Masters Project

Swooping Angels in Massive

Frank Dumont

MSc Computer Animation & Visual Effects

21st August 2009

Contents

1. Introduction

Warring angels is a current topic being tackled in the film industry, with the film *Legion* due for release in 2010[1], featuring an army of angels descending on Earth to start the apocalypse, and a version of *Paradise Lost* in pre-production[2], with the battle leading to the expulsion of Lucifer from Heaven set to be depicted[3].

The aim of this project is to produce *Massive*[4] agents for a single shot from a sequence depicting the following scene from the Book of Revelations:

> "And there was war in heaven: Michael and his angels fought against the dragon; and the dragon fought and his angels, And prevailed not; neither was their place found any more in heaven. And the great dragon was cast out, that old serpent, called the Devil, and Satan, which deceiveth the whole world: he was cast out into the earth, and his angels were cast out with him." (Revelation 12:7-9, King James Bible)

As well as being a dramatic scenario to recreate, the need for flying characters in the sequence offers an interesting challenge in *Massive*, which was originally designed for crowds that move on the ground. The sequence is imagined as a teaser trailer for a larger production on the same theme:

Outside Heaven, with clouds above and clouds below, rumbles of thunder accompany a darkening of the ground as a red mist rises. From the fog the shapes of Satan's fallen angels and demons emerge as they march to war. Ahead of this army a light breaks through the clouds above and angels fly out in large numbers to defend Heaven. Small skirmishes would be shown as the two sides meet, with angels clashing in mid-air and attacking characters on foot as they swoop by at high speed, whilst others are brought crashing to the ground. The sequence would end after following the hero character, Michael, as he flies down amongst other angels and lands on the ground, poised to join the battle.

The shot chosen from this sequence to produce was the emergence of the first angels from the clouds above the battlefield.

Initially an extreme long shot shows an empty sky before the first angel drops from the clouds into view. Immediately after the first angel appears more emerge from the same spot in the sky, following the first as it falls. A constant stream of angels begins to descend in a column until they gather enough speed to use their wings to pull up, spread out and glide towards the camera (Fig. 1.1).

