A Multi-Layered Flocking System for Crowd Simulation

Simon van den Hurk
University of Auckland
Computer Science Department
Science Faculty
+64 21 044-5621
simon.vandenhurk+cgat@gmail.com

Ian Watson
University of Auckland
Computer Science Department
Science Faculty
+64 9 373-7599 ext. 88976
ian@cs.auckland.ac.nz

ABSTRACT
The field of crowd simulation attempts to model crowd movement of both people and animals. Typical research in this field aims to develop systems which model the interaction between multiple instances of the same type of character. This paper examines two aspects of crowd simulation which are often not considered, the movement of crowds containing characters of vastly different sizes and the ability to allow characters to move underneath other characters when there is sufficient space to do so. To include these traits in a crowd simulation model a new system is proposed: the multi-layered flocking system. This system has a basis in the original Reynolds flocking model but further divides the simulation space using a series of layers. Characters in the simulation are represented using one or more navigation objects which lie upon the layers in the system. These navigation objects represent parts of the character as it moves throughout the simulation and can be either dynamic or static. Different combinations of navigation objects allow for the representation of characters of varied shapes and sizes as well as different movement styles, all of which are able to navigate using the same system. By creating a crowd which contains different character representations a more interesting overall motion can be obtained.

Keywords
Crowd Simulation, Multi-Agent Simulation, Flocking, Herd, Boid, Motion.

1. INTRODUCTION
The problem of simulating the movement of large numbers of characters in an environment is relevant to both real time and rendering applications being created today. These crowds of characters are used to breathe life into the large and often varied scenes created for a variety of mediums such as games, movies and television.

In this paper two aspects of crowd simulation are examined that are often ignored and a system is proposed that further enables designers to create more complex crowd interaction by removing these limitations.

The first aspect to be examined is the interaction between the characters of a crowd when the size of the characters differs significantly. The majority of techniques described in the field of crowd simulation concern the interaction between multiple instances of the same type of character. They therefore do not consider the effect that character size will have upon the overall movement of the crowd.

The second aspect that is to be considered is the increased level of crowd interaction that can be created when characters in the crowd are able to navigate not only around other characters, but also underneath other characters when there is sufficient space.

To allow these two aspects of crowd interaction to influence the navigation of the characters in a crowd a new system is proposed: the multi-layered flocking system. This system represents the characters and environment of a simulation by placing them upon a series of layered cells. This approach is inspired by the representation given in the original flocking paper by Reynolds[9].

The objectives of this paper therefore are:

- To define a navigation system that takes into account the size difference of characters within a simulation.
- To allow the characters in this navigation system to navigate not only around but also underneath other characters within a simulation.
- To examine the different representations and behaviours that can be used for both characters and the environment within this navigation system.

The next section provides an overview of the other literature in the field of crowd simulation. In the following section the design of the multi-layered flocking system is explained. The next section discusses the use of the multi-layered flocking system and describes its capabilities. The final section contains a conclusion for this paper.

2. LITERATURE REVIEW
The popularity of video games as an entertainment form has led to increased realism in successive titles. This realism includes the simulation of crowds to represent different aspects of the game. Recent games such as *Supreme Commander*[1] have included thousands of characters as military units while other games such as *Assassins Creed*[16] have used smaller quantities of characters moving in groups in an attempt to create a more lifelike representation of a crowded market.

2.1 Agents
The most widely known and popularized technique for large scale crowd movement is the flocking concept proposed by Reynolds[9]. Reynolds' paper presented a framework to simulate

CGAT Conference 2010, April 6–7, 2010, Singapore.

the flocking behaviour of animals. The examples provided by Reynolds involved bird-like flocking agents (referred to as boids) that moved through a 3-dimensional environment. The framework is also applicable to two dimensions (as in the flocking movement of sheep).

In Reynolds framework each agent is expressed using a simple point mass model, the direction of which is calculated by using a combination of different steering behaviours.

Reynolds defines three different types of steering behaviours in order to simulate the flocking motion. These behaviours are initially called *Collision Avoidance*, *Velocity Matching* and *Flock Centring* though in Reynolds later works he refers to these same behaviours under the names *Separation*, *Alignment* and *Cohesion* accordingly[10]. Each steering behaviour returns a force representing the desired acceleration of the agent in order to best fulfil the behaviour. The three behaviours are defined as following:

- *Separation* - Separation provides a steering force such that the movement of the agent avoids colliding with other flocking agents.
- *Alignment* - Alignment is the steering behaviour that causes entities to attempt to match their directional heading with those of its neighbours.
- *Cohesion* - Cohesion provides a steering force towards the average position of the neighbours in order to cause the boids to herd together.

