Simulation of Unmanned Air Vehicle Flocking

Neil R. Watson, Nigel W. John, William J. Crowther
Manchester Visualization Centre, University of Manchester, UK
nrw@cs.man.ac.uk, n.w.john@man.ac.uk, w.j.crowther@man.ac.uk

Abstract

Flocking style behaviour is commonplace in nature. It provides a number of benefits for groups of creatures on the move at little cost. Unmanned Air Vehicles are becoming increasingly popular for a variety of applications and are often used in groups. Manually controlling many UAVs at high densities can be difficult. The simple, efficient and safe nature of flocking as an automated flight control method, potentially make it an ideal solution for manoeuvring groups of UAVs. This paper investigates the implementation of a real-time graphical simulation of flocking UAVs. Of particular interest is simulating simplified realistic aircraft dynamics in real-time and allowing flocking rule parameters and weightings to be interactively adjusted in order to investigate the effects they have on flocking behaviour. The paper describes work carried out for an MSc. research project, which is now being utilised for the authors Ph.D research.

1. Introduction

A flock can be simply described as a group of several individuals (known as boids) clustered together, moving with a common velocity. In nature there are numerous examples of this sort of behaviour including flocks of birds, herds of land animals, swarms of insects and schools of fish. When looking at the motion of a flock of birds it seems as though there is some kind of centralised control. The flock appears to move as one fluid object, yet despite the impressive result, the motion must be simply the aggregate result of each individual making basic movement decisions based upon its local surroundings.

In the past there has been a fair amount of interest in flocking and how to simulate it. Most notable is the research carried out by Craig Reynolds [1]. Most early flocking simulations were laboriously scripted by animators creating a predetermined flight path for each boid. Instead Reynolds created a system whereby a set of simple flocking rules were applied for each boid to automatically control its motion. This method not only produced a simulation that was a lot quicker to develop, since the motion were automated, but also resulted in flocking behaviour that looked much more realistic and natural.

Creatures flock together for a number of reasons, including protection from predators, improved food search and social interaction, but why would flocking behaviour be desirable in unmanned air vehicles (UAV)? UAVs are becoming increasingly popular for both civil and military applications. They are attractive for civil uses such as aerial photography or crop dusting as they are cheaper to buy and operate than manned air vehicles. Their military uses include surveillance and even air strikes since they can be small, operate for extended periods of time and require no risk to a pilot. With ever increasing amounts of UAVs in the skies and greater use of UAVs operating closely together in teams, there is greater pressure on current air space management techniques. When flocking, birds manage to co-ordinate their movements at extremely high densities. Therefore a solution to the problem could be achieved if we were able to model groups of UAVs as birds in a flock and automatically manage their own air space based on a set of flocking rules. A flocking controlled group of UAVs would be easier to manage. For example the flock could be sent to a location by issuing a single command, upon which the flock will automatically make their own way to the location staying close together, whilst avoiding collisions with each other and other obstacles. Once there they could circle safely awaiting further commands. A flock offers advantages over traditional automated formation flying or “follow the leader”, since it has a fluid and highly adaptable structure rather than a rigid fixed one. Since no flock member relies upon any other to operate, the flock has good in-built redundancy. If one unit were to fail the rest of the flock would be unaffected.

Since Reynolds first introduced the idea of flocking rules, numerous flocking simulations have been developed [2][3][4][5]. However, the majority of these, whilst demonstrating well the principles of flocking, do not attempt to provide realistic aerodynamics or specify a particular animal or aircraft being simulated. Most simulations will model a boid as a point mass. To provide a useful simulation of flocking of UAV a more realistic model is necessary. Recent research work at The University of Manchester School of Engineering saw the development of a flocking simulation, novel in that a realistic non linear 6-degree of freedom aerodynamic model of an actual UAV (Manchester UAV) was incorporated [6]. The simulation ran off-line using

Proceedings of the Theory and Practice of Computer Graphics (TPCG'03)
0-7695-1942-3/03 $17.00 © 2003 IEEE

Matlab® and Simulink® to calculate the boid trajectories. A VRML visualisation was then used to display the simulation. Of particular interest was the development of meaningful statistical metrics that usefully quantify flocking behaviour and the investigation of the relationship between rule weighting and flocking behaviour. The work concluded that flocking does potentially offer an efficient and simplistic solution for safer management of the flight paths of large numbers of UAVs. The simulation was restrictive in that parameters have to be changed off-line, then the simulation re-run before any results can be displayed. This paper describes a project continuing on from this work and looks at the creation of a real-time version of the simulation in which flocking parameters can be controlled interactively, providing the user with immediate feedback on how the flock has been affected, whilst still providing a acceptable level of realism in terms of dynamics.