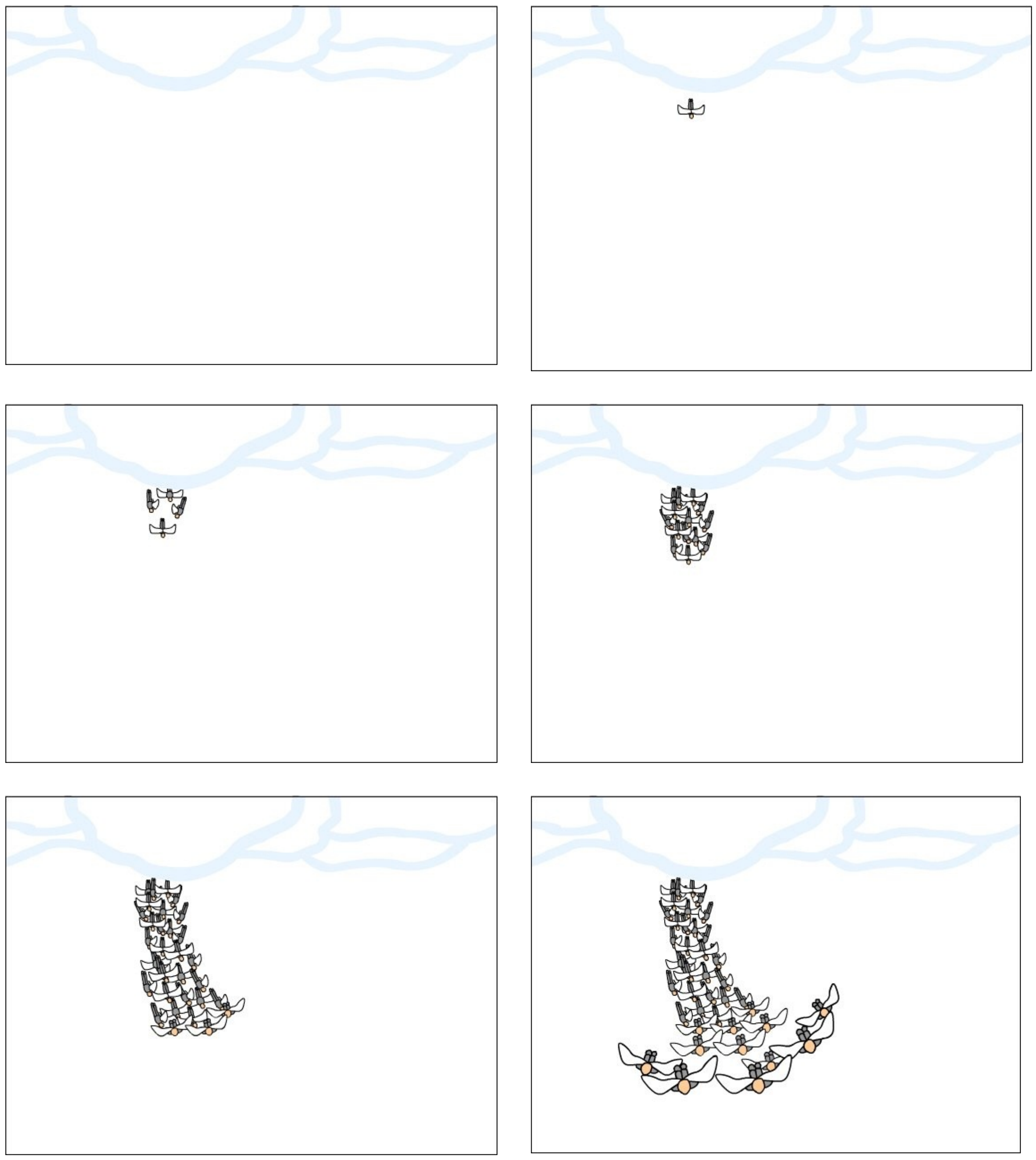


Figure 1.1: Storyboard of the intended sequence

This shot was chosen because it allows for a large number of characters but requires little variation in their appearance, and also features a lot of interesting character movement. Other parts of the sequence would also require a lot of complex character animation to be created, in the form of walk cycles or fight animations, whereas the major factor in the visual impact of the angels in the chosen shot is the way in which they move through the air as they fly.

The shot should show a large number of angels moving in a coordinated group, falling from above until they pick up enough speed to use their wings and pull up to glide past the camera.

2. Previous Work

Previous work judged most relevant to this project included films containing CGI angels and films with a large number of flying creatures moving together.

The film *Dogma*[5] features three main characters that are angels: Metatron, Loki and Bartleby. For most of the film all three characters have their wings hidden, but they are visible in two scenes and appear both computer-generated and as props. The wings used are traditional, with all white feathers, and match up well with the human bodies. Both Loki and Bartleby wear armour cuirasses, which would be appropriate dressing for the angels in this project as they are about to fight a battle.

Gabriel is a main character in the film *Constantine*[6] that is half-human half-angel, and other similar characters also appear. As in *Dogma* the wings of these angels are largely hidden, and in *Constantine* can only be seen by certain characters in the story. Unlike in *Dogma* however they are not universally white, and the wing span is much larger. Although the wings used look more impressive than in *Dogma* they may not work as well with a large number of angels in one shot, as they would need to align their bodies with other angels in order to stay close without touching each other.

As well as CGI angels *Constantine* also features an imagining of Heaven using clouds. The bright sunshine is a strong symbol of the presence of God and also allows a lot of colour in the shot which could look dull if only white clouds were used. The inclusion of a modern-looking city in the landscape is an interesting addition, although not in keeping with the notion in this project that Heaven itself is above the clouds from which the angels are descending.

X-Men: The Last Stand[7] has a single character simply named Angel, who is a human mutant with wings that are a pure white. The scenes in which his wings are visible include several with him flying as well as swooping, which provide good reference for creating flying angels in *Massive*.

The film *Pitch Black*[8] is a science-fiction film featuring winged aliens that fly together in large flocks. The way they move forms streams of creatures that stick together and move as one, with chaotic movements within the flock which make individual aliens hard to see. This gives a strong impression of a coordinated group which would be appropriate to recreate for an army of angels, although with slower, more controlled movements for individual characters.

Finally *The Matrix Revolutions*[9] was researched for its use of flying robots, which move in large numbers in the film when they are battling against humans. When the robots fly in groups they form streams similar to those in *Pitch Black*, with the additional feature of the machines at the edge of the group spiralling around the stream. This extra movement makes the group seem more solid, as if the outside members are holding the shape together and providing protection for the robots inside of the stream. This would be an interesting behaviour to try to achieve in *Massive*, whilst adding an extra dimension of coordination to the crowd of angels.

3. Technical Background

This project is created using *Massive* software, named as the acronym of *Multiple Agent Simulation System in Virtual Environment*[10]. *Massive* is designed for animating large crowds of characters, using artificially intelligent agents that can be programmed to interpret and respond to their surroundings. These agents are combined with 3D geometry and animations in order to render 3D animated scenes.

Each agent is defined as a combination of a body and a brain. The body represents the physical properties of an agent, including size and appearance, and is comprised of a skeleton coupled with 3D geometry and surface materials for rendering the agent.

The brain represents the artificial intelligence processes of an agent, and is constructed as a node network for evaluating a series of fuzzy logic operations. Channels which monitor variables within the scene are used as inputs to determine conditions that are currently affecting the agent, and fuzzy logic is applied to these inputs in order to make decisions on how to respond. This response is enacted by altering the value of output channels representing the current attributes of the agent.