The final steering force that will be applied to the Reynolds agent is then calculated. Reynolds provides two different combination methods and suggests a third method in his later paper[10]. These methods combine the forces provided by the different behaviours in an attempt to best produce a result that is desirable to all of the behaviours.

Each of the steering behaviours uses information about the surrounding entities. These surrounding entities are known as an agent's neighbours. The entities that are chosen as an agent's neighbours are dependent on the agents perception function. The simplest perception function being a perception radius describing a sphere centred on the agent.

Due to the fact that agents only require a local knowledge Reynolds suggests sorting the boids into an arrangement of bins. Each agent lies within a bin and its bin is updated as it moves throughout the simulation. This sorting of the agents allows the computational complexity of the neighbour comparison to be reduced from $O(n^2)$ to $O(9n)$.

Reynolds notes that this technique has the advantage of being deterministic but that it can be difficult to create a combination of forces that produces the desired behaviour.

Further work by Reynolds presents further steering behaviours that can be used within his original framework[10]. By combining these behaviours it is possible to create groups of agents that act as a flock, but also easily integrate with other behaviours that could be used when the agent's state is changed. Examples of these extra behaviours include: *Flee*, *Pursuit*, *Wander* and *Leader Following*, as well as behaviours to provide obstacle avoidance and specific scripted actions.

Lebar Bajec et al. also revisits the concept of using the boids model to simulate crowds and provides a deeper algorithmic analysis of the model[3].

As the agents in Reynolds framework only require knowledge of their local neighbours the flocking technique is easily scalable on multi-core hardware architecture as demonstrated by Reynolds[8]. Work by Pettré shows that in the long term a group of Reynolds agents will tend to form into a single large flock[7]. Reynolds implementation on the Playstation® 3 includes an additional steering behaviour to disperse crowds of increased density in order to prevent a single flock from forming and to reduce the loads inside a single bin.

Treuille proposes an alternative model for crowd movement that makes use of dynamic potential fields instead of individual agent perception[15]. This framework combines the search for a global path with the search for local avoidance into a single calculation. This approach produces a crowd movement that naturally forms lanes of agents moving behind each other. Work by Berg et al. also describes a technique that creates a similar lane forming style crowd[17]. The lack of individual agent goals in Treuille's approach is addressed by Sud et al. at the cost of increased computational requirements[14].

Other related work includes a suggested method by Scutt that can create simple swarms with minimal computational complexity by not guaranteeing the separation of the entities in the swarm[12].

2.2 Environment Representation

The concepts mentioned so far deal primarily with the steering behaviour of agents, rather than the representation of the environment in which these agents move.

Alternate methods of crowd simulation that have been proposed discuss the use of Voronoi diagrams in order to divide up the space on which the crowd is to traverse. Work by Pettré involved creating Voronoi diagrams around the static obstacles in the environment and then mapping a connected graph of cylinders onto this space[7]. Kamphuis et al. presents a similar approach in which the corridors of movement are created by a leading entities initial path finding, thus restricting group members to group together to form crowd movement[2].

The more recent work by Pettré et al. continues to refine the use of navigation graphs and cylinder movement spaces[6]. This work includes the concept of *Levels of Simulation* in order to simulate a crowd of 35,000 pedestrians. This involves updating agents that are closer to the camera more often in order to reduce the computational load while maintaining the apparent detail of the crowd. Maïm et al. also discusses the use of *Levels of Simulation* in order to produce a large scale crowd simulation that runs at a desirable real-time frame rate[4]. Richmond et al. suggests a level of detail implementation that utilizes a computer's GPU in order to achieve an increase number of agents within the crowd[11].

Work by Nieuwenhuisen et al. uses a combination of navigation graphs and bounding cylinders to keep groups of agents together when moving past obstacles in the environment to provide a more cohesive overall movement[5].

Silver also proposes an alternative method for path finding that is designed to cause entities in the crowd to choose paths which

cooperate with the other agents around them[13]. Silver achieves this by dividing the environment into a grid, within which each entity reserves places for future time steps.

3. SYSTEM DESIGN

3.1 Introduction

In this section a formal definition for the algorithm and data structures that compose the multi-layered flocking system is given. There are four main components to the system and each is examined in turn. These four components are the navigation agents, the layered navigation cells, the neighbourhood selection and the navigation behaviours. Each component is an extension of a two dimensional implementation of Reynolds algorithm.

