Art Directed Ant Crowd Simulation in Houdini

Tushar Kewlani

Master of Science,

Computer Animation and Visual Effects

Bournemouth University, Talbot Campus

August, 2013

Contents

Table of contents i
Abstract iii
Acknowledgements iv

1 Introduction 1

2 Related work 3
2.1 Crowd Simulation Tools 4
2.1.1 Craig Reynolds Flocking 4
2.1.2 Particle Based Tools 4
2.1.3 Massive 6
2.1.4 MPC Alice- (Artificial LIfe Crowd Engine) 6
2.1.5 Continuum Crowds 8

3 Technical Background 10
3.0.6 Solver Surface Node 11
3.0.7 VEX 11
3.0.8 HOM 12
3.1 Ant Colony Optimization 12

4 Design and Implementation 14
4.1 Introduction 14
4.2 Design 15
4.2.1 Character 15
4.2.3 Brain 17
4.2.4 Finite State Machine 17
4.2.5 Instancing/Rendering 17
4.3 Technical Implementation in Houdini 18

4.3.1 Character 18
4.3.2 Ants 18
4.3.3 SOLVER-AI Brain 19
4.3.4 Behaviours 20
4.3.5 Flow Fields 22
4.3.6 Obstacle Avoidance 23
4.3.7 Mind 25
4.3.8 Stick To Surface 26
4.4 States and State Machine 28
4.4.1 State 1.x-Wander 29
4.4.2 State 2.x-SeekFood 29
4.4.3 State 3.x-Digger 30
4.4.4 State 4.x-seekHome 31
4.4.5 State 5.x-seekPheromones 32
4.5 Geometry Instancing and Rendering 32
4.5.1 Instance 33

5 Applications and results **34**
5.1 Crowd Simulation Results 34
5.2 Pheromone Path Following Results 36

6 Conclusion **45**
6.1 Summary 45
6.2 Future work 46

References **48**

Abstract

This project presents a crowd simulation system in Houdini for use in the computer animation industry. Crowd simulation algorithms such as ant colony optimization allows for building complex simulations without overly complicated individual agents. The aim of this thesis is to design a modular agent based crowd solution that could be used to build a variety of swarm AI simulations.

The pheromone based ant foraging behaviour and other states like digging are implemented to analyse the solution developed against this behaviour model. Finally a render time Delayed Geometry Loader is used to render the crowd in an efficient manner.

Acknowledgements

I would like to thank Jon Macey and Zhidong Xiao for their feedback and guidance through the project. Peter Claes for suggesting the initial ideas and suggestions that led to the thesis. Finally I would like to thank all my professors and classmates in the MSc.

Chapter 1

Introduction

Crowd simulation for film and animation has become demanding of large scale photo-real solutions. Standalone softwares such as Massive have been extremely successful at creating crowd simulation systems that create impressive results. But the proprietary nature of these solutions and the need to export custom rigs and other data through the pipeline reduces the flexibility and can increase turn around time in projects. Also since the development of such a software is run by a separate company sometimes substantial internal development has to be done to use them for custom effects.

Hence a lot of companies build there own crowd simulation solutions. DNEG, MPC, ILM all use their own software for crowd simulations as they offer greater flexibility and even license costs can be a big factor for these decisions(Bielik 2004).

This thesis explores the option of building such a system inside of Houdini. The system developed will aim to build a pheromone based ant foraging solution using a Houdini pipeline. Keeping the engine inside Houdini offers various advantages such as attributes and agents can be modified on the fly. Geometry and attributes can be visualized inside Houdini for quick visual debugging. We will demonstrate that the system developed utilizes the algorithms used for ant foraging behaviour. Also, the final simulation produces results that can be used in a film or animation sequence. The behaviours can be art directed to follow paths which can be generated and added by artists using the digital assets created.

The development has been carried out in Houdini 12.5.316.22 on linux-x86$_6$4 $- gcc4.4platform$.

Chapter 2

Related work

Figure 2.1: *Climbing Zombie pyramid from WWZ by MPC in ALICE*

Research on crowd simulations has been of interest since the late 19th Century. Although researchers from various fields like architecture, Computer Graphics, physics etc. have been working on crowd simulations, exchange of information and ideas in this field has remained limited. A reason for this could be because most of the behaviour modelling techniques used are application specific.(Thalmann et al. 2004)

Still crowd simulation can be broadly categorized into two separate fields. One field being scientific and technology related areas where realism and quantitative results are the focus of work. These can include evacuation simulations, crowd flow bottlenecks, traffic planning etc. The results of the simulation are paramount here and not the visual appeal. The visualizations are carried out with the sole purpose of validating findings.

Another area of research is crowd simulations for the entertainment industry. With the demand for greater scale and visual realism in films and computer games crowd simulations have become an important area of development. The focus here is on visual realism and aesthetics rather than computational validity. Since the area of research for this thesis is crowd simulations for VFX, we will limit ourselves in the discussion to the second category of research.

2.1 Crowd Simulation Tools

2.1.1 Craig Reynolds Flocking

(Reynolds 1987) published his paper on modelling emergent behaviours based on dense interaction of relatively simple steering behaviours. In this system each agent was called a boid. Although each boid had access to all of the flock, the motion of the boid would be dictated by only a few of its immediate neighbours. By providing simple behaviour algorithms like separation, alignment and cohesion each boid would be a self propelling character with some apparently simple decision making ability. The motion of these boids would give an appearance of a brain.

2.1.2 Particle Based Tools

ILM create a particle based simulation tool for ”Star Wars ep1: THe Phantom Menace” . They used maya’s particle system to create the global motion of the particles based on a set of attributes the particles had. These attributes can be defined as Position, orientation, mass etc. Once the global motion of the particle has been determined an action from a collection of clips can be used on an instanced geometry on the particle. By using a random collection of geometry and texture files with varied animation cycles it is possible to create a vast array of crowd.

PDI also created a crowd system for use in the movie "Shrek 2" called Mob. The mob system allows a large crowd to be built by combining a small number of animations with small numbers of character types, body types, hair styles, clothing styles, hat styles, materials, colors, behaviours, and so on, plus customizations for final tweaking. Assignment of these features, as well as placement of the characters and their direction of motion, can be controlled and simultaneously randomized(Thalmann et al. 2004).

The mob system had a motion simulator and a rendering simulator. They also used a cycle based animation system like ILM. To meet the additional challenges during the making of "Madagascar", PDI/Dreamworks update the system to handle high level behaviours. This included providing a look at control for the animators to place. The mob system would handle the rest of the simulation(Kermel 2005).

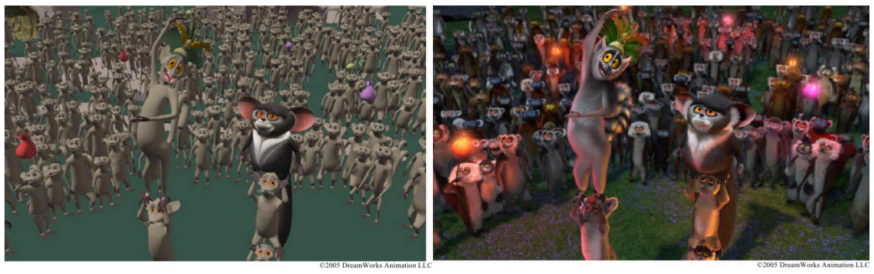

©2005 DreamWorks Animation LLC
©2005 DreamWorks Animation LLC

Figure 2.2: *Madagascar crowd with High level lookat behaviour for the foreground characters*

Double Negative also built a crowd simulation solution based on particles for their work in "Angels and Demons". By establishing a set of rules, the particles were run through a simulation to generate their motion, followed by motion capture clips to create characters. CHOPS in houdini was used to blend between multiple clips.(UKScreen 2013)

2.1.3 Massive

Massive is probably the most popular tool used for crowd simulation in VFX. Initially developed at Weta digital for the battle sequences in the Lord of the Ring Trilogy (Thalmann et al. 2004) it has since become a commercially available software.

Massive uses a brain that employs a set of fuzzy rules for each agent to determine what action it should perform next based on its immediate environment. A graph of interconnected nodes input data to the logic nodes(AND,OR,NOT) and the logic nodes using a set of fuzzy rules determine the movement of the agent. A typical brain of a warrior in lord of the rings had 6000 to 9000 nodes equating to some 2000-3000 rules per agent(Griggs 2003).

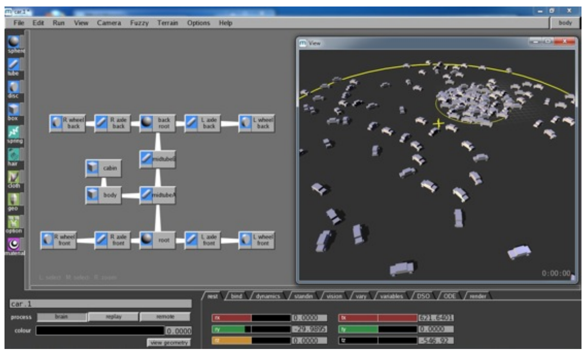


Figure 2.3: *Crowd Simulation in Massive,*

Each agent also has a motion tree, which is a set of actions that it can perform. Based on the input from the fuzzy rules an action is executed by the agent.

Massive also provides high level control of the crowds with agent placement tools, flow field editors, agent variation etc.(Thalmann et al. 2004)

2.1.4 MPC Alice- (Artificial LIfe Crowd Engine)

MPC has a properietery crowd simulation system called ALICE. Originally developed for the movie Troy, ALICE has been used on some complex crowd simulations in films. Lately in "World War Z" it was used to build crowds of zombies climbing on each other to form pyramids.

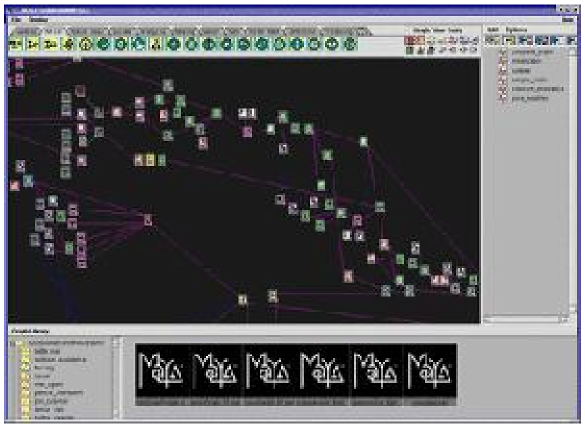


Figure 2.4: *Crowd Simulation in Maya,(Kolve 2004)*

ALICE originally built in maya has now moved away from its dependence on Maya, while still providing with a MEL bridge into maya. ALICE gives the artists the ability to connect node graphs to build complex behavioural states, which in turn drive motion clips to create animated characters on screen(Pieke 2008).

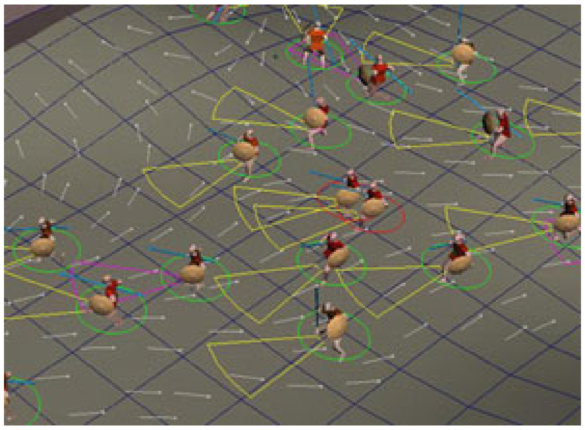


Figure 2.5: *Crowd Simulation in Maya, (Kolve 2004)*

Some of the crowd control networks built into ALICE include, (fuzzy) logic, state machices, flow control etc. Alice has also been integrated into maya's particle system for generating effects such as arrow impacts, dust kicks, blood splashes etc(Kolve). MPC has also integrated ALICE with their proprietery physics engine PAPI, to allow for transitions from motion clips to ragdoll dynamics.