Fuzzy logic in the brain is implemented using membership functions which take any value and evaluate the "truth" of that value based on a specified curve, returning 1 for true, 0 for false or any value in between. The different fuzzy logic nodes available in *Massive* are shown in Figure 3.1.

input	The input node reads the value of a channel to receive information about an agent or its surroundings. Expressions can also be added to inputs to make calculations for use in the brain.
output	The output node writes values to channels to alter the state of an agent or affect its surroundings
fuzz	The fuzz node holds a membership curve which is used to evaluate the fuzzy logic "truth" value of its input
defuzz	The defuzz node converts fuzzy logic values back into values for use in a channel
and	The AND node applies the fuzzy logic equivalent of the Boolean AND operation, and is true when all of its inputs are true
or	The OR node applies the fuzzy logic equivalent of the Boolean OR operation, and is true when at least one of its inputs is true
timer	The timer node measures time, and is started when it receives a strong enough input signal. Timer nodes can count up to a specified time in a loop or else start counting and never stop.
noise	The noise node generates random values between 0 and 1, changing at a specified rate

Figure 3.1: Table of brain nodes in *Massive*

4. Solution

Aim

After considering the previous work relevant to this project, the final aim is to produce agents in Massive that could be used to create the shot described in the Introduction section. The agents should appear as humans with wings, dressed in armour ready to fight. The angels should be able to convincingly fall under gravity in a column, and glide in a specific direction. Angels on the outside of the column should move in a spiralling motion, keeping the shape of the column together, and there should be no collisions between characters.

Massive agents

To create the angels in *Massive* three main agents were required. The angels used to form the column were separated into 'Leaders' and 'Followers'; the Leaders determining the path for the group to take and the Followers using them as a guide. The third 'Spiraller' agent represented the angels spiralling around the column in order to preserve its shape.

Each agent skeleton was created from the human skeleton used in *Massive*'s 'Ambient' agent[11]. To transform them into angels, bones were added to the skeleton to replicate the bones in a bird's wing and also allow control of the feathers.

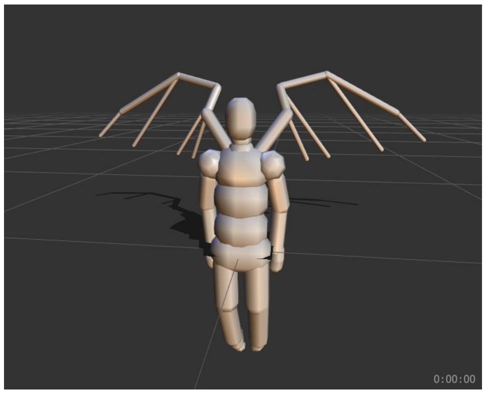


Figure 4.1: Angel skeleton in *Massive*

Using the orientation of the skeleton, the appearance of falling and flight is achieved using the 'ty' channel, which sets the speed or acceleration of an agent along its y-axis. At the start of any scene however each angel is positioned so that it is pointing 'upwards', so a node network was created in each brain to rotate the agent until it was correctly orientated to drop downwards, before it was allowed to move.

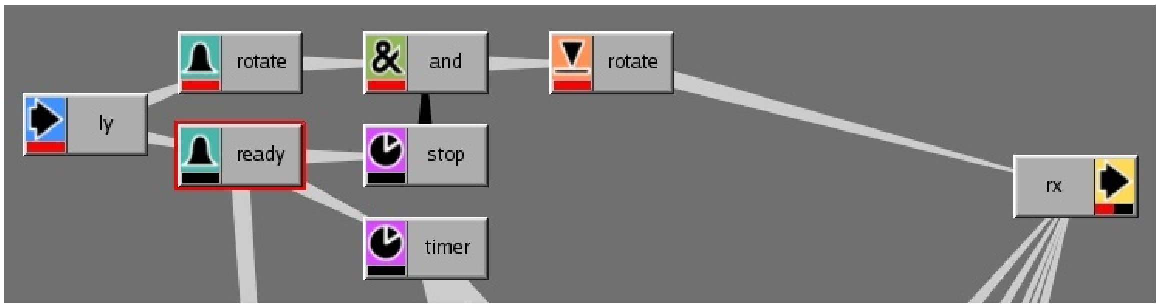


Figure 4.2: Node network for 'setup' behaviour

The 'ly' and 'rx' channels are used as the input and output of the network respectively. The lx channel monitors the y-component of a unit vector in the direction of the agent's current heading, and the rx channel affects the agent's speed of rotation about the x-axis. The 'ready' fuzz node holds a membership function which returns 1 if the value of ly is -1, and the angel is pointing directly downwards, and zero otherwise. The 'rotate' fuzz node holds the opposite membership function, so that when the angel is not pointing downwards the rotate defuzz node causes the agent to pitch down using the rx channel until it is. The 'stop' timer is set up so that once started by the ready node it stays true, and the black NOT connection stops the rotate node having any more effect. Finally the ready node also triggers a second timer, enabling the rest of the brain to start its processes.

