Crowd Simulation Pipeline

with Maya and Renderman

Thomas Melson
E9083768
MSc Computer Animation
Masters Project

September 2006

Contents

Initial Aims 1
Initial Aims 1

Previous Work 2
Procedural Animation 2
Behavioural Animation 3
Latest Behavioural Techniques and Tools 4

Technical Background 6
Rendering Techniques 6
Efficient Simulation Techniques 7

Design 9
User Interface 9
Simulation Details 9
Visualization and Rendering 9

Implementation 11
Pipeline Overview 11
User Interface 14
Simulation Implementation 16
Environment 18
Agent Movement 20
Agent Behaviour 22
Environment Interrogation 22
Environment Objects and Agent Subtypes 23
RIB Generation 25

Conclusions and Future Work 27
Future Work 27

References 29

Appendix - TMCrowd User Guide 30

Introduction

When dealing with large numbers of characters in computer animation, certain difficulties arise that would otherwise not be a problem. The simple movement and behaviour of the characters is suddenly far more complicated due to the large numbers of individuals on screen. Although probably less detailed individually, collectively they may constitute thousands of separately moving parts. Manually key framing these agents individually is not a realistic option and group behaviour can be unpredictable and chaotic. Recent advances in the power of computers have given rise to many new techniques which aim to simplify and enhance the simulation of crowds.

Initial Aims

The idea is to undertake the development of a system that can be used to efficiently produce good quality crowd simulations for animation. The system is to be used for a specific animation of a colony of ants and as such the aim is not to produce a completely generic tool for crowds. However during development of the pipeline and the various tools it comprises, re-use and flexibility is to be borne in mind so that the system can be easily adapted for further animations.

Previous Work

Procedural Animation

The processing power of computers can be utilised to generate crowds with procedural animation. The behaviour of a group of characters may be complex but the behaviour of the individuals involved may not be. The idea is to define the behaviour of a character and replicate this across the group. One common way of doing this is using particle systems, with each particle representing a member of the crowd. Industrial Light and Magic (ILM) use a Maya particle system in their crowd generation.

> "Each particle has attributes on it that allows us to determine its position, orientation, what sequence of animation (cycle) it uses, what frame of the cycle it is currently on, allowable velocities for that frame and other information such as color and geometry variation." **[1]**

The technique has been used in many films including Star Wars: Episode 1.

Figure 1. Crowds in Star Wars: Episode 1

Animation cycles are attached to the particles across the scene and rendered out. This technique is effective but can suffer from the problem of characters lacking variety, in appearance and perhaps more importantly in behaviour. If all the characters in the scene are cycling through the same animation, the eye will immediately pick up on this and the scene will not look natural. ILM got around this problem by defining extremely long cycles and offsetting the starting point for each character.

Behavioural Animation

Behavioural animation is a type of procedural animation and a crowd system is really an extension of the particle system. Agents are given an orientation and a volume and make decisions to determine their own actions. Essentially it refines the idea of a crowd as a group of individuals.

The early pioneer of this kind of technique was Craig Reynolds in his work on Boids **[2]**, a model of bird flocking. His idea was to avoid designing the behaviour of a group as a whole and to concentrate on the individual. Then a group are put together in a simulation and the computer does the rest. These kinds of systems can be very unpredictable but given careful design some extremely impressive and interesting results can be achieved. This kind of outcome is called emergent behaviour.

Reynolds' initial program was based on 3 simple rules:

Separation : Steer to avoid crowding local flockmates
Alignment : Steer towards the average heading of local flockmates
Cohesion : Steer to move toward the average position of local flockmates

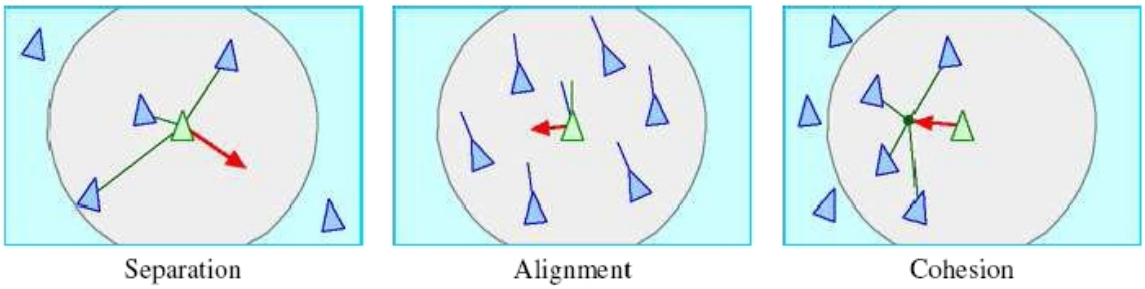


Figure 2. Boids flocking rules

These simple rules led to convincing behaviour and most observers of the animation identified the motion as flocking. The scope for this kind of system is virtually limitless. Behavioural complexity can be increased, including more details of environment, agent psychology etc.

Latest Behavioural Techniques and Tools

In recent years many systems have emerged that take advantage of emergent behaviour and combine it with various other techniques to create highly sophisticated, yet flexible tools. These tools include Plug-Ins for existing 3D software, in-house systems at animation houses and standalone commercial applications.

Massive, produced by Massive software **[3]**, is a standalone tool that was initially developed for the Lord of the Rings films to generate the huge numbers of warriors in certain scenes. Massive combines animation, motion capture, Artificial Intelligence, Rigid Body Dynamics and more to create its animation.

Massive is described in the SIGGRAPH 2006 Exhibitor Directory as follows:

> Massive Software is the world leading, Academy-Award winning software for creating autonomous, agent-driven animation and simulation, best known for crowd-related visual effects and digital stunts for film and television. **[4]**

A screenshot of the software is shown in **Figure 3**. The system is highly flexible and is scriptable so the user can generate their desired behaviour. Central to the system is the Massive 'Agent Brain'. Each agent has a brain which controls the agent's actions and interaction with other agents. At any time an agent has a number of characteristics ranging from size and strength to aggressiveness. Strength can weaken, moods can change and any number of factors may contribute to an agent action or a decision. These factors are connected within the brain by fuzzy logic rules which again can be programmed by the user. The end effect is a much more organic system, where individuals and therefore the emergent crowds have very natural looking behaviour.

Figure 3. Massive Software

One aspect to be wary of when using these kinds of techniques is that overall control is not lost, especially when producing clips of animation with a definite storyboard. So some method of user control needs to be integrated into the system. Even in the real world a director can direct a crowd of film extras.

Technical Background

Rendering Techniques

With so many separate models on screen at once in crowd scenes, the amount of geometry involved can obviously soar and efficient rendering becomes an important consideration.

Traditional model making animation encounters similar problems when building huge numbers of models for crowd scenes. These problems are alleviated by building models of varying detail, placing the more detailed, time consuming models nearer the camera. Computer animation can use a very similar technique in rendering and Pixar's Photorealistic Renderman (Prman) has implemented Level of Detail functionality which allows the user to define multiple versions of the same model, with varying levels of detail. In a crowd scene of thousands it is inevitable that the majority of characters on screen will be significantly farther away than those that are near the camera. Therefore representing these characters with vastly simplified models can remove huge amounts of geometry from a scene, increasing rendering efficiency dramatically.

This functionality has been used in many production level animations including Troy and Star Wars: Episode 1.

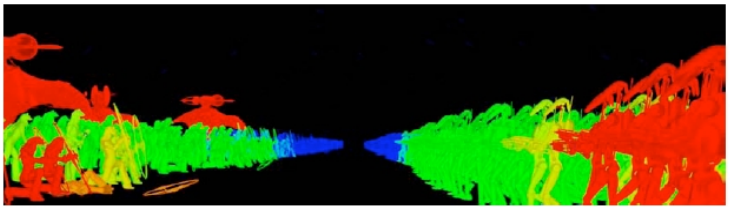


Figure 4.3: *Star Wars I* – Level of Detail RGB representations

Figure 4.4: *Star Wars I* – Gungan LOD Levels

Figure 4. Level of Detail renders for Star Wars

In 'Troy', Framestore CFC and The Moving Picture Company use Renderman to render vast crowds and a fleet of battle ships [5].