3.2 Navigation Agents

Reynolds model uses the term boid to describe the data that encompasses a characters position and movement. Each boid represents a single animal such as a bird or fish that is to be simulated. Navigation behaviours provide forces to act upon each boid and the resulting forces for this boid describe its movement. Each boid is comprised of a series of attributes in order to simulate the locomotion of the boid around the simulation space. The attributes that a boid contains are:

- *Position*: A vector representing the boids centre.
- *Heading*: A vector representing the boids direction of movement. The magnitude of this vector represents the boids current speed.
- *Mass*: A scalar value representing the mass of the boid.
- *Steering Force*: A vector representing the steering force that is to be applied to the boid.
- *Perception Radius*: A scalar value representing the radius of the circle that is used to detect neighbouring boids.
- *Navigation Behaviours*: A list of all the behaviours that are to act upon this boid.
- *Cell*: A reference to the cell that this boid currently resides in.

The simulation for each boid is performed using two update methods. The first is the physical update which calculates an acceleration vector for the boid given the boids steering force and mass. In a simple simulation this is typically done using Newton's second law of motion which is that the force is equal to the mass times the acceleration*(□=□ □)*. Using the calculated acceleration value a new heading and position for the boid can be obtained. This update is equivalent to a standard world update in a typical rendered animation or game simulation and can be performed with either a fixed or varied time-step.

The second update is the behavioural update. During this update each boid recalculates its neighbours and determines the forces that should act upon it for each of its navigational behaviours. Note that these two updates do not have to be simultaneous. A lower frequency update can be used for the behavioural update to match the needs of the simulation.

The multi-layered flocking system abstracts this concept to provide a system that is able to represent more complex character navigation. In the place of boids each character in the multi-layered system is represented by one or more navigation objects. Navigation objects are one of two types: navigation agents and navigation obstacles. The navigation agents act in a similar fashion to a Reynolds boid. Each navigation agent is located within a single navigation cell in the same way that each boid was located within a cell. Each navigation agent also contains the same attributes as a boid, with the addition of an extra scalar value representing the bounding radius. This bounding radius is used to determine the size of an object and will be used in the interaction between objects of different sizes.

A navigation obstacle will be used to represent objects in the simulation that are either stationary or whose position is dependent on another navigation agent. For example a rock that is to be avoided would be classified as a navigation obstacle. A walking human could be represented by one navigation agent to describe its overall movement and two navigation obstacles for each of the legs. Navigation obstacles do not require all of the attributes that a navigation agent is composed of and instead only requires a position, reference to a cell and the scalar value representing the bounding radius. Using these attributes a navigation obstacle is able to contribute to the navigation behaviours of other objects, such as collision avoidance, even though the navigation obstacle itself does not have any navigational behaviours.

3.3 Layered Navigation Cells

In Reynolds approach a single grid is placed over the simulation space with each boid in the simulation being associated with a single cell.

In the multi-layered system cells are also used to divide the simulation up, though unlike in Reynolds approach, the use of cells is compulsory and is used for more than just increased efficiency. Characters in the scene are represented in the simulation by one or more navigation objects. Each of these navigation objects lies within a particular navigation cell and each cell is associated with a single layer. A navigation layer is defined as a collection of cells that cover the entire simulation space. Navigation layers may contain a reference to a parent navigation layer. The navigation objects that lie within the cells of this parent layer, and any ancestor layer above that, will be used to influence the movement of the agents in the child layer.

If a layer has a parent layer then all cells within that layer will have a parent cell within the parent layer. In order for a layer to be a valid parent for another layer, the cells within the parent layer must meet certain restrictions. The entire simulation space covered by the parent cell must be completely covered by its child cells. Additionally the area that each child cell covers must lie within only a single parent. In order to meet this restriction it is often necessary for a cell within a layer to be a slightly different size to the majority of cells in that layer. This is to ensure that the entire simulation space is covered without breaking one of the two restrictions above. If the simulation space is made to wrap around the x or y axis then the cell that is of a different size must be larger than the standard grid cell size. If this is not true then the agents within the simulation may not detect neighbours correctly due to having a perception radius that spans across more than one neighbouring cell. In Figure 1 an example is demonstrated in which a parent layer requires a slightly larger row of cells.

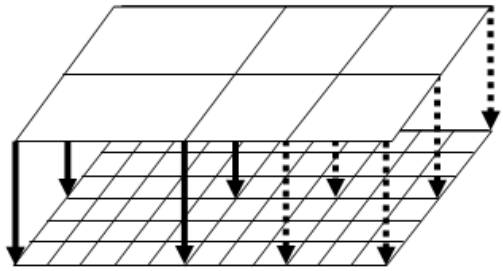

Figure 1. A demonstration of the mapping between the edges of parent and child cells upon two cell layers. The top layer has been created with three child cells per parent cell along both the x- and y-axis. The parent cells with dashed arrows map exactly to the layer below, while the parent cells with solid arrows are required to be slightly larger in order to ensure that all child cells are included within a parent cell.

The restrictions on parent size ensure that the layers are built up from the highest detail at the bottom to the lowest detail at the top. Thus the layers at the bottom will contain a larger number of smaller sized cells, while the layers at the top will contain a smaller number of larger sized cells.

