Andrew Gamlen

Submission

for

MASTER OF ARTS (ELECTRONIC ARTS) IN COMPUTER ANIMATION

Australian Centre for the Arts and Technology
Canberra Institute of the Arts
The Australian National University

Andrew Gamlen

Submission

for

MASTER OF ARTS (ELECTRONIC ARTS)
IN COMPUTER ANIMATION

Australian Centre for the Arts and Technology
Canberra Institute of the Arts
The Australian National University

This submission consists of the following sections.

1. Index and Programme Notes to Animation Portfolio

2. Sub Thesis: Investigating Flocking Using Behavioural Modelling

3. Animation Portfolio: VHS video tape containing the animation portfolio is included. Three of the animations have already been presented at ACAT performances, the remaining animations will be presented on a date to be announced.

Acknowledgments

Firstly, I would like to thank my supervisors Stuart Ramsden and David Worrall. Thanks also to all the staff and students at ACAT for their help and support.

The Australian National University Supercomputer Facility (ANUSF) provided computer and video resources necessary for animation. Thanks also to Drew Whitehouse for his help.

Index and Programme Notes to Animation Portfolio

✓ **Who Knows**

Date: June 1993 **Duration:** 2'20"

If consciousness is the result of a certain level of complexity in an organic system, such as human beings, can other complex systems, such as computers, become conscious? Will higher levels of complexity allow consciousness to evolve further, perhaps past the level of "self" so that an individual has the consciousness of a whole population?

Hardware: Amiga 500/2000.
Software: DpaintIV, Art Department Professional.

✓ **Journeys Through Scenery**

✓ **Date:** November 1993 **Duration:** 2'00'

This animation is about a journey through patterns and ideas and as such is more concerned with the overall method of travel than the final destination.

Hardware: Amiga 500/2000.
Software: DpaintIV, Adpro, Real3D, Imagine.

Formication

Date: July 1995 **Duration:** 3'10"

"If 100 ants are placed on a flat surface, they will walk around in never decreasing circles until they die of exhaustion. In extremely high numbers, however it is a different story."

Franks, N.R., "Army Ants: A Collective Intelligence," American Scientist, vol. 77, no. 2, March 1989, pp. 139-145.

A "conscious" colony of ants attempts to make contact with humanity.

Hardware: Amiga 2000, Silicon Graphics workstations, Sony Videodisk recorder.

Software: Real3D with custom RPL code, Imagine.

✓ **Wild Planet**

Date: June 1994 **Duration:** 3'30"

In the not to distant future, shopping malls have covered vast areas of the world. In the open expanses of the carparks that service these malls a new life form has emerged. "Shopping trolleys" roam the carparks in search of food, which can always be found where there are cars.

Hardware: Amiga 2000, Silicon Graphics workstations, Sony Videodisk recorder.

Software: Real3D with custom RPL code, Imagine.

Fuzzy Continuous Life

Date: July 1995 **Duration:** 4'00"

Researchers in the field of Artificial Life and those in Biological sciences cannot agree on a definition of life. The measurement of life should not be considered discrete, objects should not be considered animate or inanimate. Life should be seen as a continuous scale that ranges from the smallest particle of matter to the whole Universe. Of course a single atom is not very life-like, however atoms will seek out other atoms to form molecules and compounds. This self organising principle is only one of the properties of life.

The artificial creatures presented in this animation lie somewhere on the scale of life. They progress through their life cycle oblivious to the world outside of the computer.

Hardware: Amiga 2000, Silicon Graphics workstations, Sony Videodisk recorder .
Software: DPAINT IV, Real3D with custom RPL code, Imagine, XV.

Multiple Uses for Flocking Algorithms

Date: April 1995 **Duration:** 4'15"

This animation presents a series of short animations to show some of the technical details of my flocking algorithms:

1. Moths swarming around a lamp,
2. A school of fish,
3. Dragonflies that change from a swarm to a flock and
4. An interaction between two flocks.

Hardware: Amiga 2000, Colourpic, Silicon Graphics workstations, Sony Videodisk recorder.

Software: Dpaint IV, Real3D with custom RPL code, Imagine.

Andrew Gamlen

Sub Thesis

Investigating Flocking Using Behavioural Modelling

In partial fulfilment of the requirements
for the degree of Master of Arts
(Electronic Arts) in Computer Animation 1995.

Table of Contents

Andrew W. Gamlen

Investigating Flocking Using Behavioural Modelling

Australian Centre for the Arts and Technology
Canberra Institute of the Arts
The Australian National University

Abstract

The motions of flocks of birds, herds of land animals and schools of ocean creatures seem quite complex and beautiful. This thesis describes a simple method to simulate the motion of flocks and herds. By using a number of simple behavioural rules that each flock member executes independently, flocking behaviour emerges. Using such a behavioural approach frees the animator from having to define the motion and path of every flock member. As an extension to simulating natural flocking and herding, this thesis examines a flock-like behaviour that is not seen in nature. Following on from the generic flocking algorithm, there is some work on animating multi-legged creatures. This extends the flocking study to deal with the lower level locomotion of the creatures. The creatures are intelligent enough avoid obstacles and traverse uneven terrain, thus further freeing the animator from having to plot their every step.

1 Introduction

It is a unique delight to watch a flock[1] of birds as they twist and turn in synchrony. As a whole the flock seems to have a goal, with all the birds acting together. However if the birds are viewed as individuals, their movements and the paths they follow seem to be quite complex. In some folklore it is often thought that birds communicate changes in direction by using some form of mental telepathy, whilst others thought that most of the flock are just simply following a leader. However more recent evidence [Reynolds 87] suggests that the flock motion is the combined result of creatures[2] acting out a set of behaviours according to how they see the rest of the flock and their non-flock environment, such as obstacles, food and predators. In Reynolds' model there is no explicit communication such as telepathy, only implicit communication like the position and speed of neighbours.

To create a simulated flock using conventional computer animation methods would be extremely difficult. It would require, the plotting of a complex motion path for each bird and subsequently checking every one to make sure that none of the birds collide with any other. If a collision occurs during the animation, the paths would have to be modified and then checked again. The process would need to be repeated for all birds until there were no collisions. If an animator wanted to change the animation, the process would also have to be repeated. Obviously a method to make the animators job less tedious is required. The approach I investigate in this thesis is behavioural animation. My investigations and experiments are mainly based on the work of Craig Reynolds [Reynolds 87].

[1] In this thesis the word "flock refers generically to a group of creatures that display aggregate motion.
[2] The word "creature is used generically throughout this thesis to describe an individual flock member.

1.1 Previous Work on Flocking Algorithms

The earliest published work in computer animated flocking algorithms was by Susan Amkraut from the Computer Graphic Research Group of the Ohio State University. In 1985 Susan Amkraut in collaboration with Michael Girard created the work *Eurythmy* [Amkraut 85], which showed a flock of birds that could avoid collisions both with their flock mates and obstacles in their environment. The system developed by Amkraut uses vector force fields to control large populations. Repulsion fields placed at the centre of the creature's local co-ordinate system prevent collision among flock mates. Global force fields are used to direct the motion of the flock. Linear differential equations are used to specify force fields which are numerically integrated to compute each creature's trajectory [Amkraut 90]. By using different types of fields, such as: spirals, sinks, sources, saddles and orbits an animator can control the motion of the flock.

Creatures also have behavioural responses to force fields. These are modelled in terms of constraints on an individuals creature's orientation, direction and speed. For example, large birds have limits on their turning speeds, whereas insects are able to change direction quickly [Amkraut 90].

In 1986 Craig Reynolds made a computational model of coordinated animal motion as of bird flocks and fish schools [Reynolds 87]. This model is based on a 3D computational geometry of the sort normally used in computer graphics or computer-aided design. The software was called *Boids*. Each *boid* has direct access to the whole geometric description scene, but reacts only to flock mates within a certain small radius of itself. This basic flocking model consists of three simple steering behaviours for each flock member:

1. Separation: steer to avoid crowding local flock mates.
2. Alignment: steer towards the average heading of local flock mates.
3. Cohesion: steer to move toward the average position of local flock mates.

In addition to this model, Reynolds made a more elaborate behavioural model to include predictive obstacle avoidance and goal seeking. Obstacle avoidance allowed the *boids* to fly through simulated environments while dodging static objects. For use by animators, a low priority goal seeking behaviour caused the flock to follow a scripted path [URL 3]. Using this system, Reynolds produced the animation, *Stanley and Stella* [Symbolics 87], which shows the motion of a flock of birds and a school of fish.