Figure 5. Crowds in Troy using Level of Detail

Efficient Simulation Techniques

When dealing with large crowds a simulation can benefit from efficient methods of calculation. Areas that are particularly open to improvement are the interaction between agents and agent route planning. If agents are to observe each others behaviour when making decisions then there is potential for a lot of calculation at each step of the simulation. The simplest implementation would be for each agent to inspect the position and behaviour of every other agent at each step. This approach can cause a program to run very slowly as calculation time goes up exponentially as more agents are added to the system. In practice most agents may find that only a small subset of their neighbours are close enough to be interesting. Therefore simulation time can be greatly reduced if only those that are within a certain distance are interrogated. A popular method of breaking up space is using quadtrees and octrees. A quadtree is a system of breaking a two-dimensional space into smaller partitions. It is described by Wikipedia as follows:

> A **quadtree** is a tree data structure in which each internal node has up to four children. Quadtrees are most often used to partition a two dimensional space by recursively subdividing it into four quadrants or regions. **[6]**

Craig Reynolds' latest work has involved finding ways to locate neighbours quickly within simulation working for Sony on a PS3 games engine **[7]**. He discusses various ways of storing agents in lists in an order that reflects

their position in the environment. Then when an agent needs to interrogate its neighbours it can simply run a much shorter, local search.

Design

User Interface

The system should be easily usable by anyone so it has been decided that the work flow should start within a standard 3D tool. Maya has been chosen as it is a very high end 3D tool used extensively in production environments. Here the environment geometry can be defined as well as the animation cycles for the crowd members. MEL (Maya Embedded Language) scripts can be written to provide the user with an interface to setup the parameters for the simulation. The relevant information about the scene and the crowd to be simulated can then be provided for the simulation itself to use.

Simulation Details

The details of the simulation itself need to be written in a language more powerful than MEL script. The obvious choice is C++. Its Object Oriented nature is ideal for creating the necessary structures that will represent the crowds and their individual members. Also if designed correctly the system will be easily extensible for future work. It is also a very fast compiled language which is an important consideration when dealing with very large numbers of agents in a simulation.

Visualization and Rendering

Once the simulation has been run and the crowd simulated the user will need to view the results. Since the environment geometry is created in Maya, a sensible approach would be for the simulation to feed the results back into Maya to be visualized in the original environment. MEL can be used to import the data and to create simple representations of each crowd member, then key-frame their positions and orientations. An agent can be represented with a very simple piece of geometry, just to signify position, direction and scale. Any attempt to import the actual animated geometry to be used in the final animation would result in a massive loss of efficiency. There is no real need to view the fine details of the individuals at this stage and any benefits would be far outweighed by the reduced performance of Maya with so much geometry likely to be present in a crowd scene.

When the user has generated a satisfactory crowd and added any camera movement and extra details to the scene, the entire thing needs to be rendered. The inbuilt rendering software in Maya is not an option as the geometry for the crowd members is not present. Instead it has been decided that Prman will be used to render the sequence. Tools are available in Maya that can export RIB files for the scene complete with shaders imported from within Maya. Also the Level of Detail functionality discussed above can be made use of to improve rendering efficiency.

This still leaves the issue that the scene data exported from Maya does not actually contain the character models for the animation. This information will need to be added by a separate script or program by reading in both the environment and the animated character RIB files and combining the data appropriately to create the final animation RIBs for each frame.

Implementation

Pipeline Overview

With the basic design of the pipeline decided on, the specifics of the system need to be laid out. **Figure 6** shows the details of the workflow.

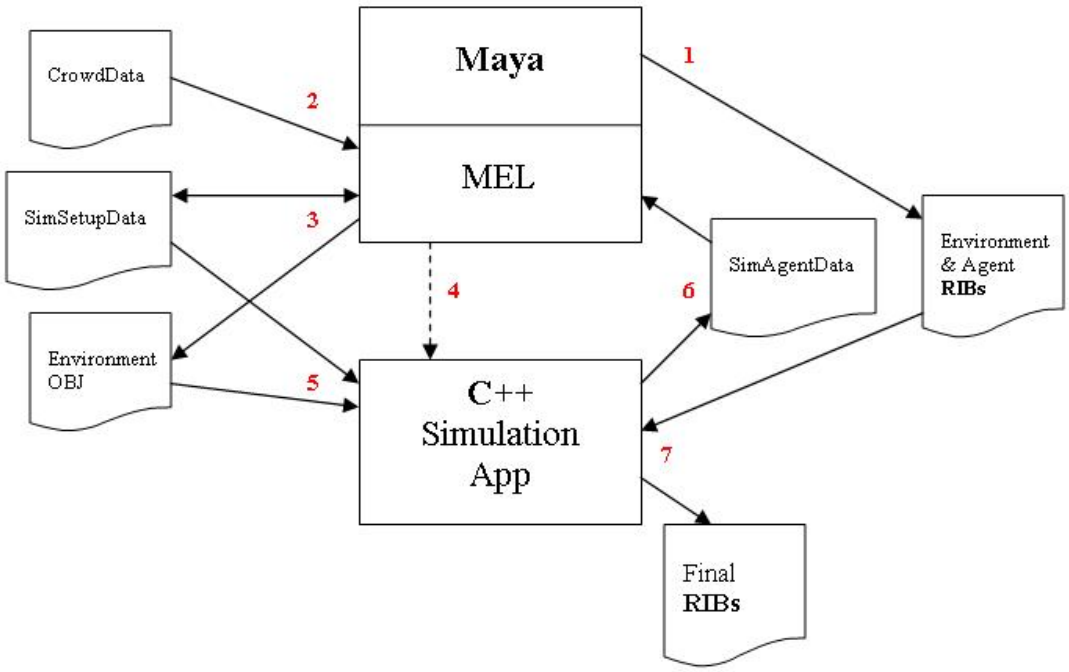


Figure 6. Crowd system pipeline

A sub directory named TMCROWD should be created by the user where the Maya project file is saved. This directory will contain the C++ application along with all the data and RIB files. The system can be broken down into 6 steps:

1. First the user will model the scene for the crowd and separately the animated characters to be used as agents. SLIM can then be used to export the scene and agents as RIB files.

2. When the scene is ready, the crowd MEL script can be run to display the UI (User Interface) and setup the crowd details. The first thing the script does is to run the Crowds C++ application and ask it to provide the CrowdData file which it then imports. This file contains

information about the type of agent to be simulated. The UI is then displayed. If the window has been previously opened, the SimSetupData file will be found and used to recall the user's previous settings.

3. The user can now configure the settings for the simulation. Part of this process is to select the environment geometry and select the Export OBJ button to export the geometry for the simulation to use. The simulation can then be run and the setup information is exported to the SimSetupData file.

4. When running the simulation the user has three choices. They can select an OpenGL visualisation, running the simulation with an interactive OpenGL display. Alternatively they can select a Maya visualization, which will do the same thing; only the results will be read back into Maya and visualized. Lastly they can select render which will run the simulation and produce the final RIB files.

5. When the C++ application is run it will read in the SimSetupData file and the OBJ file, then set up the environment and simulate the crowds. The next step is determined by the user's previous selection. For an OpenGL visualization the simulation simply exits. If a Maya visualization was requested, step **6** is next. If a render is requested, step **7** is next.

6. The application outputs the information for each agent at each frame to the SimAgentData file and exits. On exiting, the MEL script will resume, read in the data and produce the visualization by creating simple geometry and key-framing every agent at every frame.

7. The application reads in the environment and agent RIB files and as it runs through each frame will export a final animation RIB file, ready for rendering.

This pipeline allows a user to setup crowd parameters and then visually check the results before a final render is requested. A simulation can be setup and viewed with OpenGL to assess its suitability. Once a suitable setup has been found, a Maya visualization can be produced to bring the agents into the scene in Maya. Now the user is free to coordinate camera movement, offset the start of the animation and also select particular agents to be removed from the simulation. Once the scene has been fully composed, Render can be selected to produce the RIB files for the final animation.

