the various metrics we introduced into our simulation of boids, and we describe each of these in detail. In Section 4 we present the results of the simulation, and the various metrics. Then, in Section 6 we discuss the implications of these and any problems that we encountered during the process; and finally, in Section 7 we draw conclusions from the data that we collected and describe some areas for further work.

# 2 Boids and Multiple Flocks

In this section we explain the boids algorithm and show how parameters can be introduced to support different species of boid and multiple flocking behaviour.

Figure 1 shows several differing flocks of agents. One of the aspects that make it difficult to compare this work with prior research is the somewhat loosely defined boids problem. There are several areas in which the algorithm can be modified to produce radically different results. The swarming emergent behavioural effect of boids arise from the use of vector algebra in the form of three "rules". These rules are used in a decentralised manner, where every agent (or "Boid") continuously calculates them and alters their own velocity.

There are no stochastic aspects in the rules of the original algorithm. Aside from random starting positions and random initial velocities, it is completely deterministic. Reynolds named these three rules as follows (see [14]):

1. Separation: Move or steer to avoid crowding local flock-mates.
2. Alignment: Steer in such a way to align self with the heading of local flock-mates.
3. Cohesion: Move or steer toward the centre of mass of local flock-mates.

These rules frequently do not constitute an interesting (or indeed useful) simulation on their own. It is possible to add more custom rules to these in order to create meaningful movement and swarming behaviour for

the purpose of modelling. It is this flexibility that has spawned several ABM toolkits [22, 23] in the last few decades. To create some purpose, we add an additional "goal rule', in which boids steer to move towards a certain location. Boids are quite useful for behavioural animation where they have already found use in animating swarms of bats, and also flocks of penguins [15].

The original Boids algorithm is summarised in Algorithm 1.

**Algorithm 1** The original Boids algorithm by Craig Reynolds (see [14]).

initialise $N$ boids in continuous 3D space.
**for** $i \leftarrow 0$ to $i_{N-1}$ **do**
  gather all boids into set $C$ that are within the communication radius of boid $i_N$
  compute relative vector to centre of mass $V_c$ of boids in $C$
  compute average velocity vector $V_v$ of boids in $C$
  compute separation vector $V_s$ of boids in $C$ which are in the separation radius
  new $V \leftarrow V_0 + V_s + V_c + V_v$
**end for**

Each boid builds two sets of boids with whom some computation is required. Originally, this is the neighbourhood set $S_n$ with whom the boid attempts to achieve cohesion (centre of mass steering) and the other set is the separation neighbourhood $S_s$. Normally $S_s \subset S_n$, in other words, the boids which are modifying the separation vector are closer.

The conceptual for-loop in a serial solution can be done in parallel, assuming that the output data is written to a location that does not affect the input data. If the input data is modified as the new values are computed, then this algorithm can almost be seen as border-line deterministic due to the scheduling of parallel threads and race conditions. However, as it was intended in the first instance, it is a completely deterministic algorithm, only accepting random starting locations and velocities.

It is important to note that agent visibility from the original algorithm is ignored in our simulation, and communication is performed in a uniform sphere around a boid, as opposed to the 3-D extrapolation of the original 2-D visibility circle sector into a sphere with a conical shape excluded from the rear of the boid. This exclusion can be the source of more optimisation [19].

In this work we modify the original Boids algorithm in Algorithm 1 and substitute additional checking for separate flocks of Boids, which interact in a different manner. This new algorithm is summarised in Algorithm 2.