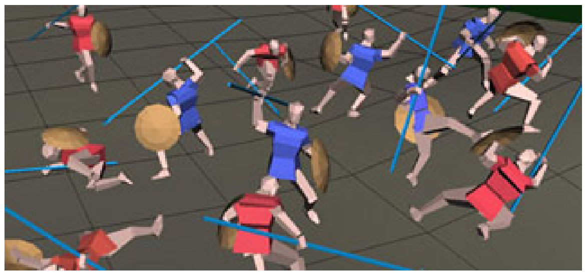


Figure 2.6: *Battle crowd in Alexander, (Kolve 2004)*

For rendering the agents, MPC used a combination of C++ DSO's and Lua scripts to control the renders. This work-flow allowed them to quickly script new features based on the changing requirements of the production(Haddon & Griffiths 2006)

2.1.5 Continuum Crowds

Another very interesting approach to crowd simulations is to drive the motion of the particles based on fluid dynamics. In this system a dynamic potential field is calculated per timestep. Simulations created using this system run at interactive frame rates for millions of particles and exhibit emergent phenomenon as well. In this system a potential function guides the particles to their goal with the use of discomfort fields to handle geographical preferences(Treuille et al. 2006)

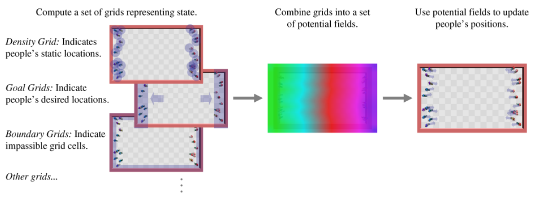

Compute a set of grids representing state.
Density Grid: Indicates people's static locations.
Goal Grids: Indicate people's desired locations.
Boundary Grids: Indicate impassible grid cells.
Other grids...
Combine grids into a set of potential fields.
Use potential fields to update people's positions.

Figure 2.7: *General Algorithm Overview(Treuille et al. 2006)*

So a slope with a higher gradient has a lower potential field thereby making sure the agent would prefer a route where the slope is lesser, hence simulating some intelligence in the particles.

Although a very efficient method for simulating a large number of particle based crowds, there is not enough control over the parameters of the simulation to make it useful for the computer animation industry. Though some aspects of this system could be adapted in a Houdini pipeline to create large crowds for wide shots. By generating flow fields for terrain using a cost potential system would allow us to avoid complex obstacle avoidance and even guide the agents around obstacles in a smooth manner.

Chapter 3

Technical Background

Houdini's open node based workflow and 'open' access to data allows a high level of flexibility and control for complex simulation work. Since the crowd simulation system developed here is primarily aimed at computer animation, Houdini provides a perfect platform for development.

Houdini provides built in point cloud architecture based on KD-trees. This allows for extremely fast lookups for neighbours. In combination with VEX and VOPS this point cloud functionality provides a robust and efficient framework for point cloud iterations. Houdini provides two separate areas for development of dynamic simulation. One is the POPS context which is essentially a particle simulation environment. The other is DOPS which is Houdini's dynamic simulation environment.

Although POPS has some powerful built in functionalities like state and event for building behaviours, there are certain limitations to working in the POP context. The simulation needs to be played from the first frame and you cannot skip ahead in the timeline.

In addition the access to point attributes and SOP geometry operators is limited. Houdini has also revamped its SOP context in Houdini 2012 with an order of magnitude jump in performance. Hence SOP's were chosen to create the agent simulation.

3.0.6 Solver Surface Node

A SOP solver allows the DOP simulation to use a chain of SOP's to evolve over time. This allows the user to access the simulation from the previous time step as a starting point for the next time step. This can be used to generate effects that evolve from oner frame to another like automata or simulation. (Side Effects Software 2013)

Usually in SOPS time dependent effects are implemented by using variables like $F to create a new seed value every frame. This results in effects that are independent of the previous frame and create random looking results every frame. Whereas by using a solver node network the effect takes into account the previous frame and appears to evolve over time.

If you jump into the solver node, it is essentially a node made up of using subnet–DOPNet–SOPsolver. Inside the solver node there are multiple input nodes. $Prev_Node$ is cooked every frame with the output of this node containing the geometry from the previous frame. The node with the "Display Flag" turned on is used as an input for the next frame. The $Input_1$ node contains the original geometry fed into the node. This is useful in case the rest position is of use in the simulation.

3.0.7 VEX

VEX is the vector expression language in Houdini. Initially implemented as a shading language its role has been expanded to create new custom operators in COPS, POPS, SOPS and CHOPS. VEX is based on the C language, and provides performance similar to compiled C/C++ code. VEX is also multi-threaded. Any nodes written in VEX will benefit from this feature as well. In many cases VEX will outperform Houdini nodes written in C++.(Side Effects Software 2013)

Houdini's more visual programming paradigm of VOPS is based on VEX code. All VOPS essentially have VEX code running underneath the hood. Although VEX code is pretty efficient at iterating point data, there are other limitations to the language.

VEX cannot generate points. It can manipulate and create attributes on pre-existing points.

Most of the behaviours which utilize iterating through the the agents and nearest neighbour search have been written in VEX for these reasons.

3.0.8 HOM

HOM or Houdini object model is an API that allows you to access information in Houdini using the python scripting language. HOM has access to point, primitive and detail/Global attribute in houdini unlike VEX. Also, new points can be created in houdini using python unlike VEX.

Since HOM does not have access to the point cloud functionality of houdini, nearest neighbour searches are a big bottleneck on the pipeline. Although using python would make some of the design choices simpler in the future due to the extensive access to all the attributes in Houdini, this was considered to be a big disadvantage and early in the project it was decided to avoid using python for any functionality that required iterations across the agents.

3.1 Ant Colony Optimization

Multi-Agent research has taken inspiration from social insects such as Ants. According to Luke and Panait(2004). A very large number of agents will yield sophisticated collective action by following some relatively simple behaviours.

Ants use Pheromones for a variety of purposes to communicate with each other. If an ant finds a food source it will drop pheromones around the food and also on the path back to the nest. Other ants can detect this pheromone and follow it back to the food source based on the strength of the scent. If the food source is finished then they don't drop any more pheromones. As the pheromones vaporize the ants seek other sources of food nearby. Over a period of time ants choose the shortest path as the pheromone being deposited is strongest on these trails.

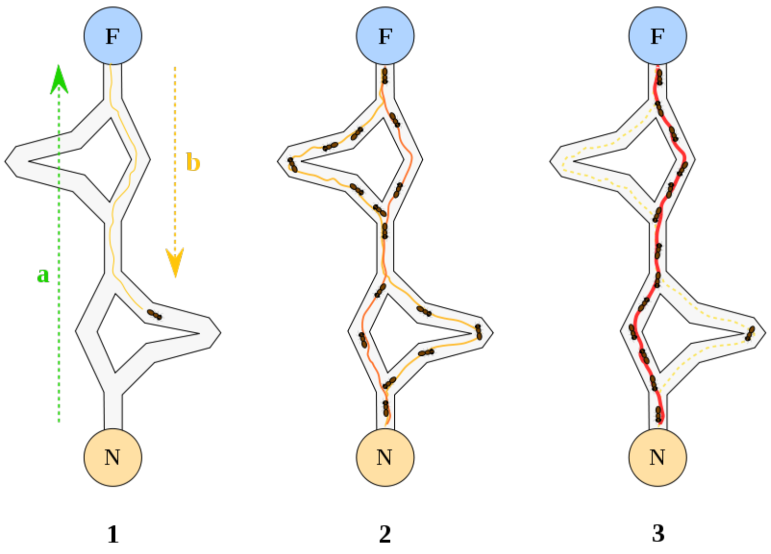


Figure 3.1: *Shortest route in ACO, (Dro 2006)*

Pheromones can have other utilities in an ant colony as well. They can be used for finding home and threat perception as well. When under threat an ant will emit pheromones notifying the ants nearby causing a chain reaction that allows the whole colony to be aware of the threat.

The ant colony optimization is widely used to solve optimization and distribution problems in networking. One of the more successful uses is in solving the Travelling Salesman Phenomenon(TSP).

In the simulation the pheromones are used for pathfinding. The use of pheromones helps reduce the complexity of individual agents as a mechanism for inter-agent communication(Panait & Luke 2004).

Chapter 4

Design and Implementation

4.1 Introduction

Agent based simulations have been used in VFX for some time now and provide a great result. Creatively being able to direct a simulation as per the director's vision is one of the main requirements of any simulator for the entertainment industry. But having said that all simulations must be based on reality. Nature provides some extremely interesting emergent behaviour that is useful for modelling complex behaviours in a multi-agent system

The purpose behind creating a multi-agent system for this thesis was to make a system that would be based on practical algorithms for emergent behaviour. It should be possible to implement accurate mathematical behaviour models in the system which are quantifiable. As a mean of evaluation, it was chosen to model the pheromone based ant foraging behaviour. Though this was the model created, it should be noted that the same system can be adapted for other behaviour models as well. This might be battle scenarios, crowd evacuation procedures etc. We will discuss this further in the next section when we discuss the design of the program. The final aim of the project was to develop the simulation with a strong bend to realism. The agents should transition smoothly from one state to another and their locomotion must be believable.

Houdini was chosen as the development environment for this system as it has an open node based architecture, but also manages SIMD multi-threaded evaluation very efficiently. This makes it ideal for point-cloud based distance calculations etc. Also due to its very open programmable expression language VEX and Python scripting integration it allowed for implementing custom algorithms.

4.2 Design

Since the plan was to have a system that was extendible in the future, at the design phase the project was broken into several modular steps that would communicate with each other. The program was broken into a set of classes that could be replaced in the future with a separate feature, without affecting how the other modules performed.

The basic design paradigm for the program.

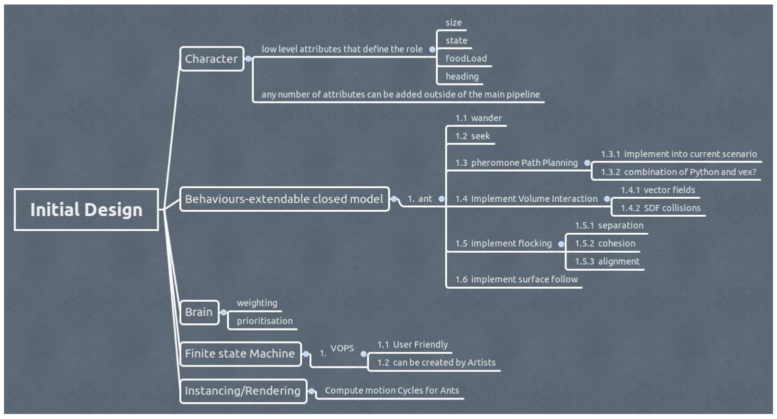

Initial Design
Character
low level attributes that define the role
size
state
foodLoad
heading
any number of attributes can be added outside of the main pipeline
Behaviours-extendable closed model
1. ant
1.1 wander
1.2 seek
1.3 pheromone Path Planning
1.3.1 implement into current scenario
1.3.2 combination of Python and vex?
1.4 Implement Volume Interaction
1.4.1 vector fields
1.4.2 SDF collisions
1.5 implement flocking
1.5.1 separation
1.5.2 cohesion
1.5.3 alignment
1.6 implement surface follow
Brain
weighting
prioritisation
Finite state Machine
1. VOPS
1.1 User Friendly
1.2 can be created by Artists
Instancing/Rendering
Compute motion Cycles for Ants

Figure 4.1: *Initial Design*

4.2.1 Character

This is where all the characteristics of an agent are defined. Attributes like size, velocity, force, age, specie can be added here. This section would control any generic AI features that the character would demonstrate before entering the simulation.