Designing a system that has discrete parts keeping the user interaction and the simulation separate has several advantages. The two parts can be worked on individually without concern for the affect on the other. The only communication they have is through the SimData files and the initial configuration CrowdData file. Once the system is working, an updated simulation program can be produced that simulates a completely different type of crowd. As long as the new program is setup to provide the altered CrowdData file to match the details of the new crowd, the new program can be seamlessly added to the pipeline.

Another advantage of keeping the two parts separate is that it doesn't tie the system to using Maya. If at a later data the simulation was needed for use with another application, this would be achievable by recreating the User Interface and visualization functionality within the new system. Since the bulk of the functionality is within the simulation program itself, this process would be relatively painless.

One factor this design necessitates is identical simulation from identical parameters. If a visualization is requested, followed by a render, the resulting simulation should obviously produce the same results that the user decided on. With the same input to the simulation, the only thing that can change the output is the use of randomly generated numbers. As with most behavioural simulations, the crowd program makes use of random numbers when using probability to decide on behaviour. Should these values be truly random each time the application is run, the resultant behaviour would differ each time. This outcome can be avoided by using a random number generator that takes a seed value to inform it where to start in its random generation. If the seed is the same, the results will be the same. It was decide to default the seed to zero, but to provide this parameter to the user to allow them to generate alternate simulations if they wish.

If the user changes certain input parameters (number of agents, input geometry etc.) the simulation will unavoidably change. However if an agent is selected to be removed from the simulation the results would otherwise be expected to be the same. The alternative would defeat the purpose. To avoid this occurrence the program will still simulate these removed agents but will not render them in the final animation.

User Interface

All the functionality required in Maya for the system is provided by MEL script. The three main areas of functionality are a user friendly interface which can accept all user input, the importing and exporting of text files to disk and the ability to automatically create and animate marker geometry representing the agents.

The interface was broken down into three areas and each area was given its own tab on the window. The first tab, shown in **Figure 7** is the Animation Details.

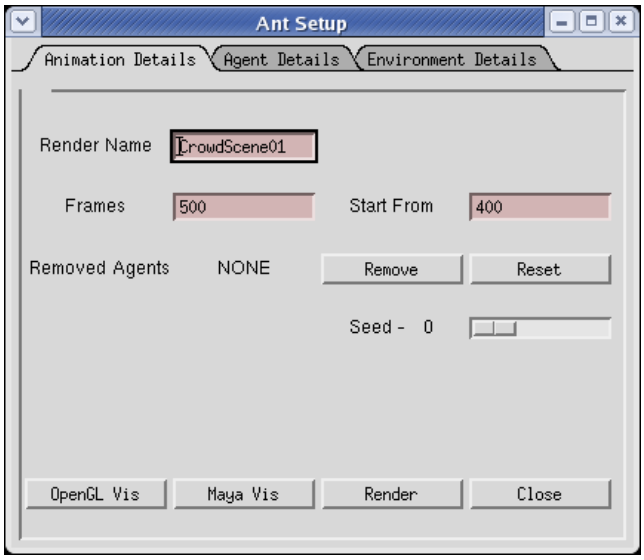


Figure 7. Animation Details

The first input control is the Render Name. The name the user gives will dictate the name of the final RIB files and allows multiple renders to be made without overwriting previous files. The name should also match any exported environment RIB files. Next is the number of frames and the Start From control which allows the user to wait a number of steps in the simulation before starting an animation of a given length. The Removed Agents controls allow the user to select any number of visualized agents on the screen and click the Remove button to remove them from the animation. The Reset button will cancel all removed agents. The Seed slider provides a way of changing the animation simulated without actually changing any of the simulation details.

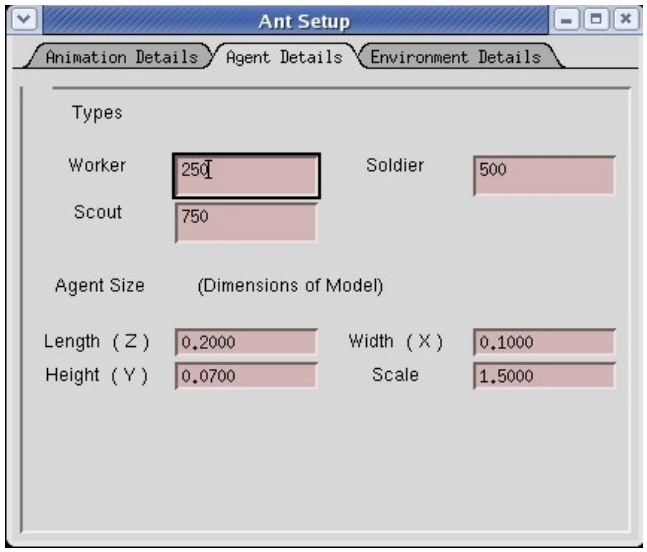


Figure 8. Agent Details

The Agent Details tab, shown in **Figure 8** sets up the number, type and dimensions of the agents in the simulation. The number of controls in the Types section is dictated by the CrowdData file which describes the different sub agents there can be. The agent Dimensions parameters should match the dimensions of the agent models being used for rendering. Precision is not vital but the values should be fairly accurate. The scale field simply allows the user to scale the models to any size within the simulation.

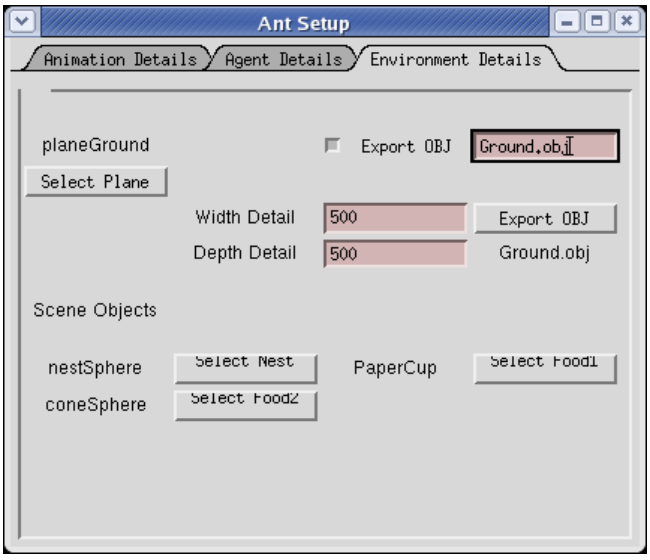


Figure 9. Environment Details

The Environment Details tab is shown in **Figure 9** and is used to setup all environmental data. If exporting the environment as an OBJ file the user must select the Export OBJ checkbox, enter a name for the file, select the geometry in the scene and click the export OBJ button. The Width and Depth Detail fields dictate the degree of detail to which the simulation will model the OBJ data. If not exporting as an OBJ file, the user should simply select a polygonal plane to export directly. The only other environment details that need exporting are any special scene objects. The number and names of buttons here again depend on the CrowdData file. Any number of objects may be asked to be provided (although the user doesn't have to export all or indeed any of them). The coordinates of these objects will be used in the simulation to affect agents' behaviour in some way.

Once all settings are configured, the user has the choice of either type of visualization or a render. MEL script is then used to write the settings out to the SimSetupData file and to execute the simulation program. On completion the script will resume and if a Maya visualization was requested, the SimAgentData file is read and the script will create, transform and key frame the agents into the Maya environment.

Simulation Implementation

The generation of the agent behaviour within the environment is all done within the C++ program. The program needs to be able to represent thousands of agents within a three dimensional environment, to keep track of them throughout the simulation and deal with all behaviour and interaction. The class structure needs to deal with this functionality and be flexible enough to accommodate future development. The main class structure of the program is shown in **Figure 10**.