Andy Kopra of VIFX expanded on Craig Reynolds original *Boids* software to produce photorealistic imagery of bat swarms that were used in the motion pictures *Batman Returns* and *Cliffhanger* [URL 3].

1.2 Particle Systems

The Flocking algorithm used by Reynolds and described later in this thesis can be seen as an expanded version of a particle system [Reeves 83]. Particle systems are used to simulate "fuzzy" objects, such as clouds, fire, smoke and water. Because these objects change shape fluidly over time, much like a flock, they cannot be simulated by means of traditional computer graphics using techniques such as affine transformations.

Particle systems contain large numbers of individual particles which can each have a number of properties:

(1) position,

(2) velocity,

(3) size,

(4) colour,

(5) transparency,

(6) shape and

(7) lifetime.

The dynamic shape of the objects being simulated is the result of these properties of particles changing over time by reacting to the environment in which they are placed.

The environment will usually be some form of simulated gravity, but it could also be any force that acts on the velocity or other properties of the particles. The flocking algorithm takes this one step further in that the creatures interact with each other as well as the environment. Particle systems generally use points or spheres as the particles, whilst the flocking algorithm needs a much more complicated geometry to simulate a bird or other creature. Because of this the creature needs another property, orientation.

2 Flocking and Artificial Life

The motion of the simulated flock can be described as emergent, in that the global pattern of flocking motion is not programmed but emerges from the interactions between the flock members. Flocking is just one example of "emergence"; the term is borrowed from the discipline of Artificial Life. Different disciplines assign "emergence" differing meanings. Crutchfield [Crutchfield 94] offers this definition:

Emergence is generally understood to be a process that leads to the appearance of structure not directly described by the defining constraints and instantaneous forces that control a system. Over time "something new" appears at scales not directly specified by the equations of motion. An emergent feature also cannot be explicitly represented in the initial and boundary conditions. In short, a feature emerges when the underlying system puts some effort into its creation.

These observations form an intuitive definition of emergence. For it to be useful, however, one must specify what the "something" is and how it is "new". Otherwise, the notion has little or no content, since almost any time-dependent system would exhibit emergent features.

The "something new" or "unexpected phenomena" are often cited as requirements for emergence. However, chaos physicists and some philosophers [Crutchfield 94] claim that unpredictability is not an appropriate criteria for emergence. They argue that unexpectedness or unpredictability means only a lack of data or an incomplete data set [Kawata 94]. Once a flock has been simulated, we are in possession of all lines of interactions and critical events that govern the simulation results. Although it may be impractical in many cases, we may be able to trace and explain how the commands of a program cause the events of the simulated flock, which suggests that these events are not unexpected and so are not "emergent". The impracticality is usually a result of the vast number of interactions that need to be traced to explain how the flock interacts. Thus there is no clear criteria to determine whether the results obtained are predictable or not. Some computer scientists restrict the definition of emergence even further. In their view if the simulation cannot change the code which controls the simulation, the results are more or less programmed in [Kawata 94].

Chris Langton proposed a more practical definition of emergence as the feedback between micro- and macro-level behaviours. Taylor [Taylor 91] describe it thus;

"The local dynamics of a set of interacting entities (creatures) supports an emergent set of global dynamical structures which stabilise themselves by setting boundary conditions within which the local dynamics operates. That is, these global structures can "reach down" to their own, physical bases of support and fine tune them in the furtherance of their own, global ends. Such LOCAL to GLOBAL back to LOCAL, inter-level feedback loops are essential to life, and are the key to understanding its origin, evolution and diversity."

Under such a definition, the flocking algorithm presented in this thesis cannot really be considered as true emergence; only a simulation. However it does fit into the earlier intuitive definition suggested by Crutchfield. Crutchfield expands on the idea that the unexpected behaviour is not only the result of not having full knowledge about a system but is also dependent on the observer's knowledge and ability to build accurate models. Since the observer's knowledge is critical in evaluating whether a system shows emergent behaviour or not, this leads to a very subjective view of emergence.

When a person uses their subjective judgement to say whether a simulated flock produces realistic looking motion, they are using previous models to judge the realism of the system. If the observer were to have no previous knowledge of flocks in nature, they would probably perceive no pattern on initially observing a simulated flock. Although the previous statement is only an assumption (thought experiment) there is an historical bases for this assumption: The Belousov-Zhabotinsky reaction, is a chemical reaction which shows spiral patterns that oscillate and change shape over time. When this reaction was first discovered a number of scientist denied that the reaction created any pattern [Crutchfield 94]. This was perhaps due to them continuing to use an older model that maintains that complex patterns could not be the result of a simple reaction between two chemicals.

3 Implementation of the Algorithm

The three steering behaviours that I have implemented to control my own animated flocks are very similar to those mentioned by Reynolds earlier, differing more in name than actual function. The short names I have chosen represent the geometrical way these behaviours are derived. The three steering behaviours I have implemented are, in order of precedence of operation:

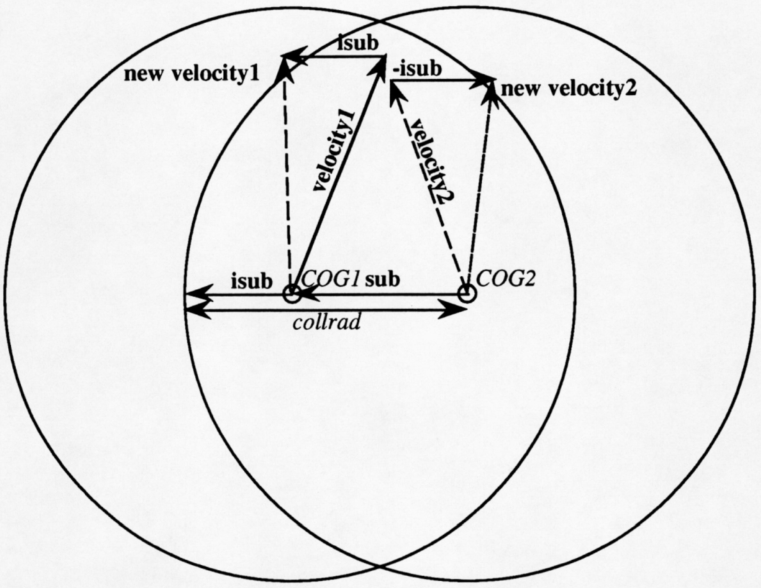
(1) Collision avoidance,

(2) Velocity matching and

(3) Flock Centring.

3.1 Collision Avoidance

The collision routine that is used is basically the simple inverse square method.



Creature1 collision radius   Creature2 collision radius

Figure 1: Collision Avoidance. For ease of viewing this diagram shows an inverse method. In an inverse square method the vectors **isub** and **-isub** are reduced to the square root of their lengths. The following equations show how the new velocity is derived.

Equation 1: $\mathbf{sub} = COG1 - COG2$

Equation 2: $\mathbf{isub} = \|\mathbf{sub}\| \times \sqrt{collrad - |\mathbf{sub}|}$

Equation 3: $\mathbf{new\ velocity1} = \mathbf{velocity1} + \mathbf{isub}$

The diagram shows the collision avoidance vector being added to both flock members, this was only used in earlier version of the algorithm. A flock member may have more than one collision and avoiding a collision with one flock member may create a collision with another. This same type of collision avoidance is also used to avoid environmental objects such as buildings, trees and cars in my animation *Wild Planet*

[Gamlen 94]. The cars presented a further problem in that they were moving faster than the herd of trolleys and are of a different size. In one test animation the trolleys collided and embedded themselves within the geometry of the car. To stop this from happening again I always made sure that objects in motion had a collision radius that was at least twice the length of their maximum velocity vector. As a spherical vector field on its own will not satisfy all shapes of creatures and static objects I implemented a tubular force field which I used in the log and reed avoidance in the dragonfly animation [Gamlen 95a]. Reynolds implemented a collision prediction method which involves a form of artificial vision to look ahead for collisions. Although this method would be much better, the number of calculations involved in determining which flock members are visible (not obscured) and how far away they are was outside the realistic time constraints with the hardware I was using.

3.2 Velocity Matching

Velocity matching behaviour enhances collision avoidance and also helps to keep the flock together. The collision avoidance is enhanced by flock members roughly travelling in the same direction; they thus don't cross each others paths and thus avoid collisions.