Each agent has a random velocity in the zx axis assigned to it. We also maintain an up vector and a forward vector at all times through the simulation. This is essential to make sure the orientation of the agent is always in the direction of velocity and the normal of the surface its travelling on. With these two axis always available calculating a third axis becomes a trivial matter of a cross product. Each agent is assigned a random mass as well to ensure variation in the crowd. Houdini assigns a point number to all points in the scene. These can be accessed at any time during the simulation, but due to the constantly evolving point counts across the nodes, and the fact that Houdini does not maintain a consistent point number it becomes extremely difficult to create or access attributes consistently across the simulation. Also, a pretty useful feature would be to just remove any point that is not required in the simulation due to technical or aesthetic reasons. Hence the point number of all agents at birth is saved as a separate attribute

This is the class where all the behaviours are modelled. A developer would program behaviours of any level of complexity. These behaviours are saved in a digital asset as a collection of geometry nodes in either VEX or python. Although the behaviour algorithms are quite complex and would have to be written by a programmer, TD's or artists can use these pre-programmed nodes. Theoretically a tree of connected nodes should allow an artist to model any behaviour required.

The behaviour section is a layered system. There are certain state specific behaviours which form the base layer of the simulation. These can be pheromone seek, wander etc. and would be considered local. The next layer is the more generic behaviours that are applicable globally to the whole simulation. These would be separation, cohesion, obstacle avoidance. Also it must be pointed out here that behaviours themselves are independent of such characterization and can be either local or global based on their usage.

4.2.3 Brain

This is the mind of the system where all the behaviours are calculated. A priority sorting model has been used along with the force weighting method. This allows for some additional flexibility with force weighting as the agents are not totally dependent on the forces and not all forces are constantly fighting to control the locomotion of the agent.

4.2.4 Finite State Machine

This is where the state changes of the agent are defined. Particular effort has been made to keep this module in the VOPS functionality of Houdini. Since this is a visual programming model even artists not very familiar can build or modify state machines. This is the area of the system that would need the maximum tweaking at a creative level and hence the decision to keep this in VOPS.

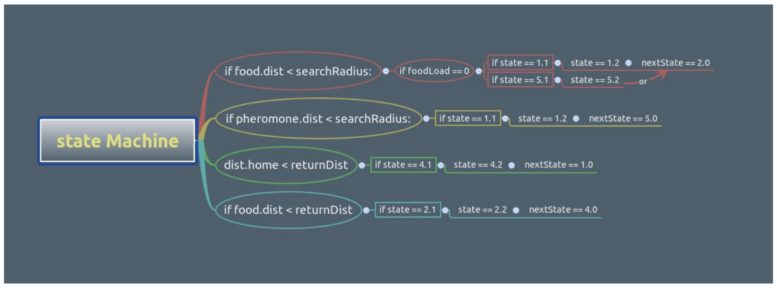


Figure 4.2: *Initial State Machine Design*

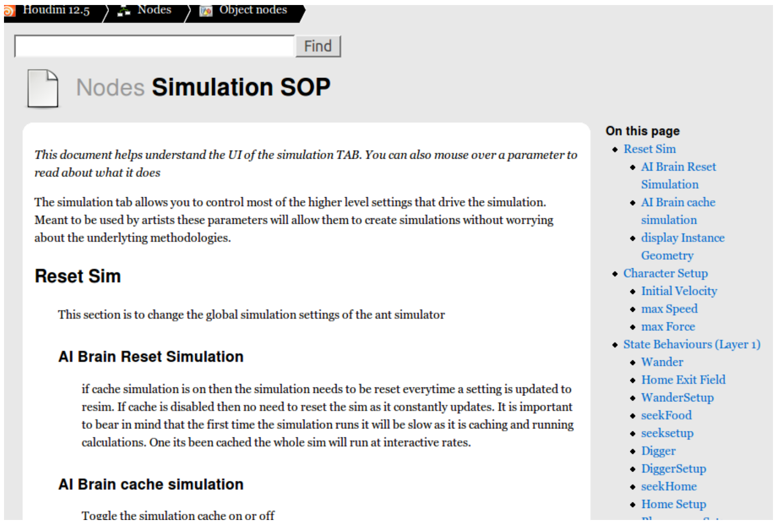
4.2.5 Instancing/Rendering

This module manages the stamping of geometry on the simulated points. Pre-cached simulations are passed into this area of the system and animation cycles are applied to the geometry. To reduce the memory footprint and speed up the work-flow point instancing is used. The animated mesh is cached to the disk. This mesh is then read back and the speed of playback is managed based on the velocity of the points.

4.3 Technical Implementation in Houdini

This section will detail the actual code and algorithm implementation in Houdini.

The actual simulation environment for the crowd has been converted to a digital asset. Since the behaviour nodes are all designed as digital assets, The system can be managed without an external hi-level UI pretty seamlessly. There is also a help card that has been generated for this parent asset which explains all the parameters in the UI.



Nodes **Simulation SOP**

This document helps understand the UI of the simulation TAB. You can also mouse over a parameter to read about what it does

The simulation tab allows you to control most of the higher level settings that drive the simulation. Meant to be used by artists these parameters will allow them to create simulations without worrying about the underlyting methodologies.

Reset Sim

This section is to change the global simulation settings of the ant simulator

AI Brain Reset Simulation

if cache simulation is on then the simulation needs to be reset everytime a setting is updated to resim. If cache is disabled then no need to reset the sim as it constantly updates. It is important to bear in mind that the first time the simulation runs it will be slow as it is caching and running calculations. One its been cached the whole sim will run at interactive rates.

AI Brain cache simulation

Toggle the simulation cache on or off

On this page

- Reset Sim
  - AI Brain Reset Simulation
  - AI Brain cache simulation
  - display Instance Geometry
- Character Setup
  - Initial Velocity
  - max Speed
  - max Force
- State Behaviours (Layer 1)
  - Wander
  - Home Exit Field
  - WanderSetup
  - seekFood
  - seeksetup
  - Digger
  - DiggerSetup
  - seekHome
  - Home Setup

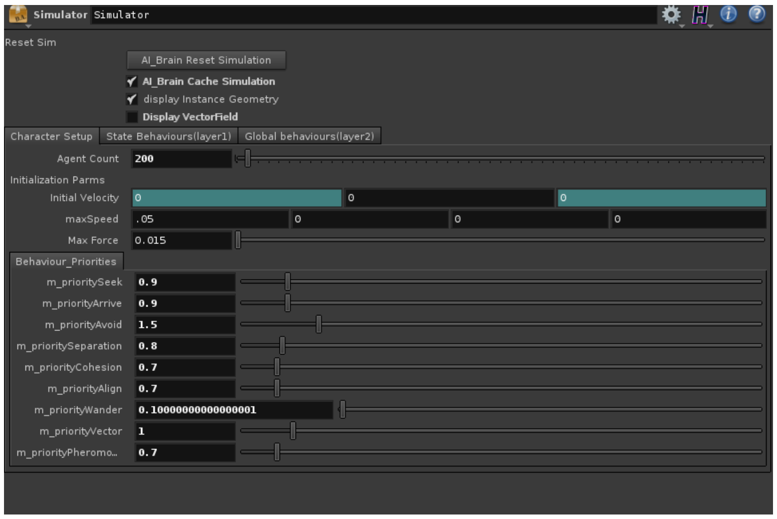
Figure 4.3: *Help Card for Digital Asset*

4.3.1 Character

This node contains all the attributes that define the generic behaviour of the agents. These are the attributes that would be common to any crowd simulation irrespective of the species being simulated. The UI allows the user to customize most of the attributes.

4.3.2 Ants

Inside this node all the AI crowd is simulated. Since ants are terrain based creatures a ray SOP is used to place the agents on the terrain



[✓] AI_Brain Cache Simulation
[✓] display Instance Geometry
[ ] Display VectorField

```
Agent Count            200
Initial Velocity       0      0      0
maxSpeed               .05    0      0      0
Max Force              0.015
m_prioritySeek         0.9
m_priorityArrive       0.9
m_priorityAvoid        1.5
m_prioritySeparation   0.8
m_priorityCohesion     0.7
m_priorityAlign        0.7
m_priorityWander       0.10000000000000001
m_priorityVector       1
m_priorityPheromo...   0.7
```

Figure 4.4: *Character Generation UI*

before being added to the simulation. An attributecreate node is used to add any ants specie specific attributes to the simulation. These are the specie specific simulation parameters that help define the behaviour of a specific specie. All pheromone and food related attributes are added at this stage.

4.3.3 SOLVER-AI Brain

This is the solver node explained in the Technical Background section. This is the heart of the engine that is updated on a per frame basis to create a dynamic simulation. For more details look through the Technical Background section of the report.

One of the challenges of using the solver node over POPS is that all points that are a part of the simulation will be carried through the simulation. Hence any additional points that need to be added to the simulation for calculation are grouped and merged with the agent points and fed into the simulation. Once inside the Solver these are then separated out at the top of the simulation tree and each set of points is only used where required. For example the SDF volume and the pheromone points are only required at the early stage of the simulation where local behaviours as per the state machine are simulated and at the end when the state machine is updated. Hence there is no reason to keep them as a part of the data stream when global behaviours such as separation and obstacle avoidance is calculated. This only reduces the efficiency of the whole program and makes the data cumbersome to manage. By splitting the points into separate data streams a remarkable increase in efficiency was observed.

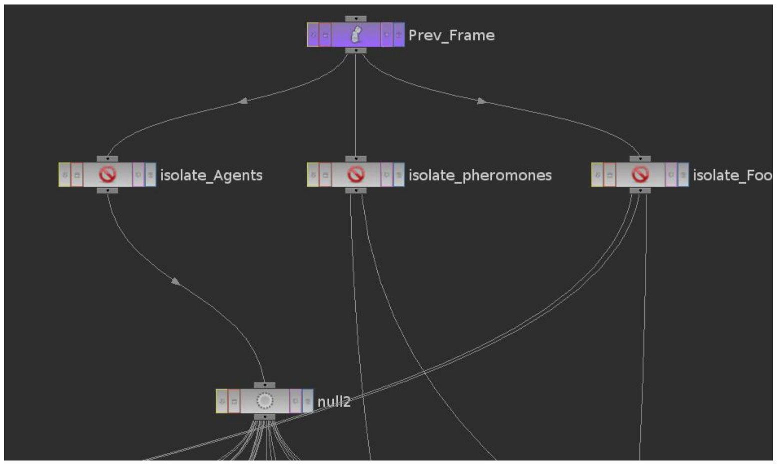


Figure 4.5: *Split Data Stream*

4.3.4 Behaviours

The first layer of behaviours are programmed according to Reynolds(1999). Some of the behaviour algorithms implemented in the crowd digital asset include Arrive, pursuit, seek, separation, cohesion and alignment. The algorithm were derived from Reynolds flocking system.

The nearest neighbour search makes use of the point cloud functionality in Houdini and is extremely efficient. As mentioned earlier VEX was used primarily for these behaviours as they handle SIMD extremely well.

Some key behaviours are described below.

Wander

The wander function projects a circle ahead of the agent. On each iteration a random point is chosen on the perimeter of the circle and a force is exerted on the agent to move towards the point.

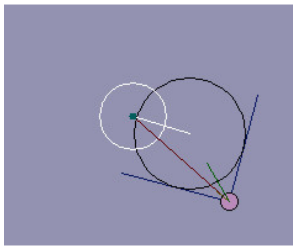


Figure 4.6: *Wander, (Reynolds 1999)*

Seek

Seek is primarily used in the simulation for tracing the path back to the home or finding the food source.

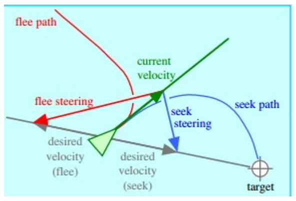


Figure 4.7: *Seek Flee, (Reynolds 1999)*

The seek behaviour implemented in this project has two options. One is to seek a single point irrespective of the distance of the point from the agent. This is particularly useful when a permanent steering behaviour is to be applied in a certain state. The other option is to use multiple points(point Cloud) for the seek behaviour. The agents only seek the point if they are within a certain user specified distance from the point.