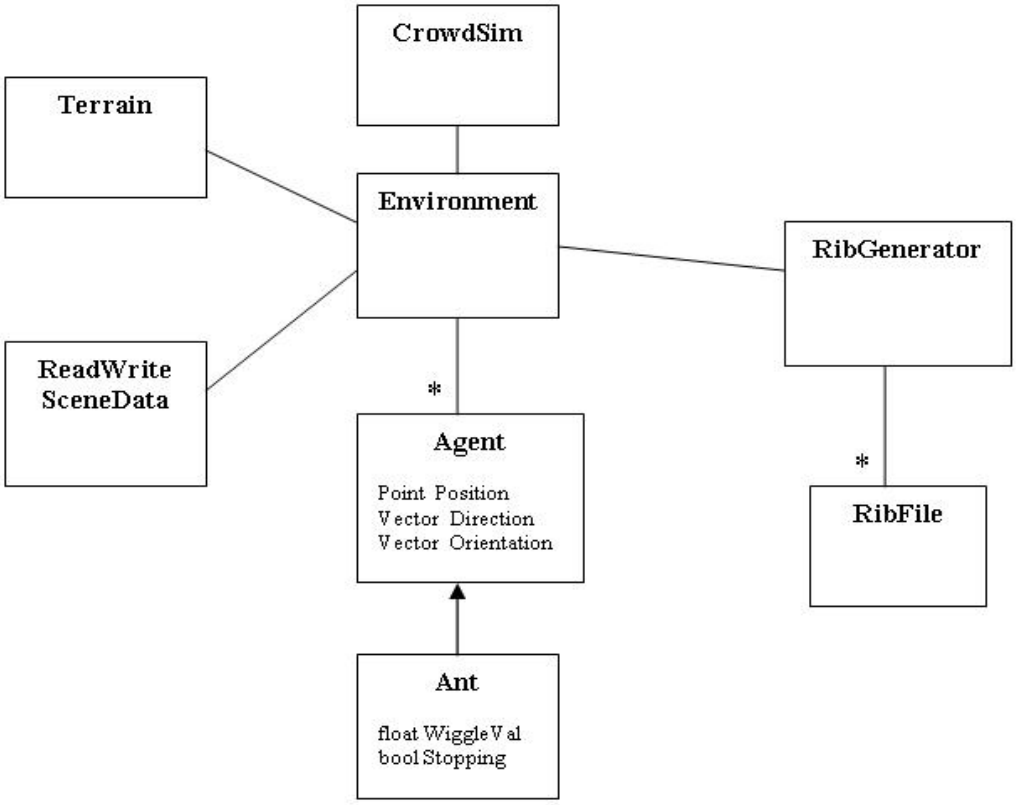


Figure 10. Simulation main class structure

The general structure is fairly simple with the singleton class Environment at the centre of the system. The Environment class creates and stores all agents in the simulation and is responsible for updating the environment and the agents it contains throughout the simulation. It uses a Terrain class to store the geometrical information about the environment and also maintains input / output classes ReadWriteSceneData and RibGenerator.

ReadWriteSceneData is responsible for all data transfer between the simulation and Maya (or other system). At the start of the simulation it reads the environment OBJ file, along with the SimSetupData file which gives the system all the information about the user's configuration. At the end of the simulation, if the user requested a visualization, it outputs the SimAgentData file, with information on each agent at every frame of the animation. The RibGenerator class handles the import and export of all RIB files, if the user has requested a render. Each RIB file is represented in a RibFile object, which maintains the structure of each file.

Each agent in the simulation is represented by an Agent object. It contains information on the agent's position, heading, orientation and speed as well

as methods to move it on within the environment. The Agent class can be instantiated directly and used to model the agents in a simulation. However this provides only basic behaviour and it is intended that the class be inherited from by others to add more specific functionality. This has been done with the Ant class, used in the simulation of the ant animation. Basic behavioural differences were added such as a slight wiggle when walking and a tendency to stop every so often. This was then added to with more complex decision making behaviour.

Environment

The representation of the environment within the program is key to the simulation. All agent behaviour takes place in the environment and it is vital that the geometry provided from Maya is represented accurately so the agents can determine where they are able to travel.

It was decided that the environment provided from Maya should consist only of polygonal geometry. This gives the simulation a discrete set of 3D points to deal with which makes calculation of surfaces easier than if complex NURBS surfaces were permitted. A second decision was made to further simplify the geometry representation to a grid of points displaced upwards or downwards to create a single deformed polygonal patch as shown in **Figure 11**.

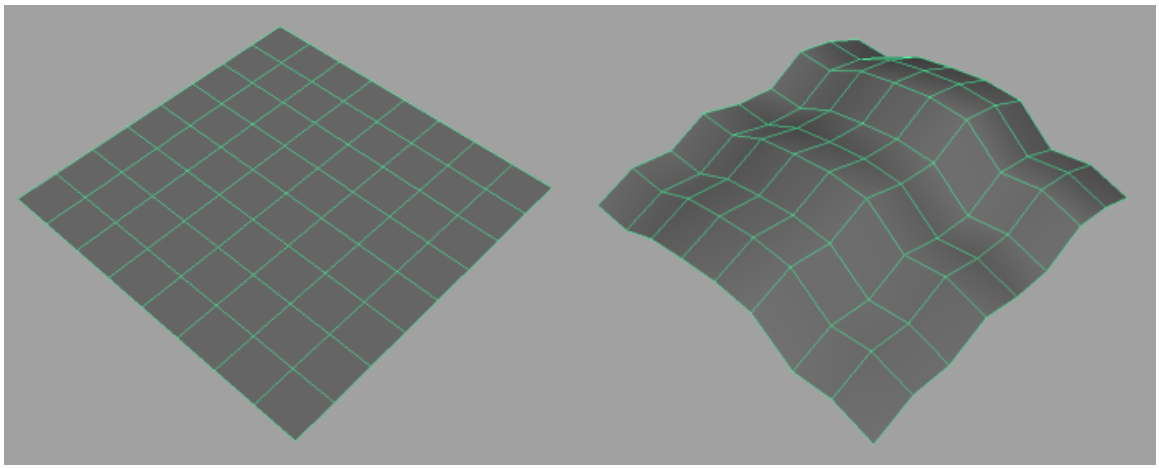


Figure 11. Flat and displaced polygonal grids

At first glance this appears to be quite a restriction for simulation environments. However if the number of columns and rows in the displaced grid are increased sufficiently, the detail of the model can

become as thorough as is required. The only real limitation is that each X-Z coordinate can only have one Y value; each point on the grid may only be displaced up or down and not in the X or Z direction to create overhangs or multiple levels in the geometry. The result is that a scene which contains multiple levels of some kind like a bridge over a road cannot be represented completely in the simulation. However a scene can be modelled containing a bridge, and by only exporting the geometry underneath, a crowd can still be successfully simulated below the bridge.

The reason for designing the system in this way is the advantages it brings when calculating agent behaviour within the environment. Each agent will have a position on the displaced grid and the terrain around them can be easily accessed when making decisions about where and how to move by simply accessing the data in the grid locations around the current position. If geometry was not represented as a grid but by a large list of polygon meshes all kinds of problems would arise. The process of locating geometry near an agent that is available to move to would be a lot less straight forward and efficient. Some kind of octree system could be used to find close geometry but then the question of whether a clear path is available between it and the agent arises. Also, complex geometry with small polygonal details could easily lead to undesired results from the movement logic. For this reason the advantages of the displaced grid system have been judged to outweigh the drawbacks and allow for more work spent on the behavioural side of the simulation amongst other areas.

To transfer the environment geometry in Maya to the C++ program an initial implementation allowed the user to create geometry in Maya by displacing a polygonal plane and exporting the positions of the vertices to the SimSetupData file. These points are imported into the program and used to recreate the geometry. However although the displaced grid method is suitable for representation within the simulation, it was seen as a significant limitation from a modelling point of view within Maya. Therefore an alternative method was provided for the user to export geometry. Any polygonal scene can be created by the user and then exported to an OBJ file for the C++ program to import. The user then creates a plane and positions it above the area of the model on which to simulate the crowd. This simply provides a quick and easy way to define the precise area the crowd should inhabit. Once the simulation program has imported the OBJ file and the dimensions of the plane, it can create a grid and displace its points so as to project it onto the model. This can be done by ignoring the Y coordinates of the polygons in the model and viewing them as 2D X-Z Polys. The following pseudo code describes this process:

```
for each Point in grid
{
       Point.Y  =  - MAX_INT
}

for each Polygon in model
{
    for each Point on Grid that lies within the bounds of the 2D Poly's vertices
    {
        if (Point lies inside the 2D Poly)
        {
            if (Polygon.Y  >  Point.Y)
                Point.Y  =  Polygon.Y
        }
    }
}
```

The process is performed once at the start of the simulation, giving the system the displaced grid to use throughout the simulation. The level of detail of the grid is decided by the user who provides the number of columns and rows to be used. This allows the user to model a polygonal environment and to export any part of it for the simulation of a crowd.

