CORE
Metadata, citation and similar papers at core.ac.uk
Provided by University of Lincoln Institutional Repository

Improving Games AI Performance using Grouped Hierarchical Level of Detail

David Osborne and **Patrick Dickinson** [1]

Abstract. Computer games are increasingly making use of large environments; however, these are often only sparsely populated with autonomous agents. This is, in part, due to the computational cost of implementing behaviour functions for large numbers of agents. In this paper we present an optimisation based on *level of detail* which reduces the overhead of modelling group behaviours, and facilitates the population of an expansive game world.

We consider an environment which is inhabited by many distinct groups of agents. Each group itself comprises individual agents, which are organised using a hierarchical tree structure. Expanding and collapsing nodes within each tree allows the efficient dynamic abstraction of individuals, depending on their proximity to the player. Each branching level represents a different level of detail, and the system is designed to trade off computational performance against behavioural fidelity in a way which is both efficient and seamless to the player.

We have developed an implementation of this technique, and used it to evaluate the associated performance benefits. Our experiments indicate a significant potential reduction in processing time, with the update for the entire AI system taking less than 1% of the time required for the same number of agents without optimisation.

1 INTRODUCTION

Computer games AI is becoming increasingly complex, with players expecting increasingly realistic behaviour from non-player characters. At the same time, the size and detail of game environments has increased dramatically, with games such as Crysis [5] having effective view distances of over 8 kilometres. Often these environments are largely uninhabited, with the game agents centred around the player's location. If agents are present in other areas, they will sometimes use simplified behaviour models to reduce computational overheads. However, unless these are carefully constructed, problems can arise when an agent's simplified model is noticeably inconsistent with its full behaviour.

Increasingly sophisticated methods of performance optimisation have been investigated. For example, attempts have been made to dynamically adjust the amount of CPU time given to each agent [17]. Others have taken the concept of *level of detail* (LoD), a system used extensively in computer graphics [4, 16] to dynamically adjusts the complexity of visual representation, and transposed it to AI behaviour functions.

For example, work by Brockington [1] presents a LoD system which was used in the commercial game *Neverwinter Nights*. This made use of reduced behaviour complexity, based on 5 LoDs, with characters at the highest level using a complete set of combat and pathfinding algorithms. Those at lower LoDs had simplified versions of the same functions. For example, at the highest level, characters could use a full pathfinding system, whereas at the lowest level no pathfinding was used: agents were simply moved in a straight line to their destination with a time delay equivalent to the travel time. Whilst effective, Brockington's implementation is designed around one specific game environment: many of the pathfinding abstractions in his implementation rely on the structure of the levels within the Neverwinter Nights game engine.

Other methods have made use of hierarchical representations of behaviour models. Champandard's [3] Hierarchical State Machines (or HSMs) makes use of nested finite state machines. Each state of a machine could contain another machine or a set of child states: transitions from the parent state are effectively connected to all the sub-states, and transitions to the parent state can be redirected to the nested states. The key to HSMs is refinement and abstraction, with abstraction allowing for a parent state to be used as a regular state, rather than using the nested states or machines. Refinement allows for the states contained within the parent state to be used instead. This system applies LoDing to individual behaviours; however, designing and visualising the HSM is difficult as complex behaviour functions have to be decomposed into a set of simpler sub-states, while also providing appropriate behavioural elements at the parent level.

Brom *et al.* [2] take a comparable approach by applying hierarchical models to goal-based task representation. Non-player character behaviours are represented as tasks, which are further decomposed into atomic actions. Brom *et al.*'s proposed LoD system controls the detail with which individual actions are modelled and executed, and is coupled with an analogous LoD representation for the environments inhabited by those agents.

