Emergent Effects in Massive Agent Swarms in Real-time Game Environments

Owen Knight, Tim Wilkin *Member, IEEE,* and Shaun Bangay
School of Information Technology
Deakin University

***Abstract*—Computational efficiency and hence the scale of agent-based swarm simulations is bound by the nearest neighbour computation for each agent. This article proposes the use of GPU texture memory to implement lookup tables for a spatial partitioning based k-Nearest Neighbours algorithm. These improvements allow simulation of swarms of 2^{20} agents at higher rates than the current best alternative algorithms. This approach is incorporated into an existing framework for simulating steering behaviours allowing for a complete implementation of massive agent swarm simulations, with per agent behaviour preferences, on a Graphics Processing Unit. These simulations have enabled an investigation of the emergent dynamics that occur when massive swarms interact with a choke point in their environment. Various modes of sustained dynamics with temporal and spatial coherence are identified when a critical mass of agents is simulated and some elementary properties are presented. The algorithms presented in this article enable researchers and content designers in games and movies to implement truly massive agent swarms in real time and thus provide a basis for further identification and analysis of the emergent dynamics in these swarms. This will improve not only the scale of swarms used in commercial games and movies but will also improve the reliability of swarm behaviour with respect to content design goals.**

***Index Terms*—swarms, steering behaviour, gpu, emergence**

I. Introduction

Agent swarms have been used in a variety of computer game and animation products, such as the penguins in “Batman 2” or the horde of Saruman in “Lord of the Rings: The Two Towers”. Collective behaviours of the swarm represent higher order system dynamics arising from the low level interactions of individual agents, both with their neighbours and with elements of the local environment. Emergent dynamics and the resulting swarm behaviours are becoming increasingly relevant as the number and density of agents increases and as environments are made more complex. An understanding of these emergent dynamics and the relationship to the underlying agent interaction model is necessary if truly massive swarms are to be simulated, with predictable behaviours that meet content design goals. However, to achieve this understanding,the computational bottlenecks that limit the scale and efficiency of swarm simulations must be overcome.

There are two prevalent questions we seek to address in this research program:

1) How can very large swarms be simulated within the performance requirements of a typical game or animation setting?
2) What are the implications of introducing very large swarms into complex game environments?

With regard to question 1), we have devised a solution for addressing the current major bottleneck in the process: the task of finding the occupants of the *local neighbourhood* of each agent, which affect its subsequent behaviour. This solution is optimized for implementation on the Graphics Processing Unit (GPU), making it suitable for current gaming platforms. Additionally, we have implemented a per-agent-customisable decision procedure and implemented this in a Single Procedure Mutiple Data (SPMD) framework, allowing us to implement the entire swarm simulation on a GPU.

As for the implications of deploying swarms into complex game environments, we have conducted preliminary studies of the emergent dynamics that arise when the numbers of agents moving through a chokepoint increases above a critical threshold. We have identified several interesting phenomena that correlate with those found in traffic and crowd simulations, as well as some unusual phenomena more akin to the bubbles seen in an hour glass as it flows.

II. Related Work

Over the years there have been a few approaches to agent-based swarm modelling, using flocking [14], steering behaviours [15] and social forces [8] to define interaction rules between local agent neighbourhoods. These approaches are open to parallelization and packages such as the OpenSteer toolkit have been adapted to use CUDA [17] which allows agent behaviour simulation to occur in parallel, improving performance.

Local interaction requires that agents be aware of the properties of their neighbours. The nearest-neighbour search is shown to be the most expensive step required by the classic agent-based methods [11], [1], [18]. The simplest algorithm for computing the nearest neighbours is the brute-force algorithm, which is inherently parallelizable as the distances between pairs of tuples are independent [10], [5]. Parallel versions exhibit speedups ranging from 47 [10] to 148 when using a KD-tree implementation [5]. Breitbart [2] has demonstrated a brute force k-NN implementation for the OpenSteer framework using CUDA and achieving a 42 speedup, simulating 4096 flocking agents at 214 fps.

Spatial data structures that exploit the coherence improve performance. Two-dimensional spatial grids [12], [13] simulate 10,000 pedestrians at 80 fps using a 10 CPU cluster. A three-dimensional static grid with a heuristic to update the neighbour information where agents are changing direction simulates 32,768 agents on the GPU at 30 fps [4]. Computations are still dominated by the neighbourhood search which takes over half of the computation time.