Offset Pursuit

A simple pursuit example would be to seek an animated target. But a smarter looking solution is to look ahead in time based on the current velocity of the target and pursue a position that predicts the location of the target at that time in the future. Also if the target is directly in front of the agent then the agent should seek out the target else it should pursue the target with a force that is proportional to the distance between the two but inversely proportional to the speed of the agent and the target(Buckland 2005)

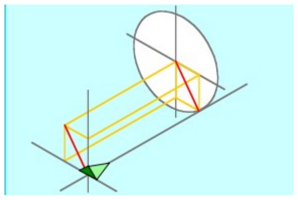


Figure 4.8: *Seek Flee, Reynolds(1999)*

4.3.5 Flow Fields

To be able to art direct the motion of the ants, flow fields were implemented into the behaviour module. Reynolds(1999) considered them particularly valuable as they provide the ability for artists to customize the movement of the characters. The utility digital asset provides the user with the ability to convert any curve into force fields. Custom force fields can also be generated in Houdini by painting or combing normals.

Force fields have been used in the current project to lead the ants into the hill and to keep them inside the terrain boundaries. This is a particularly useful method as this is a completely procedural way of setting the boundary and can be modified on the fly based on requirements.

The force fields can be generated in two separate manners. One being one being parallel to the flow of the curve or look at a target and point either towards or away from it.

Create Ray Vectors PseudoCode

```
• input curve
• resample curve to add evenly spaced points
• add normals attribute N
```

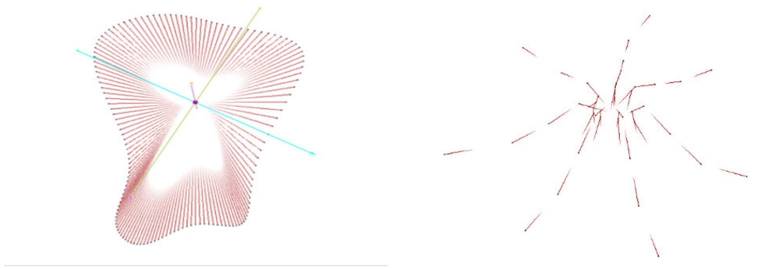


Figure 4.9: *Flow Field Target(left), Tangent(Right)*

```
• for each point i in curve compute $P_i - P_{i+1} = \mathtt{VN} = normalize\mathtt{V}$.
```

A more detailed explanation of some of the custom behaviours is provided in the relevant state machine descriptions in the sections ahead.

4.3.6 Obstacle Avoidance

Collision Detection To avoid interpenetration of agents with the environment all the agents check for collision with any obstacles in their path. The algorithm used for checking for collisions with the obstacles is a ray-triangle collision. Houdini provides an intersect function in vex that returns the (u,v) and world position co-ordinates of a ray intersection with a polygon.

For each agent three rays are calculated based on the velocity of the agent. The first ray is parallel to the direction of travel i.e. straight ahead. Two other vectors are created based on the first vector. Both these vectors are rotated around the up vector of the agent. These are essentially side vectors that are used to check for lateral proximity to the obstacles.

Create Ray Vectors PseudoCode

```
• create ray vector from agent velocity
• create a 3x3 matrix
• transform matrix by $\Theta$ angle(radians) along axis ‘Up Vector‘
```

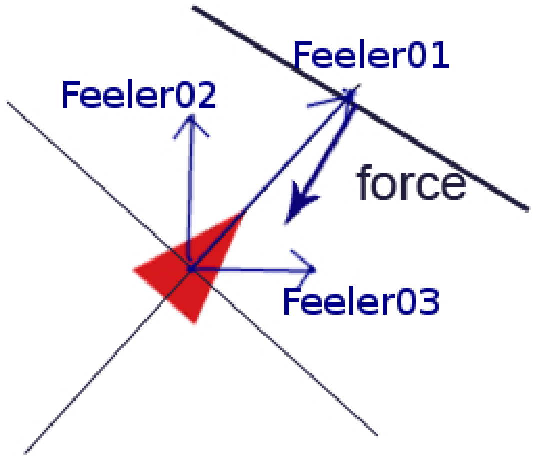


Figure 4.10: *Wall Avoidance setup*

- *`multiply the second ray vector with the matrix to rotate the vector`*
- *`transform matrix by angle(radians) along axis ‘Up Vector‘`*
- *`multiply the second ray vector with the matrix to rotate the vector`*

Collision Response Based on the (u,v) position of the hit on the polygon we can retrieve the normal of the polygon with which the collision occurred. The agent is pushed away from the obstacle based on the length of the ray penetration into the polygon. This ensures the force is inversely proportional to the distance between the obstacle and the agent.

Collision Detection and Implementation Pseudo-Code

- *`detect collision ray-primitive intersection.`*
- *`retrieve the co-ordinates of the collision.`*
- *`detect length of penetration into the polygon`*
- *`retrieve face normal at point of collision`*
- *`multiply normal with a constant and divide by length`*

- *`multiply the second ray vector with the matrix to rotate the vector`*

Although computationally a bit more expensive the additional ray vectors ensure that the agents always maintain a distance from the obstacle even at sharp angles. If not used the collision response fail if the obstacle has sharp edges. Especially with longer agents such as ants, as they move around an obstacle their rear end can turn into an object.

4.3.7 Mind

The mind is the brain of the crowd simulation engine developed here. This is where all the forces are combined to move the agent in a particular direction.

Instead of using a simple weighted sum method where all the forces are randomly truncated based on the behaviour needed, this crowd engine implements a more refined priority based system for calculating the final force.

All the forces and priorities are added to an array. The forces are sorted based on the priority defined by the user using a sorting algorithm.

Based on the order we cycle over each of the forces in the array. Before adding an individual steer force to the final steering force, we check if the current steering force has exceeded the maximum possible force set by the user as an agent characteristic. If the length of the steering force does not exceed the length of the maximum force, we add a new steer to the final steering force.

The final steering force is used to calculate the acceleration based on the Newtonian formula:

$$\vec{a} = \vec{f}/\mathrm{m}$$

Based on the acceleration we append to the current velocity of the agent using:

$$\vec{v} \mathrel{+}= \vec{a} * \mathrm{t} \ \textit{where t is the timestep}$$

Instead of offsetting the position based on velocity in this function we update the $\vec{v}$ of the agent on this frame and leave the position to the stick on surface function.

4.3.8 Stick To Surface

Since this simulation was required to implement a terrain based specie, a stick to surface node was implemented in Houdini.

Initial research presented with a few methods of implementing this feature. One was to project a grid from the top on the environment in the negative Y axis and use this grid as a ground plane. Each time step the agent would compute its new position in xz axis and the Y axis would be determined by the point of intersection of the vector from its previous position to its current position with the surface if the projected grid. One disadvantage of this method was that it required the scene geometry to be conducive to a projection. It would require a substantial set-up time for artists to create a version of the environment where this would be possible. Also if the agent has terrain based locomotion then we limit the y position of the agent to the previously calculated position of the current Y position of the terrain.Also since there can be only one Y position of the grid it is not possible to use this method with complex geometry shapes which overlap each other. Thirdly the agents would not walk on inverted geometry like overhanging cliffs or the down facing geometry of a bottle or coke can.

Another method is to use a displacement map for the ground and have the agents offset in Y based on the value of the displacement map. But this was an even more limiting option and not accurate or feasible for high quality VFX work.

A third option was chosen which used a ray intersect technique similar to the first option. But instead of just firing a ray in the negative Y axis we also fire a ray in the same direction as the velocity of the agent but rotate it upwards. Then on each frame we compute the distance from the tip of the ray to the point of intersection to the geometry. The ray that is longer determines the new position of the agent which is the point of intersection for that ray.

$$P \mathrel{+}= \vec{v} * t \textit{ where t is the timestep}$$

Along with the new position we also calculate the new heading for the agent. The heading ensures that the agent is always facing the direction of travel.

We have already defined an "up" vector for the agent earlier and updated it based on the normal from the terrain.

The copy sop in Houdini updates the direction of the template geometry based on the normal of the points being copied on to. Hence we update the normal of the point with the new heading to orient the geometry being copied.

Another approach could be to create a quaternion name "rot" based on the axis of rotation and the angle of offset of the new heading from the basis vector 1,0,0. But for this engine we have chosen to update the normal as it provides for better debugging and we don't have to create any additional attributes.

Stick to Surface PseudoCode

```
• create a ray dir01 in the negative y direction
• create a ray dir02 parallel to the velocity of the agent
• calculate $\vec{side}$ = cross(heading,up)
• create a rotation matrix to rotate by angle $\Theta$ along axis $\vec{side}$
• multiply dir02 with rotation matrix
• calculate intersection of dir01,dir02
• calculate normal $\vec{N}$ at point of closest intersection I
• update the position P = i
• update heading at new position where heading = cross($\vec{N}$,$\vec{side}$)
• update the normal $\vec{N}$ of the agent where $\vec{N}$ = heading
```

By decoupling the heading from the velocity and smoothing the heading over a few frames we can get rid of the jitters that can be caused by conflicting forces acting on an agent(Green, 2000).

4.4 States and State Machine

The initial challenge of implementing a state machine was to be able to define multiple states in Houdini in an efficient manner. Since Houdini attribute types are limited to int, float, vector and string it was not a simple matter to store states that would be easy to use and remember.

The most effective method for this project was chosen to be to define each state as a floating point value. Each state would have an entry and exit floating point value. All state entry floating point values were saved as 1.0, 2.0 etc and exit values depending on the number of states as 1.x,2.x etc. where x != 0.

Although adding entry and exit states and switching them using the FSM was a complex design problem, the effort put in at an initial stage was quite useful in the end. While changing the food and pheromone values it was a simple matter of adding an attribute create node and modifying these values as an entry and exit state for each ant. In the current implementation multiple motion clips have not been implemented for each state but it will be a simple matter of detecting the state and adding a motion clip based on the current state of the agent.

In addition to the current state of the ant, each agent has two other state attributes.

Previous State- This attribute defines the previous state of the agent. This is particularly useful when transitioning to states which are utility states and don't change the locomotion of the agent. It was especially useful while transitioning to and back from the digger state. Since the ants would need to transition to the digger state and once dug immediately transition back from it. The various states implemented in this program and their descriptions are detailed in the next few sections.

Next State This attribute defines the next state of the agent. Every agent on a state change has its current state updated to the exit state of the current state and its next state attribute updated to the future state it needs to transition to. Once the exit state attributes have been updated for the ant the ant transitions to the next state.

4.4.1 State 1.x-Wander

Wander state is the basic state of the ants. This is also the state that the ants retreat back to when they have no other function to perform in the simulation. It essentially just implements the wander function and switches to the next state based on input.

Unlike other forces wander does not reset to zero every update cycle. Hence the ants have a graceful random motion that evolves and not a jittery random motion that updates every frame.

4.4.2 State 2.x-SeekFood

Seek food function implements the seek behaviour with the food node as a target. Since this function is only activated once the ant is within a specific distance of the food source, it steers the agent to the closest found food source. This state imitates the ability of an ant to smell a source of food from some distance away.

Each food item has a foodHome attribute. This attribute defines the quantity of food the item carries. Every time an ant seeks the food source the foodLoad value of the food item reduces. Keeping track of which item the ant has tracked and updated turned out to be quite tricky since it is not possible to save an array or a list in Houdini. To achieve this each food item was given an ID and each agent keeps track of which food ID it has visited.