**Algorithm 2** The modified Boids algorithm for inclusion of separate flocks of Boids, in order to simulate flocking behaviour.

initialise $N$ boids in continuous 3D space.
**for** $i \leftarrow 0$ to $i_{N-1}$ **do**
  gather all boids *of the same flock* into set $C$ that are within the *flock* communication radius of boid $i_N$
  compute relative vector to centre of mass $V_c$ of boids in $C$
  compute average velocity vector $V_v$ of boids in $C$
  compute separation vector $V_s$ of boids in $C$ which are in the separation radius
  gather all boids *of foreign flocks* into set $D$ that are within the *foreign flock* communication radius of boid $i_N$
  compute relative vector to centre of mass $V_{c2}$ of boids in $D$
  compute average velocity vector $V_{v2}$ of boids in $D$
  compute separation vector $V_{s2}$ of boids in $D$ which are in the separation radius
  new $V \leftarrow V_0 + V_s + V_c + V_v + V_{c2} + V_{v2} + V_{s2}$
**end for**

The main modification being the addition of a separate set $D$ which is the set of foreign boids that are within the foreign flock communication radius.

At this point, yet another parameter has been introduced that radically modifies behaviour: the foreign flock communication radius. Even in the original algorithm, there are parameters that are not included which also greatly modifies behaviour. The complete list of parameters that are included in the simulation we describe here are shown in Tables 1 and 2.

| *Name* | *Default Value* |
|---|---|
| Group Count | 2 |
| Rule 1 Enabled | Yes |
| Rule 2 Enabled | Yes |
| Rule 3 Enabled | Yes |
| Goal Rule Enabled | Yes |
| Bounding Box Height | 2.7 |
| Bounding Box Width | 5.0 |
| Bounding Box Depth | 5.0 |

Table 1: Generic parameters in the simulation.

For most of the parameters listed in Tables 2 and 1, any slight modification makes a great difference in behaviour. There are a few ways of interpreting the parameters listed in Table 2, so to diffuse any ambiguity, some are briefly described here.

| *Name* | *Default Value* |
|---|---|
| Opponent Dislike | 0.02 |
| Friend Dislike | 0.02 |
| Neighbourhood Size | 0.5 |
| Close Distance | 0.01 |
| Opponent Close Distance | 0.2 |
| Max Velocity | 0.3 |
| Group Velocity Affinity | 0.02 |
| New Velocity Affinity | 0.1 |
| Goal Affinity | 0.01 |
| Group Centre Affinity | 0.01 |

Table 2: Specific parameters in the algorithm itself.

The parameters named "Opponent Dislike", "Friend Dislike", "Group Velocity Affinity", "New Velocity Affinity", "Goal Affinity" and "Group Centre Affinity" are all treated as ratios of the relative vectors they each refer to, which are then added to the boid's velocity vector. The others are simply distances used in conditionals. To illustrate this point, assume that the vector $d$ is the relative vector from boid $n$ to a *foreign* boid. Assume also that the magnitude of $\vec{d}$ is such that it is within the appropriate communication distance for separation with foreign boids ("Opponent Close Distance" $c$), i.e. $|\vec{d}| < c$. The "Opponent Dislike" parameter ($o$) is used as follows:

$$\vec{v}_n = \vec{v}_n - o\vec{d}$$

Where $\vec{v}_n$ is the velocity of the $n$th boid. This action is repeated for every boid that is within the communication distance set for foreign boids. A similar calculation is used for alignment and cohesion.

The first of the metrics we intended to use to explain the behaviour resulting from this algorithm is spatial clustering. As in image clustering for object recognition, we are using a simple algorithm to obtain cluster data. This technique is summarised in Algorithm 3. Essentially it is matching the cluster ID of boids that are close to each other, and performing multiple passes in order to coalesce them into larger groups and eventually converge upon the true number of groups. Sometimes determining the number of groups visually is not an entirely objective procedure, and this uncertainty is represented in the constant $L$ shown in the algorithm. It should be calibrated appropriately.