Swooping

In order to achieve a convincing swooping motion in *Massive* it was decided to first create a program to simulate the behaviour using C++, OpenGL and the GraphicsLib library[12]. The Glide program uses simplified equations to calculate the lift, drag and gravity forces acting on a glider, based on its current speed and angle of flight. The glider starts from a stationary position pointing downwards to replicate the angels beginning their descent. At each step in the simulation the three forces are calculated and combined to get the net force, which is used to adjust the speed and inclination of the glider. OpenGL is used to draw the glider as a cuboid, following the motion to demonstrate how it looks. The forces acting on the glider are also rendered as coloured lines of varying length to show how they change as the glider gains speed and pulls up.

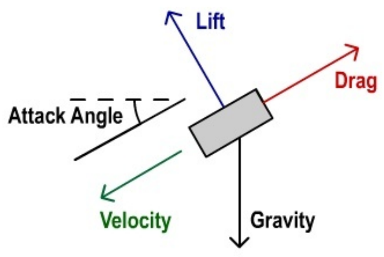


Figure 4.3: Diagram illustrating forces acting on a glider

The lift force l acts perpendicular to the velocity vector and the drag force d acts in the opposite direction to the velocity vector. The magnitudes of these forces are calculated as

$$l = \frac{1.225\pi}{4} b^2 v^2 \alpha$$

$$d = 0.02 * \frac{1.225}{2} bcv^2 + \frac{1.225\pi}{8} b^2 v^2 \alpha^2$$

[13]

where v is the current speed of the glider, α is the attack angle of the wing and b and c are the width and span of each wing respectively. To simplify the equations further, the angel wings are assumed to have a 1 metre width and 2 metre span, and the angle of attack is assumed to be the angle from the velocity vector to the horizontal, so that

$$l \approx 0.962 v^2 \alpha$$

$$d \approx (0.0245 + 0.481\alpha^2) v^2$$

The gravity force acts downwards and is calculated as 10 times the mass of the glider.

These equations combined in the simulation to create a smooth convincing swooping motion as the glider picks up enough speed to generate lift and pull itself into a gliding position.

The first method that was tried to bring the swooping motion into *Massive* involved reproducing the force equations using the variables and mathematics available in *Massive* (Fig. 4.4). Because *Massive* doesn't support vector arithmetic the calculation of the net force acting on the glider was separated into x and y components, and the clamp function was also incorporated to prevent divisions by zero. However this method failed because of the recursive nature of the equations, which use the current speed and attack angle of the glider to calculate the new speed and angle. Because the variables are node based they are not initialised, and start without values as a "not a number" error. In a non-recursive equation when one value is set it allows the calculation of the next, pushing this error message along the nodes until it is no longer needed. In the gliding equations however the error propagates in the loop until every node gives a "not a number" error.

A modified method to calculate the swoop in *Massive* used agent channels instead of new variables wherever possible to make use of their initial values, but this did not resolve the problem.