1.1 Level of Detail and Group Behaviours

Whilst the above methods have explored LoD type representations of the behaviour of individual game agents, our interest is the simulation of large numbers of agents acting in groups. Games have often made use of groups of agents working together, and one of the best known examples is Reynold's flocking algorithm [11]. Agents are controlled using a small number of steering behaviours that are applied individually, based on their neighbourhood, in order to create a believable emergent dynamic. Reynold's example of *boid* flocks, that is, flocks of artificial bird like creatures, made use of three very basic components (alignment, cohesion and separation). Later work [12] introduced other components for features such as path following and obstacle avoidance.

Other systems have been based on human behaviour. For example, van der Sterren presented two methods based on military com-

[1] University of Lincoln, Lincoln, email: david@digital-clouds.com, pdickinson@lincoln.ac.uk

bat. These compared emergent, decentralised command structures [13], where agents passed information to each other and acted on their local knowledge of the world, with centralised command [14] where one agent made decisions for an entire group. Both of these approaches provide good results: however, due to the high complexity of the behaviours, they are better suited to smaller sized groups.

Aside from military oriented models, a significant body of related work has developed around the simulation of human crowd behaviour. The objective of these systems is to create believable emergent group dynamics in everyday situations which are recognisable to the player or user. In most cases, such systems are comparable with Reynold's steering behaviours but employ considerably more complex behavioural models coupled with typically human environmental influences. Early work by Musse and Thalman [6] considered the use of emotional and sociological models of human behaviour for simulating crowd activities in simple contexts, such as visiting a museum. Later work by Sung *et al.*[15] focussed on situation dependent behaviour functions, and scalability, whilst Pelechano *et al.* [9] developed a model of interaction which generates interesting and realistic emergent behaviours in queues and crowded spaces.

Works by Brockington, Champandard, and Brom *et al.* which considered the optimisation of behaviour functions for individual agents has already been mentioned. Similar approaches to the optimisation of crowd simulations have also been proposed, such as that of Pettre *et al.* [10] which used reduced update times for non-visible agents, combined with simplified individual behaviour functions. However, systems comprising group-based agents offer the potential to further consider both interactions between agents, and representations of the entire group. Some existing research has approached LoDing from this perspective.

For example, Niederberger and Gross [7], presented a hierarchy of agents in a group, where each individual was assigned to a discrete LoD value. This determined both how each agent's behaviour was implemented as an individual, and also how computational resources were distributed through the group. For example, an agent could opt to pass its scheduled computation resources to another, superior, agent in the hierarchy.

Work by O'Hara [8] is of particular interest to us. This considered the representation of a group of agents engaging in Flocking behaviour (using Reynolds steering behaviours [11]), and sought to reduce the number of entities required to maintain a persistent model. Agents with stable trajectories and similar spatial characteristics are dynamically clustered into sub-groups called *stable groups*. These were then abstracted to a single entity which represents the stable group's dynamic and spatial properties, removing the need to update each member individually. However, as stable groups are defined by their member's dynamic characteristics, the group structure of the entire flock must be validated and maintained. For example, stable groups were merged, and others split, in response to interactions with other groups.

1.2 Our Approach

Like O'Hara [8], the aim of our work is to optimise the simulation of flocking agents through the abstraction of individuals. However, we approach this more from the perspective of player perception and experience (as is the case for LoDing in graphics). Moreover, we consider the problem of maintaining an entire environment rather than a single flock: whilst O'Hara applied his system to a single group of hundreds of agents, our experiments have simulated hundreds of thousands of agents, organised into separate and distinct groups occupying the same world. Our motivation is that this type of arrangement represents a likely structure for a large-scale game environment; for example, a sweeping plain populated by herds of different types of animals.

The novelty of our method derives from our approach to LoDing a *group* of agents. Existing approaches (like O'Hara's) use a bottom-up approach, and seek to reduce the representation required for a number of discrete agents acting in a group. Instead, we take a top-down approach in which we consider the group itself to be a single entity, and then add or remove agents as required to achieve an appropriate level of detail.