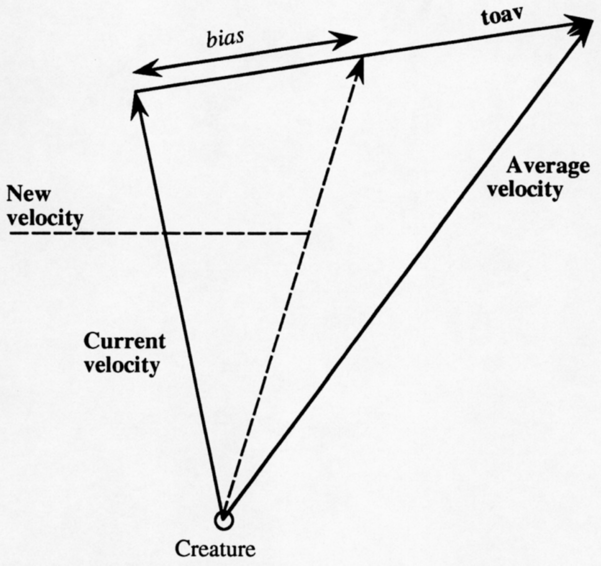


Figure 2: Velocity matching. The following equations show how the new velocity is derived.

Equation 4: **toav = average velocity - current velocity**

Equation 5: **new velocity = toav** $\times$ *bias* + **current velocity**

By varying the velocity matching bias, the animator can have an effect on how well the flock members match each others velocities and hence avoid collisions. This can be used to make up for some of the shortcomings in the collision avoidance routine. However making the velocity matching bias very high has the unpleasant side effect of making the flock motion very "stiff" as it negates the flock centring behaviour. This motion looks like a squadron of aeroplanes flying in formation and would be a good way to simulate this motion as the aeroplanes would still be free to make slight course deviations. This type of motion is usually animated by making all the aeroplanes follow the same path, creating the ultimate "perfect" flying as the planes always keep exactly the same distance from each other. As this would be totally impossible for human pilots, it looks unrealistic.

3.3 Flock Centring

Flock-centring behaviour causes individual flock members to seek the centre of the flock. This behaviour is important as it simulates the flock members need for safety from predators. In the wild, one of the main reasons that animals flock is to appear larger than they really are; this deters predators. Being at the centre of the flock means that a predator would probably take the flock members on the outskirts of the flock instead. It is this safety instinct that creates the interesting turning motion of flocks as members turn and try to get to the centre, other flock members respond by turning away to shift themselves closer to the centre. This behaviour is calculated in much the same way as the previous behaviour.

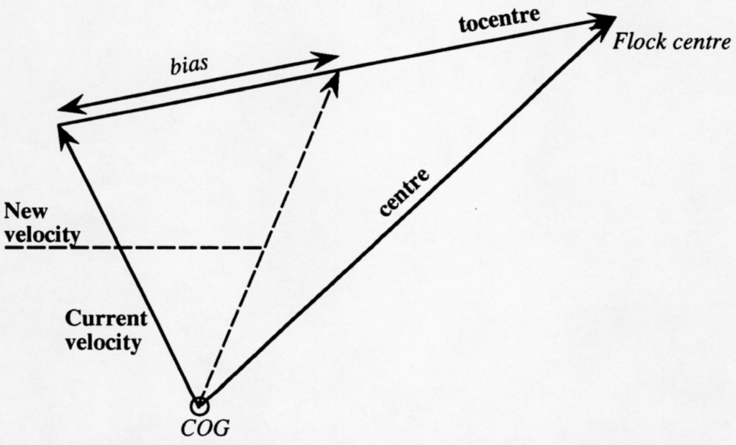


Figure 3: Flock centring. The following equations show how the new velocity is derived.

Equation 6: $\mathbf{centre} = \mathit{Flock\ centre} - \mathit{COG}$

Equation 7: $\mathbf{tocentre} = \mathbf{centre} - \mathbf{current\ velocity}$

Equation 8: $\mathbf{new\ velocity} = \mathbf{tocentre} \times \mathit{bias} + \mathbf{current\ velocity}$

By using large centre-bias values the flock shows a motion that is more like that of a swarm of insects hovering around a light or food source, than that of a flock of birds.

3.4 A Flock Member's View of The World

As mentioned in the previous section, a creature's behaviour depends on its perception of its local neighbours. In the earliest version of my algorithm, every flock member "looked at" every other flock member to modify their behaviour. In this global model the type of behaviour that usually emerged was more like swarming than flocking. This was probably due to all the flock members wanting to be at the centre of the flock and because the flock centre perceived by flock members was the actual flock centre, there was no impetus to fly in any direction. They simply wanted to swarm around the flock's centre. It is very unrealistic to assume that every flock member can see every other flock member, however it is a lot easier to implement this on a computer as the algorithm doesn't have to calculate which creatures can be seen by other creatures. To overcome this problem I gave the creatures a form of artificial vision. This involved calculating a spherical segment volume of vision for each creature. Any flock members who were within this volume are used to influence a flock members behaviour.

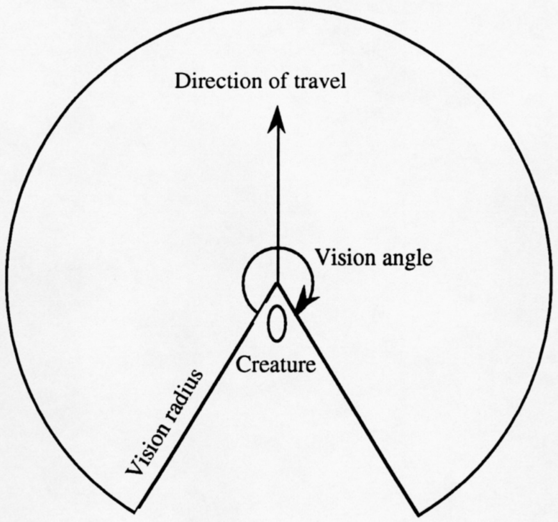


Figure 4: Artificial vision volume.

Two calculations are used to generate the vision volume. Firstly any flock members that lies within a user-defined vision radius of a particular member are said to be visible. This provides a distance or depth to which a creature can see, which is used to simulate the effect of close creatures obscuring those that are further away. As this is only a simulation it is of course possible that if the creatures had real vision there would be other creatures within this radius that would be obscured from vision and creatures outside the radius that would be visible. However this method works well within my simulated environment. Secondly, a limited vision angle is calculated to either side of an imaginary axis running lengthwise through a creature's body. This simulates the forward-looking vision that most animal possess. Because the angle and view radius are user definable the earlier global model can be replicated by setting the viewing angle to 360 degrees and the visible radius to a suitably large number that it encompasses the whole flock.

3.5 Directing the Flock

If a flock of creatures is given a set of initial velocities and positions and the flocking algorithm is set in motion, the system will provide a realistic simulation of natural flocking. However the flock is free to fly where ever it wants. Although the simulated flock is not sentient (it does not actually decide where it wants to fly) the complex nature of the steering calculations could not possibly be predicted. Since it is impossible to predict the direction of travel of the flock I wrote a flock tracking routine so that the computer graphics "camera" always followed the flock. This tracking routine has two flock-following options:

1. Follow the centre of the flock or
2. Follow a particular flock member

and two camera options:

1. Stationary camera (camera pans and tilts to follow flock) or
2. Moving camera (camera stays a fixed distance from the flock).

Of the four combinations of these options the moving camera following the flock centre is the most useful for observing the flock, as it tracks the flock even if the creatures try to fly out of view. However the method I relied upon most is a fixed camera method which follows the flock centre, as this is more like a real camera trying to follow a flock.. Using these tracking routines it is similar to being a camera person for a nature documentary, this is especially so when using a stationary camera, as the flock can move so far away that they become mere specks. You then wait hoping that they will turn and come back into view. Viewing simulated flocks like this provided the inspiration for the animation *Wild Planet* [Gamlen 94] which has a format similar to that of a television nature documentary.

The algorithm would be enhanced by a method of corralling and directing the flock. Initially I tried to use force fields similar to that used by Amkraut [Amkraut 85].

This was simple to implement as these fields are just invisible static objects. However I found this method hard to use, mainly because I only had two types of static objects and hence two types of force fields: spherical and cylindrical. Complex paths would have to be made up of many sub-fields otherwise some flock members would escape the fields. The circular nature of the fields caused some members to reflect in one direction and others in an opposite direction causing the flock to split and rejoin for no apparent environmental reason.