Our work is based on the CUDA based static grid implementation [6]. Here agents calculate a bin identifier and the list of identifiers is sorted to find agents in the same bin. The highly optimised parallel radix sort used is shown to be 23% faster than an optimised multicore CPU sorting algorithm [16]. This process is simulates 65,536 particles at 120 fps.

An alternative to the agent-based approach is the macroscopic modelling of crowd dynamics [19], using a global potential field which drives the motion of the individuals. Fields can be based on potential functions, radial basis functions, fluid dynamics, gas-kinetic models, incompressible flows and continuum dynamics. Macroscopic models produce a number of emergent phenomena witnessed in real crowds, but due to their resolution and the continuous nature of the field, other dynamics are hard to model using this method. The underlying functions are also not easily parallelizable, which precludes the macroscopic models from benefiting from GPU acceleration. The continuum method also does not scale well with increase in resolution of the underlying grid representation [9].

Continuum models produce smooth flows of motion but can omit some of the emergent effects that occur through interactions of discrete individuals as found in most game settings. Examples include traffic jams that occur in choke points when crowd of people attempt to pass through a narrow opening [3]. Flows of traffic are also inherently unstable due to the erratic behaviour of individual vehicles [7] and a chain reaction of followers. One small fluctuation caused by a single "near miss" can grow as the chain of following vehicles also overreact to avoid accident. If the density of vehicles is sufficiently high this can lead to a "phantom traffic jam" which then propagates upstream into the traffic flow. This is one of the effects we demonstrate with our massive swarms.

III. GPU OPTIMIZED SWARM SIMULATION

Agent simulation is based on steering behaviours, where the forces that determine the movement of each agent are related to the position and other properties of other agents and obstacles within a local neighbourhood around that agent. We use a bounded k-nearest neighbours (kNN) algorithm to determine the k nearest neighbours for each agent that affect its behaviour. The key steps during the simulation are as follows:

```
for each iteration
  for each agent
     solve kNN
  for each agent
     resolve decisions using kNN
  physically simulate behaviour
  resolve collisions
```

Our implementation is incorporated in the OpenSteer framework allowing ready customization of the group behaviour layer with each of the steering behaviours implemented as a separate CUDA kernel. The simulation environment used for considering the emergent behaviour is created as a plugin that uses this layer. We extend the behaviour blending code to include a prioritized weighted blending scheme which allows some behaviour calculations to be bypassed when other more urgent demands exist (such as the need to avoid obstacles).

The nearest neighbour computation has the highest complexity and dominates the other steps. We address this by employing an algorithm with reduced complexity and optimize this for usage on the SPMD architecture of the GPU. Green's method [6] is used as a starting point where the environment is logically decomposed into a grid of cells. Agents are sorted according to cell index on the GPU using key/value parallel radix sort. The start and end position for each cell index in the sorted list is a coherent sequence that identifies the agents occupying each cell. This can further be exploited for later cache coherence by rearranging the agent data in the same order.

We make a number of refinements to this process to further improve performance. Since the spatial decomposition is static we look up the index of each cell using 3D texture memory. While the texture cache provides some acceleration, this step relies on the efficient thread scheduling to hide the latency of the texture read operation. This permits future use of non-uniform grids. Texture addressing also allows boundaries to be clamped or wrapped for toroidal topologies. The k nearest neighbours within a bounded number of cells about that containing each agent are identified by computing the distance to each agent within these cells. A texture look up process is used to exploit a precomputed table identifying which grid cells are within the desired neighbourhood which further reduces run-time computation by a small amount (<10%).

We have compared the results of our refinements with:

- our implementation using only the original process described by Green [6] on the GPU and
- the CPU based grid approach implemented by Breitbart [1] also under OpenSteer (their GPU implementation was slower than the CPU version).

Our implementation of Green's process runs at comparable levels to the original reported: achieving 80 fps for 65,5366 agents as opposed to the 120 fps reported which we attribute to the greater generality in the OpenSteer framework. This version still runs 67 times faster than the Breitbart CPU code for 262,144 agents. The CPU code has higher performance only for less than 512 agents. Our implementation using texture lookups consistently improves performance, exceeding our implementation of Green's processby a factor of 97 for 1,048,576 agents.

IV. PERFORMANCE ANALYSIS

We investigate scaling to massive swarms by measuring the agent update rate as the number of agents changes to establish the complexity of our implementation in practice.