We have developed a hierarchical model to represent a group of agents engaged in flocking behaviour (using Reynolds steering behaviours [11]). Unlike O'Hara, who maintains adaptive clusters of agents, our abstraction is based on a predefined hierarchical tree structure. Leaf nodes in the tree represent the active group, and each branching node defines a LoD abstraction of it's subtree. LoDing is implemented by simply expanding or collapsing individual nodes: in our experiments we limit the depth of the tree, but this is an arbitrary restriction. Our structure also affords the potential to optimise memory usage by deallocating collapsed nodes until required. Whilst we arguably do not maintain the same level of persistence as O'Hara, we are able to retain consistent behaviours for very large numbers of agents, and support seamless transitions through different LoDs by incrementally adjusting the representation of each subtree.

We proceed by presenting our hierarchical group structure in Section 2. In Section 3 we describe our test implementation, where we apply our approach to agents controlled using Reynolds flocking method. Sections 4 and 5 presents the results of our evaluation, and discussion.

2 GROUPED LEVEL OF DETAIL

The main concept behind our approach is that agents of the same type are often instantiated and managed in a distinct group. For example, a single herd of animals may comprise a large number of agents who act together. Within this context, we use LoDing as a means of controlling the representation of the group as a whole (rather than the representation of individuals with the group). We refer to this idea as *grouped level of detail*.

LoDing applied to a 3D graphical models is implemented by varying the number of polygons used to render that model. This may be implemented procedurally, or by using a set of pre-constructued models with the same key geometrical features. Either way, the model itself it considered to be a single entity, and the polygons are the building blocks used to construct it: in principle any number of polygons could be used to construct a particular model.

Our approach to AI grouped level of detail is analogous to the graphical case. We consider that the agents themselves represent the building blocks from which the group is constructed, and vary the complexity of representation of the group by varying the number of agents used to model it. For example, if we want to reduce the group complexity, a set of agents is dynamically removed from the group and replaced by a single agent which retains their spatial and dynamic characteristic. When more detail is required again, these *abstracted* agents are expanded back into the original set.

2.1 Balancing Performance and Representation

Consider a simple case where a group has only two pre-defined LoDs. In one case, the group may be represented with a full and

complete number of agents: this provides the maximum representational complexity, but also requires the highest processing resources to maintain the group's behaviour. In the second LoD, the entire group is represented by just a single entity which abstracts all of its constituent agents. This representation has minimal processing requirements, but only a very simple representation. However these LoDs are managed, transitions between them are likely to present inconsistencies to the player. A key point of our approach is to mitigate this effect by varying the number of agents in the group incrementally.

In order to achieve this we have used a hierarchical tree structure to represent the group, with nodes representing agents at various levels of abstraction. Our tree has a nominal maximum depth, such that leaf nodes at this depth effectively correspond to individual agents in the group. Higher level nodes represent incremental abstractions of their subtrees, and can be independently collapsed or expanded again in response to a requirement for less or more detail. For example, using a proximity-based LoD, subtrees representing agents which are closer to the player are expanded to a greater depth than those which are further away.

An example is shown in figure 1, where a LoD *membrane* passes through a representation of the fully expanded tree to show its current actual expansion. The dark coloured leaf nodes which lie along the surface of the membrane represent the whole group at its current LoD.

Only leaf nodes run full behaviour functions, and they are also used to visualise the entire group during rendering. Nodes below the membrane represent higher LoDs which are not currently instantiated. For example, in figure 1 there are currently 4 leaf nodes. The far left hand side is expanded to its highest level of of detail, with 2 nodes at the lowest depth instantiated. These represent individual agents running full behaviour functions. The two adjacent agents have been abstracted to a lower LoD, and are modelled by a single node. The remaining four agents are at an even lower LoD, being represented by a single node on the far right-hand side. If more detail were required, this single node could be expanded back into either 2 or 4 nodes again.

2.2 Node Update and Management

The relationship between tree nodes and LoDs is expressed in the update procedure used for each node. Leaf nodes lying on the membrane surface receive a full update, using Reynold's flocking algorithm. These leaf nodes are the agents used to represent the group at its current LoD. Nodes below the membrane are currently abstracted out of the group, and effectively represent higher LoDs which may be utilised when required. These nodes are not updated.