**Algorithm 3** Clustering algorithm for use in 3D space with boids.

```
initialise constant L with a value describing minimum distance required to be a cohesive cluster
repeat
  for i ← 0 to i_{N-1} of N boids do
    for j ← 0 to j_{N-1} of N boids do
      if distance from boid i to boid j is less than L then
        set cluster ID of boid i to cluster ID of boid j
      end if
    end for
  end for
until no more changes are made
```

# 3 GPU Implementation

When simulating ABM in large scale, it is often necessary to make use of parallelism that is normally inherent in such simulations. GP-GPU is particularly useful for parallel applications, due to how modern GPU architectures have evolved from accommodating massively parallel fragment and pixel shaders operating on textures in on-board memory. NVidia's CUDA resulted after researchers around the world spent several years of GP-GPU work done using "ping-pong" buffering via these fragment shaders that modify textures. CUDA makes GP-GPU programming much more accessible. Multi-core systems can also decrease computation required, but to a much lesser extent than GPUs, provided of course there is a sufficient number of agents, otherwise the time required to copy memory from the host to the device becomes too great and hence defeats the purpose. Usually though, this number need only reach into the thousands before the performance increase becomes very clear.

The typical lifespan of a program utilising CUDA is as follows:

1. Initialise agent data on host
2. Copy agent data to device
3. Execute CUDA "kernel" on device
4. Copy modified agent data back to the host, render and repeat.

This process can be improved by directly rendering from the CUDA "kernel", eliminating the need to copy the agent data back from the GPU. The CUDA "kernel" is the smallest atomic section of code that can be executed by one CUDA core. Modern NVidia cards have

upwards of around 512 of these cores, which can be used to their full extent by a carefully designed program.

Implementations of ABM, such as boids, face a unique set of problems when attempting performance increases. Performance is often an important aspect of ABM, as researchers usually express a general need to explore the emergence resulting from the simulation of tens and even hundreds of thousands of agents. In these simulations, emergent behaviour frequently becomes more prominent with greater numbers of agents [23].

Since it is expensive to perform clustering every single frame, we seek to improve performance in a couple of ways, beyond just implementing the algorithm on the GPU. Firstly, we use a tiling algorithm devised by Nyland et al. [24,25] in order to perform the all-pairs style computation shown in Algorithm 3. This is a well known method for hiding global memory latency in CUDA-enabled devices. It uses shared memory of the CUDA enabled device to load "tiles" of the total number of boids, which is then processed by the block being run on the streaming multiprocessor (SM). CUDA organises sections of boids into "blocks" of threads which are executed by SMs. Modern NVidia cards have several SMs and these execute blocks until they are complete. Shared memory is local to the block currently being executed on an SM. It is effectively a very fast cache, since accessing global memory on the GPU is very expensive, reaching into the hundreds of clock cycles to fetch data from global memory.

Shared memory is used by defining it first as follows:

```
__shared__ float xshare[tile_size];
```

At this point, this arrays can be used, but it will go out of scope as soon as the block has finished executing. To perform tiling, you can expect to see a section of code in the shape of the following:

```
for (int i=0; i < (N/tile_size); ++i) {

  globalindex = i * tile_size
      + thread_index_in_block;

  xshare[thread_index_in_block]
      = x[globalindex];

  __syncthreads();

  // Perform processing here

  __syncthreads();
}
```

Where x is the array in global memory. A "tile" is loaded into shared memory, and then the threads in the block are synchronised, after which some processing follows. Finally, another synchronisation follows the processing. This is done so that the shared memory is not replaced by threads that finish the processing section first before others are finished using the shared memory. In our simulation the processing contains many conditionals to determine which boids belong to which set. These conditionals come at a performance loss, but it allow us to observe interesting emergent behaviour with more than one flock of boids.

# 4 Performance Results

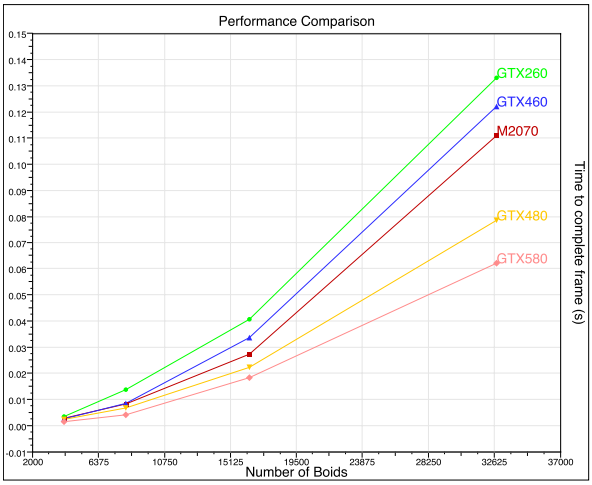


Figure 2: Performance scaling with increasing numbers of simulated boids on various GPU models.