To overcome the above mentioned problems, I implemented a path following steering behaviour. The path is a smooth spline curve with an invisible object moving along it. To see the effect on behaviour the path object can be made visible . This behaviour was given an even lower priority than the three original steering behaviours (see section 3.6 Overall Control Structure). If it were given a higher priority the creatures would be more interested in following the path than flocking and the motion would look much like a squadron of aeroplanes. The path object is usually well ahead of the flock, causing very large path-following vectors to be generated. To counteract this, the path bias is usually kept small. Path bias works in the same way as does the flock centring and the velocity matching biases. The path-following method is a system with which most animators are already familiar. Using this method the flock can be located anywhere it is wanted giving the ability to interact with other motion pathed or key framed parts of an animation. If some part of the animation is not synchronised then all the animator has to do is change the path and run the simulation until the desired results are achieved. The major problem with the path method is finding an appropriate ratio between the path object's velocity and the velocity of the flock. If the path object gets too far in front of the flock, the path-following vector can get excessively large and cause the flock motion to become stiff. Conversely, if the flock catches the path object then the flock will have no impetus to move forward and will swarm around the path object.

The main problem with finding a ratio that will produce realistic motion for the full length of the path is that the velocity of the path object can be varied over the path, simulating acceleration and deceleration. This leads to a non-linear relationship between the average velocity of the path object and the velocity of the flock. I have not implemented an algorithm to handle this problem, instead I have a couple of heuristic rules that I use by hand:

1. If the animation is too "swarm-like",
    a. reduce the number of frames in the animation, or
    b. decrease the maximum velocity of the creatures.
2. If the animation is too stiff,
    a. increase the number of frames, or
    b. increase the creatures maximum velocity.

note: increasing or reducing the number of frames in the animation is equivalent to reducing or increasing the velocity of the path object.

These rules work well and realistic results are usually obtained by the second run of a simulation.

3.6 Overall Control Structure

The four steering behaviours: Collision Avoidance, Velocity Matching, Flock Centring and Path Following each generate a vector, signifying the desired motion of the creature. Since there are four behaviours the simplest way to generate a single vector for an individual flock member would be to average all three vectors. This system will work fairly well in most situations, however in critical situation there can problems [Reynolds 87]. In crowded environments, where there are many creatures and static objects, the averaging of vectors can produce collisions. For example a creature may have a flock centring vector pointing in one direction and a collision avoidance vector pointing in another direction and the averaging effect on these two vectors will sometimes swing the final vector back towards the static object that the creature is trying to avoid. To deal with this problem my algorithm uses a prioritised scheme of assigning velocities similar to that used by Reynolds in his *Boids* software [Reynolds 87]. The main difference between Reynolds method and mine is that Reynolds assigns accelerations rather than velocities to his flock members. Using accelerations rather than velocities is slightly more complex and allows creatures to be given strength limits past which they can't accelerate. This creates more realistic motion than simply using velocities, as creatures can accelerate and decelerate in certain circumstances such as; collision avoidance and rejoining the flock. The added computational and programming expense of using acceleration over velocities is not warranted for the small increase in realism which would be gained, as the execution of the three steering behaviours in my algorithm changes the creature's velocities over time.

Collision avoidance is given the highest priority since if a natural creature was involved in a collision it may be harmed or die. Also a collision in three dimensional computer graphics looks very unrealistic as it appears as if one object is passing through another. The second highest priority is given to velocity matching, since it enhances collision avoidance. The lowest priority is given to flock centring, since giving it higher priority causes the flock to remain moving around a fairly stationary point, which is a behaviour more like swarming than flocking.

A secondary issue with flock centring is that if the priority is too high, a large number of collision avoidance calculations are necessary as creatures on the outside of the flock try to dive into the centre and the creatures that are already near the centre want to stay at the centre and not move to make way for the other creatures. This large number of calculations not only slows the algorithm but can also cause an individual creature to have multiple collision calculations with other flock members.

The collision avoidance routine is simple and straight forward as it only deals with collision involving two creatures faithfully. If the collision avoidance routine had to deal with multiple collisions the vectors that it produces for the first collision can be negated be the second and subsequent collisions hence the first collision may not be avoided. When the flock is run in directed mode, the path following behaviour is given the lowest level of priority (see section 3.5 Directing the Flock). The three behaviours; Velocity Matching, Flock Centring and Path Following all have a user definable bias which can be used to create different types of flocking behaviour (see section 3.8 Different Types of Behaviour).

In the simulation the velocity matching and flock centring routines are executed in the reverse order of priority as the vector produced by the flock centring routine is then taken to be the current velocity which is then used by the velocity matching routine. This order of execution means that the effect of flock centring routine is lessened by the velocity matching routine and hence the velocity matching routine has higher priority since it has the last say. The collision avoidance routines are executed separately from all other behaviours. If a creature has to take collision avoidance action then none of the other behaviours are executed at that point in time. Collisions with "static" objects are given a higher priority than collision with other flock members in the current version of the algorithm. This is because in the animation *Wild Planet* [Gamlen 94] the "static" objects were actually cars that were moving at higher velocity than the herd. If the "static" objects are actually static then the program can be simply modified.

3.7 Physical Simulation

As has been already mentioned, increasing the complexity of the algorithm for only very small gains in realism is not worthwhile. The problem of complexity and realism of a simulation versus the complexity of the controlling algorithm is very common in computer graphics. Recent attempts have been made to overcome this problem by using genetic algorithms to "evolve" creatures and their control systems [Sims 94]. My approach to physical simulation was to use none and observe the results and see what was needed to create a more life-like simulation. In some of my early simulations the flock would slowly wind down and eventually stop. Whilst this was due to an error, it prompted me to implement a minimum and maximum velocity constraining routine. Although it would not look unrealistic for a sea or land creature to have a zero or very small velocity it would be quite unusual for a bird. Another problem was that during simulations creatures would sometimes turn nearly 180 degrees in a single frame usually while trying to avoid a collision. This type of sudden movement is not out of place under the microscopic and with insects, however the direction of larger creatures is more constrained by their momentum. Therefore I implemented turning angle constraint so that larger creatures could be more faithfully simulated. Although these two physical constraints are rather simple they have a remarkably realistic effect on the motion.

Reynolds [Reynolds 87] implemented a simulation of the laws of gravity, so that his *boids* would decelerate when climbing and accelerate when diving. The *boids* could also stall if they tried to climb without significant velocity. Tu took physical simulation even further in a simulation of artificial fishes [Tu 94] by simulating a number of different physical phenomena. The physics of water were simulated. The dynamics of the muscles of the artificial fishes and plants that were part of the environment were simulated using a spring mass system. These plants were seen as food by some of the fish and the whole simulation was quite complex; containing numerous different species of fish, including predators and prey. Each fish possessed a sophisticated perceptual system that could use vision to sense light and dark patterns and judge distances. Even mating behaviour was incorporated. This information was used by the artificial fish to build an internal model of it's environment. With such a complex system the fishes can be seen as autonomous agents living in a simulated environment. Since the fish are autonomous agents it would be difficult for animators to make them do what they would like them to do, so this type of highly complex simulation may have more a role to play in understanding complex systems such as real fish and artificial life.

3.8 Different Types of Behaviour

Flocking is not the only group behaviour that can be simulated by the algorithm presented here. In this section I will discuss a number of natural behaviours and how I have used the flocking algorithm to simulate them. I discovered swarm-like motion by accident when testing my global algorithm and found that the velocity matching behaviour was not operating. Once I had my algorithm fully working I could still repeat this kind of motion by setting the velocity matching bias very low and giving the creatures 360 degree vision, similar to real insect vision. A high centre bias can also be useful to keep the creatures constrained to a particular location.

In the first section of the animation, *Multiple Uses for Flocking Algorithms* [Gamlen 95a], moths swarming around a lamp are simulated. My flocking algorithm was slightly modified so that the moths perceived the lamp to be the centre of their swarm, ensured that they would not fly away. Collisions with the lamp are avoided by the use of a spherical "force field" which repels the moths. This force field and the turning angle constraint (see section 3.7 Physical Simulation) keeps the moths from trying to return immediately to the lamp.

Although fish in nature use slightly different behaviours than those in my flocking algorithm, a reasonable simulation can still be generated. The rules used by Tu [Tu 94] to simulate fish are:

1. Each individual maintains an empty space around itself. In general, only one neighbour at a time is at the preferred distance from a particular fish.
2. Fishes tend to keep their neighbourhood at a particular preferred angle with respect to their body angle.