Nodes above the membrane surface may be used to reduce the current LoD (by simply collapsing them as required). These receive a simplified update procedure which approximates the properties of their subtrees. In our implementation we simply update these nodes by averaging the dynamic attributes (position, velocity, etc.) of their children. This is effected using a recursive *bottom up* tree update procedure.

As mentioned, we use a player proximity test to determine when nodes should be expanded or collapsed. For example, when a node at a higher level of abstraction moves close enough to the player it is expanded into child nodes (where the number is determined by the branching factor), up to the maximum depth of the tree. Conversely, when the two leaf nodes moves far enough from the player to warrant an abstraction to a lower LoD, they are removed and their parent becomes a new leaf.

We wish to maintain consistency in abstraction, as far as possible, and this raises the issue of how child nodes are best re-initialised when their parent is expanded. We have experimented with two methods: firstly, attributes of abstracted nodes are retained relative to their parent, and used for re-initialisation. This method is preferable in terms of persistence, especially where nodes are quickly expanded and collapsed, or smaller groups are used. However, it does also require that full storage for nodes is maintained down to the lowest depth.

Alternatively, nodes can be re-initialised randomly (within reasonable limits) relative to their parents. We have used this second method in our experiments as it allows for nodes to be deallocated when not in use, and thus utilises memory resources more efficiently. Note that the actual number of individual agents represented by a hierarchical group is given by:

$$Num.\ Agents = BranchingFactor^{Max.\ Depth} \tag{1}$$

The depth and branching factor directly affects how many nodes will be given a full update at each cycle. The optimal structure thus depends on several factors, such as the number of agents per group, and how the LoD system is controlled (e.g. whether or not it is based on player proximity).

3 IMPLEMENTATION

Our test implementation was developed in C++, and uses OpenGL to provide a simple graphical representation of the 2-dimensional game world. For each node undergoing a full update procedure we use the three base flocking behaviour components suggested by Reynolds [11], as well as an additional *wandering* behaviour [12]. The random wandering component is important as the three base behaviours utilise relationships between neighbouring nodes in the same group. At the lowest LoD (where the root node represents an entire group) these components are effectively abstracted out of existence. However, the wander component is an underlying behaviour which is retained at all levels.

Reynolds uses a physical model, where behaviours are modelled by applying forces to an agent. To implement this model, we model each node as a point mass with a fixed value of $m_a = 50Kg$. In addition, each node has persistent position and velocity attributes $\mathbf{p}_a$ and $\mathbf{v}_a$ respectively. Each of the four steering components is implemented as a physical force, which is calculated and applied to each leaf node at each update cycle. These are described as follows.

Alignment applies a force to change the heading of the node towards the average heading of its neighbours. The force is applied along the direction of the average heading, which is calculated as:

$$Average\ Heading = \frac{1}{n}\sum_{i=1}^{n}\frac{\mathbf{v}_i}{\|\mathbf{v}_i\|} \tag{2}$$

where n is the number of other nodes within its immediate neighbourhood. This neighbourhood is defined by a circle of radius N around the node. The velocity $\mathbf{v}_i$ is the velocity of node i in the neighbourhood. The heading represents a normalised target velocity direction.

Cohesion applies a force to steer a node towards the centre of its neighbours. This is implemented by first calculating the centre point, $\mathbf{C}$, of the neighbourhood:

$$\mathbf{C} = \frac{1}{n}\sum_{i=1}^{n}\mathbf{p}_i \tag{3}$$

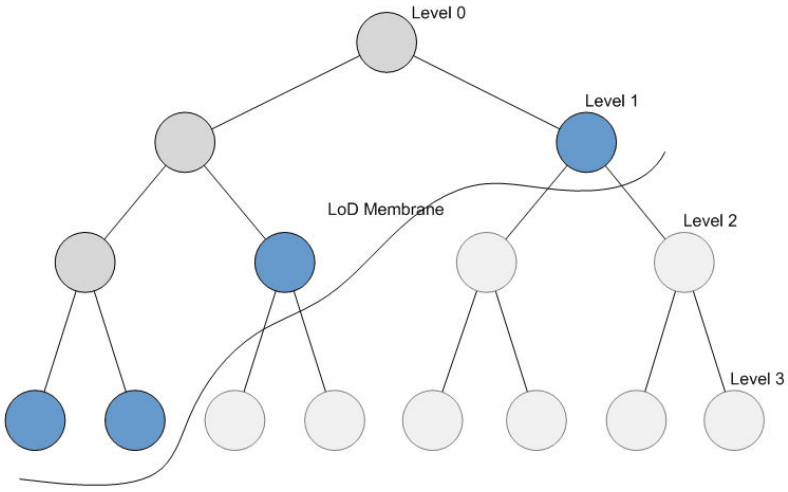

Figure 1. Hierarchical representation of a group of agents

where $\mathbf{p}_i$ is the position of node i. Once $\mathbf{C}$ has been calculated, a *seek* function is used to steer the node towards $\mathbf{C}$. This applies a force given by the following:

$$Seek\ Force = \frac{\mathbf{C} - \mathbf{p}_a}{\|\mathbf{C} - \mathbf{p}_a\|} v_{max} - \mathbf{v}_a \tag{4}$$

where v_{max} is the maximum speed.

Separation has an opposing effect, steering nodes away from very close neighbours. This force is given by the following:

$$Separation\ Force = \sum_{i=1}^{n} \frac{\mathbf{p}_a - \mathbf{p}_i}{\|\mathbf{p}_a - \mathbf{p}_i\|^2} \tag{5}$$

where $\mathbf{p}_i$ is the position of neighbour node i.

Wander is the final behaviour used, and provides a pseudo-random component. A force is applied to steer each node to a point $\mathbf{T}$ offset randomly along the perimeter of a circle of radius $\mathbf{w}_r$ projected a distance $\mathbf{w}_s$ ahead of the node's current position and heading (we used values of $\mathbf{w}_r = \mathbf{w}_s = 3$). The force applied is then given by:

$$Wander\ Force = \mathbf{T} - \mathbf{p}_a \tag{6}$$

Maximum force and velocity values were also used in order to prevent nodes moving too rapidly. These were set to 500 and 20 respectively. The full update function for a node is thus performed by calculating the steering forces, summing them, and applying them using the following dynamic update equations:

$$\mathbf{s}_a = \mathbf{v}_a t + \frac{1}{2} \frac{TotalForce}{m_a} t^2 \tag{7}$$

$$\mathbf{v}_a = \mathbf{v}_a + \frac{TotalForce}{m_a} t \tag{8}$$

where $\mathbf{s}_a$ is the change in position, and t the time since the last update. The magnitude of $\mathbf{s}_a$ is limited to a maximum based on the time since the last update, and $\mathbf{v}_a$ is initially set to the velocity calculated at the end of the previous update. This full 4-component update model is applied to each leaf node (along the membrane surface in figure 1). Nodes above this level are updated by simply computing the average position and velocity values of their children.

We also made use of *prioritised dithering*, a method presented by Reynolds which uses probabilities to determine whether a particular force is applied on a particular update. Each force component is thus assigned an associated probability, and also a weighting factor, which can be used to fine-tune the behaviour functions. The values we used for these two additional parameters are given in Table 1.

Parameter	Value
Alignment Probability	0.4
Alignment Weight	20
Cohesion Probability	0.2
Cohesion Weight	50
Separation Probability	0.5
Separation Weight	2
Wander Probability	0.3
Wander Weight	20

Table 1. Prioritised Dithering Parameters

As mentioned, Reynold's flocking behaviour is used to update the current leaf nodes (that is, nodes lying along the expanded tree membrane). Nodes at higher levels of detail (below the membrane) receive no update as they are not instantiated. Nodes above the membrane (at lower LoDs) receive a reduced update procedure where their position and velocity attributes are simply calcuated by averaging those of their children.

