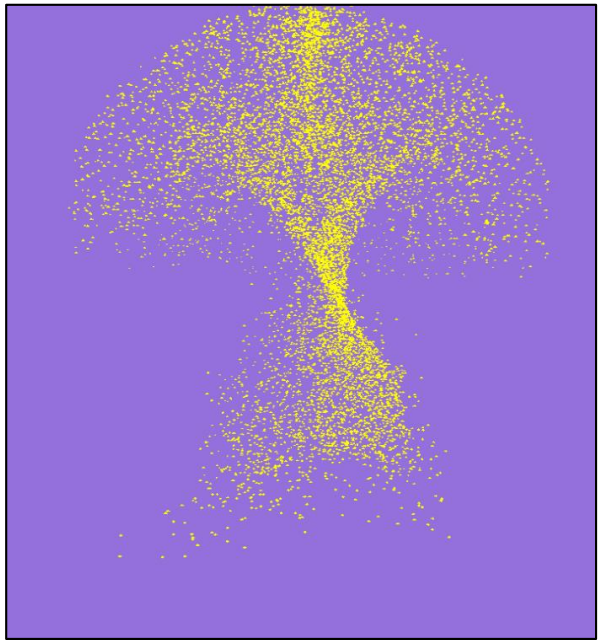
Advanced Technologies: Swarm Intelligence

Dudley Dawes [15017153]
Department of Computer Science and Creative Technologies
University of the West of England
Coldharbour Lane
Bristol, UK
dudley2.dawes@live.uwe.ac.uk



Abstract
The aim of this project is to simulate the swarm intelligence technique, particle swarm optimisation, with up to one-hundred thousand meshes, using the DirectX11 API. This paper will discuss the research leading up the simulation, the outcome and an evaluation of the process.

Author Keywords
Swarm Intelligence; Particle Swarm Optimisation; Artificial Life; GPGPU

Introduction
Particle Swarm Optimisation (PSO) was originally developed to optimise continuous nonlinear functions and derives from artificial life, in particularly Swarm Behaviour (Kennedy and Eberhart, 1995). The behaviour exhibited by particle swarm simulations are visually akin to flocks of birds and schools of fish.

Due to the high dependency on stochastic processes, the PSO algorithm can be likened to Genetic Algorithms (GA) and Evolutionary Programming (EP) (Kennedy and Eberhart, 1995). This is because the concept of the fitness function, known as *crossover* in GA and EP, is used in PSO algorithms in the form of the global and local best positions (Coppin, 2004).

The early applications using PSO including training a Neural Network to solve the nonlinear Schaffer f6 function and the Fisher Iris Data Set (Kennedy and Eberhart, 1995). Due to its inexpensive computation, PSO is a candidate for integration within a wider Game Engine.

Background Research

The five basic principles of swarm intelligence (Kennedy and Eberhart, 1995) summaries the aspects of Swarm Intelligence which make it ideal for modelling human behaviour. PSO is a type of Swarm Intelligence or collective behaviour, which is a simplified social milieu. Moreover, the original aim of the PSO algorithm was to simulate the flocking of birds, similar to the Boids technique (Reynolds, 1987).

PSO shares traits with other Swarm Intelligence algorithms and genetic algorithms through its fitness function, known as the global and local best position. Different types of PSO algorithm have been developed to satisfy different desired characteristics including; Nearest Neighbour Velocity Matching, The Cornfield Vector, Eliminating Ancillary Variables, Multidimensional Search, Acceleration by Distance and the Simplified Version (Kennedy and Eberhart, 1995).

The Ant Colony Optimisation (ACO) optimisation strategy is another form of Swarm Intelligence designed to optimise a set of data. This strategy is a type of local search optimisation which typically finds local maxima (Coppin, 2004). The simulation of pheromones dropped by ants searching for food is the metaheuristic. The more regularly the pheromones are renewed by any passing agents, the stronger they are, indicating the length of the path. Thus, ACO is good at solving the Travelling Salesmen Problem and any pathfinding problem.

Different ways of optimising a system for a computationally expensive simulation include; Multi-Threading, a performance optimisation technique which utilises multiple CPU cores; Spatial Partitioning, a type of Broad Sweep technique using binary space partitioning, is used to optimise collision detection by targeting specific areas of interested; Frustum Culling, the technique which speeds up rendering by only rendering objects which are visible to the camera (Luna, 2012); Tweaking Physics updates so objects are not updated as regularly; using the Compute Shader, a general purpose graphics processing unit (GPGPU) programming technique use for offloading calculations to the GPU and Instancing, which is the technique where a buffer uses the same set of triangles to render multiple meshes.

GPGPU programming, in particular, the use of the Compute Shader provides an excellent solution to common problems with computationally expensive simulations. There a few different ways a Compute Shader can be initialised for GPGPU programming. One initialisation technique is to utilise the *Append/ Consume Buffer*, another is to utilise a *Structured Buffer* (Luna, 2012).

The *Append/ Consume Buffer* selects a particle at random from the input data structure, then the shader applies the forces and finally removes it from the data structure. This technique works well for simulations with thousands of meshes as, in such a simulation, the order in which meshes are updated does not matter. The other type of Buffer is the *Structure Buffer*. This is

the standard Buffer which has no special features. Unlike the *Append/ Consume Buffer* the order in which objects are taken from the *Buffer* is linear.