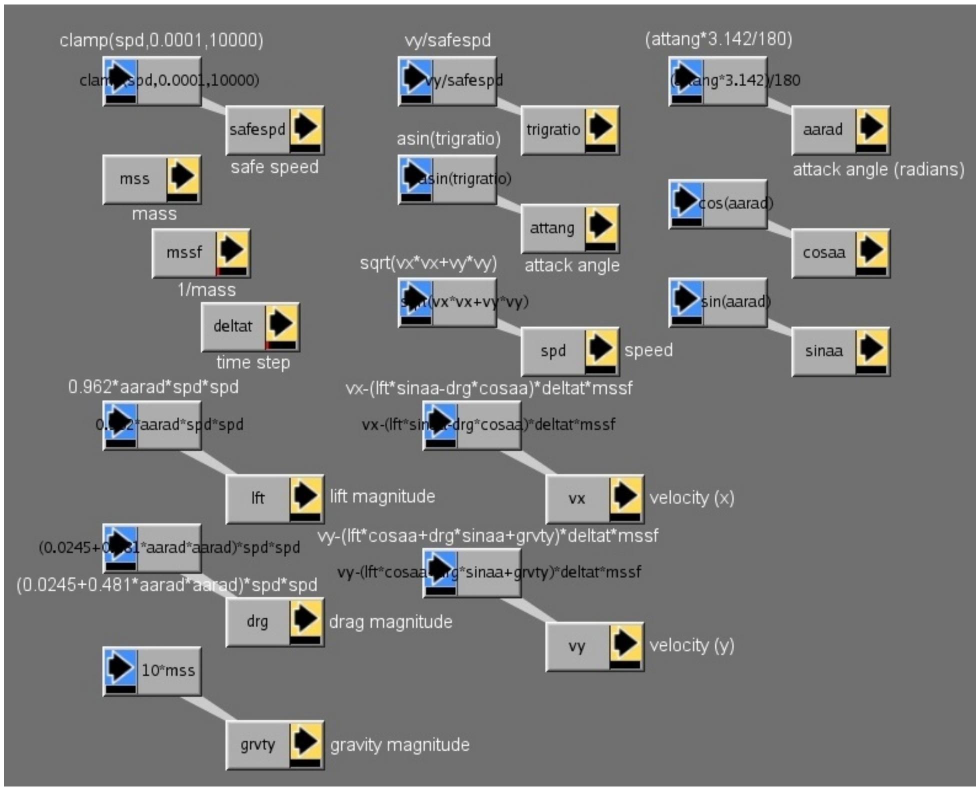


Figure 4.4: Annotated brain attempting to reproduce force equations in *Massive*

The alternative to calculating the swoop dynamically was to use dead reckoning, repeating the motion produced in the simulation directly without taking into account the state of the agent. To achieve this the Glide program was altered to output the attack angle and speed of the glider after each calculation, and these values were brought into a spreadsheet to produce graphs of the values changing as the glider pulls up (Fig. 4.5).

These graphs could then be approximated in *Massive* by splitting them into sections that fitted the shape of *Massive*'s membership curves, and using Boolean switches to control which curve is used at a specific time (Fig. 4.6). The current membership curve is then used to blend between the values at the start and end of the relevant section of the simulation graph.

In the early test agents both the speed and angle graphs were recreated in *Massive* in this way, but it was eventually decided that the speed changes were not crucial to the impression of the swoop, and that the speed would be better controlled elsewhere in the brain.

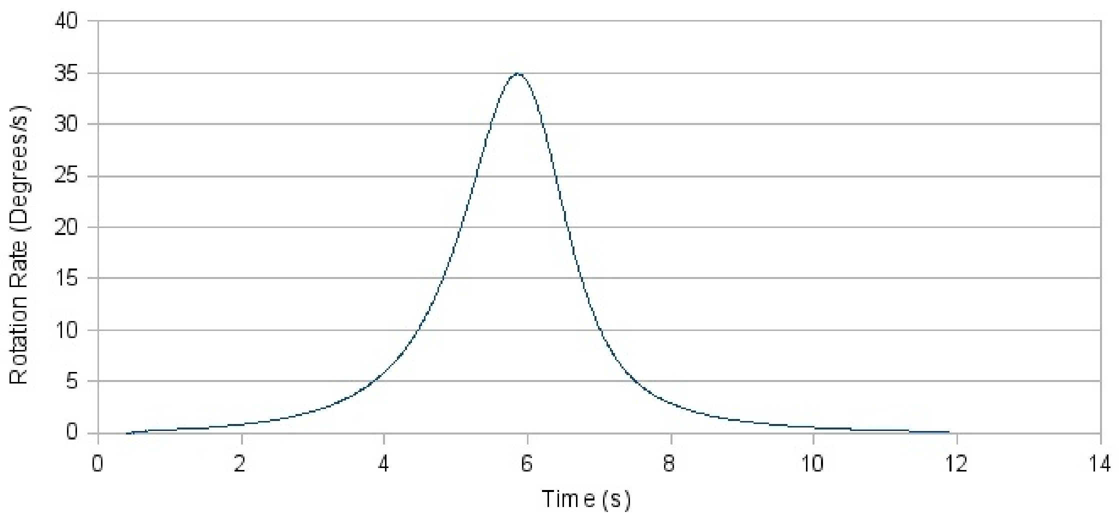


Figure 4.5: Simulation graph showing the angle change per second as the glider pulls up

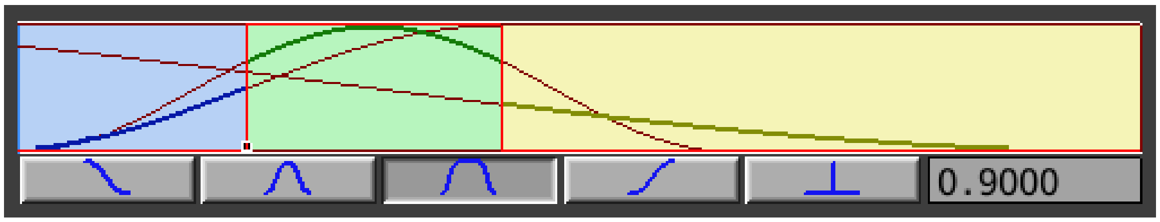


Figure 4.6: Coloured membership curves replicating the angle change graph. The coloured boxes show the time periods over which the curve sections are active. Each curve section is squashed or stretched depending on the range of the values it blends between.