Figure 2 shows a pseudo-quadratic increase in frame completion time, as is expected from the $O(N^2)$ nature of the *all-pairs* computation in the Boids algorithm. We tested the implementation across 5 different NVidia cards (GTX260, GTX460, GTX480, GTX580, Tesla M2070) with 4096, 8192, 16384 and 32768 boids.

It seems that the coefficient of the $O(N^2)$ curve may be quite low resulting in a steady but delayed increase in frame computation time. This could be attributed to the effect of the tiling technique used to hide global memory latency in the CUDA implementation. However, since the implementation does not include any spatial data structure or sorting mechanism, computation is still based on *all-pairs*, which would explain the clearly non-linear nature of the curve.

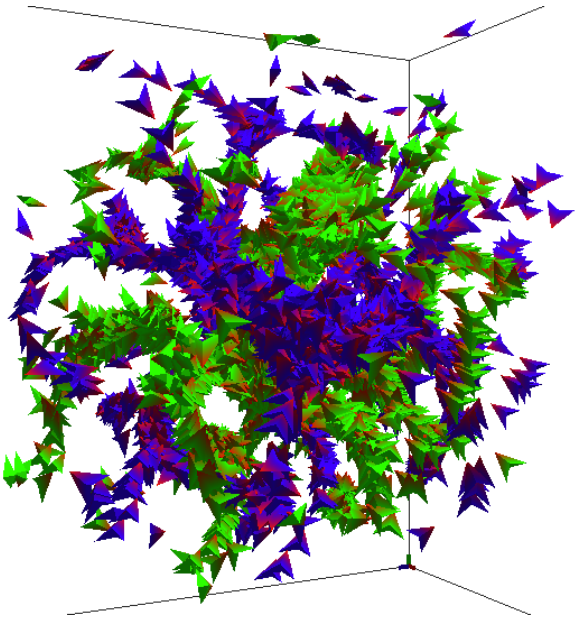

Figure 3: Tribe/flock interaction creating complex emerging patterns.

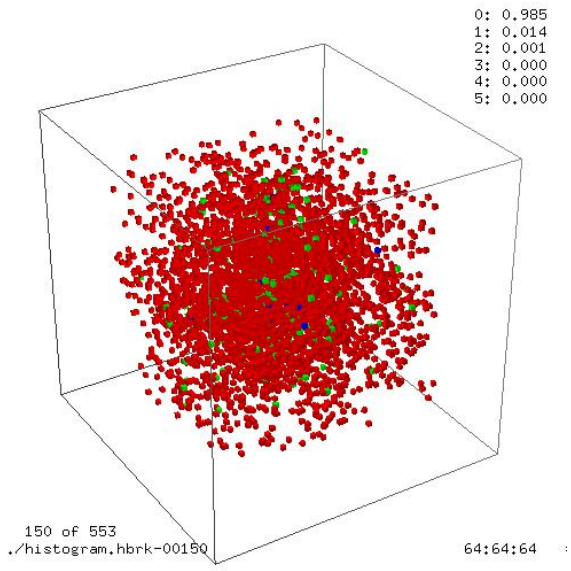


Figure 4: 3D histogram of space occupation with 4096 boids, near beginning of simulation.

## 5 Flocking Separation Results

An interesting metric we use to study the flocking model behaviour is histogramming in 3D-space. Essentially the 3D space is discretised and then each boid's position is mapped to a discrete cube in the 3D space, and each corresponding cube has its occupancy incremented by one. We expect the pattern to be spatially homogeneous - with no preferred direction or asymmetries, providing the model is correctly implemented.

Figure 3 shows an early pattern of the spatial occupancy histogram. A pattern is already beginning to emerge showing that the probability of boids occupying a particular region in the confinement box is spatially symmetric but weighted towards the centre of the box, due to the residual effect of the "goal rule". Boids swoop and swirl around the box avoiding the walls.