Agent Movement

With the environment represented in the simulation, the positioning and movement of the agents themselves was addressed. At this time the system is designed to simulate crowds that are constrained to the surface of the geometry i.e. agents that cannot fly. Therefore since the environment is a deformed grid, an agent's exact location depends on its X-Z coordinates within the environment; its Y coordinate will simply match that of the grids at this point. Also when moving, their choice of direction can be viewed as some direction across the X-Z plane, although they may actually be travelling up or down very steep slopes at times. This idea is shown in **Figure 12**.

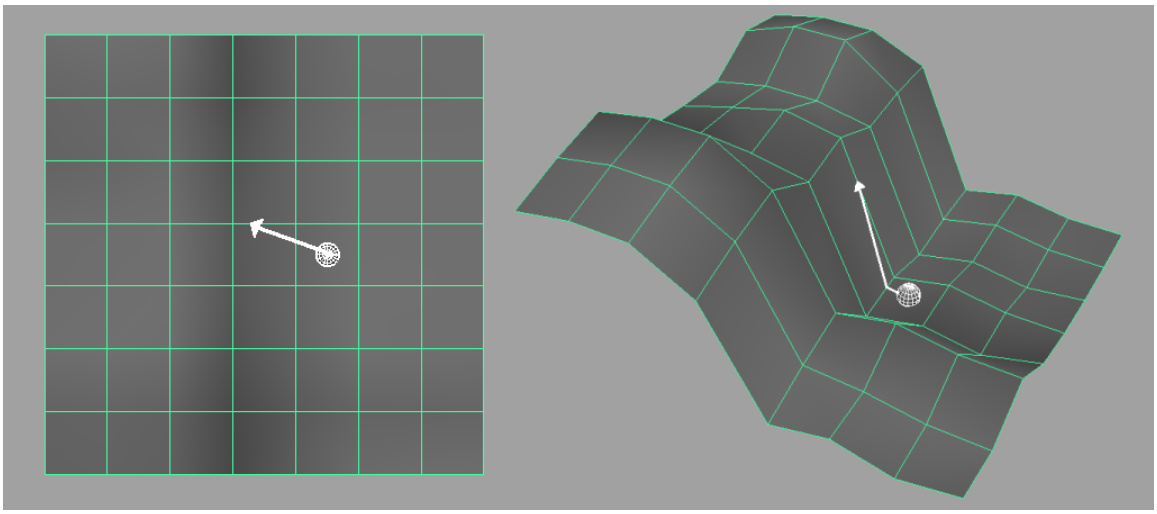


Figure 12. Aerial and perspective view of agent moving in environment

An agent is shown on a deformed grid in an orthographic projection from above and a perspective view. From above the grid structure of the environment is clearly apparent and the agent is heading in a clear direction across the grid. All decisions about which direction to take are implemented in the Agent class by viewing the environment in this two-dimensional way. The squares of the grid around the agent are assessed and the agent decides on a direction across the grid in which to travel. The perspective view shows what is actually happening when the agent moves, taking the vertical displacement of the points on the grid into account.

Once the direction has been decided and the agent actually has to move, it is important to stop viewing the terrain in a two-dimensional way and to move in three dimensions. It would be possible to move the agent across the grid in two dimensions, and then simply adjust the agent's position vertically to match the terrain. However this would lead to undesirable behaviour. If travelling up or down a steep slope, one small grid square could actually cover a large amount of terrain once the displacement is accounted for and agents would appear to speed up whilst travelling up or down hill. For this reason the following technique is used:

Direction is decided on in two dimensions.
Using the direction, the next point where the agent will cross a grid line is calculated.
The three-dimensional points of this line are then interpolated to find the position in three-dimensional space where the agent will meet the line.
This point is then used as the agent's short-term target.

Agent Behaviour

With the agents able to move around an environment, the implementation of the decisions behind their behaviour was addressed. As described above, basic behavioural methods are implemented in the Agent class. It contains a Move method which is called at each step of the simulation. This method will move the agent in its current direction and decide whether or not to change direction. In the Agent class this just takes the form of a random direction change from time to time. Any more complex decision making for an agent is left to any derived class but additional methods are provided in the Agent class to give information about the environment around the agent.

Environment Interrogation

To aid in the collection of environmental data the environment is divided into a grid of squares (separate to the geometrical displaced grid). Each square is represented in a two- dimensional array and holds a GridData object that stores information about that square, including a list of the agents inside. It was decided that a full quadtree system was not necessary as the system does not necessarily need to run at a high interactive rate, being for animation rather than a game engine. Nevertheless at present the system can maintain interactive speed whilst accommodating several thousand agents. The Agent class method CollectEnvironmentData is called at each step for each agent. It inspects the contents of the surrounding squares and adds all other agents to a vector list, CloseAgents.

The method FindCollidingAgent in the Agent class will iterate through the CloseAgents list and provide variables containing information about any impending collisions with other agents. It takes variables VisionAngle and VisionDistance defining a cone of vision within which to take any agents into account. Using the position, direction and speed of any agents inside it calculates if there is likely to be a collision between that agent and this within a certain tolerance. **Figure 13** shows the angle of vision of an agent with an expected collision shown in red. If a likely collision is detected, a Colliding flag is set along with a pointer to the agent and a Point representing the expected point of collision. If an agent is within a very small distance away, a collision is determined to have occurred and a Collided flag is set along with a pointer to the agent collided with. If multiple expected collisions are found, the closest takes priority. This method is for use by derived classes; the Ant class calls it at each step and

defines behaviour to take evasive action and to stop briefly if an actual collision is detected.

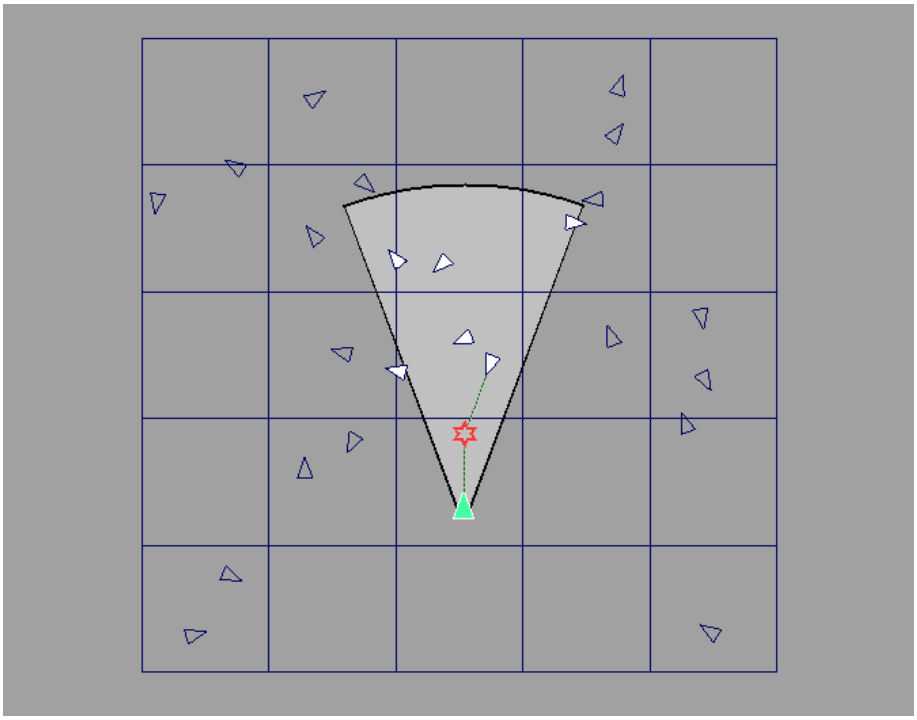


Figure 13. Agents detects a likely collision

Scope for additional behaviour in derived classes is virtually limitless. The CollectEnvironmentData method in the Agent class is declared as virtual and can be overridden. The Ant class overrides the method and calls the parent version before adding its own functionality. The GridData objects contain Tag variables which can be used to store any data required. The ants use it to store a quantity called PheromoneLevel which represents a marker left on the ground by an ant. Ants can add to these values and inspect the level of surrounding squares and use it to decide on behaviour.