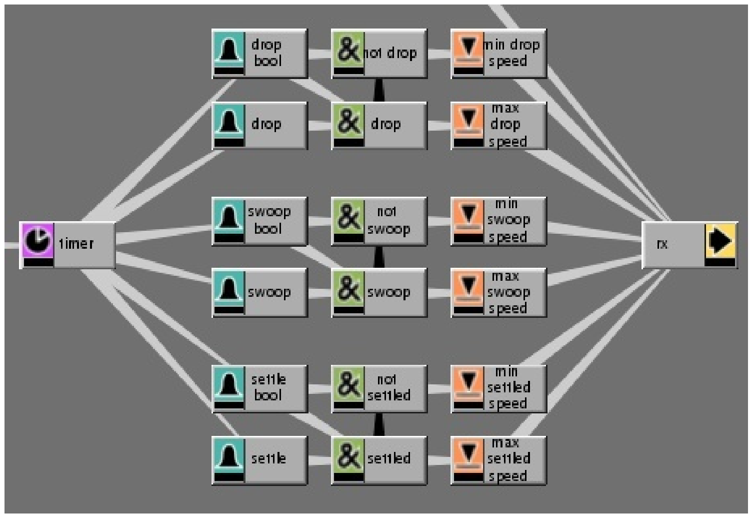


Figure 4.7: Node network for the 'swoop' behaviour

Each section of the graph is recreated using two fuzz nodes, two AND nodes and two defuzz nodes (Fig. 4.7). One fuzz node holds a membership curve designed to mimic the shape of the simulation graph whilst the other holds a Boolean function which is true for the time interval that the curve is valid. The defuzz nodes hold the start and end values of the relevant section of the graph, to be used as the rotation speed about the x-axis. Finally the AND nodes are set up so that one is the inverse of the other and they always add up to 1, blending between the start and end values at a rate dictated by the curve.

Once an agent is in the right position, this node network can be triggered by starting the timer and the agent will repeat the motion of the simulation and swoop into a gliding position.

Following

The starting point for the Follower agents involved using sound to determine where surrounding agents were. Using the distance and direction to other agents, each falling angel could move away from those that came too close and move towards others that weren't.