Most schools of fish are organised on the same lines of preferred distance and angle [URL 2], [Tu 94]. Fish use organs called lateral lines to sense pressure variations in the water. This organ is used to sense the distance to other fish and to detect pressure vortices that can be used to increase swimming efficiency [Triantafyllou 95]. Other experiments with fish seemed to show that vision provided an attractive force between school members while the lateral lines provided a repulsive force [Partridge 82]. Most of the studies done on schools of fish have been on species of fish that are consumed. Schooling among these species reduces the probability of being eaten, since the probability of detection is reduced by forming a school. The school may also be mistaken for a larger organism. As a school gets larger a greater percentage of its members are protected within the "shell" of the surrounding members. Some predators also form schools, not to gain a protective advantage but to increase their food searching area by spreading out, until the other members are barely in the sight. Then if one member finds food the other members can take advantage of the find [URL 2]. To add some realism to the school of fish in my animation [Gamlen 95a] a number of techniques were used:

1. the collision radius of the fish was made much smaller than that of the moths, forcing them to school closer together. An unexpected outcome of this change was that the fish looked as if they were moving their tail fins in a swimming motion, when in fact their bodies were completely rigid. This illusion is due to fish trying to turn into the school centre and then turning away as it tries to avoid a collision with another fish.

2. The turning angle constraint was set at 5 degrees so that the fish would not try any unrealistic turns.

3. The path that they follow was drawn to mimic the motion of real fish.

The third example on the video [Gamlen 95a] shows a number of dragonflies that begin by swarming around some reeds and then fly off in a flock along a river. By the end of the animation the dragonflies are nearly all flying in perfect formation. This was achieved by varying the velocity of the path object. At the beginning the path object is stationary and thus the dragonflies swarm around it. As the path object accelerates the dragonflies follow, at first looking like a mobile swarm then a flock and finally the motion changes to look similar to aircraft flying in formation. Throughout the animation the dragonflies avoid a log, rocks and a clump of reeds which cause them to disperse and reform. This animation in no way parallels real dragonflies and was done to show how a flock could change its behaviour over time.

The final animation [Gamlen 95a], is an example of a completely synthetic flocking behaviour. It shows the interaction between two flocks of simple ellipsoidal creatures. The creatures have subtle markings, one flock has aqua veins on a green body while the other has the colours reversed. Although the two flocks have a reasonably complex double helix path to follow any evidence of this path is lost as the flocks combine but see each other as "static objects" rather than members of the same flock. This animation suggests a future series to investigate a number of flocking motions and behaviours not seen in nature (see section 4.3 Extensions and Future Work).

Use of my flocking algorithm is not only restricted to creatures that can travel in three dimensions. By limiting creatures movements to two dimensions, herds can be simulated. In natural herds such as wildebeests and buffalo, the individuals view of other herd members is considerably diminished by occlusion due to the 2 dimensional structure of the herd. To simulate this with the flocking algorithm, the creature's vision radius must be in the order of 2 to 4 creature lengths. Reducing a herd member's vision can present problems. If a creature is separated from the herd it may not be able to "see" the herd and will not know which direction to move in to return to the herd. This problem can be solved by either increasing the vision radius, at the loss of some degree of realism from the simulation, or creating a new behaviour to allow the creature to re-find the herd. The behaviour I implemented was to have creatures that couldn't see any other creatures, move toward the path object.

Another feature of natural herds is that the distance between herd members will decrease (the herd will bunch up) in response to danger [Calvin 90]. It is possible to simulate this behaviour be decreasing the creatures collision radius. The size of the collision radius to simulate this behaviour could be linked to the distance between the flock and a predator. A minimum and maximum distance limit would need to be set. If the predator was outside the maximum distance then the creatures would ignore it.

Once the predator comes within this distance the algorithm would decrease the creatures collision radius and they would bunch together. The creatures would stop bunching up when the predator reached a certain minimum distance so that they would not intersect with each other (see section 3.1 Collision Avoidance).

Simulating the raiding behaviour of army ants presented a challenge for my flocking algorithm. Army ants of different species have different raiding behaviour which follows a roughly similar procedure. Ants search for food while leaving a chemical trail, when food is found they spread out to gather the food (which may be another insect colony) and when an ant has gathered some food it returns to the nest by following the chemical trail. Trail formation and simulation in ants is a well studied area. Army ants, like other self-organising insect societies, exist in large colonies.

"If 100 ants are placed on a flat surface, they will walk around in never decreasing circles until they die of exhaustion. In extremely high numbers, however it is a different story." [Franks 89]

Artificial life researchers have used genetic algorithms to "breed" generations of simulated ants that become more adept at following trails, with each new generation [Levy 92]. However to date nobody has simulated the trail forming behaviour of ants [Levy 92]. It has been suggested by biologists that the simulated ants probably failed to leave trails because the food in the simulated environment was distributed in small randomly spaced clumps. Ants in nature usually do not leave trails in similar circumstances [Levy 92]. Army ants move along a pre-marked trail fairly rapidly, until they reach new territory where upon they will advance only a few centimetres while marking the new trail and then turn around to rejoin the main swarm of ants [Schneirla 71]. To simulate this type of behaviour in my animation *Formication* [Gamlen 95b], the path objects velocity was kept quite small so that the ants would swarm around the front of the flock in an approximation of the above-mentioned behaviour. This did not produce a very realistic simulation, however the animation is more concerned with complex global emergent behaviour arising from simple local interactions than with realism.

4 Other Uses of Flocking Algorithms

As can be seen, flocks of birds are not the only type of motion that can be simulated using a flocking algorithm. Flocking algorithms could be used to control a crowd of synthetic actors in a feature film. At the Graphics, Visualisation and Usability Centre of the Georgia Institute of Technology [URL 1], Jessica Hodgins and students are using an herding algorithm to simulate the movement of groups of robots and cyclists. This work is very focused on the dynamics of the herd members. By comparing simulations with a dynamically controlled herd and a herd of point masses that have perfect velocity control they have shown that more realistic results can be generated using dynamical control. However since the dynamically controlled herd members can only alter their movement while their "legs" are in contact with the ground the herd was not as robust as the point masses and some collisions occurred [URL 1].

Simulated flocks can play an important role in understanding natural flocks. The behaviour of simulated flocks can be interpreted much more easily than that of natural flocks. Simulations can be run many times changing a single parameter such as visual perception of flock members and observing how this effects the flock. However biologists have been slow to accept the validity of simulated flocking experiments [Levy 92]. In the fledgling science of Artificial Life there are many groups working with systems similar to flocking algorithms. In these systems, complex global behaviour emerges from the interaction of many simple local agents. To facilitate a standard amongst researchers the Santa Fe Institute has been developing a simulation system which they call The Swarm [URL 4], [Hiebeler 94]. The flocking algorithm could be easily implemented within The Swarm Simulation System. Many other phenomena can be simulated using the Swarm system:

1. *Economic models, with economic agents interacting with each other through a market.*
2. *Social insects building nests, foraging for food, or performing other actions.*
3. *Molecules interacting in artificial chemistry.*

4. Traffic simulation.
5. Ecological simulations.
6. Artificial intelligence applications.
7. General studies of complex systems, artificial life, emergent phenomena, etc. [Hiebeler 94]

This powerful system will help to unify researchers in different fields and may give Artificial Life a higher standing in the biological sciences.

4.1 Computing Environment

My initial locally oriented flocking algorithm was developed on the Amiga platform, firstly 68030 and later, a 68040 based system. The Amiga is a good low cost graphics platform and about the only personal computer capable of playing full screen animations at video rates (25 frames per second and faster). The flocking code was written in RPL, a FORTH like language which is part of the commercial package REAL3D. REAL3D was used because of its a total integration, eliminating the need to program a three dimensional renderer or to integrate flocking data with other programs. The lack of debugging facilities in REAL3D made programming difficult for complex sequences. The final animation work was rendered in 24 bit colour at a resolution of 768 by 576 pixel's on the Amiga and test animations were viewed on a Silicon Graphics Indigo work station. The final animations were output to Sony video disk before being edited onto U-MATIC SP video tape. The rendering process was quite time consuming. The dragonfly animation [Gamlen 95a] took almost 2 weeks to render. This animation runs for 48 seconds in real time and contains no complex rendering attributes, such as shadows or multiple light sources. The wire frame test run of the same animation, 1200 frames, took 6 hours to render although there are only 19 dragonflies in the swarm/flock.