The simulation scenario tested consists of a choke point simulation where each agent performs basic flocking. Anti-penetration testing and collision forces control agent overlap with each other and the walls of the environment. Agents steer towards two target points: in the mouth of the choke point and

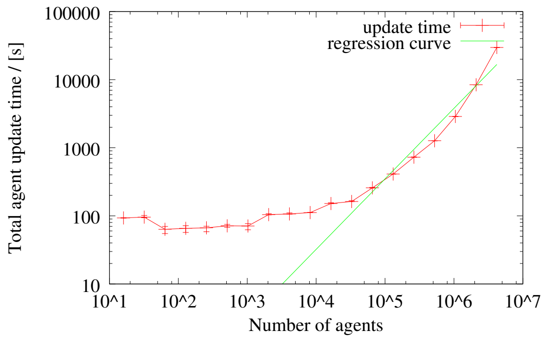


Figure 1: Performance achieved for large scale simulations.

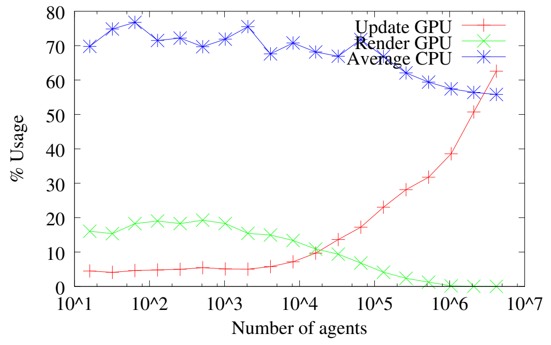


Figure 2: Processor usage during simulation.

then, once that is attained, to a distant point in line with the gap. Agent dynamics are implemented using an Euler step update. We use these same settings to demonstrate emergent effects triggered by critical swarm size in the next section.

We measure the amount of time spent during 50,000 iterations of the agent update step, representing the nearest neighbour and agent simulation calculations. Explicitly excluded is the rendering time which varies with quality of rendering and the number of agents visible at any point.

We perform all measurements using a dual core AMD Athlon II running at 3.1 GHz and two NVIDIA Geforce GTX 570 graphics cards. One GPU is dedicated to the agent update while the other is used for rendering. We report the results of 7 iterations, showing both the average result and the minimum/maximum values achieved. These values are presented in Figure 1. Both horizontal and vertical axes are shown using logarithmic scales. Some variation in times between runs is attributed to CPU-GPU synchronization in each frame as this effect is both non-deterministic and unrelated to the numbers of agents.

For numbers of agents below 2^{15}, the update time is effectively constant. The process is CPU bound and dominated by transferring data to the GPU and CPU/GPU synchronization. Variance is caused when several agents occupy the same bin improving efficiency. When the number of agents increases above 2^{15} the amount of work required by the GPU becomes significant. This is relevant considering previous approaches have often utilized relatively small numbers of agents.

The CPU/GPU shift is visible in the results from the NVidia performance monitor shown in Figure 2. For larger numbers of agents the usage on the update GPU increases while the CPU has increasing proportions of idle time.

The complexity of the GPU bound upper half of the data set (for number of agents $\geq 2^{15}$) fits the regression curve shown in figure 1: $2.32 \times 10^{-3} n^{1.0353}$ for n agents ($R^2 = 0.961$) indicating complexity is close to linear in practice and substantially below the $O(n^2)$ required by a pairwise comparison of all agents. Agent numbers are limited only by available GPU memory. The speed and scale of this simulation is greater than what have been achieved previously on comparable hardware and would continue to be relevant as GPU hardware continues to advance.

V. Large scale emergent behaviour

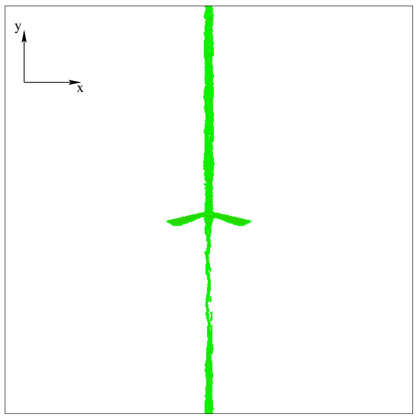
Massive agent simulations produce novel forms of emergent behaviour. Examples of desirable, undesirable and unexpected emergent patterns in the context of the choke point scenario are presented.

Agents are placed in a continuous environment of dimensions 2048×2048. Agents start in a rectangular region of dimensions 900×450 in the bottom half and are directed upwards through a centrally located gap that is 40 long and 27 wide, representing a choke point that all agents have to funnel through. The walls outside the choke point are inclined at an angle of $13°$ to divert agents towards the gap. Agents are initially packed in a regular grid fashion. For large simulations agent separation is below the width of each agent. Each agent tries to remain at least 0.5 units from all other agents. The environment is wrapped toroidally to keep agent numbers constant.