Figure 4and Figure 5 show early and later snapshots of the spatial occupancy histograms. The pattern is spatially symmetric - on average, with smoothness increasing as we average over more independent runs. The "boids" are most likely to occupy space in the centre of the box, with decreasing probability of occupancy near the walls and corners.

A useful means of analysing the multi-flock model is to consider the number of distinct spatial flocks that can be identified as the model evolves in time. We can use the boid-boid separation distance parameter as discussed above to define whether any two individual boids are "connected". An instantaneous breadth-first graph search of connected boids followed by labelling the boid components or separate flocks. There are a number of algorithms that can be employed to compute the number of flocks quickly, including the component labelling approach that can be carried out on GPUs using data-parallelism [26]. There are various sophisticated analyses we could perform such as tracking the moments of the geometrical shapes of flocks but for the purposes of this present paper we report only the number of distinct flocks as it changes with time. We can obtain $< N_{\text{flocks}}(\text{time}) >$ where the average is over many independently (randomly seeded) runs with the same parameters. The findings are shown in Figure 6.

From visual inspection it seems that regardless of the spatial dimensions of the simulation or indeed the number of boids, the clusters will more or less always converge at some constant value, provided that the number of boids is the only variable. In the results shown in Figure 6, it seems that this value is approximately 1000 time steps (or animation frames). Using the GPU-accelerated simulation program we have been able to explore quite large boid numbers and many independent runs. Even with a large difference in the number of boids, it still seems very clear that this constant value does not change.

Figure 7 shows the number of flocks plotted on a logarithmic scale. This emphasises the change in behaviour after the first 1000 time steps or frames. Independent

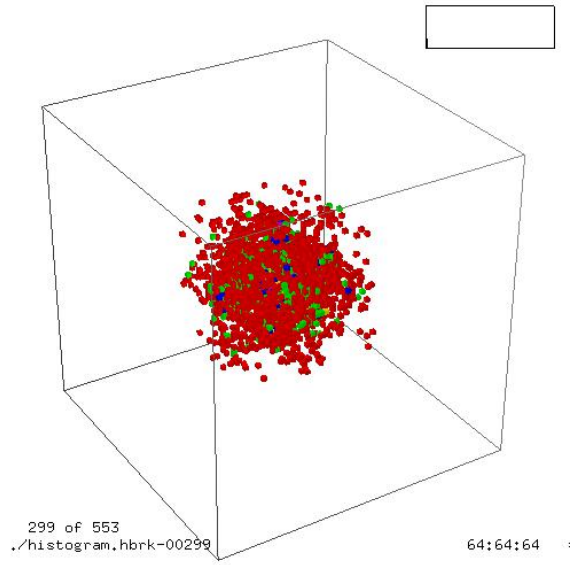


Figure 5: 3D histogram of space occupation with 4096 boids, differing colours represents higher levels of occupancy. This histogram was generated at a couple of hundred time steps later than Figure 4.

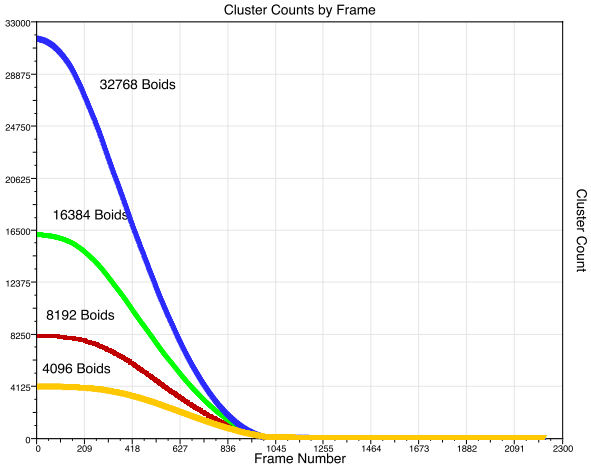


Figure 6: The distribution of clusters in linear time, each colour representing a number of boids ranging from 4096 (yellow) to 262144 (green).

of the number of number of boids in the system the curves approximately intersect after 1000 steps. This particular time is likely related to the typical velocity and box dimensions. After this time, the flocks have had a chance to thoroughly explore the box and mix or unmix accordingly. After this time regime the system tends towards more convergent behaviour.