2. Background

Reynolds' flocking simulation was based upon an adapted particle system [7] in which each particle represented a single boid at a point in space. To simulate flocking this basic model needed to be adapted in two main ways. Firstly unlike particles, boids need to have orientation, since they have shape. Secondly boid behaviour is dependant on interaction with other nearby boids, whereas in most particle systems particles exert behaviour independently of one another.

In order to control the motion of each boid, Reynolds developed a small set of flocking rules, which were designed to mimic behaviours seen in natural flocks. Each rule is applied once for each boid, for each frame of animation. A rule uses information about the position of nearby boids to produce an acceleration demand. The resulting demands from each rule are then combined into a single demand, which is used to alter the velocity and position of the boid.

Reynolds suggested that three behaviour rules are essential for good flocking simulation: collision avoidance, velocity matching and flock centring. Collision avoidance can be divided into the sub-rules evasion (provides an urge for boids to accelerate away from imminent collisions with other boids) and separation (ensures boids fly at a safe distance from one another by urging boids to accelerate away from their neighbours). Velocity matching (also known as alignment) urges boids to move with a similar direction and speed. Flock centring (also known as cohesion) works in the opposite way to separation, drawing the boids closer together. Other rules can be used that, whilst not essential for a basic flocking behaviour, provide additional desirable features, for example: migration and obstacle avoidance.

2.1. Simulated Perception

In order to make use of these rules boids need to have some perception of what is around them. In a real flock, each bird senses what is around it primarily through vision. To accurately simulate vision would be complicated and prohibitively expensive in terms of processing time. Since the positions and headings of each boid are known in a simulation we could just provide these to a boid as its perception of the world. However in real flocks, birds don't have total global perception. Nearby flock mates obscure the view of others and factors such as fog, cloud or low light will also limit perception distance. Yet real flocks still manage to operate successfully under such conditions, indicating that only local perception is necessary. In fact flocking relies on local perception. For example, consider a large flock in which each boid has global perception. In trying to fulfill the urge for flock centring every boid, even those on the very outskirts of the flock, will be trying to get to the centre of the whole flock simultaneously. Global perception would also make it extremely hard for a flock to carry out manoeuvres such as splitting up to fly around a large building. A good compromise, for the boid perception model, has been found to be using a spherical area of space around each boid [1]. Each boid can only detect other boids within a certain radius (which varies depending on the rule in question). This model is simple, efficient and provides the localised perception required.

2.2. Combining Rules

Each rule returns an acceleration demand it thinks the boid should use. The simplest way to combine these demands into one acceleration vector that can be applied to the boid is to give each rule a weight value and take a weighted average of the demands. This approach is generally a sound one, but can have problems if acceleration demands of conflicting rules happen to cancel each other out.

Other, more complex, methods could also be used to combine rules. For example priority ordering of the rules, or neural networks could be trained to take values describing the boids current state and surroundings and output rule weightings or priorities accordingly.

3. Methods

In our project, two separate implementations of the same flocking software were developed for two different platforms. The first runs on a standard PC (Linux or Win32) to allow portability and convenience. The software was then ported to the SGI® Onyx® 300 and enhanced in order to take advantage of the immersive

Proceedings of the Theory and Practice of Computer Graphics (TPCG'03)
0-7695-1942-3/03 $17.00 © 2003 IEEE

large screen stereographic display facilities provided by The Visualization Immersive Projection Laboratory (VIP Lab) at MVC. The VIP lab comprises an SGI® Reality Centre powered by a six pipe SGI® Onyx® 300 visualisation server and provides active stereo via LCD shutter glasses. Both implementations were written in C and use OpenGL®. The PC software was developed using GLUT (OpenGL® Utility Toolkit) and the Onyx® software using the OpenGL Multipipe™ SDK.