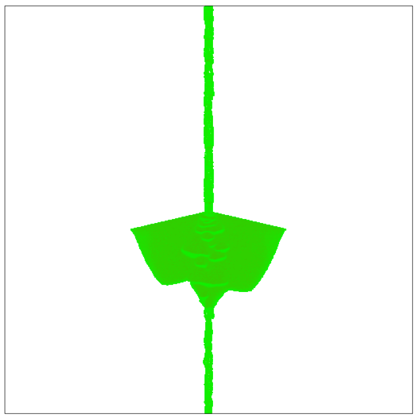
Examples of the configurations achieved are shown in Figure 3. with darker shades of green indicating higher agent densities, changing to red for very high levels. For relatively small numbers of agents (Figure 3a) the choke point has little impact on speed of agents. Agents align into a coherent stream. Some minor congestion occurs at the entrance to the choke point. Significantly, variation in density occurs in the outflow of the choke point where the minor variations in agent speed and direction can become amplified as agents press in from behind.

Congestion causes increases in local agent density as shown in Figure 3b. Bubbles of low density regions form before the choke point. Wave-like patterns in the density plots are visible at a global level. Similar structures of high density regions exists beyond the outflow of the choke point and propagate opposite to agent movement. These regular structures are present only when the discrete agents are available in sufficient *density*.

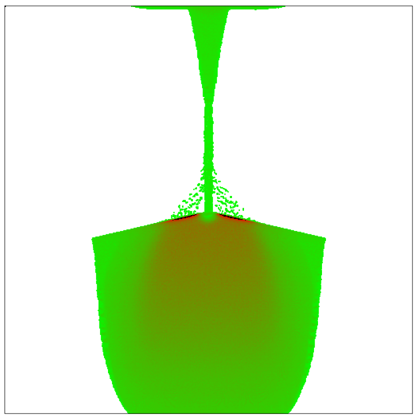
Very large numbers of agents shown in Figure 3c create very high agent densities from the start of simulation, sufficient to prevent any low density bubbles. Bubbles are confined to boundary regions. Peak agent densities do not occur within the choke point, but in a region before it. Agent densities are high enough to overcome barrier forces, allowing some agents to tunnel through. This undesired behaviour is due to the stiffness causing numerical integration failure for these very high densities.



(a) 32768 agents.



(b) 262144 agents.



(c) 2097152 agents.

Figure 3: Agent density across the simulation grid.

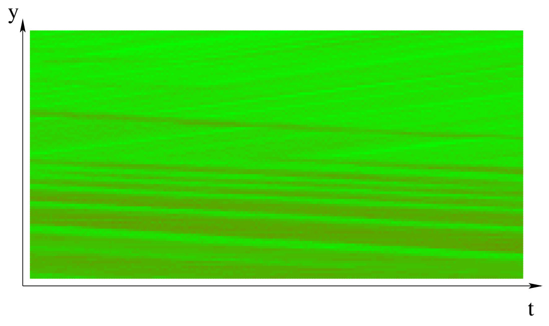


Figure 4: Space-time density variation.

The repeated patterns that emerge resemble waves but lack wave properties. The interval between wave fronts is not constant (Figure 3b) and we observe that waves do not travel at consistent speeds. Fronts overtake and merge with neighbours. Most are triggered by build up of agents about the choke region but agent collisions within the critically packed stream after the choke point can also produce density bubbles. The space-time density diagram shown in Figure 4 allows further analysis of these emergent structures. This shows a slice through the center of the choke point for successive frames of the simulation. Linear structures with positive gradients represent the trajectory of agent clusters moving upward. The wave-like structures are present as linear features with negative gradient.

Gradient (speed) is constant for each structure. What does change is coherence - density peaks diffuse outwards and merge with neighbours. It appears that the trailing edge of the density peak moves more slowly than the leading edge. This is consistent with actions at an agent level: agents in the leading edge (for the wave moving downwards) are instituting rapid braking to avoid collision, which those at the trailing edge are gradually accelerating. Hence agents are likely to pack up more rapidly at the leading edge.

VI. Conclusions

The techniques described use off-the-shelf GPU hardware to achieve near real-time simulation of massive agent swarms. The approach improves on similar previous work through further optimizations and applies them within a framework for agent simulation. The technique scales efficiently to larger swarms than have been typically employed before. Such dynamics will be found in commercial games within the next few years. Emergent effects, such as those illustrated, become apparent as swarms become massive. These effects, whether desired or unexpected, will be present in the next generation of computer games. Analysis of the emergent behaviour offers insight into the choices made in the steering behaviour of individual agents.