We can further analyse the evolution of the number

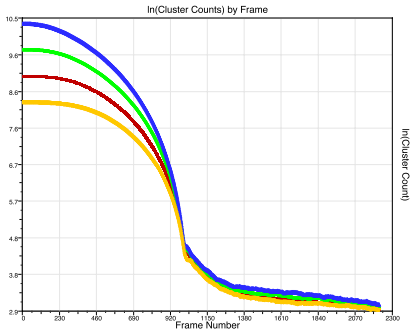


Figure 7: Log-linear plot of the cluster (flock) count averaged over independent runs.

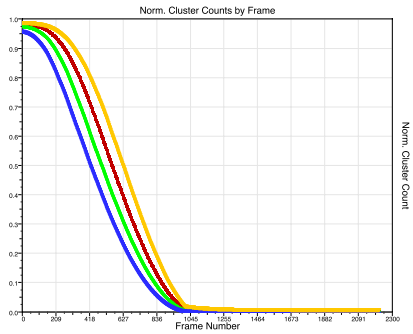


Figure 8: Log linear xy plot of the cluster count.

of flocks by considering a normalised version of the data thus allowing a comparison with different system sizes Figure 8 shows how similar the curves are in form-factor when this normalisation is carried out. It shows that the end of the early mixing regime is delayed when there are fewer boids in the system. Larger boid numbers lead to an earlier fall in the number of flocks present.

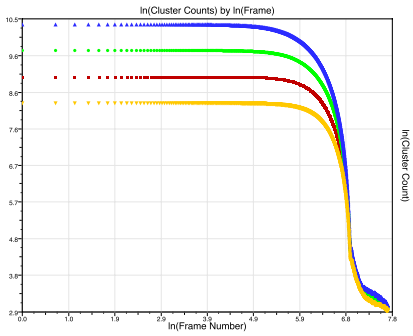


Figure 9: Log Log xy plot of the cluster count.

Figure 9 shows the normalised data plotted on a log-log scale further emphasising the sharp cutoff as the boid flocks have explored their interaction space and have

separated out apart from some stragglers and some dynamical fluctuations. The y-intercept values correlate with increased boid numbers in the system but the cut-off point when temporal behaviour changes is shown even more sharply and universally, independent of the number of boids.

# 6 Discussion

The boids model appears to scale well on GPUs. The performance gains from the use of GP-GPU are satisfactory and there are opportunities for other layers of parallelism such as job-task management by deploying on multiple-GPUs either controlled by a single CPU or hosted on separate nodes of a compute cluster. We employed relatively simple data partitioning techniques to manage agent-agent communications across s the spatially localised agents in the model. There is scope to employ space partitioning techniques such as oct-trees to speed up the $O(N^2)$ communications to nearer $O(N \log N)$ which should support even bigger agent numbers.

We have successfully extended the boids model to support multiple flocks but at the expense of quite a number of adjustable parameters. It is likely that some of these are not in fact independent and may be reduced down to fixed ratios in terms of box size. There does appear to be a very clear indication of a fixed and universal time changeover point in behaviour - the exact time is likely related to our velocity parameters and their relationship to box edge length parameters of the simulation.

The component labelling of the boid clusters/flocks is not especially computationally expensive compared to the computational costs of time-stepping the model. This will not necessarily scale up beyond the $\approx 10^5 - 10^6$ boid numbers that we explore in this paper.

We have tried to interpret our model in terms of other reported work on simulated boids. The precise nature of the parameters and multiple-flock behaviours is unfortunately not yet generally agreed upon, making it difficult to make unambiguous comparisons. This is certainly true when comparing against other ABM models which have highly constrained and constant communication.