The shape of the cells can be implementation specific, though a rectangular shape is most likely due to the efficient detection of a point within an axis aligned grid.

3.4 Neighbourhood Selection

In Reynolds boids framework the neighbours of a boid determine the influencing behavioural forces. The neighbours of a boid are determined by finding the nearby boids within the simulation and then testing these boids against a perception function to determine if they will influence the boid in question.

The neighbourhood selection algorithm returns all the boids within the selected boids cell and all those boids within the neighbouring cells. In a common 2D grid the neighbouring cells would be those cells that are above, below, right, left and diagonal to the current cell, giving a total of eight cells. Then each of the potential neighbours that lie within these cells is checked to see whether it lies within the perception radius of the current boid. In this implementation the boid considers a neighbour to be within its perception if the neighbouring boid's position is within the circle defined by the perception radius.

The approach must be modified for the multi-layered system in order to account for the neighbouring navigation objects that are of a different size. Navigation objects that are of a large size may be within the perception range of another navigation agent without the centre of the navigation object being within the perception radius i.e. only the edge of the large navigation object lies within the perception circle. Without taking this into account there could be missed interactions between navigation objects resulting in unwanted collisions or unusual navigation behaviour.

In order to avoid this problem the navigation agents in the simulation are given an additional scalar value, the bounding radius. The original radius, the perception radius, determines the distance at which an agent perceives its neighbours. The second, the bounding radius, defines a circle describing the bounds of the agent itself. The perception function for the agents is then changed such that an agent perceives a potential neighbour if any part of the bounding circle of the neighbour lies within the perception radius. Therefore a potential neighbour is considered to be within the perception distance if the distance from the centre of the agent to the neighbour agent is less than the sum of the agent's perception radius and the neighbours bounding radius.

In the multi-layered flocking system the neighbourhood definition is extended to include some of those navigation objects that lie within the cell layers above the current agent's layer. Since the navigation objects in the layers above are within cells that are of equal or greater size there is potential that the navigation objects that are to be included have a larger bounding radius. For this reason the neighbourhood is modified to include the surrounding cells of each of the ancestors of the cell that the current agent lies within. Thus the neighbourhood of an agent is defined as the cell of this agent, plus all of its ancestors and the cells surrounding the current cell and each ancestor cell. A formal listing of the algorithm that finds all navigation objects within an agent's neighbourhood is given in listing Algorithm 1: Neighbourhood Selection.

Input: NavigationCell *cell*, NavigationAgent *agent*, Boolean *includeNeighbours*, Boolean *includeParents*
Output: List of NavigationObjects *neighbours* within *cell*, excluding *agent*, including neighbouring cells if *includeNeighbours* is true, including parent cells if *includeParents* is true

```
neighbours ← Ø;
foreach NavigationObject navObj in cell do
    if navObj ≠ agent then
        if DistanceTo(agent,navObj) <
(agent.perceptionRadius + navObj.boundingRadius) then
            Add navObj to neighbours;
        end
    end
end
if includeNeighbours then
    foreach NavigationCell nCell in neighbours
        neighbourObjs ← NeighbourSelect(nCell,agent,
                        false,false);
        Add each NavigationObject in neighbourObjs to
neighbours;
    end
end
if includeParents then
    parentObjs ← NeighbourSelect(cell.parent,agent,
                 includeNeighbours,true);
    Add each NavigationObject in parentObjs to neighbours;
end
```

Algorithm1. Neighbourhood Selection

All of the objects within these cells are compared with the perception function of the current agent to determine if they are valid neighbours, and then the navigation behaviours use these neighbours to produce their recommended steering forces. Figure 2 provides a visual example of the neighbourhood selection algorithm being run.

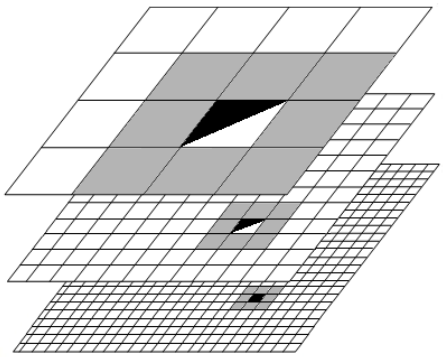

Figure 2. An isometric view of the Neighbourhood Selection algorithm being run on a three layered navigation system. The original cell that the agent is positioned in is shown in black. Ancestor cells are shown with a black triangle and the neighbouring cells are shown in grey. The remaining cells in white are not examined.