4.2 Algorithmic Efficiency

The standard implementation of my flocking algorithm grows in complexity and time in proportion to the square of the flock's population $O(N^2)$. This is because each creature must calculate data about every other creature even if only to decide that some creatures cannot be "seen" (see section 3.4 A Flock Member's View of the World) and should not be involved in further calculations. When the flock size increases, the amount of time taken to calculate the position of the flock increases exponentially. To counteract this characteristic of the algorithm I used the Monte Carlo method. Each creature only has to look at a predefined number of other creatures that are chosen at random. This technique made it possible to render an animation containing 104 fish in 10 hours without any perceivable difference in realism from the standard algorithm. This method makes the flocking algorithm into a linear time problem $O(N)$ as long as the random sample is keep constant. However there is a problem in that the collision avoidance section of the algorithm is still in the order $O(N^2)$. Reynolds investigated dynamical partitioning of the flock to solve this problem. A creature would keep a list of its neighbours and when it moved closer to one of its neighbours it would only need to test the list of it's neighbour to avoid collisions and check for new members in it's neighbourhood [Reynolds 87]. Another method is Girard's PODA system [Girard 85] which uses a system of incremental collision detection. This uses a partial solution that described the situation just a moment before a change in the flock. The algorithm need only be concerned about the changes and so can run much faster, assuming that the incremental changes are small. This algorithm is apparently linear time $O(N)$ for typical cases [Reynolds 87]. The system of linear differential equations used by Amkraut [Amkraut 90] achieves linear time at the cost of not be able to simulate vision.

The algorithm could be speeded up further by using parallel processing hardware. Each flock member could be assigned a processor on a multiple CPU computer. This type of distributed processing parallels nature, where each bird in the flock can be considered as a separate information processor.

4.3 Extensions and Future Work

For the animation *Formication* [Gamlen 95b] I started using dynamic simulation to control the legs of ants. The Amiga computer was not capable of calculating in a reasonable time, so key frames from a single dynamic walk cycle were used to generate the motion for the ants legs. The dynamical leg control algorithm was largely based on based Rodney Brooks' Subsumption Architecture for robotics [Brooks 89], [Badler 91], [Franke 95], [URL 5]. The basic premise of Subsumption architecture is that it involves a number of low level computer processes (finite state machines) that operate in parallel and can be subsumed by higher level processes. For example the walking behaviour of Brooks' six legged robots is not programmed but emerges from a network of simple reflexive behaviours. This fits in well with the emergent flocking behaviour as it can act as a higher level process and subsume the lower level walking dynamics.

To take the step from computer simulation to the real world I have constructed a Braitenberg vehicle [Braitenberg 84], [Cheeseman 88]. This vehicle is not controlled by a digital computer. It uses simple analog connections between two light detectors and two drive motors and is capable of simple behaviours, such as light avoidance and light following. A number of these vehicles acting together would produce an interesting experiment and could be the basis for co-operative robotics [URL 2].

Porting the code to run on faster machines such as Silicon Graphics work stations or fast Pentium based machines would allow the implementation of a number of the more advanced features mentioned in this thesis, such as more complex environments, larger numbers of creatures, together with obstacles and features such as wind and ocean currents. Individual creatures could be made more individual by varying their size, shape, strength and other physical attributes. Further experimentation with flock like behaviours that are not found in nature, such as multiple flock interaction and using other behaviours to create flocking should prove useful.

Conclusion

The emergent model of flocking presented in this thesis is capable of producing animations that evoke a sense of the natural world. This model can be executed on simple personal computers, although more realistic animation could be generated using more complicated dynamical simulation on more sophisticated computers.

The level of realism generated by the algorithm allows an animator to change parameters to create a number of different flock-like behaviours. With advances in flocking algorithms and general purpose simulators like The Swarm, flocking algorithms will have a greater role in computer animation and the study of complex behaviour.

Bibliography

[Amkraut 90] "Eurhythmy: Concept and Process", Susan Amkraut and Michael Girard, Journal of Computer Visualization and Computer Animation, VOL 1: 15-17, 1990.

[Badler 91] "Making Them Move", Eds. Norman I. Badler, Brian A. Barsky and David Zelter, Morgan Kaufman Publishers, Inc, 1991.

[Braitenberg 84] "Vehicles: Experiments in Synthetic Psychology", Valentino Braitenberg, MIT Press, 1984.

[Brooks 89] "A Robot that Walks: Emergent Behaviors from Carefully Evolved Network", Rodney Brooks, Artificial Intelligence Memo 1091, Massachusetts Institute of Technology (February 1989).

[Calvin 90] "The ascent of mind: ice age climate & evolution of intelligence.", Calvin, William H, Bantam, 1991.

[Cheeseman 88] "Experiment with 'Synthetic Psychology' Build a Braitenberg vehicle", Electronics Australia: 60 - 64, March 1988.

[Crutchfield 94] "The Calculi of Emergence", James P. Crutchfield, Physica D (1994)special issue on the proceedings of the Oji International Seminar, Complex Systems - from Complex dynamics to Artifcial Reality, held 5 - 9 April 1993, Numazu, Japan.

[Franke 95] "Brooks, Subsumption, and Mobile Robots", Jerry Franke and John R. Surdu, Departmental Technical Report number 95-061, Computer Science Department, Florida State University, April 1995.

[Franks 89] "Army Ants: A Collective Intelligence", Nigel R. Franks, American Scientist, vol. 77, no. 2, pp. 139-145, March 1989.

[Girard 85] "Computational Modeling for Computer Animation of Legged Figures", Michael Girard and AnthonyA. Maciejewski, ACM SIGGRAPH Vol 19, #3:263 - 270 July 1985.

[Hiebeler 94] "The Swarm Simulation System and Individual-based Modeling", David Hiebeler, proceedings of Decision Support 2001: Advanced Technology for Natural Resource Management, Toronto, September 1994.

[Kawata 94] "From artificial individuals to global patterns", Masakado Kawata and Yukihiko Toquenaga,
TREE 9(11), 1994, 417-421.

[Levey 92] "Artificial Life", Steven Levey, Vintage Books, NY, 1992.

[Partridge 82] "The Structure and Function of Fish Schools", Brian L. Partridge, Scientific American, June 1982.

[Reeves 83] "Particle Systems-A technique for Modeling a Class of Fuzzy Objects", William T. Reeves, ACM SIGGRAPH Vol 17, #3:359 - 376 July 1983.

[Reynolds 87] "Flocks, Herds and Schools: A Distributed Model", Craig W. Reynolds, ACM SIGGRAPH Vol 21, #4:25 - 34 July 1987.

[Schneirla 71] "Army Ants", T. C. Schneirla, W. H. Freeman and Co., 1971.

[Triantafyllou 95] "An Efficient Swimming Machine", Michael Triantafyllou and George Triantafyllou, Scientific American, March 1995.

[Tu 94] "Artificial Fishes: Physics, Locomotion, Perception, Behaviour", Xiaoyuan Tu and Demetri Terzopoulos, Proceedings of SIGGRAPH 94:43 - 50, July 1994.

Videos references

[Amkraut 85] "Eurythmy", Amkraut, S., Girard, M., Karl, G., Issue #21, item 2, SIGGRAPH Film & Video Show, 1985.

[Gamlen 94] "wild Planet", Andrew Gamlen, produced at The Australian Centre for the Arts and Technology, 1994.

[Gamlen 95a] "Multiple uses for Flocking Algorithms", Andrew Gamlen, produced at The Australian Centre for the Arts and Technology, 1995.

[Gamlen 95b] "Formication", Andrew Gamlen, produced at The Australian Centre for the Arts and Technology, 1995.

[Symbolic 87] "Stanley and Stella: Breaking the Ice", Symbolics Graphics and Whitney Demos Productions, Issue #36, item 1, SIGGRAPH Film & Video Show, 1987.

HTML references

[URL 1] Graphics, Visualization, and Usability Center Animation
http://www.cc.gatech.edu/gvu/animation/Animation.html

[URL 2] The Armyant Project
http://armyant.ee.vt.edu/armyant-project.html

[URL 3] Boids, Flocks, Herds, and Schools:
http://reality.sgi.com/employees/craig/boids.html

[URL 4] Swarm Web Pages
http://www.santafe.edu/projects/swarm/

[URL 5] Thing
http://piglet.cs.umass.edu:4321/~macdonal/thing.html

Note: The first section of the html references is a title. The line begining http:// is the universal resource locator (URL).