The interactive model performs well and the flocking and flock-separation behaviour is visually appealing as an animation. It appears very similar to the behaviours observed of fish such as herring, where the rapid movements of individuals belies the emergent collective behaviour of the shoal. There is open potential to augment the model with new behaviours and indeed to apply genetic evolutionary effects [18] to introduce artificial intelligent behaviours amongst the agents semi-automatically.

# 7 Conclusion

We have extended Reynold's Boid model to support multiple distinct flocks or clusters of individual agents. We have shown how this model can be implemented using GPUs to support interactive performance of models that are still large enough to capture logarithmic multiple-time and multiple-length scale behaviours.

There are a number of additional features we hope to add to the model including different containment geometries and the application of external driving forces to bias the agents such as gravity. These effects are well understood in terms of the physical mechanics but have not been widely explored in the context of potentially intelligent agent-based models. The effect of agents having goals and "will power" to choose according to a strategy is likely to significantly change the thermodynamically driven baseline behaviour of non-intelligent physical systems. This raises interesting issues in artificial intelligence concerning just how intelligent life forms like fish or birds are and how intelligent they need to be to explain their observed collective behaviours.

At present our multi-flock boids are still very simple agents, with a tribal preference but still with no more artificial intelligence than single-flock boids. We are experimenting with other simple agent intelligence behaviours such as predation, reproduction and the use of simple feeding chains. We expect these effects will be sustainable providing a large enough number of agents can be simulated.

The GPU devices we have employed are entirely adequate for a user-interactive multi-flock model as might be used in a game to animate or drive character movements. The statistical mechanical analyses we hope to pursue will likely require the use of multiple-GPUs to scale up the simulation and analysis algorithms to suitable time and length scales however. It is likely that the number of cores and floating point units available in GPUs will offer considerably more power for accelerating the performance of agent-based models such as the multi-flocking boid system.

# References


[1] Ferber, J.: Multi-Agent Systems - An Introduction to Distributed Artificial Intelligence. Addison-Wesley (1999)

[2] d'Iverno, M., Luck, M.: Understanding Agent Systems. Number ISBN 3-540-41975-6. Springer (2001)

[3] Milner, R.: The Space and Motion of Communicating Agents. Number ISBN 9780521738330. Cambridge (2009)

[4] Chan, W.K.V., Son, Y.J., Macal, C.M.: Agent-based simulation tutorial - simulation of emergent behavior and differences between agent-based simulation and discrete-event simulation. In: Proc. 2010 Winter Simulation Conference, Baltimore, MD, USA, 5-8 December. (2010) 135–150 ISBN 978-1-4244-9864-2/10.

[5] Macal, C.M., North, M.J.: Tutorial on agent-based modeling and simulation part 2: How to model with agents. In: Proc. 2006 Winter Simulation Conference, Monterey, CA, USA. (3-6 December 2006) 73–83 ISBN 1-4244-0501-7/06.

[6] Lerner, A., Chrysanthou, Y., Shamir, A., Cohen-Or, D.: Data driven evaluation of crowds. In: Proceedings of the 2nd International Workshop on Motion in Games. MIG '09, Berlin, Heidelberg, Springer-Verlag (2009) 75–83

[7] Kaup, D.J., Clarke, T.L., Oleson, R., Malone, L.C.: Crowd dynamics simulation research. In: Proc. 16th Conf. on Behaviour Representation in Modeling & Simulation (BRIMS'07), Orlando, FL, USA. (2007) 173–180

[8] Gerdelan, A.P.: Fuzzy Motion Controllers and Hybrids. PhD thesis, Computer Science, Massey University, Albany, New Zealand (2011)

[9] Thanngiah, S.R., Shmygelska, O., Mennell, W.: An agent architecture for vehicle routing problems. In: Proc 2001 ACM Symposium on Applied Computing, Las Vegas, USA (2001) 517–521 ISBN 1-58113-287-5.