3.1. System Overview

Two different flocking simulation versions were created (UAVFlock V1 and V2) which incorporate different dynamics and flocking rule systems. The boids inhabit a 3D world referenced by a right-handed co-ordinate system with the up direction along the positive z-axis. There is a ground plane in the x/y plane at z=0 and a number of axis-aligned skyscraper-like buildings close to the origin on this plane. Each UAV boid is represented by a polygonal aeroplane model. Each boid has an associated state that describes its position, orientation and movement. There are initially ten boids that start from random mid-air positions around the origin. The user can vary the number of boids.

3.2. UAVFlock V1

V1 was intended as a general flocking simulation. It features a number of flocking rules and a basic model of dynamics. The emphasis was on creating a good looking, if not wholly realistic, visualisation of UAV flocking, with variable rule parameters, which demonstrates a variety of flocking concepts.

A boid's state description includes position, velocity and acceleration vectors. These are recalculated for each boid at every frame of animation. In the V1 dynamics model a boid's new position X_i' at each frame is calculated from its current velocity V_i, current position X_i, and elapsed time since the last frame dt, using the fourth order Runge-Kutta numerical integration method (RK4). Its new velocity V_i' is then calculated from its current acceleration A_i, V_i and dt, also using RK4. The UAVs have minimum/maximum possible speeds. If required the velocity is scaled to meet these.

It is assumed that the UAVs fly oriented towards their velocity. Pitch and heading angles are calculated to rotate the UAV model to point along the velocity vector. A real plane will bank (roll about its local x-axis) by altering the positions of its ailerons, in order to accelerate laterally and initiate a turn. This is not simulated in this simple model of dynamics. In this model acceleration is calculated directly from the flocking rules, rather than controller commands (e.g. positions for ailerons, elevator, rudder, flaps, slats etc.) being calculated from the rules, then plane acceleration being calculated from the effects of those. However, to make the simulation look more realistic, a bank angle is calculated and applied to the UAV model so that it rolls as it turns. The bank angle, ϕ, of a plane in a horizontal turn can be calculated by:

$$\phi_i = \arccos\left(\frac{1}{n_i}\right) \quad (1)$$

where n_i is the boid's current load factor. The load factor is not considered in this version of the simulation, however the load factor is approximately proportional to the lateral acceleration. Therefore we can get a pseudo bank angle from the equation:

$$\phi_i = \arccos\left(\frac{1}{s \times Ay_i}\right) \quad (2)$$

where A_{yi} is the lateral component of the boid's acceleration, aligned with the boid's local y-axis and s is a scaling factor. The bank angle (1) only applies for a horizontal turn. The boid may travel vertically as well as horizontally, meaning this equation would be incorrect, however it will suffice for the approximation of bank angle that we require.

Each boid i has its acceleration calculated by the application of the flocking rules to the boid and its surroundings. In this version of the simulation there are seven rules applied: alignment, cohesion, separation, evasion, migration, floor collision avoidance and building collision avoidance. Each rule returns an acceleration demand vector (i.e. the direction it thinks the boids should accelerate in). The vectors that the rules return all have magnitude in the range [0, 1]. Each rule has its own individual parameters associated with it, as well as a single weight each. The weights can take any positive value and are all initialised as 1.0. The acceleration demands of each rule are multiplied by their associated weight and summed to form one acceleration vector Ai.

Real aeroplanes generally accelerate more gradually vertically than they can horizontally. To approximate this in a simple way, the z component of Ai is scaled down by a user selectable factor in the range [0, 1]. It is initialised to a value of 0.5. Ai is scaled down if it is greater than a maximum acceleration value. The change of Ai from one frame to the next is also limited, to ensure motion is not too oscillatory.

The rules work basically as those described by Reynolds, but with key enhancements for our application. The first rule considered is cohesion. The cohesion perception radius is initialised to 100 units. When applied to a boid, the rule calculates a weighted average of the vectors from the current boid to each of the other boids that lie within the cohesion radius and uses that as the demand. The vectors are weighted to provide acceleration

demand vector that is most influenced by close boids. The zone of perception is split up into two regions, A and B. Zone A is defined by a sphere of radius 20 units around the boid and zone B by a sphere of radius cohesion radius. Figure 1 show a cross section of perception zone. If a boid is within zone B (and not in A), the vector is scaled by 1/20, then weighted by 1 over the magnitude of this scaled vector squared. This inverse square weighting means that the influence a boid has on the resulting acceleration demand is inversely proportional to the square of the distance to that boid. The distance is scaled so that the