Environment Objects and Agent Subtypes

As described above the CrowData file can be configured to allow the user to select various objects in a scene in Maya as specific environmental objects to be used by the simulation. The positions of these objects are imported into the program via the SimSetupData file and they are passed in to the Agent objects on Initialisation. The Agent class ignores them but derived classes are free to keep a record of their positions. The Ant class

takes the first object position and stores it as a Point variable called Nest. Any subsequent objects are stored in an array of Points called Food.

It will often be the case that an animation requires more than one different type of agent in a crowd. To accommodate this any number of types can be configured in the CrowdData file. This information will be provided to the user and the simulation will be told how many of each type to generate. Each agent stores an enumerated type which describes what type of agent it is. These types can be taken into consideration when deciding on behaviour.

The Ant class is setup to provide three different types of ant, Workers, Soldiers and Scouts. The names are really arbitrary but are provided to the CrowdData file to be displayed in the UI. When making decisions, an ant's behaviour will depend on its type. The Workers will simply crawl around but the Soldiers will periodically decide to visit the Nest (if one is setup by the user) using the object stored in initialisation. The Scout type on the other hand will constantly check the position of the Food objects against its current position. If it finds it is on top of one it will return to the nest, dropping pheromones on the way. Scouts are also setup to detect and follow pheromone trails and in this way, lines of marching scout ants are formed between the nest and various food sites in the environment, as shown in an OpenGL visualization in **Figure 14**.

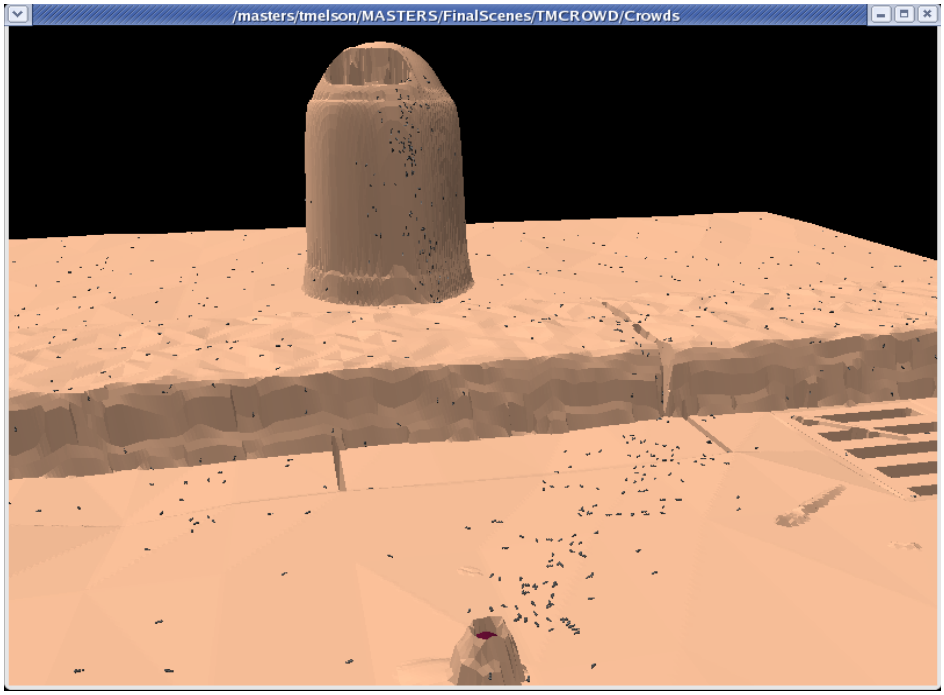


Figure 14. Lines of ants collecting food

RIB Generation

When a render is requested by the user the simulation must produce RIB files for the final animation. In order to achieve this, the program must have access to both the environment and the animated agent RIB files. It is the user's responsibility to provide these files under the TMCROWD directory. SLIM can be used for RIB exporting, complete with render settings and shaders.

Initially the system required that the environment be exported to RIB at each frame, complete with camera animation. This was regarded as inefficient as environment geometry can contain a lot of data and in a large animation this can mean high export time, high import time and high RAM usage. The system was amended to expect one environment RIB file which can be used for each frame. Camera animation is imported separately, one RIB file for each frame with all geometry hidden.

Animated agent RIBs should be provided by the user in a separate directory. A sequence is required for each level of detail and for each action the agent can perform. The program will look for RIB files with the name of the agent provided in the CrowdData file (e.g. Ant) followed by three numbers separated by stops. The first number is the level of detail, the second the action number and the third the frame number. The user is free to decide how many levels of detail to provide. The program will determine this number when reading in the agent RIB files and implement their use accordingly.

The part of the system that decides on the use of different actions is in the Agent class and derived classes in the simulation. The Agent class will always set the action number to one but derived classes can change the number as required. The availability of different animated actions to the system can be used by the simulation in two ways. Firstly when an agent changes its behaviour, for example it stops moving, it might want to change the action from a walk cycle to a stationary one. Secondly agents with different sub types may use different actions, for example two types of soldier may wear different uniforms. The classes also have control over the cycle frame number so if desired the animation can be sped up, paused of synchronized with other actions.

When run, the program will read in the environment RIB and agent RIBs, strip out everything except the geometry and write them back out to file. Then at each step of the simulation the appropriate camera RIB is read in

and a ReadArchive command is added for the environment. Then DelayedReadArchive commands are added for each agent in the system along with appropriate translation commands to position them.

Some renders from RIB files produced with the system for the ant animation are shown below.

Conclusions and Future Work

The development of the crowd simulation pipeline has been very successful. Using the MEL scripts and the C++ program a user can setup a scene in Maya and simulate crowds within it in a user friendly and efficient manner. Simulations can be visualized, rendered and re rendered with different settings quickly and easily, with helpful options to achieve the desired results.

The C++ program handles a large amount of the logistical work by automatically providing the specific crowd details to Maya and constructing the final RIB files with level of detail functionality and archived geometry to maximize efficiency.

The simulation is also very extensible. Through using polymorphism and providing many useful methods within the Agent base class, it is relatively simple to inherit from this class and create complex behaviour tailored to specific animations. The scene objects, agent actions and agent subtypes functionality is all integrated with the Maya interface and can be used within a derived agent class to provide power and flexibility to the programmer.

Future Work

There are several areas in which the system could be extended. The system as it stand has set up an interface between the simulation and Maya using text files. Since the pipeline is modularized in this way it would be relatively easy to provide the same functionality within different 3D packages. A package such as Houdini for example could be setup to provide complex procedural geometry which might make a very interesting environment for crowds.

The way the simulation itself represents the environment could be reexamined, giving more time to the representation of fully three-dimensional models allowing multiple levels and overhanging objects. The model could be stored in an OBJ type format and an octree used to aid in its exploration by agents.

As discussed in the Previous Work section of the paper, the latest crowd simulation tools provide all kinds of functionality. The tools are vast and flexible, providing options for directing certain aspects of a crowd, for

combining animation and motion capture data and to blend animation cycles together. The scope of these kinds of tools is very large but it would be interesting to attempt to add some of these techniques into the crowd pipeline.

The system already allows the export of specific object locations within a scene for the agents to interact with. Extra functionality could be provided to export areas of geometry or curves to be followed by certain agent types. The logic that links particular actions and behaviour to different types of agent could be made open to the user to some degree, giving them more power over the behaviour of a crowd. Some work would be required to expand the user interface as well as the interface with the simulation to transfer the more complex user settings.

References

[1] Crowd Systems at ILM, Christophe Hery, Hiromi Ono, Douglas Sutton, Industrial Light & Magic, April 15, 2004

[2] Craig Reynolds. "Flocks, Herds, and Schools: A Distributed Behavioural Model", *Proceedings SIGGRAPH, 1987* pp 25–34 (1987).