3.5 Navigation Behaviours

Reynolds specifies a number of different navigational behaviours that allow for character movement, of which *Separation*, *Alignment* and *Cohesion* are used to create flocking[10]. The forces provided by these different behaviours can be combined in different ways to achieve a final movement direction. One method is to assign weights to each behaviour, with higher weights being assigned to more important behaviours such as obstacle avoidance. A second method called prioritized acceleration allocation can also be used. This technique attempts to allocate the total amount of acceleration that the boid can provide by considering the steering behaviours in order of their priority. A third method is suggested where one behaviour can be evaluated randomly for a given probability for each update of the simulation, thereby spreading the computational load of the behaviours across multiple updates of the simulation.

For the multi-layered flocking system the same approach can be used to determine the final force. The different navigational behaviours can all be applied to the simulation, though modifications may need to be made to take into account the influence of the layers of navigation objects. For example the separation behaviour is normally calculated as the sum of the difference between the current agent in question and its neighbours. This sum is then scaled by the inverse of the separation distance ($1/d$). In a multi-layered system the difference between the current agent and the neighbours must take into account the bounding radius of the objects and can additionally be multiplied by a factor representing the difference in layer height. This additional weight can then make agents prefer to avoid collisions with agents in layers above, which in turn may create a more natural steering behaviour for the specific simulation.

3.6 Cell and Agent Property Relations

The closely tied relationship between cells and the navigation objects within them place certain restrictions and dependencies upon their attributes. Firstly the perception radius of the agents in a particular layer must always be less than both the width and the height of the cells in their layer. This ensures that agents on the border of a cell cannot perceive objects that lie further than one cell away (as these cells are not included in the neighbourhood selection algorithm).

The ratio between the bounding radius and the perception radius must also be considered. An agent requires that the difference between these two distances be sufficiently large such that two agents moving directly towards each other do not intersect after a single behavioural update. Since these attributes depend entirely on the simulation being created, e.g. the maximum speed of the agents in each layer, they are implementation specific.

4. SYSTEM CAPABILITIES

4.1 Introduction

This section examines the different type of movement that are possible when using the multi-layered flocking system. It describes the possible representations for agents and obstacles, the use of properties and the behaviours which can be applied to agents to provide different types of movement.

4.2 Types of Movement

In order to explain the types of movement that are possible a simple example involving two types of characters is presented. They will be used to show the different representations of characters that are possible in the multi-layered system. Consider a simple simulation that is composed of two types of characters. The first group consists of standard human sized characters; the second consists of giant humanoid characters that are significantly larger.

4.2.1 Single Layer Character Representation

The simplest way to represent these two groups of characters is to simply represent each individual character with a single navigation agent. All of these navigation agents then reside on a single cell layer that represents the traversable area in the simulation.

Using this representation the characters can be given appropriate behaviours and would traverse about the simulation without collision. The size of the cells in the cell layer would need to be quite large as it needs to be big enough to meet the conditions of the bounding radius of the giant characters. A large cell size will potentially hold a large number of the smaller human characters and as such will cause the number of objects in a neighbourhood to be large. This in turn will increase the number of comparisons required to determine the force caused by each behaviour. To avoid this problem the simulation can be extended to use another cell layer.

4.2.2 Multi-Layered Character Representation

An additional cell layer needs to be added to accommodate the giant characters. Both groups maintain their representation of a single navigation agent at their position, but the giant characters have their navigation agents placed upon the top cell layer.

With this current definition the multi-layered system has divided the two groups of characters. Now the characters representing the giants lie upon the top layer and are unaware of the characters upon the bottom layer. The characters on the bottom layer will consider both the other human characters and the giant characters to be within its neighbourhood and will avoid both. The cell size on the top layer will be the same as in the previous example, but the cell size on the bottom layer can be a much

higher fidelity and will therefore reduce the number of comparisons to determine an agent's neighbours.

With this representation the characters that lie on the lower cell layer are not able to move beneath the larger characters. The characters on the lower level will treat the navigation agents that represent the giant characters with equal avoidance to the other neighbours. In order to address this issue the concept of properties is introduced.

4.3 Properties

Properties are attributes that particular navigation objects in the simulation have in order to enhance their resulting movement. Properties are directly coupled with the behaviours associated with an agent and as such are specific to the simulation being created.

4.3.1 AffectsBelow Property

This use of properties is essential to further define complex behaviours, especially concerning the interaction between characters on different levels. As mentioned previously the multi-layered system is able to represent characters of vastly different sizes and it may be desirable to create movement where the smaller characters move underneath the body of the larger characters. In order to create this type of movement an additional property is used, the *affectsBelow* boolean value. By default this value is true, but when it is false, the navigation object with this property is ignored by agents that lie on any cell layer below the current navigation object. This therefore allows for Navigation Objects on a parent layer to interact with their neighbours on the same layer, but to not affect those agents that lie in any child layer.