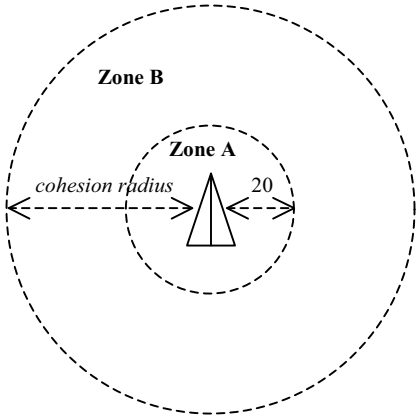

Figure 1: Boid perception zones

returned value does not become too small, too quickly as the distance increases. Figure 2 shows a graph of the magnitude of the vectors returned compared to the distance. If a boid is within zone A then the vector is weighted by 1 over the distance. This means that any boids within this zone are treated equally regardless of distance and return unit vectors (vectors of magnitude 1). This is to ensure all the vectors returned are in the range [0, 1]. If an inverse square weighting was used across the

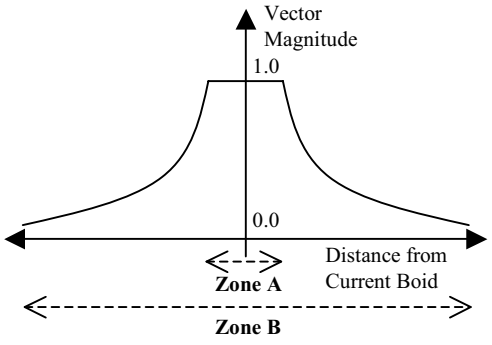

Figure 2: Boid to boid vector weighting

whole zone B sphere, including zone A, then there would be no upper bound on the weighting and it would become very large if two boids got very close to each other.

The separation rule works in exactly the same way as cohesion, but with the vectors directed from each other boid towards the current boid. It uses the same two-zone perception system. The separation radius is initialised to 50 units. The zone A radius is still 20 units.

The alignment rule again works in the same way, except the vectors calculated for each boid within perception distance are the difference between the velocity vectors of each other boid and the current boid. The alignment radius is initialised to 100 units. Zone A has a 20 unit radius.

The evasion rule has a small perception sphere of radius evasion radius, centred at the point evasion radius units directly in front of the current boid. If any boids fall within this zone, vectors from these boids to the current boid are calculated. The vector with the smallest magnitude (that of the closest boid) is normalized and returned as the evasion acceleration demand. If no boids fall within this zone a vector of magnitude 0 is returned.

For the migration rule, a vector is calculated from the current boid to the user selectable migration point. The acceleration demand towards the migration point needs to be stronger the further away the boid is from it, so the square of the distance is used to scale the demand. However to ensure that the boids do not receive too great a demand from the rule if they are far away from the migration point and to keep the rule output in line with those of the other rules it is limited at a magnitude of 1. If the rule output were unbounded, at a long distance from the migration point it would be so great that boids would effectively ignore the other rules. The migration range parameter governs at what distance from the migration point the output reaches the maximum. If the distance is less than migration range the acceleration demand vector is scaled to have magnitude distance squared over migration range squared. This scaling means that once the boids are near the migration point and the rule outputs relax and balance out, the boids will circle around the migration point at a distance dependant on the weights and parameters of this rule and the others.

To stop boids from colliding with the ground a ground collision avoidance rule is included. This takes a force field approach. The ground avoidance height parameter specifies the distance above the ground plane at which the rule starts producing an acceleration demand. This region is divided in half. If a boid is in the top half it receives an upward acceleration demand vector of magnitude 0 to 1. The magnitude is linearly proportional to how far below the floor avoidance height the boid is. If the boid is in the bottom half it is too close to the ground and must pull up, so anywhere in this region it receives an upward acceleration demand vector of the maximum strength of 1.