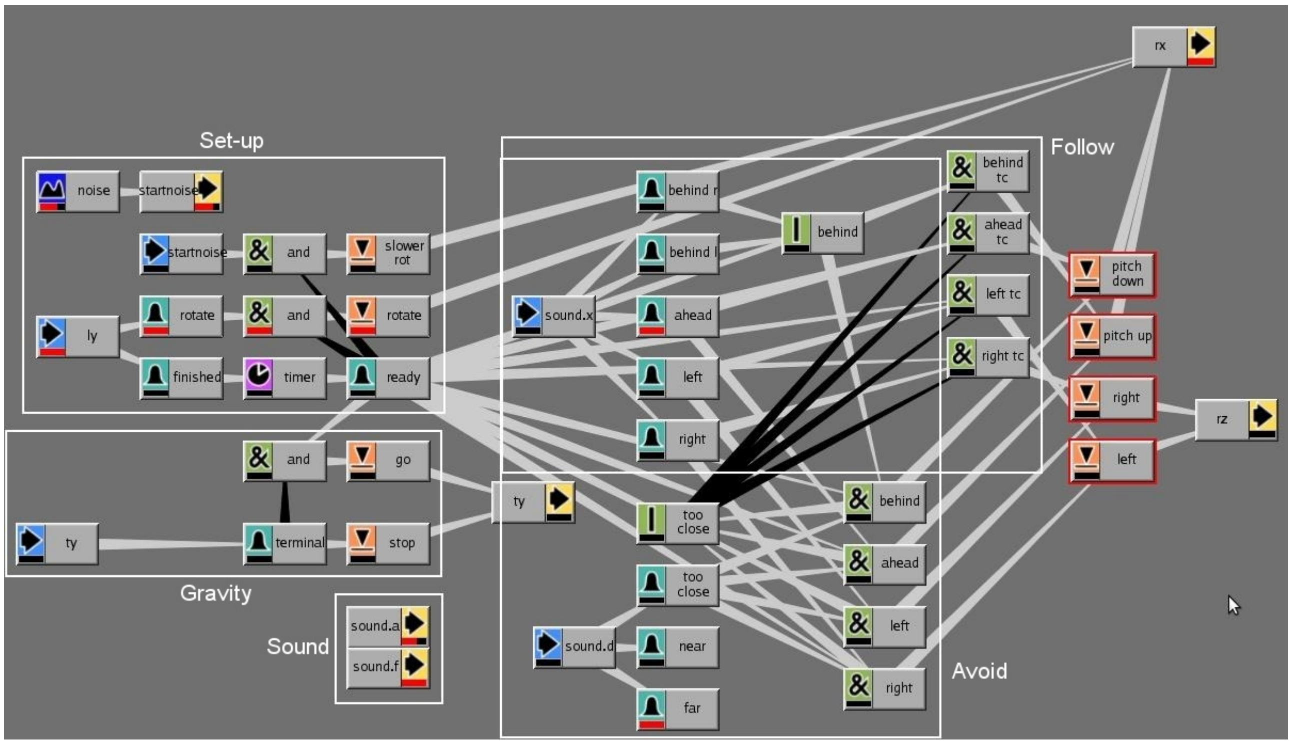


Figure 4.8: Node network for an early Follower agent

The early Follower brain can be separated into five sections, each representing a behaviour carried out by the agent (Fig 4.8). The set-up behaviour uses the same method explained previously, with a noise node incorporated to vary the time it takes for each agent to be ready to move. The gravity nodes control the speed of the agent, accelerating at 10 m/s^2 using the 'go' node until the 'terminal' fuzz node is true, and the angel has reached terminal speed. The go node is then balanced with the 'stop' node to keep the speed steady at the terminal value. The sound behaviour outputs constant values to the 'sound.a' and 'sound.f' channels to emit sound at a specified amplitude and frequency respectively. The 'sound.x' channel gives the angle in the xz-plane from an agents current heading to the source of any sounds that the agent can hear, and is used by the avoid and follow behaviours to find out which direction the other agents are in. To decide whether the avoid or follow behaviours should be active the distance to the nearest sound is found using the ‘sound.d’ channel, so that when the ‘too close’ fuzz node is true the nearest agent is too close and must be avoided. The direction to the nearest agent is used to trigger a rotation in the opposite direction, avoiding a collision. The black connections between the avoid and follow nodes stop the follow behaviour being active whenever an agent is too close, giving priority to the avoid behaviour. When the agent is not avoiding other agents, the follow behaviour is enabled and the agent uses the direction of sounds heard in order to turn towards other agents in an attempt to follow them.

This early brain network kept the follower agents together in a group as they fell and ensured there were no collisions, but because the followers and the leader all emitted the same frequency sound there was no distinction made and the group could ignore the leader agent. Another problem was that whenever one agent tried to move towards another it would pass the spot it was aiming for before realising and taking time to start moving the other way, causing it to “rock” back and forth behind the agent it was following. This behaviour in a group of agents trying not to collide meant that all the followers were likely to start rocking, taking away the impression of the angels moving as a coordinated group.

At this point it was decided that the node networks in the brain of each agent were getting too complicated and disorganised to understand and alter. To fix this, individual behaviours were encapsulated into separate macros (Fig. 4.9). Each macro was named after its purpose and given an AND node as a switch. Each switch is wired to a behaviour network so that the behaviour is entirely active if the switch value is 1, but has no affect on any output channels if it is 0. The switch takes as inputs any conditions which must be true for the behaviour to start, and any conditions which should stop the behaviour can be input to the switch using black NOT connections. As well as making each brain easier to follow, the macros also allowed each behaviour to be switched on and off in different combinations for testing purposes.

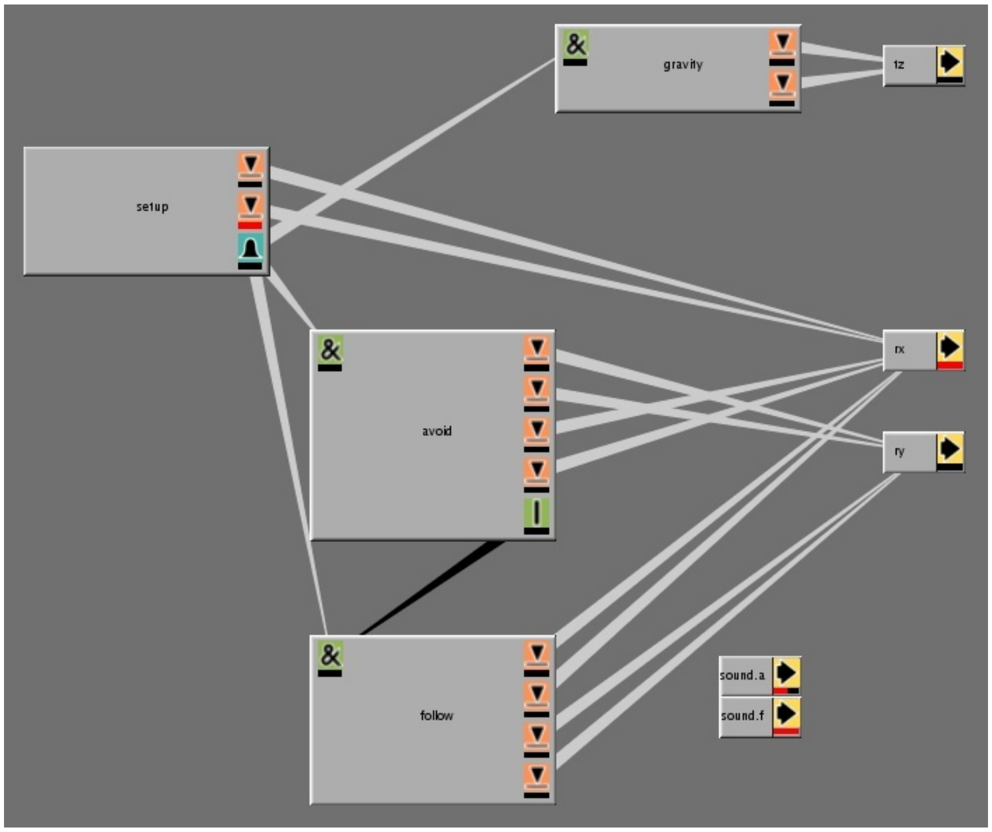


Figure 4.9: First Follower brain using macros

To stop the problems with angels rocking back and forth, a new ‘orient’ behaviour was added so that when an agent came close to the sound it was following it would start to rotate itself to point in the same direction as its target. This behaviour reduces an agent’s movement relative to its target and stops it from overshooting as it gets near.