3.1 Controlling the Level of Detail

The LoD is controlled using a proximity based test, where the distance between the player and each node is used to adjust the corresponding level during each game update. We also experimented with a simpler implementation where we used only the distance between the player and the group root node. However, we found that managing each node separately gave a superior representation, with

partial expansions allowing smoother transitions between different group LoDs.

We set distances (from the player) which define the LoD for each subtree. Each of these defines a radial LoD zone around the player, and there is necessarily one zone for each possible depth level in the tree. Thus, if a leaf node moves into a zone which represents a higher LoD, then the node is expanded into child leaf nodes. As mentioned, these nodes are initialised randomly: in our implementation we used a maximum offset radius of 10M, and allowed the magnitude of the randomised velocity to be anything below the maximum defined for an agent.

Conversely, child nodes need to be collapsed to their parent if they move far enough away from the player. We detect this condition by examining the position of the parent of each leaf-pair. If, after the parent has been updated, it lies in the zone corresponding to its own LoD then the children are removed from the tree and the parent becomes a new leaf.

3.2 Running the Simulation

A software timer was used to call the AI update function every 500 milliseconds. This interval was chosen as a minimum which provided smooth behaviour for large numbers of agents. We also used a separate software timer to measure the amount of time actually spent on each AI update cycle: we used this to quantify our evaluation, presented in the section 4. Our implementation also used a single player entity, which could be moved using keyboard input, and we rendered a top down view of the 2-dimensional world inhabited by the agent groups. Note that other than the player and the agents, no other objects existed in the game environment. Figure 2 shows a visualisation of one rendered update frame from our LoD system whilst running.

4 EVALUATION

In order to test the performance of our system we undertook a series of experiments. These were executed on PC running the Windows 7 (32-bit) operating system, with a dual core 2.53GHz processor and 4GB of memory. The neighbourhood distance, N, used by the flocking algorithm was set to 20M, and the distance between successive LoD zone thresholds (from the player) was set at 50M. This means that, for example, groups whose nodes were all closer than 50M would be represented at their highest detail level by a fully expanded tree.

An experiment with a large number of agents and groups was used to compare the performance of our grouped LoD system with an analogous system that did not make use of optimisations. The LoDed system used a tree with a maximum depth of 5, and a branching factor of 2. This provided a simulation of 32 agents per group, and the test was run with 2048 groups giving a total of 65,536 agents. The experiment was run for 500 update cycles, and then the mean update times were taken for each test. Figure 3 shows the comparative results. As can be seen, there is a very significant reduction in update time when grouped hierarchical level of detail is used.

We ran a second experiment to determine how many agents could be reasonably represented while keeping the AI update time below 100 milliseconds. A similar set up to the previous test was used, with a maximum depth of 5 and a branching factor of 2 giving a group size of 32 agents. We started with 2048 groups in the simulation, and repeatedly doubled the number of groups up to 65,536 (2,097,152 agents). However, beyond 32,768 groups (1,048,576 agents) the program became unstable due to lack of memory. The results of this test are shown in figure 4.

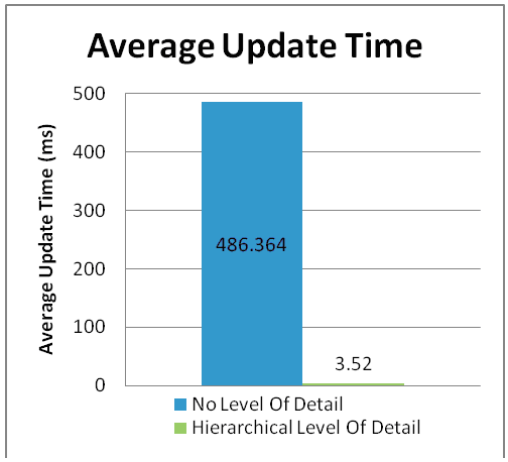

Figure 3. Average update time comparison.