Using a demand vector pointing straight up could often lead to oscillatory up and down boid motion at the start of a simulation run if the vertical acceleration weight was set quite high (the user wants to simulate very agile planes). Boids would get pulled down towards a migration point then pushed up again as they get close to the ground. To

Proceedings of the Theory and Practice of Computer Graphics (TPCG'03)
0-7695-1942-3/03 $17.00 © 2003 IEEE

reduce this, instead of having the acceleration demand pointing straight up, the vector pointed in the same x/y plane direction as the boid and up at an angle of 45° to the horizontal. Having the lateral component encourages the boid to stay more level when avoiding the ground, rather than trying to head straight up.

The final rule used was building collision avoidance. To enable the boids to avoid buildings the "steer-to-avoid" (STA) approach was adopted. Boids can detect the presence of a building in front of them if it intersects with the boid's STA perception vector. This vector runs in the direction of the boids current velocity, from the boids current position with a magnitude of STA perception distance. A test for intersection between this vector and each face of each building is carried out. If there are any intersections the building face of the closest intersection is naturally chosen to avoid.

Vectors are calculated from the boid's current position to all of the points one boid length outside the edges of the face, perpendicular to the intersection point. The vectors are normalised and the difference vectors between each of them and the normalised boid current heading vector are calculated. The difference vector with the smallest magnitude is chosen, normalised and returned as the acceleration demand, as shown in Figure 3. The boid heads in the direction of an edge that is most similar to its current velocity.

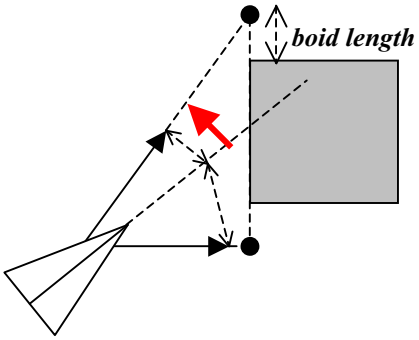

Figure 3: Steer-to-avoid obstacle avoidance

A problem with the basic STA model is that consistent obstacle avoidance is hard with only a single line of perception. A boid will not see a building slightly to its side and could clip its wing on it as it passes, or not see a building just to its side ahead and turn towards it not leaving enough time for the evasive acceleration to take affect. One solution to this problem is to use a cone of perception and calculate the intersection between this and a building face and use the resulting curve in some way to find the best way to evade the obstacle. A compromise between this and the single line approach that was experimented with in the simulation is to use four extra lines of perception around the original, aligned with the boid's local y-axis and z-axis, pointing out from the original line at a certain angle θ. If more than one of these lines intersects then there is a choice of which resulting acceleration demand to use. The resolution method selected was to choose the line that returns an acceleration demand closest to the direction of velocity

In the current simulation implementation only axis-aligned cuboid buildings are used. However, even with this simple model, obstacle avoidance behaviour for more complex shapes could be approximated by surrounding them in bounding boxes.

Some rules such as evasion, building avoidance and floor avoidance when a boid is flying close to the ground are absolutely critical for the safety of the boid. If the flocking rules were being used to control real UAV a single failure of one of these rules could easily result in a crash. Therefore these rules are given precedence over the others when they are needed. To address this in the simulation, if any one of the evasion rule, the building avoidance rule or the floor avoidance rule (when a boid is in the lower section) are activated, the weights for the non-critical rules (cohesion, velocity matching and migration) are temporarily set to 0. In a critical situation it does not matter whether the boids are heading in the same direction or are as close together as possible. Safety is paramount so these rules are ignored for the boid in question for that frame. The original values of the weights are then restored once the acceleration demand for the current boid has been calculated.

3.3. UAVFlock V2

V2 uses a more realistic model of the aeroplane dynamics. The original intention was to integrate the Simulink® model of the Manchester UAV into the program. However, recent research at Manchester [8] suggested that similar results, adequate for the purpose of investigating the effects of flocking parameters on behaviour, could be achieved with a more simplistic model. Computing a simulation run with the previous off-line simulator took approximately three times the length of the run. For a real-time system the more efficient simplistic model is definitely preferable.

The V2 flocking simulation is intended less as a visualisation of real aeroplanes flocking and more as a research tool. It incorporates a more realistic model of the way in which aeroplanes turn. However, in its current incarnation it only uses two flocking rules (cohesion and alignment) and its boids only operate in a single plane of space. Therefore, boids will happily move through each other and any other obstacles. Its intended use is for investigating the behavioural effects of different cohesion/alignment weight ratios.