Digger State 3.x PseudoCode

```
• if an ant is within range of food and foodLoad $\doteq$ 0
```

- seekfood
- if ant has reached food or exit state activated
- set foodLoad == 1
- check for food item accessed and update index value on agent
- accumulate agent count with index values per food item
- deduct foodLoad based on total count per food id.
- update phReward attribute for the ant

4.4.3 State 3.x-Digger

This is a fairly complex state of the ant behaviour. This state allows the ant to dig through areas of the environment that are considered to be soft.

To achieve this the geometry of density the object to be dug is converted to an SDF with an iso surface representation in Houdini. Then with the help of Houdini Volume VOP the ant that initially reach the isosurface tunnel through it and leave a trail for the ants that follow can follow. Once a surface has been tagged as being dug the ants that are in wander state either avoid the surface or if they are within a certain distance of the entrance to the tunnel they follow the trail left behind by the digger ants and pass through the obstacle.

Although the actual algorithm once deduced appeared trivial it turned out to be especially challenging.Firstly VEX does not generate points but only allows manipulation of point data. Since python allows creation of geometry, a python node was created to birth points to dig through the geometry. But python iterations across multiple points was extremely inefficient. Eventually a pop operator was used to dig through the geometry. This turned out to be one of the most challenging parts of the project and consumed a substantial amount of time.

Digger State 3.x PseudoCode

- if SDF value at agent position ¡ threshold
- update density attribute on agent = δ
- add all agents with density == δ to group diggers
- generate particles at position of agents in group delta
- generate a vector field parallel to particles
- for all SDF sample where distance ¡ threshold set value = 1
- if an agent is within threshold distance of entrance agent $\vec{V} = \vec{VF}$

where $\vec{VF}$ is the vector field

4.4.4 State 4.x-seekHome

If an agent has picked up food it returns home. In the current implementation of the project the agent always knows where its home is. There is no pheromone path set for them to follow. Though if required this would be pretty trivial as the workflow would be the same as the seek-food workflow that has already been defined for the food pheromones.

If an ant is carrying food it deposits pheromones along the route it follows home. Every time an ant finds the food source there is a positive reward that is provided to the ant, this increases the strength of the pheromone laid by the ant.

$$U_p(s') = R(s') + \gamma \max_{s'' \in S''} U_p(s'')$$

Figure 4.11: *Panait & Luke (2004)*

This by itself is not a sufficient factor. A key ingredient in the simulation was to generate a falloff in the pheromone strength deposited as the ant moves away from the food source. Initially it was decided to update the distance of the ant per frame from the food source visited and divide the pheromone value being deposited. But this was eventually determined to be too computationally expensive and a simpler solution was found.

The pheromone trail is stored on the points in the ground plane. Each point has an attribute phFalloff which stores the distance of the point from the food source. Since each agent has to access the point to update the pheromone strength once per time increment it is trivial to factor it by the distance attribute stored on the point as well.

seekHome State 4.x PseudoCode

```
    • if found Food and foodLoad ¿ 0
    • seek home
    • find the closest point in the pheromone trail grid
• update the phStrength of the point where phStrength += phReward $\div phfalloff$
```

To compute the decay in the pheromone strength a separate VOPSOP was used higher up in the chain. This was important as the decay was independent of the state of the agent and would update per time step.

4.4.5 State 5.x-seekPheromones

In this state the agent query the pheromone points. Each agent iterates through 16 pf its closest points to find the point with the greatest phStrength value. A value of 16 was chosen as this would ensure that the search area is limited to the 9 grids in the immediate vicinity of each agent at any time. VEX does not allow updating the primitive attributes of a geometry.

4.5 Geometry Instancing and Rendering

Houdini's implementation of the copy tool is pretty powerful for generating a wide variety of complex effects by copying a geometry on template points. Any template point attribute can be stamped back to the source geometry tree to offset or drive any random attribute. Although this is a powerful tool it comes at a prize. For a large number of points it can

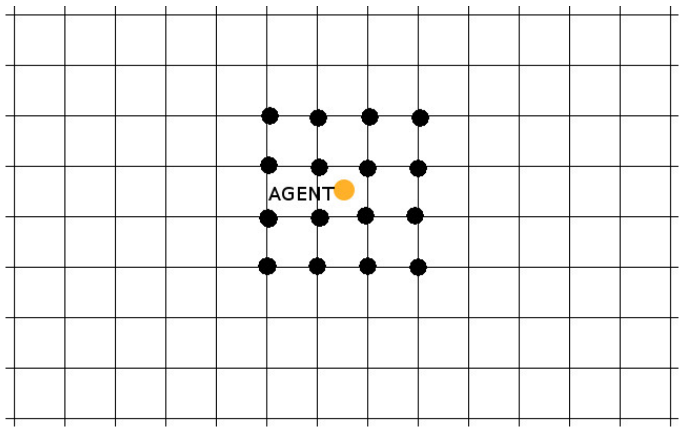


Figure 4.12: *Pheromone Search Range*

be a serious bottleneck as a copy of the geometry is stored in memory for each point.

This would lead to a large memory footprint during rendering. A more efficient method is to generate geometry at render-time. Houdini offers a Delayed Load Shader that loads a geometry at render time, similar to the delayed read archive functionality in Renderman. A .bgeo sequence of an ant walk cycle is exported from Houdini and loaded back in this shader.

4.5.1 Instance

To load the geometry at rendertime, the instance node has been utilised. The point simulation of the agents is referenced inside the instance node. The Delyed Load Procedural material is applied to the instanced geometry. By overriding the geometry load expression the speed of the walk cycle is modified procedurally using an expression.

Although this works to some extent it does not give a very accurate result. A better method might have been to calculate the length of one footstep and use that in conjunction with speed to drive the value of the walk cycle. The optimum solution is to calculate the position of each step independently for each leg based on the position offset. This would require skinning the object at render time on to the skeleton rather than a pre cached mesh.

Chapter 5

Applications and results

5.1 Crowd Simulation Results

Here are a few renders of the actual crowd simulation program.

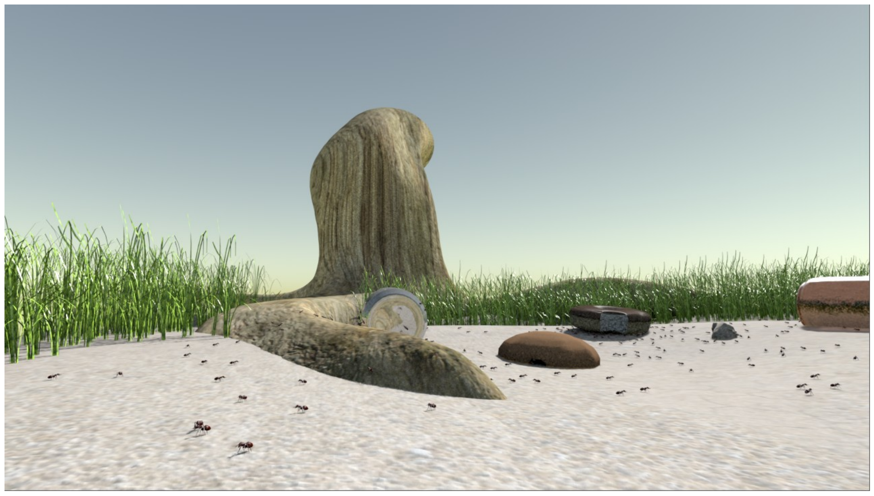


Figure 5.1: *Crowd Render*

500 ants

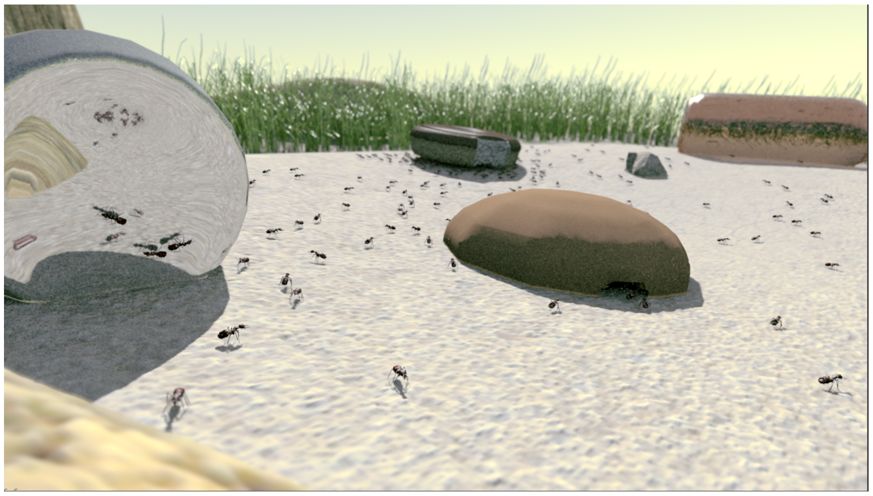


Figure 5.2: *Crowd Render*

ants Travel through tunnel and climd on the can

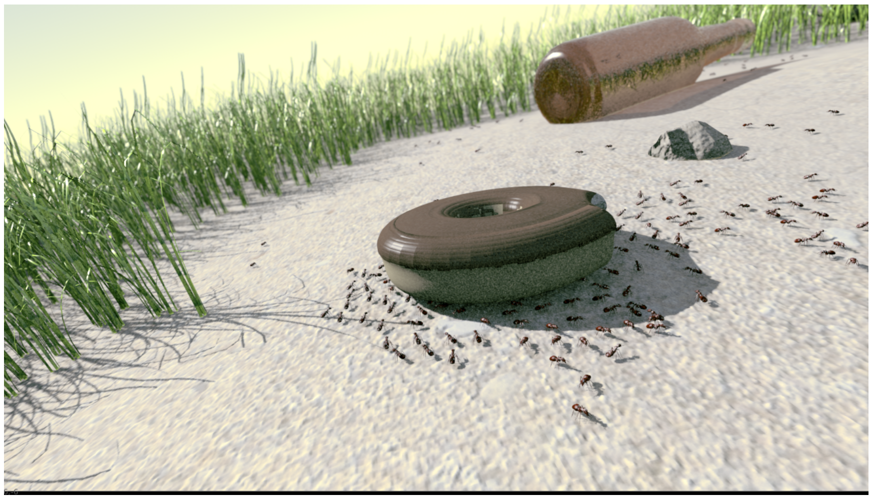


Figure 5.3: *Crowd Render*

ants seek the food

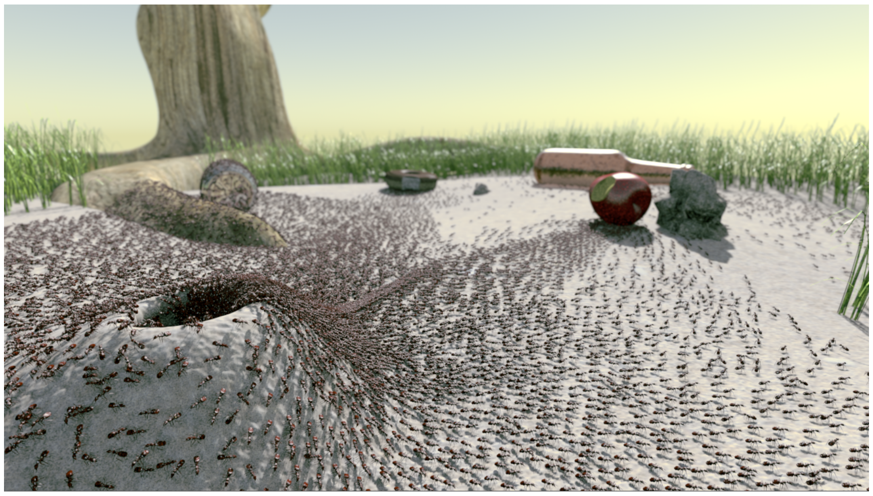


Figure 5.4: *Crowd Render*

15000 ants simulated

5.2 Pheromone Path Following Results

The pheromone based path following that has been implemented can be used for investigating the influence of a variety of factors on the output of the simulation. A comparison of the results is published below. The standard simulation values that have been compared are given below. ALl simulations have been carried out using these default parameters except for the parameter values mentioned in the captions which have been changed to obtain results.