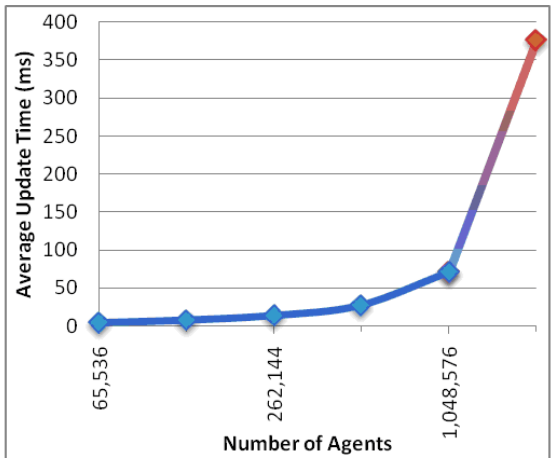

Figure 4. Update times for varying numbers of agents.

Whilst very large numbers of agents could be represented effectively, it is worth emphasising the point that system memory is a limiting factor. In practical applications we expect that the actual number of agents would be significantly lower.

5 CONCLUSION

We have proposed a group level of detail system, with the objective of achieving performance optimisations in systems simulating large numbers of agents organised into groups. Our implementation has integrated our approach with Reynolds' force-based steering behaviours.

Our experiments suggest that large performance gains can be achieved while maintaining a relatively seamless representation for the player. However, it should be noted that while the number of

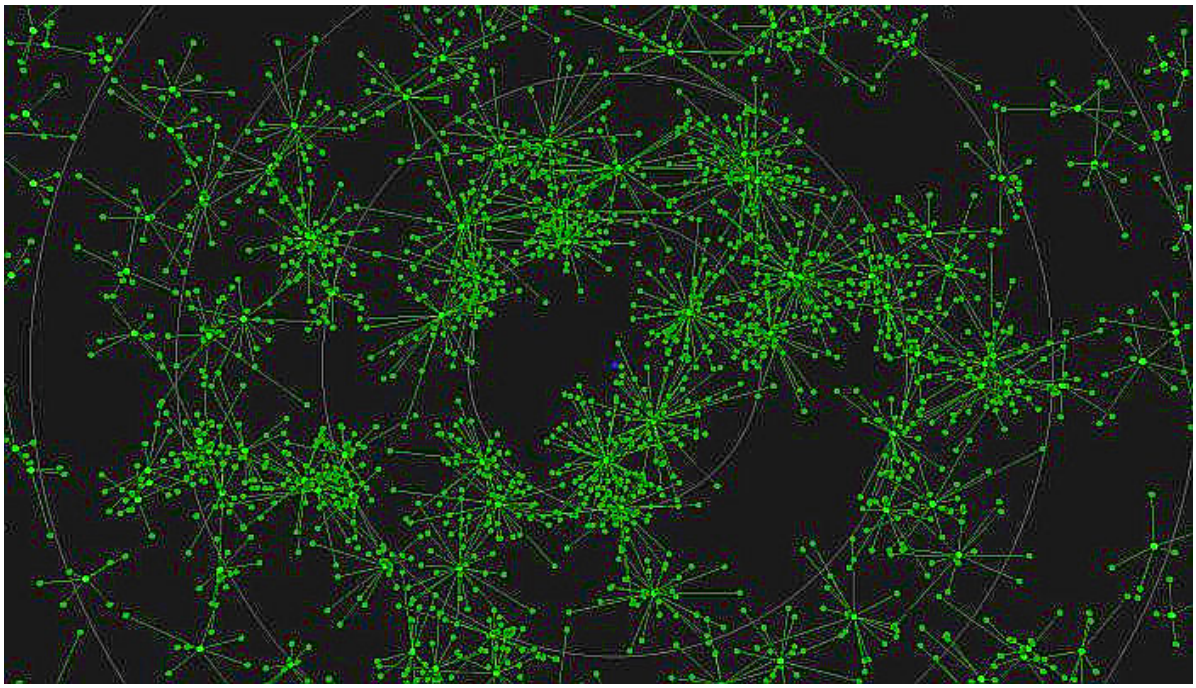

Figure 2. Visualisation of our implementation running. The central point of each cluster represents the root node of a group (its mean position). Each connected point is an instantiated child node running a full update.