Next the representation of the giant characters on the top layer is modified to use this new property. The character is now divided up into two parts, the legs and the main body. The main body is represented by a navigation agent that lies on the top layer. This main body has the *affectsBelow* property set to false. The legs of the character are represented by navigation obstacles that lie on the child layer and their *affectsBelow* value is left to the default value of true. See Figure 3 for the visual example of this representation.

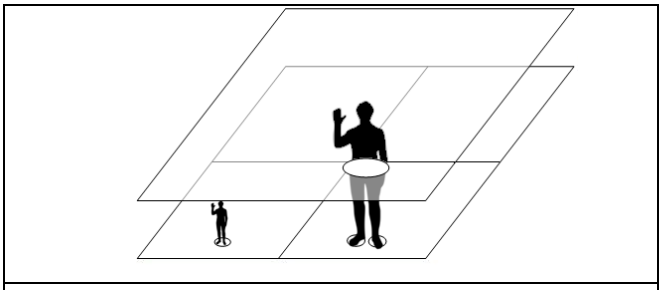

Figure 3. A human and giant character represented on two cell layers. The human character is represented by a navigation agent on the bottom cell layer; while the giant character is represented using multiple navigation objects on both the top and bottom cell layers.

As the character's body moves throughout the simulation the position of the navigation obstacles that represent the legs of the character are updated to match the animation of the character.

Now given this setup a new behaviour that modifies the existing separation behaviour is created so that it takes into account the additional *affectsBelow* property. The separation behaviour now only uses those navigation objects whose cell layer is equivalent in depth or whose *affectsBelow* property is true. Using this representation of the human characters will move past the agents on the parent layer by avoiding their legs but without taking a preference for moving beneath the parent layer.

By extending this property to use a scalar value the agents on lower levels can be made to prefer or avoid travelling underneath the bodies of the agents on the parent.

The movement of the navigation obstacles that represent the legs depends on the position of the body agent that lies on the upper layer. The position of these obstacles needs to align with the visual representation of the object in the simulation to ensure that the collision avoidance is consistent with the model being displayed. If the model being represented lifts its legs high enough such that the agents on the child layer could potentially pass underneath it then the navigation obstacle can be removed or deactivated while the leg is in the air. It was found during a prototype implementation of this property that the sudden return of the navigation obstacle to the child layer caused child agents to clip through the feet of the parent agent. This can be considered a desired property if the simulation is intended to show child agents being crushed by the feet of the parent agents. If this is undesirable the navigation obstacle that represented the leg can be re-added a few seconds in advance of the foot being returned to the child layer. This causes agents to begin moving out of the way in advance of the foot being returned to the ground.

This concept of using two layers to represent a character to allow smaller characters to move underneath them is not limited to only two layers. The same approach can be extended indefinitely to allow for even more complex interactions.

4.4 Environment Representation

The representation of the static environment in the simulation should also be adjusted to match the agents that are in the simulation. There are a number of ways that objects can be represented and several different choices are examined below.

Continuing from the previous example, obstacles in the environment that affect the agents on both layers should be represented by a navigation obstacle that lies on the top cell layer. Such a navigation obstacle will be included in the neighbourhood of the navigation agents in both the bottom and top cell layer and thus both these groups of agents will avoid it.

If on the other hand a piece of the environment only affected the children and was crushed by the feet of the characters whose navigation agents lie on the top layer, then the object can be represented by a navigation obstacle on the lower cell layer. The obstacle in the environment can then be destroyed when a parent agent's foot obstacle collided with it.

A further type of environment, where the child agents could move through it, but the parent agents would be forced to go around can also be represented. Such an obstacle in the environment could be a building, through which the smaller human characters can move about, but which the larger giant characters must navigate around. To create this type of movement the object in the environment could be represented by

a series of navigation obstacles on the lower layer, and a single larger navigation obstacle on the top layer that has the *affectsBelow* property set to false. See Figure 4Figure 5 for a visual example of this representation involving a tree.

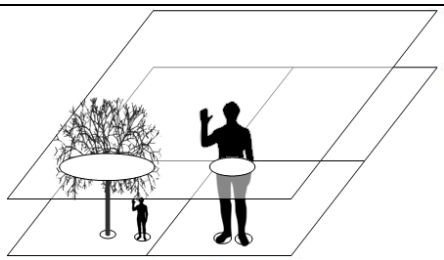

Figure 4. An example representation of a tree as part of the environment. The tree is represented on two layers which ensure that the smaller human sized characters are able to navigate close to the trunk while the larger giant sized characters are restricted to navigating around the branches.

4.4.1 Example Properties

Two additional properties that are generally useful in a simulation are *isActive* and *isBlocking*.