- Use VectorFields - **off**
- Pheromone Reward - **20**

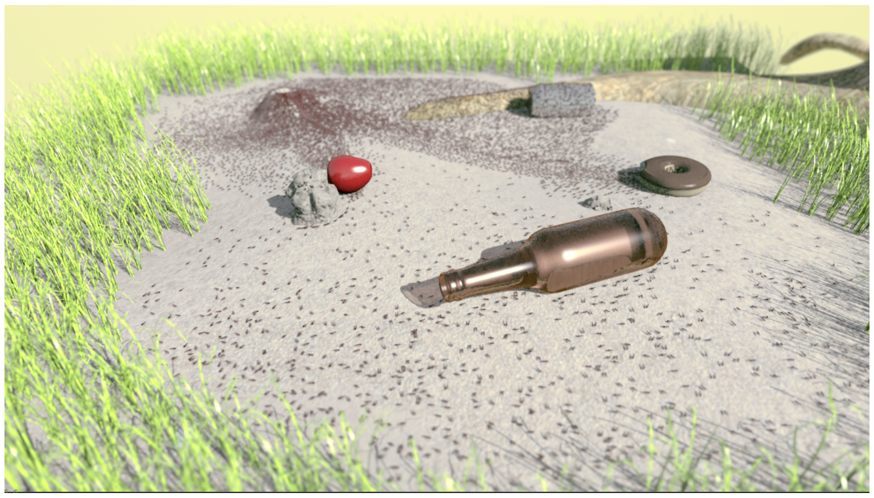


Figure 5.5: *Crowd Render*

15000 ants simulated, they bottle and stones are set as obstacles.

- Vaporisation Factor - **0.9**
- separation FOV - **35**
- seekPriority > separationPriority

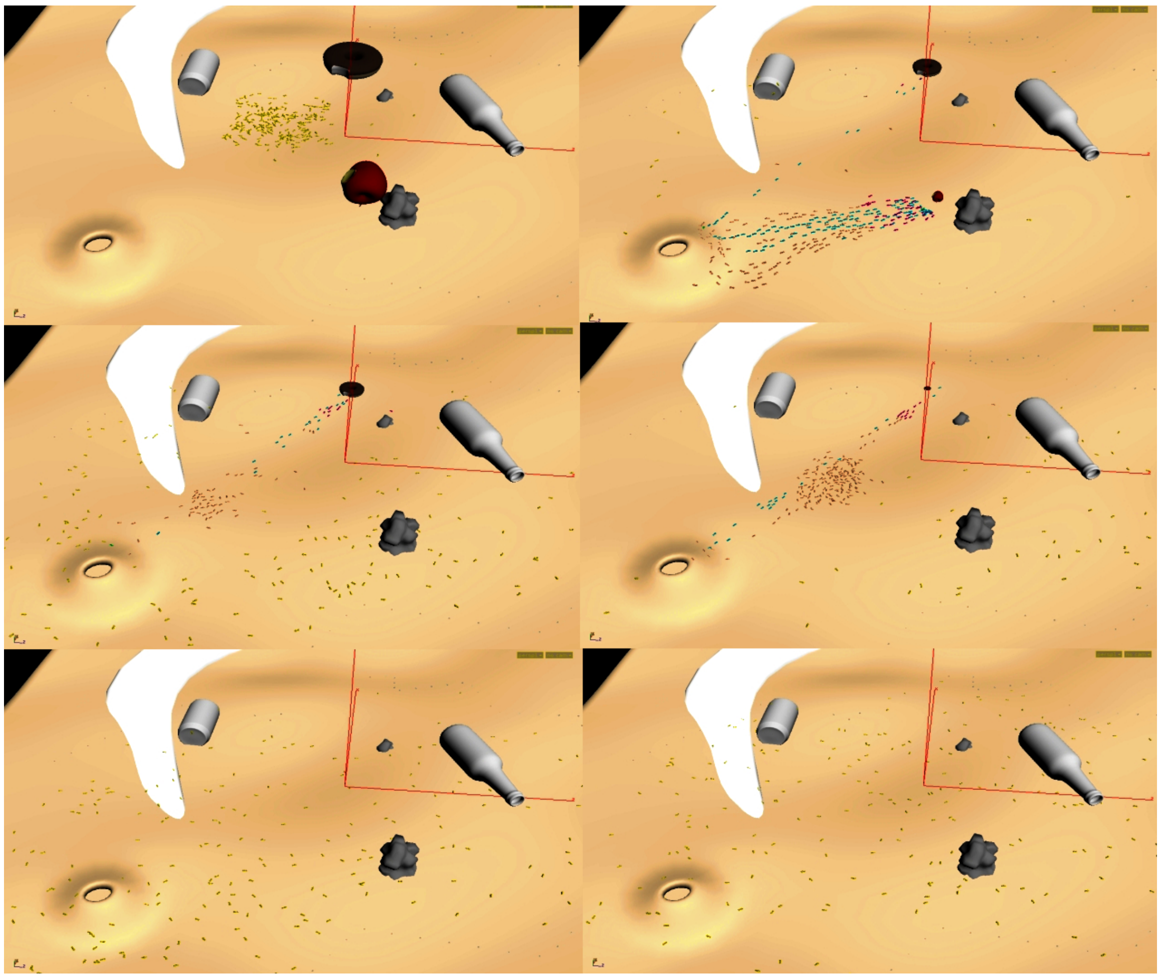


Figure 5.6: *Default Parameters*

Simulation run on default parameters

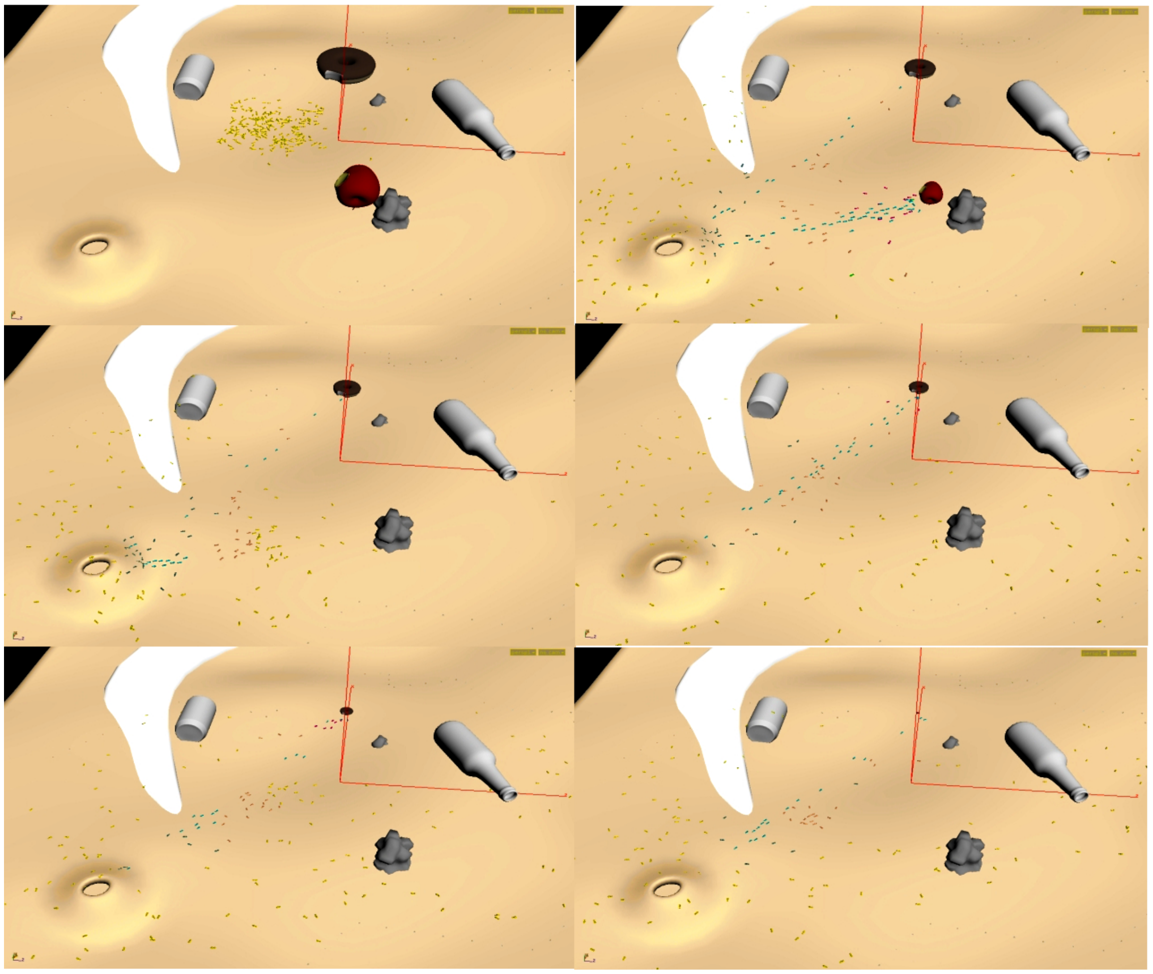


Figure 5.7: *Pheromone Reward **0***

Pheromone trail is weak. Ants are able to form a trail to the apple but the Donut is too far away and the trail keeps dissipating. Donut is still left on frame 5001

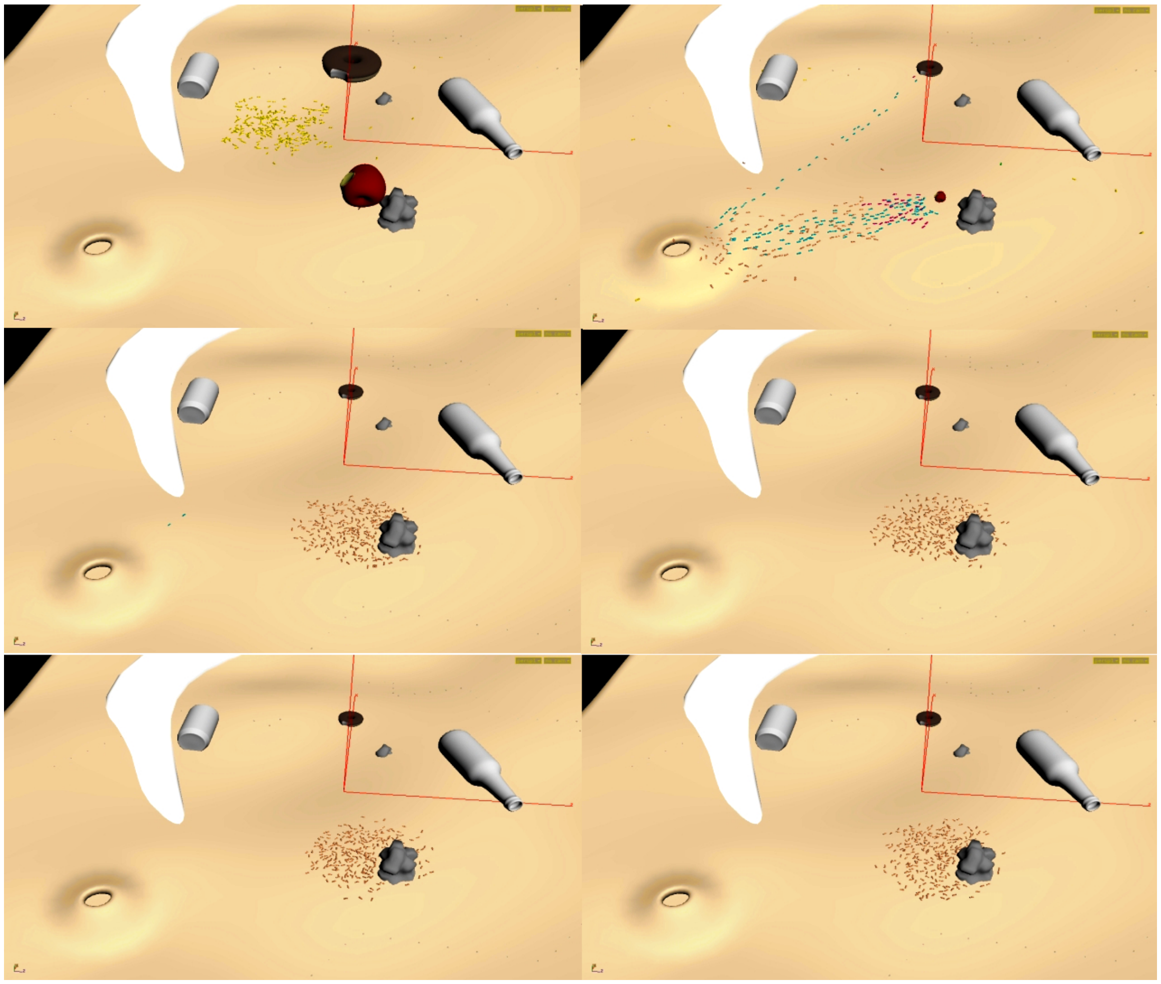


Figure 5.8: *Vaporisation Factor **1***

The pheromones do not vaporize and stay in the scene. The ants get confused and once the first food source is finished they move around in cicles following the everpresent trail. The Donut stays unfinished on frame 5000 and the ants do not create a new trail.