Appendix A
RPL Code

```
"r3d2:rpl/sys/vectors.rpl" LOAD
"r3d2:rpl/sys/objects.rpl" LOAD
"r3d2:rpl/sys/tags.rpl"    LOAD

VARIABLE obja VARIABLE objs VARIABLE obj1 VARIABLE obj2
VARIABLE obs
VARIABLE addstor
VARIABLE locary VARIABLE mem
VARIABLE current VARIABLE loci
VARIABLE nof VARIABLE fa
VARIABLE col-flag VARIABLE car-flag

( floating point variables

FVARIABLE midbias FVARIABLE avbias FVARIABLE speed
FVARIABLE locrad FVARIABLE angel FVARIABLE con-ang
FVARIABLE pathbias FVARIABLE collrad
FVARIABLE min-vel FVARIABLE max-vel
FVARIABLE sang FVARIABLE lenstor

FVARIABLE tcount FVARIABLE period

( Vector variables

VARIABLE center VARIABLE substor
VARIABLE vtmp VARIABLE velstor
VARIABLE vstor VARIABLE vswap

VARIABLE newv VARIABLE oldv

VARIABLE one VARIABLE two VARIABLE tre

( flock objects are referenced by name
( O_FINDWILD will find all occurences of fly

: GetTargets
"/level/ant*" O_FINDWILD
;

( All flock objects are place in the array obja

: array-targets
BEGIN
   DUP
WHILE
   objs @ 4 * obja @ + !
   objs DUP @ 1 + SWAP !
REPEAT
   DROP ( EOobjs marker
;

( get "static objects by name ellip*

: array-cars
"/level/cars/ellip*" O_FINDWILD
BEGIN
   DUP
```

```
WHILE
   nof @ 4 * fa @ + !
   nof DUP @ 1 + SWAP !
REPEAT
   DROP ( EOobjs marker
;

( Keep a copy of each objects previous velocity

: remember
objs @ 0 DO
   I 4 * obja @ + @
   "VVEL" O_FINDTAG 4 + @ V@   ( get velocity
   I 24 * mem @ + V!
 LOOP
;

( Calculate the local flock centre using flock members from
( the array loci.  Result is stored in vtmp

: FindMid
0 0 0 vtmp @ V!  ( initialize average
 loci @ 0 DO
   I 4 * locary @ + @ obja @ + @
   iOP_COG O_PROP
   vtmp @ V@ VADD
   vtmp @ V!
 LOOP
vtmp @ V@ 1.0 loci @ F/ VMUL ( HOPEFULLY I HAVE AN AVERAGE
vtmp @ V!
;

( fixedmid is used to give a swarm a stationary point to
( swarm about, used in place of FindMid.  Used in moth
( animation

: fixedmid
"/level/center*" O_FIND
iOP_COG O_PROP center @ V!
;

( calculate flock centring vector for current object

: mts
   FindMid
 ( fixedmid
   vtmp @ V@                    ( get center
   current @ obja @ + @         ( get object address
    iOP_COG O_PROP              ( get COG
    VSUB                        ( vector from object to center
   current @ obja @ + @
   "VVEL" O_FINDTAG
   4 + @ DUP addstor ! V@       ( get curren vel
    VSUB midbias F@ VMUL        ( scale vector from current to
      center @                 ( centre by midbias
   addstor @ V@ VADD            ( add to origonal vector
   addstor @ V!                 ( put it back
;

( Inverse collision routine. Used for insect like behaviour

: colls
obj1 @ iOP_COG O_PROP
```

```
obj2 @ iOP_COG O_PROP
VSUB VLEN
collrad F@ F<
   IF
      obj1 @ iOP_COG O_PROP
      obj2 @ iOP_COG O_PROP
      VSUB substor @ V!
      obj1 @ "VVEL" O_FINDTAG
      4 + @ DUP V@
      substor @ V@ VADD
      4 ROLL V!
      obj2 @ "VVEL" O_FINDTAG
      4 + @ DUP V@
      substor @ V@ -1.0 VMUL VADD
      4 ROLL V!
   ENDIF
;

( Inverse square collision routine

: colls-inv
obj1 @ iOP_COG O_PROP
obj2 @ iOP_COG O_PROP
VSUB VDUP vstor @ V! VLEN
collrad F@ F<
   IF
      vstor @ V@ VNORM collrad F@ VMUL
        vstor @ V@ VSUB VDUP vstor @ V! VNORM vstor @ V@ VLENSQRT VMUL vsto
@ V!
      obj1 @ "VVEL" O_FINDTAG
      4 + @ DUP addstor ! V@
      vstor @ V@ VADD addstor @ V!
        1 col-flag !
   ENDIF
;

( collision checking loop to make sure flock members check
( for collision with all other members.

: coll-loop
current @ obja @ + @ obj1 !
   objs @ 0 DO
            I 4 * obja @ + @ obj2 !
            obj1 @ obj2 @ <>
            IF
                  colls-inv ( change to colls for insect
            ENDIF       ( like behaviour
    LOOP
;

( Check for collisions with static objects

: check-cars
 current @ obja @ + @ obj1 !
   nof @ 0 DO
        I 4 * fa @ + @ DUP obj2 !       ( get the static object
                                        ( from feild
            iOP_SIZE O_PROP  2.1 F*     ( get cars size / 2 = radius
            collrad F@ F+ lenstor F!    ( add trolleys collision radius
            obj1 @ iOP_COG O_PROP
            obj2 @ iOP_COG O_PROP 2.1 F*
            VSUB VDUP vstor @ V! VLEN
            lenstor F@ F<
```

```
                    IF
                      vstor @ V@ VNORM lenstor F@ VMUL
                      vstor @ V@ VSUB vstor @ V!
                      objl @ "VVEL" O_FINDTAG
                      4 + @ DUP addstor ! V@
                      vstor @ V@ VADD addstor @ V! 1 car-flag F!
                    ENDIF
     LOOP
;

( Calculate the average vector for all objects in the array loci.
( i.e all objcts local to/can be seen by current.

: av-vel
0 0 0
 loci @ 0 DO
   I 4 * locary @ + @ 6 * mem @ + V@ VADD
 LOOP
1.0  loci @ F/ VMUL
velstor @ V!
;

( Calculate velocity matching vector

: do-av
av-vel
   velstor @ V@ current @ obja @ + @
   "VVEL" O_FINDTAG 4 + @ DUP addstor ! V@
   VSUB avbias F@ VMUL addstor @ V@ VADD
   addstor @ V!
;

( Calculate path following vector

: patb
 "/level/path*" O_FINDWILD
 t F@ u F@ v F@ O_EVAL
   ( eval pnt on curve in time
   current @ obja @ + @          ( get object address
    iOP_COG O_PROP               ( get COG
    VSUB                         ( vector from obja to curve
   current @ obja @ + @
   "VVEL" O_FINDTAG
   4 + @ DUP addstor ! V@        ( get curren vel
    VSUB pathbias F@ VMUL        ( scale vector from current to curve
                                 ( by pathbias
   addstor @ V@ VADD             ( average between current
   addstor @ V!                  ( put it back
DROP
;

( Calculate path following vector with no bias.
( Used in simulating herds with short visual ranges.
( If a herd member can't see the flock it heads for the path.

: pat
 "/level/path*" O_FINDWILD
 t F@ u F@ v F@ O_EVAL
 ( eval pnt on curve in time
 current @ obja @ + @ iOP_COG O_PROP VSUB        ( V to point on curve
 current @ obja @ + @ "VVEL" O_FINDTAG 4 + @ V! ( DUP addstor ! V@ VSUB
 DROP
;
```

```
( to experiment. If a flock member can't see other members it slows down.

: slowdown
 vtmp @ V@ current @ obja @ + @ DUP iOP_COG O_PROP VSUB ( V to last loc center @
 0.1 VMUL 4 ROLL "VVEL" O_FINDTAG 4 + @ DUP addstor ! V@ VADD
 addstor @ V!
;


( orient object geometry from normal to direction of velocity vector.

: dir
objs @ 0 DO
 I 4 * obja @ + @
 0 SWAP DUP
 iOP_COG O_PROP          ( GET OBJECTS CENTER OF GRAVITY AS ROT CENTER
 4 PICK
 "VVEL" O_FINDTAG