The first, *isBlocking* is a boolean value that determines whether a navigation object should be included in the multi-layered flocking system with respect to other character's neighbourhood. By disabling this value we are able to stop this object having any influencing behaviour on other characters. It can be used to represent buildings that have been destroyed or ethereal objects such as ghosts.

The *isActive* property serves a related purpose. It is used to enable or disable whether a character should be influenced by the behaviours it uses. With this property a character can still affect the movement of its neighbours without itself being affected. This property can be used to represent characters that have died but whose body still provides an obstacle that other agents must avoid.

4.5 Cell Layer Hierarchy

Up until this point the simulations presented have followed the format of smaller agents lying on a plane with larger and larger agents being placed in successively higher layers. The representation of layers in these examples corresponds to an increasing height value, but this is not the only ordering that the layers are able to represent. Furthermore it should be noted that the examples so far have also only contained cell layers where each parent layer has a single child layer. By sharing parent layers, simulations are possible in which there are smaller agents both below the larger agents and above them. A possible example is that of a flock of birds that flies at the head height of the giant characters while human sized characters move around their feet. The simulation would contain two cell layers for the humans and birds, both of which have the third layer with giant characters as their parent. Note that the child cells of both layers must match up to the boundaries of the parent layer to conform to the conditions described in the algorithms section.

4.6 Behaviours

This next section discusses the different behaviours that could typically be used in the multi-layered system, first discussing common movement behaviours and then examining the development of more complex behaviours that allow for the integration of the multi-layered system with path-finding systems.

4.6.1 Standard Movement Behaviours

The multi-layered system uses the same force combination methods as the Reynolds model. This allows the system to use any movement behaviour that can be defined as a force upon the navigation agent at a point in time in the simulation.

Using the standard weighted combination or force priority techniques mentioned in the algorithm section, steering behaviours can be developed that deal with the majority of standard movements required in animations and games. Reynolds provides an overview of these typical behaviours[10]. The steering behaviours that Reynolds covers include: *Seek*, *Arrive*, *Pursuit*, *Wander* and *Path Following* in addition to the standard *Separation*, *Cohesion* and *Alignment* that create flocking movement. Additional behaviours can easily be created by adapting or extending any of these behaviours.

4.6.2 Multi-Layered Movement Behaviours

By utilizing the common movement behaviours that are applicable to the multi-layered system, simulations can be made that involve a range of different movement types. Combining several of the above common behaviours allows a designer to produce varied types of movement within a simulation. The *Separation*, *Cohesion* and *Alignment* behaviours can be used to create a standard crowd. In Figure 5 an example is displayed that shows a crowd of characters moving through the legs of a larger four-legged character. This simulation was created with the multi-layered flocking system by combining the standard behaviours of a Reynolds model with a *Wander* behaviour for the four-legged character. The same simulation could be modified to make the crowd flee the four-legged character, swarm beneath it, or use the four-legged character as protection from another larger character by simply adding slightly modified versions of the *Separation*, *Seek* or *Pursuit* behaviours.

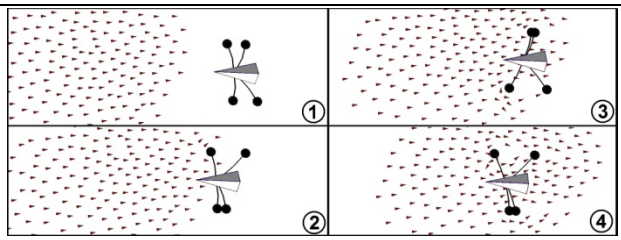

Figure 5. Example screenshots of a four-legged character moving through a crowd of smaller characters. The simulation is set up with two cell layers; the four-legged character is represented by a navigation agent on the top layer and four navigation obstacles on the bottom layer. The smaller characters are represented by a single navigation agent on the bottom layer.

An additional behaviour can be made that acts as the opposite of the *Separation* behaviour. Instead this behaviour can be used to cause characters to be drawn towards another object in the simulation. Such a behaviour can be combined with the *affectsBelow* property to create movement where child agents are attracted towards a parent agent or navigation object, giving the impression that the child is hiding beneath the parent or environment for safety. This behaviour is not limited to hiding

beneath other agents and could also be used to attract characters towards an area, whether static or moving.

4.6.3 Integration with Pathfinding Systems

The multi-layered system restricts a character's knowledge to that of its local neighbourhood and as such would perform poorly in a simulation that requires complex pathfinding. Despite this, complex path-finding is easily added to the simulation by providing behaviours that act as a bridge between more complex pathfinding models and the multi-layered system. For example, a new behaviour that extends the *Path Following* behaviour can be created that performs an A* search. This behaviour can then follow the computed path by returning a force towards the next waypoint in the path.