[3] Massive Software, www.massivesoftware.com/

[4] SIGGRAPH2006 Exhibitor Directory,
http://esub.siggraph.org/cgi- bin/cgi/idEDetail.html&CompanyID=884
Accessed on 8th August 2006

[5] MORAN, P., 2004. *VFXTalk.com Interviews Framestore- CFC Compositors & TD's for 'Troy'* Available from:
http://www.vfxtalk.com/forum/showthread.php?t=2523

[6] Wikipedia, http://en.wikipedia.org/wiki/Quadtree
Accessed on 8th August 2006

[7] Craig Reynolds, Crowd Simulation on PS3, Game Developers Conference 2006
available online:
http://www.research.scea.com/research/pdfs/GDC2006ReynoldsTemp.pdf#search=%22Crowd%20Simulation%20on%20PS3%22

Appendix - TMCrowd User Guide

Setup

Copy the following scripts into the scripts directory of the version of Maya in use:

SetupCrowd
OutputCrowd
GetProjPath
GetSelectedObjects
GetSelectedPlane
ExportSelectedAsOBJ
RemoveAgents

Create a subdirectory named TMCROWD in the directory where the Maya .mb file to be used for crowd simulation is located.

Under this directory create the following subdirectories:

Environment
Camera
Agents
Final

Copy the Crowds C++ executable into the TMCROWDS directory.

Agent RIB Files

Create the models of the agents to be added to the scene and animate their cycles. Different models can be used for different agent actions if desired. Also each action will need to be recreated for each level of detail to be used.

The animation of each model should be exported to RIB format (SLIM can be used for this purpose). The name of each RIB should be composed of the agent type followed by three numbers separated by stops. The first number is the level of detail (1 being the most detailed) and is padded with zeros to 2 digits. The second number is the action, padded to 3 digits (walk is 1) and the third is the frame number, padded to 4 digits. An example for the Ant animation is as follows:

Ant.01.002.0005.rib

This is the fifth frame of the highest detail level of the second ant action.

All agent RIB files should be exported to the Agents directory.

Environment RIB Files

A name should be chosen for the current simulation. This name can be anything but will be used by the system to refer to your environment RIB files. From here on this name will be referred to as the SimName.

Open / create the Maya scene where the crowds will be generated. All parts of the scene that will contain crowds must be polygonal geometry. The scene does not need to be fully completed in order to test the addition of crowds, but when a render is to be requested, the scene should obviously be fully modelled, shaded, lit etc.

As the crowd system uses prman, any shaders used must be renderman shaders. SLIM can be used to create defaults or import custom shaders).

The system only needs one environment RIB containing the geometry. This RIB file should be given a name matching the SimName followed by .0001.rib For example the Ant scene environment RIB was named StreetScene.0001.rib

This RIB file should be exported to the Environment directory.

The camera path for the animation should be exported separately, one RIB file for each frame of the animation.

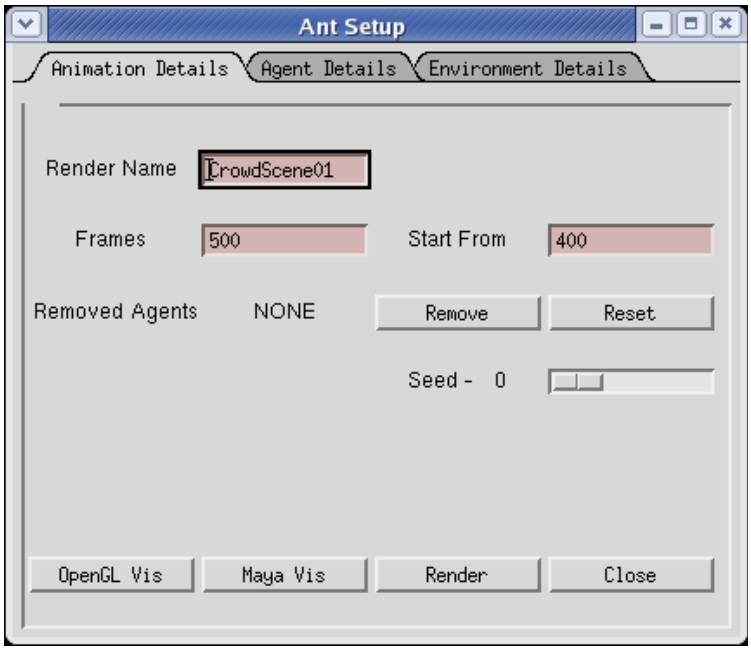
Key frame the path of the camera, along with any other camera settings and then hide all geometry in the scene. Then export all frames to RIB files with the name SimName followed by a stop, then the frame number padded with zeros to 4 digits. For example in the Ant animation the name StreetScene.0010.rib was used for frame ten of the camera RIB files.

These RIB files should be exported to the Camera directory.

User Interface

Open the environment scene and run the SetupCrowd script. The Setup window should appear as shown below:

Animation Details



The window is split into three tabs. The first is the Animation Details tab, shown above and is used to setup details of the simulation.

Enter a name for the simulation in the Render Name field. This should match the SimName mentioned above.

Enter the number of frames in the animation in the Frames field.

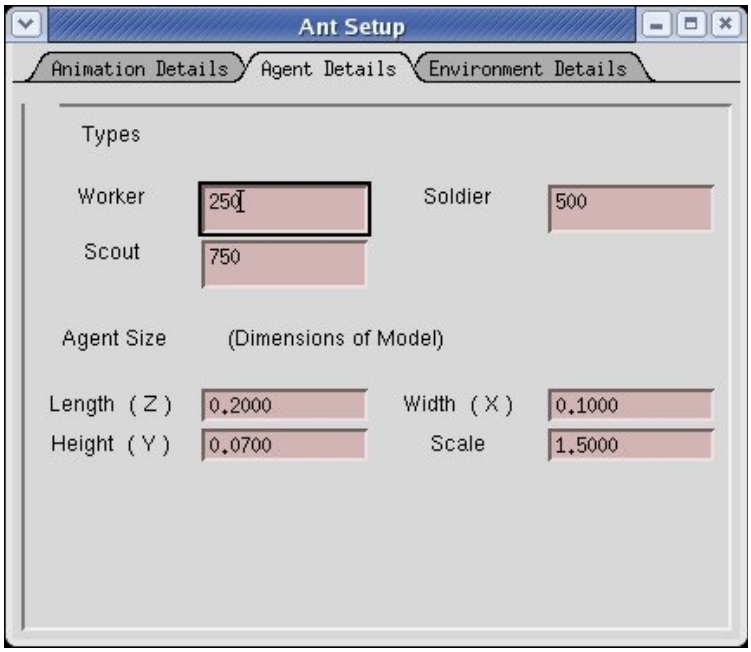
If the simulation does not take the desired shape until a number of steps in, the Start From field can be used to enter a step in the simulation to start from.

When a Maya visualisation has been run, agents can be removed from the simulation by selecting them in the scene and clicking the Remove button. The Reset button will place any removed agents back into the simulation.

The Seed slider can be used to change the specific outcome of the simulation, without changing any input parameters.

Agent Details

The Agent Details tab is shown below:



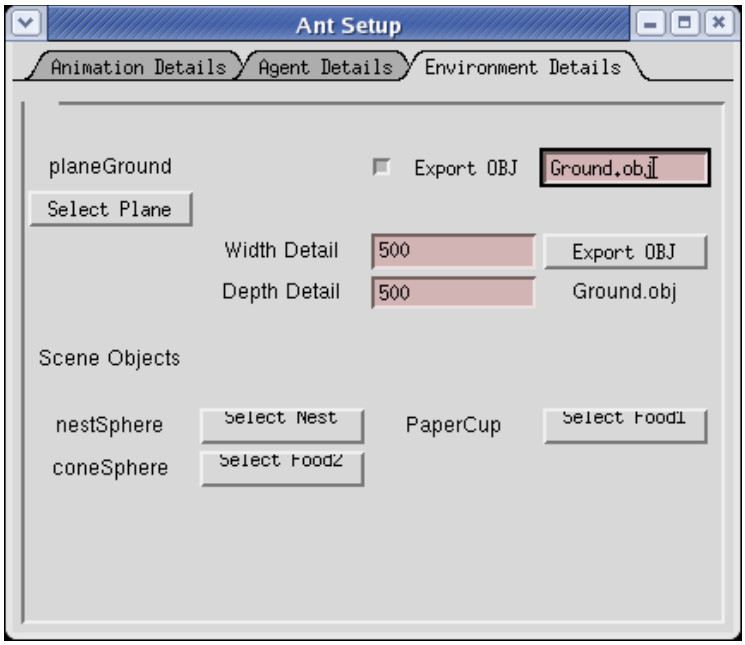
Configure the number of agents in the simulation by entering the number of individual subtypes. The number and names of these fields will depend on the details in the CrowdData file, provided by the C++ program.