In this version of the simulation boids are constrained to move at a constant speed in a horizontal plane. The boids motion is based upon a simple model of an aeroplane performing a level turn [9]. For an aeroplane flying in a straight line, at a constant altitude, with a constant speed, to make a level turn it must use its ailerons to force the aeroplane to bank at an angle ϕ, as shown in Figure 4. In its original state the lift force, L, created by the wings is equal, but opposite, to the force of the weight of the aircraft, W. Since the wings are no longer horizontal, L is no longer acting in the opposite direction to W and is inclined at the angle ϕ. For the turn to be level the vertical component of L must be equal to the weight. This leaves the horizontal component of L, F, which causes the angular acceleration that forces the aeroplane to turn.

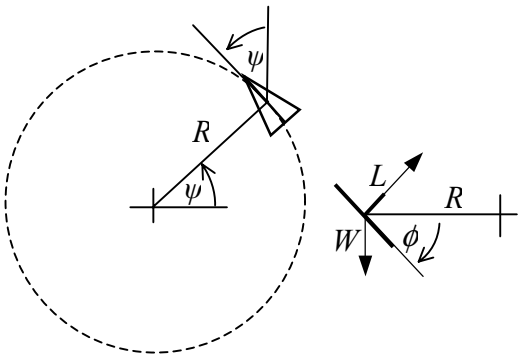

Figure 4: An aeroplane making a level turn

The planes load factor, n, is defined as:

$$n = \frac{L}{W} \tag{3}$$

It is normally quoted in terms of "g's" and is a measure of the planes angular acceleration. The bank angle of the plane can be found by:

$$\phi = \arccos\left(\frac{W}{L}\right) = \arccos\left(\frac{1}{n}\right) \tag{4}$$

A plane experiencing a load factor of 2 g's has a lift force equal to twice its weight being exerted on it and will be banked at an angle of 60°.

From Newton's second law it can be found that, for a plane moving in a circular path at a constant velocity of V_∞, the radius of the turn is:

$$R = \frac{V_\infty^{\,2}}{g\sqrt{n^2 - 1}} \tag{5}$$

and that the angular velocity (i.e. the rate of change of its heading angle ψ) is:

$$\dot{\psi} = \frac{d\psi}{dt} = \frac{g\sqrt{n^2 - 1}}{V_\infty} \tag{6}$$

where g is gravitational acceleration.

Associated with each boid i is its position vector X_i, its velocity vector V_i and its heading angle ψ_i. There are m boids in the flock. To produce the boids motion these values are recalculated at each frame of the simulation. First the new heading angle ψ_i' is calculated from $\dot{\psi}_i$, ψ_i, and elapsed time since the last frame dt, using RK4 integration.

Next the new value of $\dot{\psi}_i$ is calculated as in Equation (6). The load factor, n, is calculated first. At each frame, for each boid, n is found by applying the cohesion and alignment flocking rules. It is then used in the calculation of (4) and (6). The V1 model used the flocking rules to calculate the acceleration vector. In the same way the V2 model applies its rules to calculate the load factor.

The boid's linear velocity is then calculated by:

$$\dot{x}_i = V_\infty \cos\psi_i, \quad \dot{y}_i = V_\infty \sin\psi_i \tag{7}$$

Finally its new position x_i', y_i' is found with $\dot{x}_i$, $\dot{y}_i$, x_i, y_i, and dt, using RK4.

In this model the boids are assumed to have global perception and thus all other boids are considered in the calculation of each boid's load factor. A boid i is located at position X_i. The flock has a cohesion centroid C located at X_C. Each boid has its own cohesion vector, B_i, $(X_C - X_i)$ oriented at an angle β_i. The flock also has an alignment vector A, which is common for all boids and oriented at an angle α. The way in which the cohesion centroid and alignment vectors are calculated is dependent on the flocking type selected and is described shortly. Figure 5 illustrates the state of a boid i in terms of the vectors and angles associated with it.