agents simulated in our experiments is very high, our agents are using only simple steering behaviours, with minimal additional processing. Nevertheless, most games requiring advanced behaviours will not require anywhere near as many agents as we have used, and additional performance optimisation techniques could also be utilised, such as reducing the amount of update time scheduled for lower LoDs. One aspect of our work which we have not considered is that of the interaction between different groups: for example, how interactions between nodes at different LoDs might be effected. We leave this issue as future work.

REFERENCES

[1] M. Brockington, "Level-Of-Detail AI for a Large Role-Playing Game," *AI Game Programming Wisdom*, Charles River Media, pp. 419-425, 2002.

[2] C. Brom, O Sery and T. Poch, "Simulation Level of Detail for Virtual Humans," Proceedings of the International Conference on Intelligent Virtual Agents, pp. 1-14, Paris, 2005.

[3] A. Champandard, "Hierarchical State Machines," AI Game Development: Synthetic Creatures with Learning and Reactive Behaviours, New Riders Publishing, pp. 557-571, 2003.

[4] J. H. Clark, "Hierarchical Geometric Models for Visible Surface Algorithms," *Communications of the ACM,* 19(10), pp. 547-544, 1976.

[5] Crytek, "CryENGINE 2 - Specifications," [Online] 2007. [Cited: 19 December 2009.] http://www.crytek.com/technology/cryengine-2/specifications.

[6] S. Musse and D. Thalmann, "A Model of Human Crowd Behavior: Group Inter-Relationship and Collision Detection Analysis," Workshop of Computer Animation and Simulation (Eurographics97), Budapest, 1997.

[7] C. Niederberger and M. Gross, "Level-of-Detail for Cognitive real-time Characters," The Visual Computer: International Journal of Computer Graphics, 21(3), pp. 188-202, 2005.

[8] N. O'Hara, "Hierarchical Imposters for the Flocking Algorithm," Computer Graphics Forum, 21(4), pp. 723-731, 2003.

[9] N. Pelechano, J. Allbeck and N. Badler, "Controlling individual agents in high-density crowd simulation," ACM SIGGRAPH/Eurographics symposium on Computer animation, pp. 99108, , 2007.

[10] J. Pettre, P. de Heras Ciechomski, J. Mam, B. Yersin, J.-P. Laumond and D. Thalmann, "Real-time navigating crowds: Scalable simulation and rendering," Computer Animation and Virtual Worlds, 17(3) pp. 445455, 2006.

[11] C. W. Reynolds, "Flocks, Herds and Schools: A Distributed Behavioural Model," Computer Graphics, 21(4), pp. 25-34, 1987.

[12] C. W. Reynolds, "Steering Behaviours for Autonomous Characters," Games Developers Conference, San Jose, CA, pp. 763-782, 1999.

[13] W. van der Sterren, "Squad Tactics: Team AI and Emergent Maneuvers," AI Game Programming Wisdom, Charles River Media, pp. 233-246, 2002.

[14] W. van der Sterren, "Squad Tactics: Planned Maneuvers," AI Game Programming Wisdom, Charles River Media, pp. 247-259, 2002.

[15] M. Sung, M. Gleicher and Stephen Chenney, "Scalable Behaviors for Crowd Simulation," EUROGRAPHICS 23(3), 2004.

[16] L. Williams, "Pyramidal Parametrics," Computer Graphics, (Proc. Siggraph 83), 17(3), pp. 1-11, 1983.

[17] I. Wright and J. Marshall, "Egocentric AI Processing for Computer Entertainment: A real-time Process Manager for Games," in Proceeding of International Conference on Intelligent Games and Simulation, pages 42-46, 2000.