Enter the dimensions of the agent models used in the simulation. The values do not have to be absolutely precise but should be fairly accurate.

The Scale field can be used to scale the size of the agents up or down in the simulation.

Environment Details

The Environment Details tab is shown below:



Create a polygonal plane in the scene and position it over the area of the model where the crowd is to be simulated (its Y coordinate is of no consequence). Select the plane and click the Select Plane button.

The plane itself may be modelled as the simulation environment by translating its points up and down. This information will be exported and the crowd generated on the plane.

However it is recommended that an OBJ file is exported instead. To do this, ensure the Export OBJ checkbox is checked and enter a name for the OBJ file.

Use the Width and Depth Detail fields to dictate the detail level to which the environment will be modelled in the simulation (500 is a fairly good values for these).

In the Maya scene select the geometry where the crowd is to be generated. The whole scene need not be selected, just those objects that are underneath the plane created above.

Click the Export OBJ button and the environment will be exported to an OBJ file in the TMCROWD directory.

Select any objects in the Maya scene to be exported to the simulation and click the Scene Object buttons to set them. The number and names of these buttons will depend on the details in the CrowdData file. The

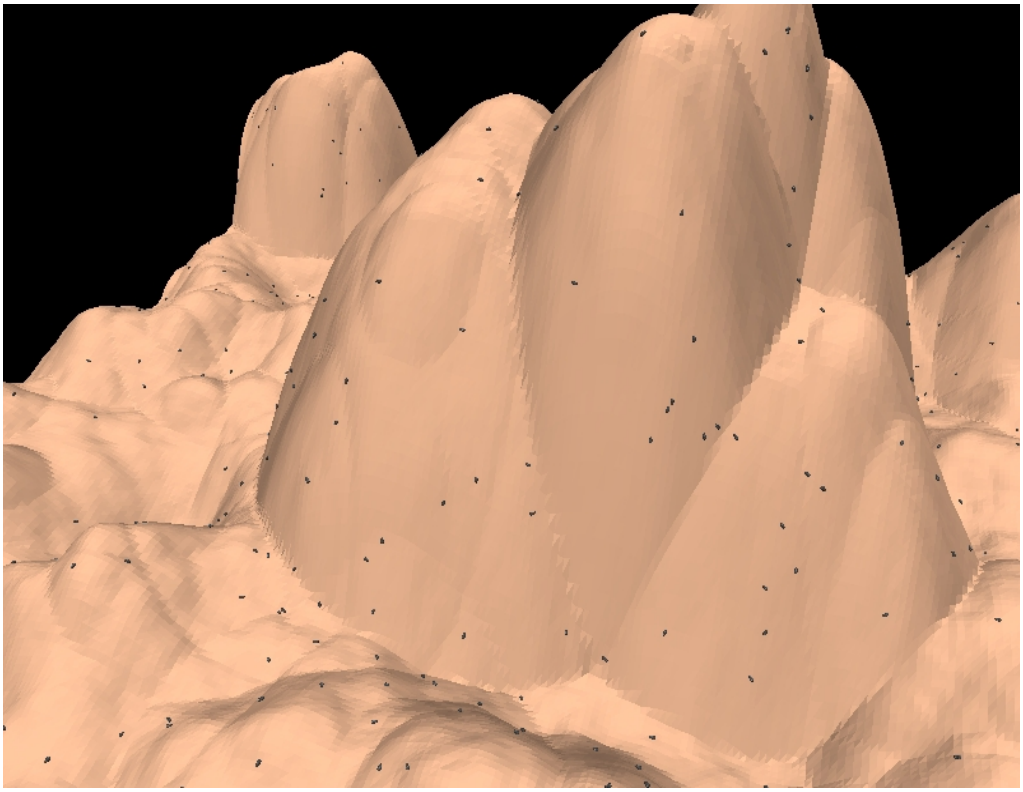
coordinates of these objects will be provided to the simulation and will be used to affect agents' behaviour. It is not enforced that all or any of these objects are exported.

Running the Simulation

Once the simulation details are entered and OBJ file exported, the simulation can be run. There are three options when running the simulation.

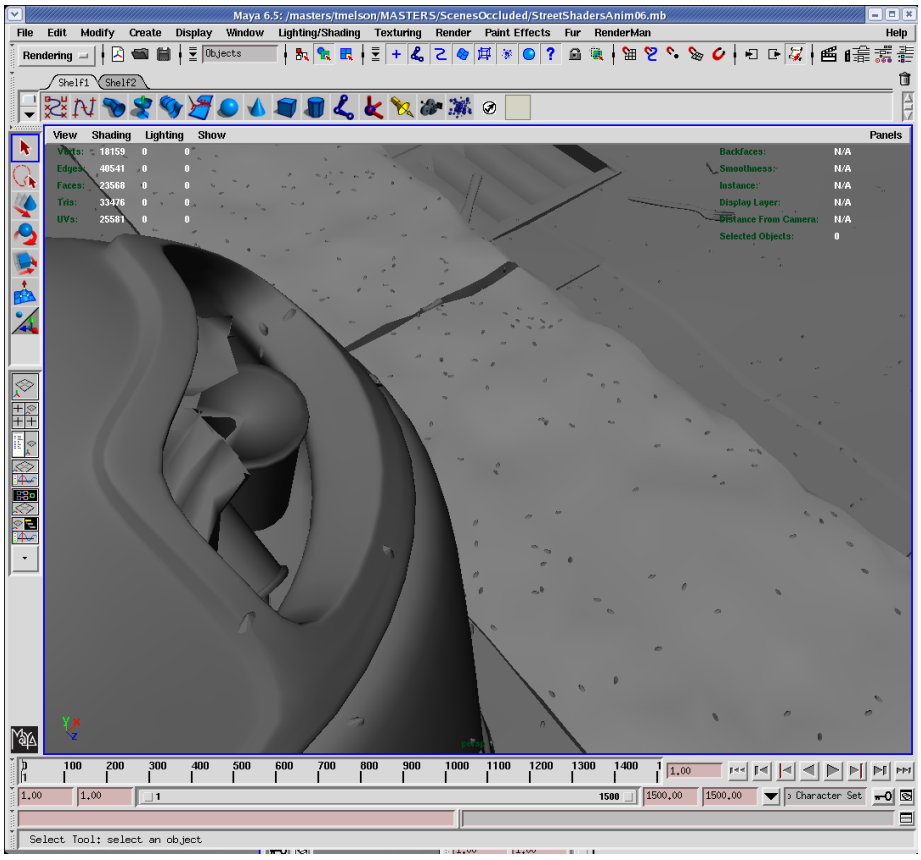
OpenGL Visualisation

An OpenGL visualisation can be requested by selecting the OpenGL Vis button. This is just a quick way to view the results of your settings as a crude representation. An OpenGL simulation view is shown below:



Maya Visualisation

Selecting the Maya Vis button will run the simulation, again showing an OpenGL view whilst it runs. However on completion Maya will import details of the simulation and visualize the results in the scene by creating animated markers that represent the agents. A Maya scene view with agents added is shown below:



Final Render

When the simulation is satisfactory a full render can be requested by selecting the Render button.

The simulation will run showing an OpenGL view. As each step of the simulation is processed, a corresponding final RIB file is and written out to the Final directory