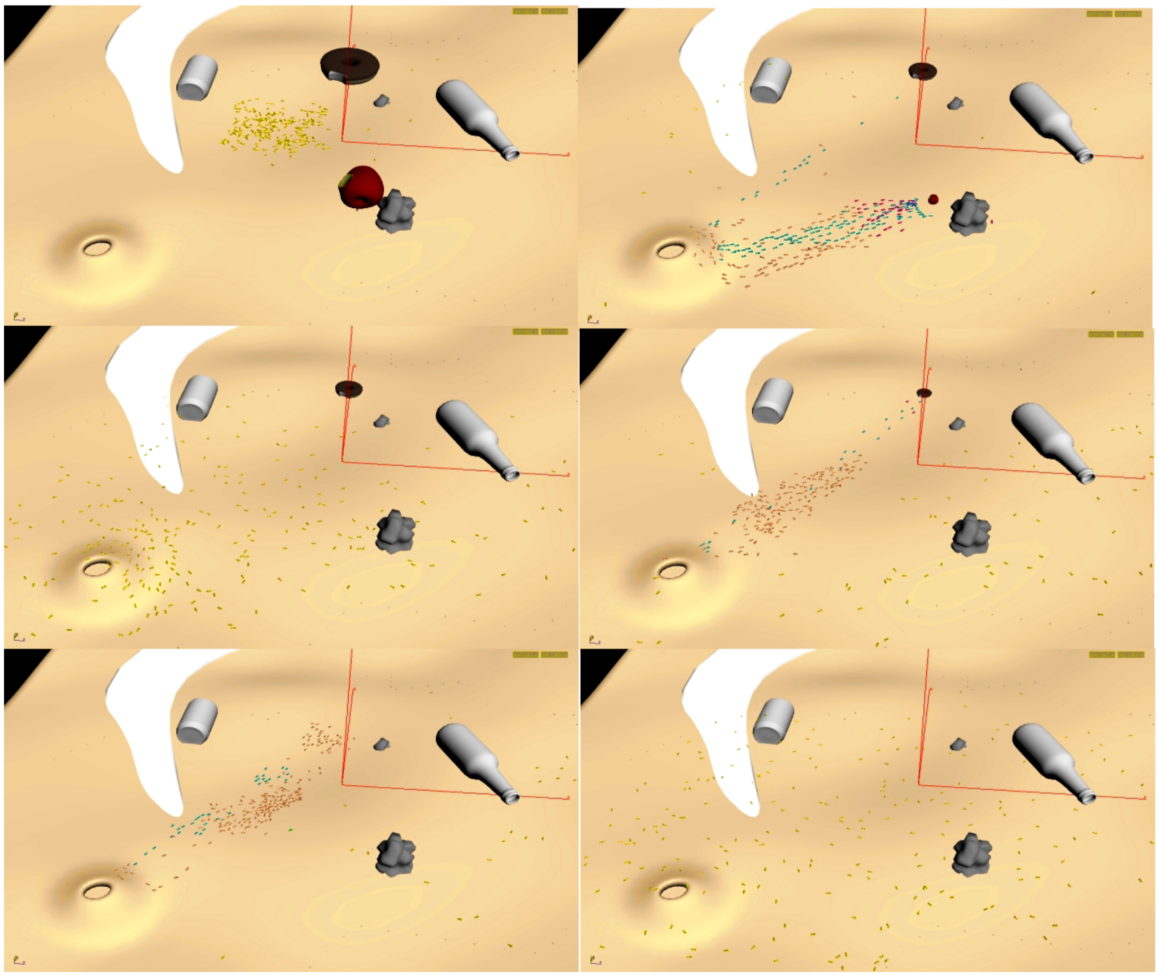


Figure 5.9: *Pheromone Reward* ***50***

Though the pheromone trail is formed, the excess pheromones take longer to dissipate and the ants take longer to move to the second food source. Donut is over on frame 3831 as compared to frame 3261 in figure 5.1.

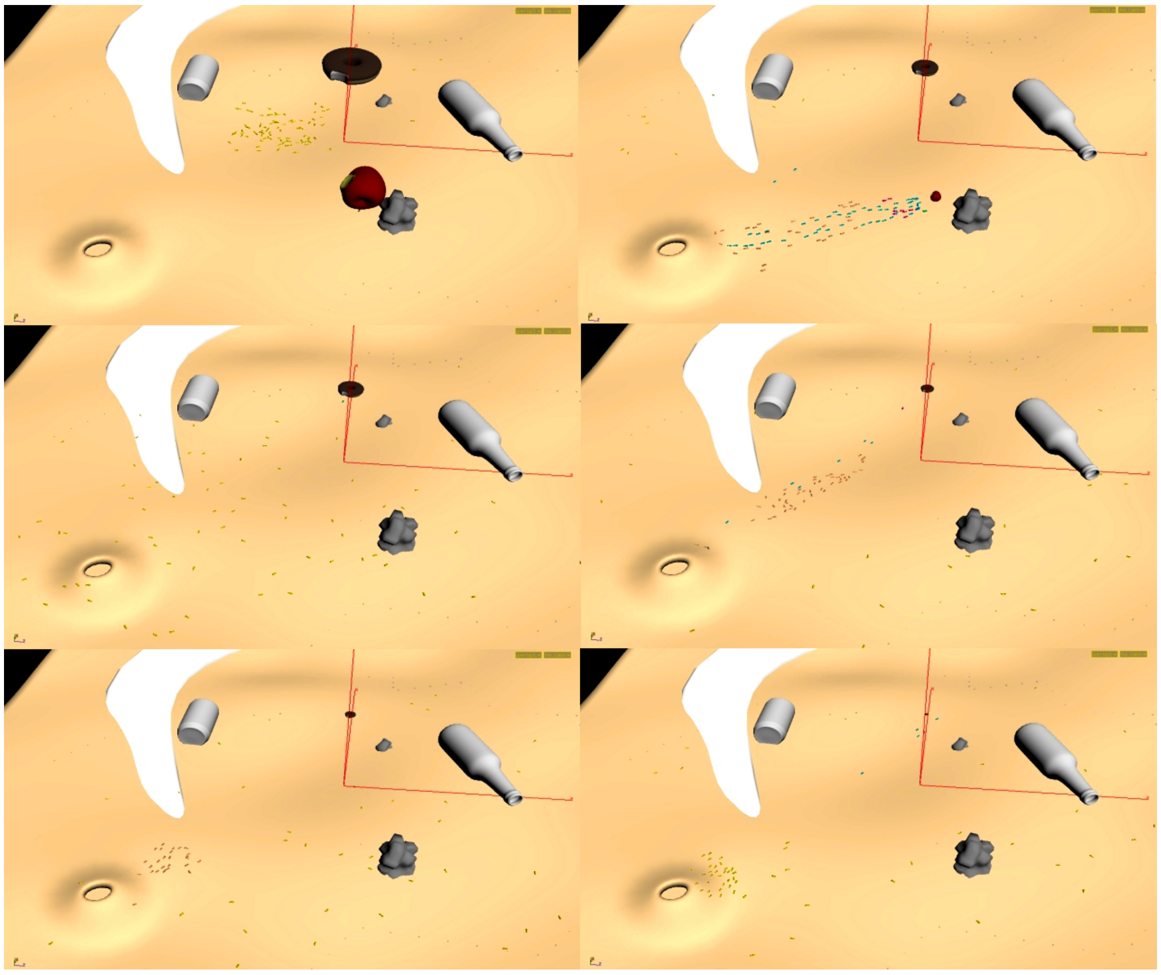


Figure 5.10: *Agent Count* ***100***

A smaller number of agents are quite efficient at finding the apple and consuming it but are not able to make a trail to the Donut that is further away.

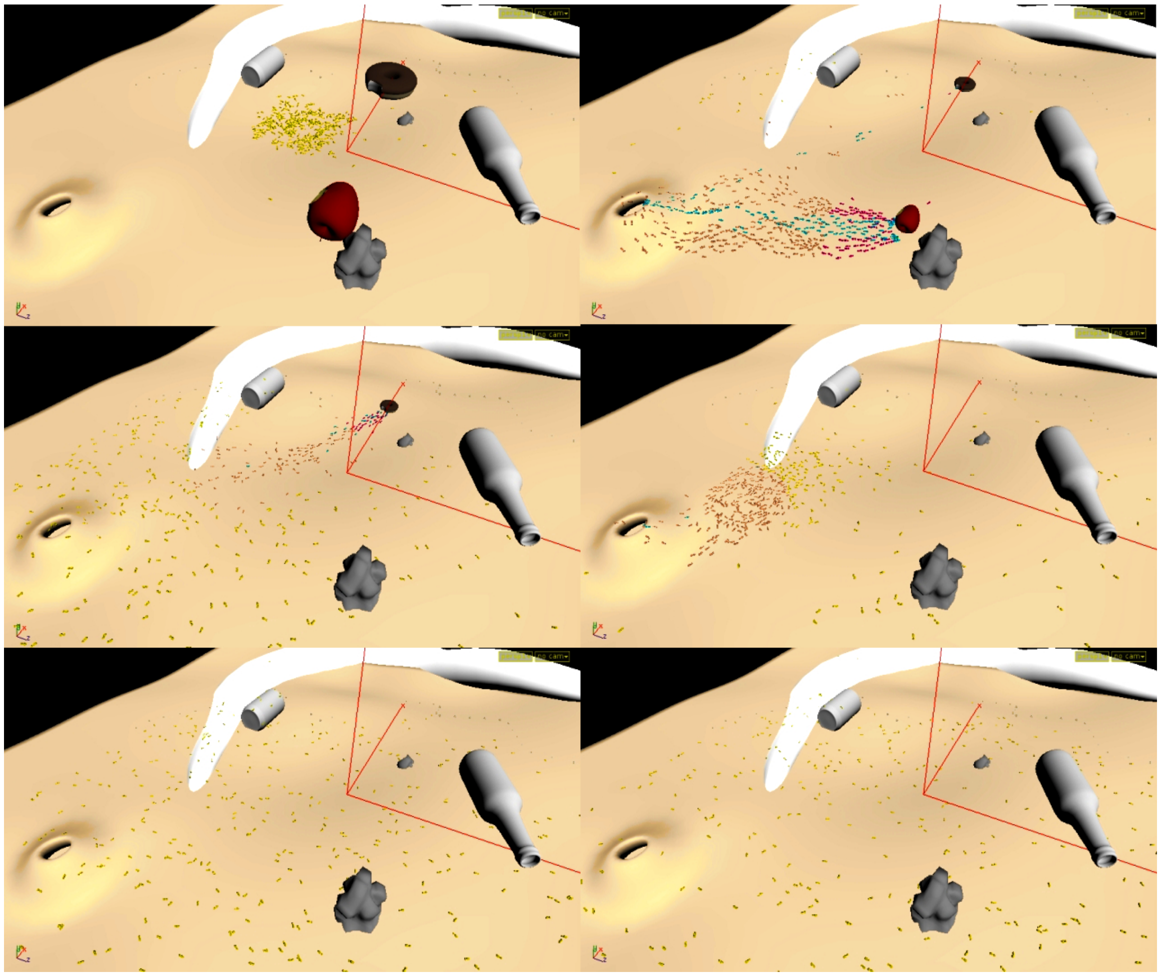


Figure 5.11: *use VectorFields* ***ON***

With vectorfields in place to guide the agents into the anthill and out, we get a much more visually appealing result with ants actually entering the ant hill and streaming out of it. The trails are also quite linear. The results are skewed though as the ants take only 2571 frames to finish the Donut as compared to 3261 in figure 5.1.