The load factor for a given boid is calculated by finding the differences between the boids current heading angle and the alignment angle α, and the cohesion angle β_i. These values represent the heading angle errors $\psi_{\alpha i}$ and $\psi_{\beta i}$ (i.e. the angles the boid must turn through to be heading in the direction of the alignment angle and cohesion angle respectively). They are defined as:

$$\psi_{\alpha i} = \alpha - \psi_i, \; -\pi < \psi_{\alpha i} < \pi \tag{8}$$

$$\psi_{\beta i} = \beta_i - \psi_i, \; -\pi < \psi_{\beta i} < \pi \tag{9}$$

To find the load factor these values are first multiplied by the cohesion and alignment rule weights W_α and W_β. They are then summed together, giving the overall heading error. The final load factor value is given by:

$$n_i = 1 + \frac{1}{\pi}\left(W_\alpha \psi_{\alpha i} + W_\beta \psi_{\beta i}\right) \tag{10}$$

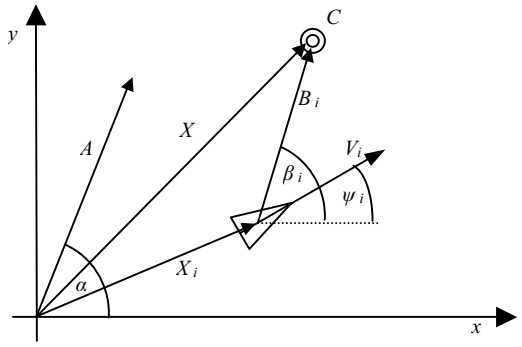

Figure 5: V2 boid vectors and angles.

A plane flying in a straight line has a load factor of 1. In (10) if both of the heading errors or both of the weights are zero, the returned load factor demand will be 1 as required. The overall heading error is scaled by 1/π for convenience, so that integer weights give an integer maximum load factor. By limiting the size of the weights, the maximum load factor allowed for a boid can be set.

A plane banked at an angle of ϕ to its left side has the same load factor as if it was banked to its right side by the same angle (i.e. greater than or equal to one). As it is, (10) may give a value of less than one. If the overall heading error is negative (i.e. the demand is for the boid to accelerate to its right), the value of n_i will be less than one. To deal with this (10) becomes:

$$n_i = 1 + \frac{1}{\pi}\left|\left(W_\alpha \psi_{\alpha i} + W_\beta \psi_{\beta i}\right)\right| \tag{11}$$

and when the overall error $\left(W_\alpha \psi_{\alpha i} + W_\beta \psi_{\beta i}\right)$ is calculated it is tested to see whether or not it is greater than one. If it is greater than one, then n_i is calculated as in (11). The boids angular velocity, $\dot{\psi}_i$, is calculated as in (6) and will be accelerated to the left of the boid's heading (anti-clockwise). If, however, it is less than one, the fact that acceleration to the right side of the boid is required (clockwise) is recorded. The value of n_i is then calculated with (11). $\dot{\psi}_i$ is then calculated as in (6). Since acceleration is in the right direction the value of $\dot{\psi}_i$ is negated. Finally when the boids bank angle, ϕ_i, is calculated, it is also negated so that the boid rolls to the right.

For a real aeroplane the load factor must change gradually. In the model described so far a boid's load factor can change instantly. For example, in the most extreme case, from a maximum load factor left turn to a maximum load factor right turn (or visa versa). In order to avoid this, a maximum allowable rate of change for the load factor is set. In the current implementation this is initialised to a maximum change of ±0.3 g's per second. The mechanism limiting the change takes into account the direction of turning with the load factor. So, for example, it recognises that a change from a right turning load factor of 2, to a left turning load factor of 2 is 2 and not 0.

In V2 three different flocking types, used to produce different sorts of behaviour, are defined: A, B and C. In type A flocking a fixed alignment vector is used and a cohesion centroid that is the average position of all the boids. This allows the flock to be directed to fly on a certain heading. The alignment angle is calculated as:

$$\alpha = \arg(A) \tag{12}$$

and a boid's cohesion angle as:

$$\beta_i = \arg(X_C - X_i) \tag{13}$$

where

$$X_C = \frac{1}{m}\sum_{j=1}^{m} X_j \tag{14}$$

In type B flocking the alignment angle is calculated as an average of all the boids alignment angles. In fact, the alignment angle is actually calculated by taking the average of the boids' heading vectors, then converting this to an angle. This is to avoid the problem that arises with the [0, 2π] boundary. For example, the average of the heading angles 1/4 π and 7/4 π should be 0 whereas a straight average calculation would give π. In type B flocking the cohesion centroid is fixed. Having a fixed cohesion centroid corresponds to the idea of a migration point. Boids will fly towards the point and then circle around it. The alignment angle is calculated as:

$$\alpha = \arg\left(\frac{1}{m}\sum_{j=1}^{m} V_j\right) \tag{15}$$

and a boid's cohesion angle as:

$$\beta_i = \arg(X_C - X_i) \tag{16}$$

The final flocking type, type C, uses a cohesion centroid that is the average position of the flock, as in type A, and an average alignment angle, calculated as in type B. This allows the boids to flock freely together, with no predefined guidance influence. The alignment angle is calculated as (15) and cohesion angle as (13).