5. Conclusion

The multi-layered flocking system allows designers to create crowds that are able to contain members of varying sizes and that allows for characters in those crowds to travel not only around but also underneath and above larger characters. The resulting movement produced by the system is able to represent characters moving in groups of varying size, and these groups are able to move through, around and underneath other characters in the simulation. By only providing the minimum amount of required knowledge to each character the system is able to reduce the computational requirements. The division of the simulation space also leads itself well to multi-core architectures.

There are some disadvantages and issues that need to be considered when using the system. Firstly fine tuning the forces to find a desired movement can be time consuming. Secondly the entire structure of the multi-layered system uses the assumption that in a situation with a smaller object and a larger object, the smaller object is expected to navigate around the larger object. This structure is what maintains the efficiency of the model in the simulation. Though this structure is appropriate for the situations suggested above, it is not true of every situation. If an agent requires information in the layers below it a careful implementation is required to ensure that the complexity of these behaviours is kept to a minimum.

This system allows each character to define its own behaviours. Thus it is possible to create both group behaviour such as crowds and combine this with specific individual goals and desires. Furthermore the system can use specific navigation behaviours to incorporate more complex navigation techniques such as pathfinding.

5.1 Future Directions

The multi-layered flocking system in this paper is an abstraction from the 2-dimensional version of Reynolds flocking method. It is theoretically possible to create a similar multi-layered flocking system that uses the 3-dimensional version of Reynolds flocking method as a basis. Such a model would use a 3-dimensional cell representation, to provide spatial partitioning using an abstract volume. Such an implementation may prove to be complex to understand for a designer, but warrants investigation.

An in-depth study of the varying types of behaviours which are possible to create using this system would provide for a useful overview to present to designers who wish to use the system. This study could also include the ease at which alternate techniques could be integrated within the system using specialized behaviours.

6. REFERENCES

[1] Gas Powered Games. 2007. Supreme Commander, THQ.

[2] Kamphuis, A. and Overmars, M.H.2004. Finding paths for coherent groups using clearance. In Proceedings of the 2004 ACM SIGGRAPH/Eurographics symposium on Computer animation, 19-28.

[3] Lebar Bajec, I., Zimic, N. and Mraz, M. 2007. The computational beauty of flocking: boids revisited. Mathematical and Computer Modelling of Dynamical Systems, 13 (4). 331-347.

[4] Maïm, J., Yersin, B. and Thalmann, D. 2008. Real-time crowds: architecture, variety, and motion planning. SIGGRAPH Asia '08: ACM SIGGRAPH ASIA 2008 courses, ACM, Singapore, 1-16.

[5] Nieuwenhuisen, D., Kamphuis, A. and Overmars, M.H. 2007. High quality navigation in computer games. Science of Computer Programming, 67 (1). 91-104.

[6] Pettré, J., Grillon, H. and Thalmann, D. 2008. Crowds of moving objects: navigation planning and simulation. ACM SIGGRAPH 2008 classes, ACM, Los Angeles, California.

[7] Pettré, J., Laumond, J. and Thalmann, D. 2005. A navigation graph for real-time crowd animation on multilayered and uneven terrain. In First International Workshop on Crowd Simulation.

[8] Reynolds, C.W. 2006. Big fast crowds on ps3 In Proceedings of the 2006 ACM SIGGRAPH symposium on Videogames, 121.

[9] Reynolds, C.W. 1987. Flocks, herds and schools: A distributed behavioral model Proceedings of the 14th annual conference on Computer graphics and interactive techniques, 25-34.

[10] Reynolds, C.W. 1999. Steering behaviors for autonomous characters. Game Developers Conference.

[11] Richmond, P. and Romano, D. 2008. Agent Based GPU, a Real-time 3D Simulation and Interactive Visualisation Framework for Massive Agent Based Modelling on the GPU. International Workshop on Super Visualisation.

[12] Scutt, T. 2002. Simple Swarms as an Alternative to Flocking. Game AI Programming Wisdom, Charles River, 123-456.

[13] Silver, D. 2006. Cooperative Pathfinding Game AI Programming Wisdom 3, Charles River.

[14] Sud, A., Andersen, E., Curtis, S., Lin, M. and Manocha, D. 2008. Real-time path planning for virtual agents in dynamic environments. ACM SIGGRAPH 2008 classes, 55.

[15] Treuille, A., Cooper, S. and Popović, Z. 2006. Continuum crowds. ACM SIGGRAPH 2006 Papers, 1168.

[16] Ubisoft Montreal. 2007. Assassin's Creed. Ubisoft.

[17] van den Berg, J., Patil, S., Sewall, J., Manocha, D. and Lin, M. 2008. Interactive navigation of multiple agents in crowded environments. In Proceedings of ACM Symposium on Interactive 3D Graphics and Games.