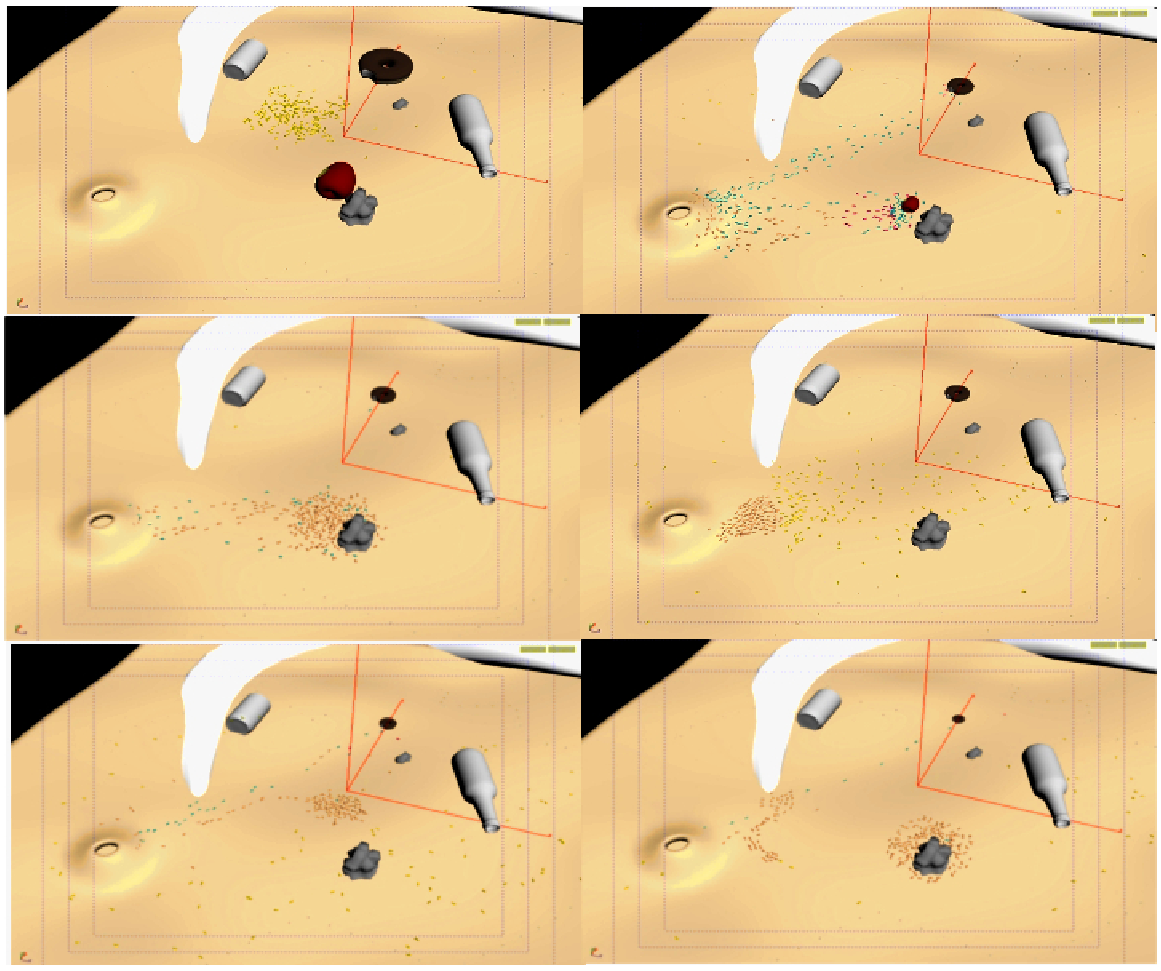


Figure 5.12: *SeparationPriority > seekPriority*

In figure 5.7 we see that the order of the priority of forces can change the simulation by an large magnitude. By keeping the separation priority higher than the seek priority we see that with a larger crowds the agents find it tough to find their way back home. The ants take a long 1491 frames to finish the apple as compared to the 1121 in figure 5.1. Single agents tend to get isolated for a long period of time as the pheromone value around the ant is always greater than the environment. One way of tackling this would be to give the ant a limited pheromone amount which would depreciate with each time step. Eventually if the ant does not find its way back home in a certain amount of time it will stop emitting pheromones.

Chapter 6

Conclusion

The aim of creating a crowd simulation system that is driven by underlying behaviours but still is art direct able has been successful.Based on the needs of a particular species sop nodes can be added using any number of Houdini's built in tools like vex, python or c++ to create a range of crowds.

6.1 Summary

This project demonstrates that by decoupling the behaviours from the decision making we can build a modular system that can be extended ahead in the future. A lot of time was spent in managing data in Houdini as it has limited attribute type. Although arrays have been introduced in VEX they are limited to being used for computation and cannot be used as an attribute.

Python was explored and although it would simplify the development time it would greatly affect the simulation times.

As a comparison some nodes were initially built in python and VEX both. For just one python node in the simulation the difference in performance was staggering.

- VEX 1000 frames in 17 sec which amounts to 58.8 frames/sec

- Python 239 frames in 60 sec which amounts to a low 3.98 frames per sec.

Although trying two separate workflows initially increased the development time as familiarity with both API's has to be created, the results speak for themselves and led to a efficient design choices later in the cycle.

Performance was not the primary criterion in design of the solution, the crowd system is pretty efficient. The biggest bottleneck is the volume data that has to be carried by the engine every timestep for the digger state. The ray intersect checks for collision are also expensive with 3 feelers per agent. In the future a bounding box based collision detection as a level of detail enhancement can be added. Based on the distance from camera, agents can just check for collisions against the bounding box of the geometry in the scene.

- With the volume data and 500 agents we got a speed of 9 fps
- with the volume data and 15000 agents we got a speed of 3 fps.

Although we increased the number of agents by a factor of 30, the performance reduction was only by a factor of 1/3. Also, based on some tests removing volume data from the simulation can increase the fps by about 5.

Another area that of Houdini that needs to be explored further in the future is floating point precision. VEX in Houdini still uses 16-bit floating point precision. This made comparison of float values tough. To work around this problem all floating point values were compared to a min and max range. A better floating point solution would make the code more efficient and cleaner.

6.2 Future work

The current walk cycle implementation of the ant is based on their velocity. A more powerful system would be allow the agent to place the footstep per frame based on the new position. This would enhance the realism of the simulation.. Instead of the finite state machine a fuzzy logic decision making system can be implemented. Although the state machine was successfully setup in the VOPSOPS a fuzzy logic system might be more complex and the possibility of exploring the Houdini HDK to expand on to the current setup is being can be explored.

The digging behaviour was a challenge and has been implemented Houdini's standard volume tools. Although most of the challenges of creating a robust digging solution have been solved, it needs to be refined further before it can be used effectively. The state machine is simplistic and a more complex decision making structure can be implemented to create interesting effects. A separate simulation where ants just dig an anthill out of a volume could be implemented as well to create an anthill with tunnels going through it for added realism. The use of VDB can be explored further as the standard volume tools make the system too slow.

The current simulation is extremely dependent on weighting of forces and the priority assigned. A more procedural solution and introducing flow fields with a cost function could be explored as well. The use of flow fields would make the ants easier to art direct and the cost function would procedurally weight the forces based on various factors like elevation, terrain etc.

VEX in houdini allows users to make custom header files that can be loaded into VEX operators in Houdini. Currently only a seek behaviour is implemented as a header file. By adding the simpler behaviour to these header files we can keep the code in the VEX operator a lot cleaner and simpler. In fact a whole library of VEX header files could be made as the crowd simulation got more and more complex to simplify the code in the VEX operators.

The rendering module is pretty simplistic and has to be built upon further. While the principles of instancing and deferred rendering will remain the same, the asset is currently dependent on the cycles created. In the future as mentioned, by developing a system that allows skinning of the mesh on a skeleton driven by the crowd system at render-time, we can build a more powerful system where the logic of the agent would define what geometry is rendered and would be fully controlled by the AI. This would also help generate variations in the crowd. For example there could be a library of arms, legs, hats, costumes etc with the same topology that would be skinned individually based on the AI. This would lead to a wide array of characters from a smaller group of assets.

Bibliography

A. Bielik (2004). 'Troy: Innovative Effects on an Epic Scale'. Available from: http://www.awn.com/ [Accessed 12.07.2013].

M. Buckland (2005). *Programming Game AI by Example*. Wordware game developer's library. Wordware Pub.

J. Dro (2006). 'File:Aco branches.svg'. Available from: http://en.wikipedia.org/ [Accessed 18.07.2013].

K. Griggs (2003). 'Assault on the senses [PC-run computer program for movies]'. *IEE Review* **49**(3):24–27.

J. Haddon & D. Griffiths (2006). 'A system for crowd rendering'. In *ACM SIGGRAPH 2006 Sketches*, SIGGRAPH '06, New York, NY, USA. ACM.

L. Kermel (2005). 'Crowds in Madagascar'. In *ACM SIGGRAPH 2005 Courses*, SIGGRAPH '05, New York, NY, USA. ACM.

C. Kolve (2004). 'alice1'. Available from: http://www.kolve.com/vfxwork/vfxwork.htm [Accessed 12.07.2013].

L. Panait & S. Luke (2004). 'A Pheromone-Based Utility Model for Collaborative Foraging'. In *Proceedings of the Third International Joint Conference on Autonomous Agents and Multiagent Systems - Volume 1*, AAMAS '04, pp. 36–43, Washington, DC, USA. IEEE Computer Society.

R. Pieke (2008). 'The Digital Eye: MPC's RD Confronts a Changing Industry'. Available from: http://www.awn.com/ [Accessed 12.07.2013].

C. Reynolds (1999). 'Steering Behaviors For Autonomous Characters'.

C. W. Reynolds (1987). 'Flocks, herds and schools: A distributed behavioral model'. In *Proceedings of the 14th annual conference on Computer graphics and interactive techniques*, SIGGRAPH '87, pp. 25–34, New York, NY, USA. ACM.

Side Effects Software (2013). 'Houdini: 3D Animation Tools'. Available from: http://www.sidefx.com [Accessed 18.07.2013].

D. Thalmann, et al. (2004). 'Crowd and group animation'. In *ACM SIGGRAPH 2004 Course Notes*, SIGGRAPH '04, New York, NY, USA. ACM.

A. Treuille, et al. (2006). 'Continuum crowds'. *ACM Trans. Graph.* **25**(3):1160–1168.

UKScreen (2013). 'DOUBLE NEGATIVE RECREATE THE VATICAN FOR "ANGELS AND DEMONS"'. Available from:/news/articles.htm?aId=2245 [Accessed 08.08.2013].