A possible solution to enable the rendering one-hundred thousand meshes is Instancing. This technique renders multiple copies of the same mesh using a single buffer, by duplicating the vertex data across each instance. Many game engines have Instancing available including the rendering APIs DirectX11 and OpenGL.

Outcomes

The rendering API used for this application was DirectX11. This common rendering engine, used in Microsoft hardware, allows the user to custom-build a game engine. Using DirectX11 to develop a plugin or library (*.lib* or *.dll*) provides the opportunity to trim away much of the unnecessary libraries which come packaged in user-friendly game engines such as the Unity and Unreal Engine.

The application was designed with a decoupled architecture. This is apparent with the Physics and Swarm Manager systems which are independent of each other. This type of set up makes the systems easier to port, edit and debug without the systems affecting each other when changed.

To think about multiple areas of optimisation the decision for simplicity over complexity for the swarm algorithm was taken. The solution to this was to implement the simplified PSO algorithm. This algorithm was first implemented on the CPU and later ported to the Compute Shader on the GPU which, due to the simple nature of the algorithm, was not difficult. With a more complex algorithm, the syntactical differences between the CPU C++ code and the HLSL code can be greater. The simplified PSO algorithm mitigates this risk.

The GPU was used to calculate the global and local best position of each particle, the Physics and the Collisions. Being updated on the GPU makes the application more efficient by handing over some computation to the graphics card. The initialisation of the Compute Shader happens in the Swarm Manager which is not with the other buffer initialisations.

To render the one-hundred thousand objects Instancing was used. Once objects are created using an Instance Buffer, depending on the usage type they will be stored in different places in RAM. To notify the GPU that the buffer needs to do a memory transfer at runtime the *D3D11_USAGE_DYNAMIC* flag is applied. This means that the positions can be changed and updated with the *Map*/ *Unmap* functions.

Evaluation

The simulation was designed as a plugin, however, due to the low frame rate of the final implementation, integrating the simulation with another engine would have a negative impact. Despite this, the design of the plugins architecture had a positive effect on the development process. For example, the Swarm Manager was built with an internal data set, allowing testing and debugging to be conducted faster as no meshes needed to be rendered. Another positive outcome of the applications architecture is the potential of controlling how much information is sent between the Swarm Manager and the Model Manager in a single update. This means the system has the potential to

only update a fraction of the models in each update cycle.

Once the Physics calculations worked on the CPU they were ported to the Compute Shader on the GPU. This had a great impact on the performance of the application. However, an improvement on this system could be made. Transferring data between video memory and RAM more expensive than keeping all data on the GPU. If this system was developed further the output from the Compute Shader could be used as the input to the Vertex Shader, thus utilising the GPU shader pipeline by keeping all data in video memory.

When initialising the Compute Shader there is a choice between two types of Buffers the *Structured Buffer* and the *Append/ Consume Buffer* (Luna, 2012). The *Append/ Consume Buffer* has no knowledge of the order, as particles are taken at random from the Buffer. This is perfect for particle systems and simulations which use thousands of meshes, as skipping or appending a particle twice in an update cycle will not be noticed. This is another example of a technique which would be utilised if the application was developed further.

If done correctly Spatial Partitioning using binary space partitioning, k-d tree, grid and bounding volume hierarchy, is an effective Broad Sweep, collision optimisation technique (Nystorm, 2014). The implemented technique was designed to work in the same way as Spatial Partitioning but instead utilised the C++ standard library vectors. The aim of this technique was to work around the exponential growth problem, $\log^2(x)$ which was stopping the application from running. This failure led to the use of the GPU for collision detection.

The positive aspect of the implementing collision system is that it uses the Compute Shader. The setup for the Compute Shader is complicated due to a lack of experience. However, it's a powerful asset to understand.

This application does not run as smoothly as originally anticipated. With further development and research into the use of Multithreading and Spatial Partitioning, a more concrete solution to the low frame rate issue may be found. However, due to the decoupled architecture of the application, further development and implementation of new systems will be easy to integrate.

References

[1] Luna, F. (2012) *Introduction to 3D Game Programming with DirectX 11.* Herndon, VA, USA: Mercury Learning & Information.

[2] Coppin, B. (2004) *Artificial Intelligence Illuminated.* Sudbury, MA, USA: Jones and Bartlett.

[3] Ji, G., Wang, S., Zhang, Y. (2014) A Comprehensive Survey on Particle Swarm Optimization Algorithm and Its Applications. *Mathematical Problems in Engineering* [online]. 2015 [Accessed 02nd March 2018].

[4] Kennedy, J., Eberhart, R. (1995) Particle Swarm Optimization. *Neural Networks.* Perth, WA, Australia. *IEEE International Conference on*, pp. 1942-1948.

[5] Nystorm, R. (2014) *Game Programming Patterns.* Genever Benning.

[6] Reynolds, C. W. (1987) Flocks, Herds, and Schools: A Distributed Behavioral Model. *SIGGRAPH '87 Conference Proceedings.* Anaheim, CA, USA, July 1987. Computer Graphics, pp. 25-34