In order to measure flock behaviour in a quantitative way some flocking statistics are calculated and displayed on screen. These are boid velocity standard deviation, position standard deviation and mean radius.

4. Results

UAVFlock provides an environment with which a user can simulate UAV boids and experiment with them by adjusting flocking rule weightings and parameters interactively. Performance is such that results of changes made to rules and parameters can be seen immediately as the changes happen. Two example screen captures of the UAVFlock software are shown in Figure 6.

Figure 6: UAVFlock screen captures.

For a simple example of how UAVFlock can be used to interactively experiment with weightings to discover emergent behaviours UAVFlock V2 is started in type B flocking mode with the cohesion weight set to 1.0 and the alignment weight set to 0.0. The boids are initialised in random starting positions. All boids will gradually move in towards the cohesion point and start circling around it (some clockwise and some anticlockwise). As the alignment weight is gradually increased the boids cluster together into two groups traveling in opposite directions circling round the cohesion point. At an alignment weight of 2.0 or greater the two boid groups to split off from the circle and each start heading in straight lines away from the cohesion point and the other group. This is caused by the alignment demand being equal and opposite to the cohesion demand and occurs most often when there are an equal number of boids each side of the cohesion point and the average heading line. The higher the alignment weight is initialised the more likely it is that another, more desirable, behaviour prevails. If the simulation is restarted with alignment set to 8.0 the boids will all group together and circle in the same direction.

5. Conclusions and Future Work

UAVFlock is the first interactive simulation to our knowledge specifically designed for UAV flocking. The current implementation provides a tool for semi-realistic interactive UAV flocking simulation and a foundation upon which improvements and additional features can be built. It provides the basis of a potentially useful tool for aeronautics engineers interested in investigating the use of flocking algorithms to control UAVs.

Some ideas for future work include firstly separation of the guidance, integration and graphics calculations. Currently each set of calculations is performed once per frame. Guidance calculations are not necessary as often as once per frame and greater accuracy may be achieved if integration steps were more often than per frame. More accurate integration methods could also be used and possibly more complex dynamics models, if not too detrimental to performance. The adaptation of the V2 into 3D and combining more flocking rules with the V2 model could also be investigated.

Currently the author is working on Ph.D research into producing physically realistic animation of the flapping flight of birds and insects. In the future this research could be combined with that outlined in this paper to provide greater realism in animal flocking simulation.

References

[1] C. Reynolds, “Flocks, Herds and Schools: A Distributed Behavioral Model”. *Computer Graphics*, 21(4), pp 25-34, 1987.
[2] C. Reynolds, Boids – Background and Update. http://www.red3d.com/cwr/boids/ , 2001.
[3] R. Platt, 3D Boids Simulation. http://www.abs2net.com/robert/3d_boids/ , 2001.
[4] C. Parker, Boids. http://www.vergenet.net/~conrad/boids/ , 1996.
[5] J. Greenbank, My Creature Behaviour Simulator. http://members.fortunecity.com/jngreenb/sim.html , 2002.
[6] W. J. Crowther, “Flocking of unmanned air vehicles”. Bristol UAV conference 2002. http://www.eng.man.ac.uk/Aero/wjc/CV.htm#Publications .
[7] W. T. Reeves, "Particle Systems - A Technique for Modelling a Class of Fuzzy Objets", *Computer Graphics*, 17(3), pp. 359-376, 1983.
[8] W. J. Crowther, “Rule-based guidance for flight vehicle flocking”, 2002, http://www.eng.man.ac.uk/Aero/wjc/CV.htm#Publications .
[9] Anderson, J., *Introduction to Flight*. McGraw-Hill, fourth edition, 1999.