[10] Settembre, G.P., Scerri, P., Farinelli, A., Sycara, K., Nardi, D.: A decentralized approach to cooperative situation assessment in multi-robot systems. In: Proc. of 7th Int. Conf. on Autonomous Agents and Multiagent Systems (AAMAS 2008). Number ISBN:978-0-9817381-0-9, Estoril, Portugal (May 2008) 31–38

[11] Hawick, K.A., Scogings, C.J., James, H.A.: Defensive spiral emergence in a predator-prey model. Complexity International (msid37) (October 2008) 1–10 ISSN ISSN 1320-0682.

[12] Scogings, C., Hawick, K.: Altruism amongst spatial predator-prey animats. In Bullock, S., Noble, J., Watson, R., Bedau, M., eds.: Proc. 11th Int. Conf. on the Simulation and Synthesis of Living Systems (ALife XI), Winchester, UK, MIT Press (February 2008) 537–544

[13] Jim, K., Giles, C.: Talking helps: evolving communicating agents for the predator-prey pursuit problem. Artificial Life **6**(3) (2000) 237–254 Summer.

[14] Reynolds, C.: Flocks, herds and schools: A distributed behavioral model. In: SIGRAPH '87: Proc. 14th Annual Conf. on Computer Graphics and Interactive Techniques, ACM (1987) 25–34 ISBN 0-89791-227-6.

[15] Reynolds, C.: Boids background and update. `http://www.red3d.com/cwr/boids/` (July 2011)

[16] Mekni, M.: Hierarchical path planning for situated agents in informed virtual geographic environments. In: Proc. SIMUTools 2010, 3rd International ICST Conference on Simulation Tools and Techniques, Torremolinos, Malaga, Spain. (2010)

[17] Reynolds, C.W.: Steering behaviors for autonomous characters. In: Proc. Game Developers Conference, San Jose, CA, USA. (1999) 763–782

[18] Birt, A., Shaw, S.: Evolving boids: incorporating machine learning into artificial life. Volume 18., USA, Consortium for Computing Sciences in Colleges (May 2003) 290–291

[19] da Silva, A.R., Lages, W.S., Chaimowicz, L.: Improving boids algorithm in gpu using estimated self occlusion. In: Proc. SBGames'08, Belo Horizonte, Brazil. (10-12 November 2008) 41–46 ISBN 85-766-9217-1.

[20] Richmond, P., Romano, D.: Agent based gpu, a realtime 3d simulation and interactive visualisation framework for massive agent based modelling on the gpu. In: Proceedings International Workshop on Super Visualisation (IWSV08), Press (2008)

[21] Richmond, P., Coakley, S., Romano, D.M.: A high performance agent based modelling framework on graphics card hardware with cuda. In: Proceedings of The 8th International Conference on Autonomous Agents and Multiagent Systems - Volume 2. AAMAS '09, Richland, SC, International Foundation for Autonomous Agents and Multiagent Systems (2009) 1125–1126

[22] Aaby, B.G., Perumalla, K.S., Seal, S.K.: Efficient simulation of agent-based models on multi-gpu and multi-core clusters. In: Proc. SIMUTools 2010, 3rd International ICST Conference on Simulation Tools and Techniques, Torremolinos, Malaga, Spain. (15-19 March 2010) 1–10

[23] Perumalla, K.S., Aaby, B.G.: Data parallel execution challenges and runtime performance of agent simulations on gpus. In: SpringSim '08: Proceedings of the 2008 Spring simulation multiconference, New York, NY, USA, ACM (2008) 116–123

[24] Nyland, L., Harris, M., Prins, J.: Fast n-body simulation with cuda. In Nguyen, H., ed.: GPU Gems 3. Addison Wesley Professional (August 2007)

[25] Playne, D.P., Johnson, M.G.B., Hawick, K.A.: Benchmarking GPU Devices with N-Body Simulations. In: Proc. 2009 International Conference on Computer Design (CDES 09) July, Las Vegas, USA. Number CSTN-077 (July 2009)

[26] Hawick, K.A., Leist, A., Playne, D.P.: Parallel Graph Component Labelling with GPUs and CUDA. Parallel Computing **36** (2010) 655–678