This work only hints at some of the issues that game developers will soon encounter. The choke point simulation is also only one of the cases typical of large strategy games. Future work will apply massive agent simulation to other common game scenarios and explore further categories of emergent behaviour that may be revealed.

References

[1] Jens Breitbart. Case studies on gpu usage and data structure design. Master's thesis, University of Kassel, Germany, 2008.

[2] Jens Breitbart. *A framework for easy CUDA integration in C++ applications*. University of Kassel, Germany, University of Kassel, Germany, 2008.

[3] Helbing Dirk, Farkas Illes, and Vicsek Tamas. Simulating dynamical features of escape panic. *Nature*, 407(6803):487–490, 2000.

[4] Ugo Erra, Rosario De Chiara, Vittorio Scarano, and Maurizio Tatafiore. Massive simulation using gpu of a distributed behavioral model of a flock with obstacle avoidance. In *Proceedings of Vision, Modeling, and Visualization 2004 (VMV 2004)*, pages 233–240, 2004.

[5] V. Garcia, E. Debreuve, and M. Barlaud. Fast k nearest neighbor search using gpu. In *IEEE Computer Society Conference on Computer Vision and Pattern Recognition Workshops, 2008: CVPRW '08*, pages 1–6, 2008.

[6] Simon Green. Cuda particles. Technical report, NVIDIA Corporation, 2008.

[7] Dirk Helbing. Traffic and related self-driven many-particle systems. *Rev. Mod. Phys.*, 73(4):1067–1141, 2001.

[8] Dirk Helbing and Peter Molnar. Social force model for pedestrian dynamics. *Phys. Rev. E*, 51(5):4282–4286, 1995.

[9] Xiaogang Jin, Jiayi Xu, Charlie C.L. Wang, Shengsheng Huang, and Jun Zhang. Interactive control of large-crowd navigation in virtual environments using vector fields. *IEEE Computer Graphics and Applications*, 28:37–46, 2008.

[10] Shenshen Liang, Ying Liu, Cheng Wang, and Liheng Jian. A cuda-based parallel implementation of k-nearest neighbor algorithm. In *International Conference on Cyber-Enabled Distributed Computing and Knowledge Discovery, 2009: CyberC '09.*, pages 291–296, 2009.

[11] Rahul Narain, Abhinav Golas, Sean Curtis, and Ming C. Lin. Aggregate dynamics for dense crowd simulation. *ACM Transactions on Graphics*, 28(5):122–1, 2009.

[12] Michael J. Quinn, Ronald A. Metoyer, and Katharine Hunter-zaworski. Parallel implementation of the social forces model. In *in Proceedings of the Second International Conference in Pedestrian and Evacuation Dynamics*, pages 63–74, 2003.

[13] Craig Reynolds. Big fast crowds on ps3. In *Proceedings of the 2006 ACM SIGGRAPH symposium on Videogames*, Sandbox '06, pages 113–121, New York, NY, USA, 2006. Acm.

[14] Craig W. Reynolds. Flocks, herds and schools: A distributed behavioral model. In *Proceedings of the 14th annual conference on Computer graphics and interactive techniques*, SIGGRAPH '87, pages 25–34, New York, NY, USA, 1987. Acm.

[15] Craig W. Reynolds. Steering behaviors for autonomous characters. In *Proceedings of Game Developers Conference 1999*, pages 763–782, 1999.

[16] Nadathur Satish, Mark Harris, and Michael Garland. Designing efficient sorting algorithms for manycore gpus. In *Parallel Distributed Processing, 2009. IPDPS 2009. IEEE International Symposium on*, pages 1–10, 2009.

[17] Simon Stiefel. Parallelizing the opensteer toolkit for simulation of steering behaviors of autonomous characters using graphics processing units. Technical report, Faculty of Information Technology, Esslingen University of Applied Sciences, 2009.

[18] Wen Tang, Tao Ruan Wan, and Sanket Patel. Real-time crowd movement on large scale terrains. In *Proceedings of the Theory and Practice of Computer Graphics 2003*, TPCG '03, page 146, Washington, DC, USA, 2003. IEEE Computer Society.

[19] Adrien Treuille, Seth Cooper, and Zoran Popovic. Continuum crowds. *ACM Transactions on Graphics*, 25(3):1160–1168, 2006.