 4 + @ V@ VNORM VDUP one @ V! DUP -1.0 F* 0.0 5 PICK VNORM VDUP tre @ V!
 VCROSS VNORM one @ V@ tre @ V@
 0
 M_ROTATE
LOOP
;

( return object to normal orientation.

: ndir
objs @ 0 DO
 I 4 * obja @ + @
 0 SWAP DUP
 iOP_COG O_PROP          ( GET OBJECTS CENTER OF GRAVITY AS ROT CENTER
 4 PICK
 "VVEL" O_FINDTAG
 4 + @ V@ VNORM VDUP one @ V! DUP -1.0 F* 0.0 5 PICK VNORM VDUP tre @ V!
 VCROSS VNORM VDUP  two @ V! DROP DROP
               one @ V@ DROP DROP
               tre @ V@ DROP DROP
               two @ V@ ROT DROP DROP
               one @ V@ ROT DROP DROP
               tre @ V@ ROT DROP DROP
               two @ V@ ROT ROT DROP DROP
               one @ V@ ROT ROT DROP DROP
               tre @ V@ ROT ROT DROP DROP
 0
 M_ROTATE
LOOP
;

( track flock centre or track named flock member by removing comments.
: track
( object camera tracking
 "/level/camera/aim*" O_FINDWILD
( track first selected obj
(  obja @ @ iOP_COG O_PROP
( track average center @
( vtmp @ V@
  0 0 0   ( initialize average
 objs @ 0 DO
  I 4 * obja @ + @
   iOP_COG O_PROP
    VADD
    LOOP
 1.0 objs @ F/ VMUL ( HOPEFULLY I HAVE AN AVERAGE
 0 M_MOVECOG
;

( Stop creatures turning to fast

: constrain
 objs @ 0 DO
   mem @ I 24 * + V@ oldv @ V!
  I 4 * obja @ + @
   "VVEL" O_FINDTAG 4 + @      ( find add of cur. vel tag
   DUP addstor ! V@ VDUP newv @ V!
   oldv @ V@
   VDOT F+ F+                  ( VDOT needs to extra F+s
   oldv @ V@ VLEN
   newv @ V@ VLEN F* F/ ACOS
   DUP sang F!
   con-ang F@ F>
   IF
      newv @ V@ oldv @ V@ VSUB
      con-ang F@ sang F@ F/
      VMUL oldv @ V@ VADD
      addstor @ V!               ( replace current vel with constraint
   ENDIF
 LOOP
;

( Stop creatures going to fast or to slow

: vel-limits
objs @ 0 DO
   I 4 * obja @ + @ "VVEL" O_FINDTAG 4 + @ DUP addstor ! V@ VLEN DUP
   max-vel F@ F>
   IF
      addstor @ V@ VNORM max-vel F@ VMUL addstor @ V! DROP
   ELSE
      min-vel F@ F<
   IF
      addstor @ V@ VNORM min-vel F@ VMUL addstor @ V!
   ENDIF
   ENDIF
LOOP
;

( constraint herds to 2D

: flat
objs @ 0 DO   ( keep the herd flat
   I 4 * obja @ + @ "VVEL" O_FINDTAG
   4 + @ DUP V@
   SWAP DROP 0 SWAP
   4 ROLL V!
LOOP
;

( move objects "along their velocitiy vectors

: process
objs @ 0 DO
   0
   I 4 * obja @ + @ DUP
  "VVEL" O_FINDTAG 4 + @ V@
```

```
   4 PICK iOP_COG O_PROP VADD
   0 M_MOVECOG
LOOP
;

( initially multiply all objects velocities by a constant.
( to speed up or slow down all objects

: sped
objs @ 0 DO
   I 4 * obja @ + @
  "VVEL" O_FINDTAG 4 + @ DUP V@    ( get vel
   speed F@ VMUL                   ( multiply by factor
   4 ROLL V!                       ( put the vel back
LOOP
;

( routines to add new objects to the flock over time

: init-add
( adding time
 0.2 obs @ objs @ - F/ DUP period F!
  tcount F!
;
: add-obj
( t F@ tcount F@ F>
      IF
     objs @ obs @ <
      IF
       tcount F@ period F@ F+ tcount F!
       objs @ 1 + objs !
      ENDIF
      ENDIF
;

( initialise all variables and user definable parameters

: init
   GetTargets
   DEPTH 4 * 0 MEM_ALLOC  obja   !
   DEPTH 4 * 0 MEM_ALLOC  locary !
   DEPTH 32 * 0 MEM_ALLOC  mem    !
      32 0 MEM_ALLOC  vtmp    !
      32 0 MEM_ALLOC  velstor !
      32 0 MEM_ALLOC  substor  !
      32 0 MEM_ALLOC  vstor   !
      32 0 MEM_ALLOC  vswap   !
      32 0 MEM_ALLOC  one     !
      32 0 MEM_ALLOC  two     !
      32 0 MEM_ALLOC  tre    !
      32 0 MEM_ALLOC  newv    !
      32 0 MEM_ALLOC  oldv   !
      32 0 MEM_ALLOC  center  !
   array-targets

( uncomment special words when needed in animations

   ( array-cars
   ( init-add
   ( fixedmid

( parameter input disabled because it is easier to change only the
( parameters that need changing by altering the code
    0.55 collrad F! ( "Collision Radius" GET_FLT DROP    drop the return value
    0.09  midbias F! ( "Middle Bias"     GET_FLT DROP    0 = bad value
    0.45 avbias  F! ( "Velocity Match"   GET_FLT DROP    1 = flt O.K.
    30   locrad  F! ( "Local Radius"     GET_FLT DROP
    110  angel   F! ( "Angle of View"    GET_FLT DROP
    6 con-ang F!    ( "constrain angle"  GET_FLT DROP
    0.05 min-vel F! ( "minimum velocity" GET_FLT DROP
    0.1  max-vel F! ( "maximum velocity" GET_FLT DROP
    0.02 speed   F! ( "Speed multiply"   GET_FLT DROP
    0.09 pathbias F! ( "path bias"       GET_FLT DROP

    angel F@ >RAD angel F!
    con-ang F@ >RAD con-ang F!
  sped
  flat
  dir
;

( release memory and print out a summary of user definable parameters

: clean
obja   @ 32768 MEM_FREE
locary @ 32768 MEM_FREE
mem    @ 65536 MEM_FREE
fa     @ 32768 MEM_FREE
0 objs ! 0 nof !
collrad F@ "Collision Radius"     PUTS F.
   midbias F@ "Middle Bias"       PUTS F.
   avbias  F@ "Velocity Match"    PUTS F.
   locrad  F@ "Local Radius"      PUTS F.
   angel   F@ "Angle of View"     PUTS F.
   con-ang F@ "constrain angle"   PUTS F.
   min-vel F@ "minimum velocity"  PUTS F.
   max-vel F@ "maximum velocity"  PUTS F.
   speed   F@ "Speed multiply"    PUTS F.
;

( Main control loop, calculates which members are local

: control
objs @ 0 =
 IF
   init
 ENDIF
 ndir
 remember
 objs @ 0 DO
   I 4 *  current !
   0 loci !
   objs @ 0 DO
    I 4 * obja @ + @ DUP DUP
    current @ obja @ + @ <> IF
      current @ obja @ + @ SWAP
      iOP_COG O_PROP 4 ROLL iOP_COG O_PROP VSUB VDUP newv V! VLEN
      locrad F@ F< IF      ( check vision radius
         current @ obja @ + @ "VVEL" O_FINDTAG 4 + @ V@ VDUP oldv V!
         newv V@ VDOT F+ F+
         newv V@ VLEN oldv V@ VLEN F* F/ ACOS
         angel F@ F< IF   ( check vision angle
            I 4 * loci @ 4 * locary @ + ! DROP
            loci DUP @ 1 + SWAP !
               ELSE
```

```
                  DROP
               ENDIF
         ELSE
            DROP
         ENDIF
   ELSE
     DROP DROP
    ENDIF
   LOOP
loci @ 0 >
IF
  0 car-flag !
  0 col-flag !
  check-cars 0 car-flag @ =
  IF    ( 1 = static object collision avoidance
    coll-loop 0 col-flag @ =
    IF ( 1 = flock object collision avoidance
      patb    ( path following
      mts     ( flock centring
      do-av   ( velocity matching
    ENDIF
  ENDIF
ELSE
  pat ( if no local objs follow path
ENDIF
 LOOP
constrain
vel-limits
dir

( call camera tracking routines if necessary

( track
( follow
 process

 t F@ 1 F= IF clean .S ENDIF  ( cleanup when at animation end time = 1
1 ( return 1 to REAL3D to say OK
;

( Create animation method hook for REAL3D
& control "TRY" MTH_CREATE DROP
```