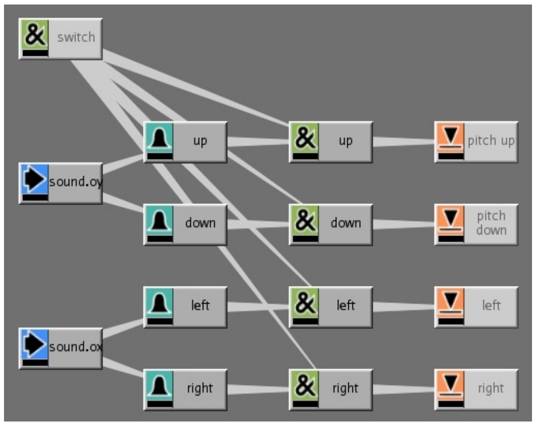


Figure 4.10: Macro for early ‘orient’ behaviour

The 'sound.ox' channel used measures the relative heading of another agent to the current agent in the xz-plane, and the 'sound.oy' channel measures the relative angle from the agent to the xz-plane. Because *Massive* was intended for characters travelling along the z-axis and staying on the ground, these channels do not measure the roll rotation of agents. This meant that with the agents orientated along the y-axis these channels could not measure the necessary relative-yaw angle between angels. As a result the Leader and Follower agent skeletons were rotated to point them along the z-axis, and the brains were altered to move using the 'tz' channel, and rotate differently as described in Figure 4.11.

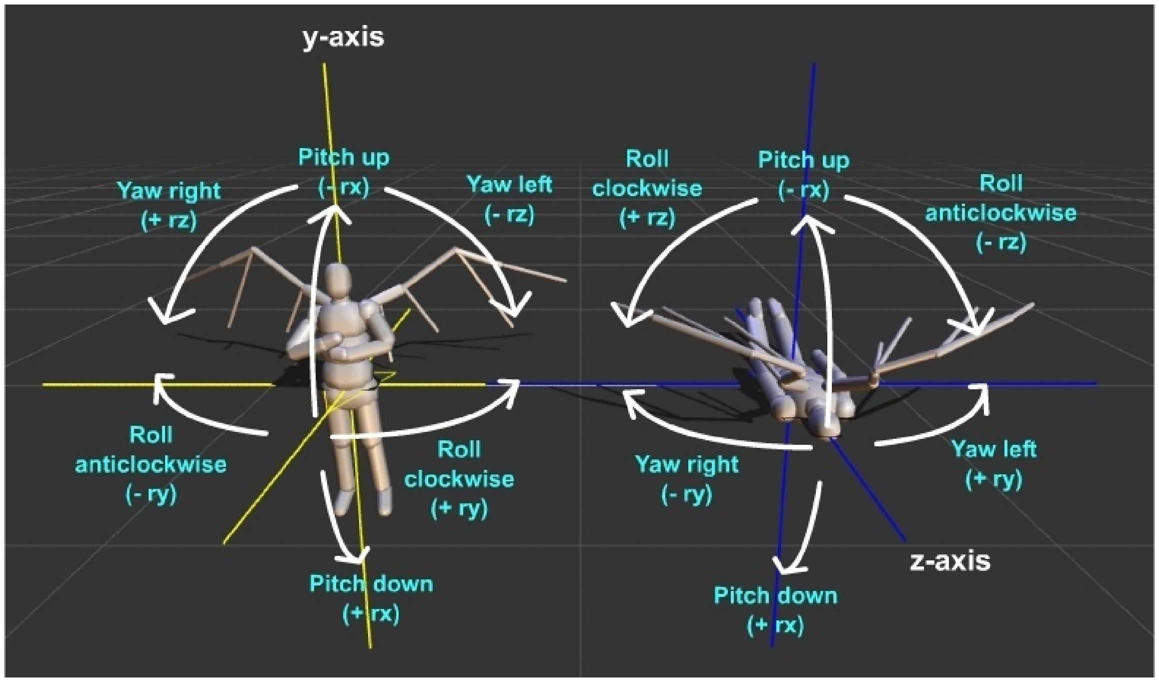


Figure 4.11: Diagram illustrating the different channels used for rotations for agents aligned along the y and z-axes

To allow the Follower agents to distinguish the leader as their main target, the leader agent was altered to emit a lower frequency sound, and the follow behaviour (Fig. 4.12) in the Follower brain was changed to only listen for this frequency. Although this meant that the leader now controlled the group of angels, all of the Followers aiming for a single agent meant they would stick very close together and spend a lot of time avoiding each other. This problem was solved by also changing the Follower agents so that if they were settled behind the Leader agent, they would start to emit the same frequency as that Leader, forming a chain in the group that kept all of the angels following the leader, directly or indirectly.

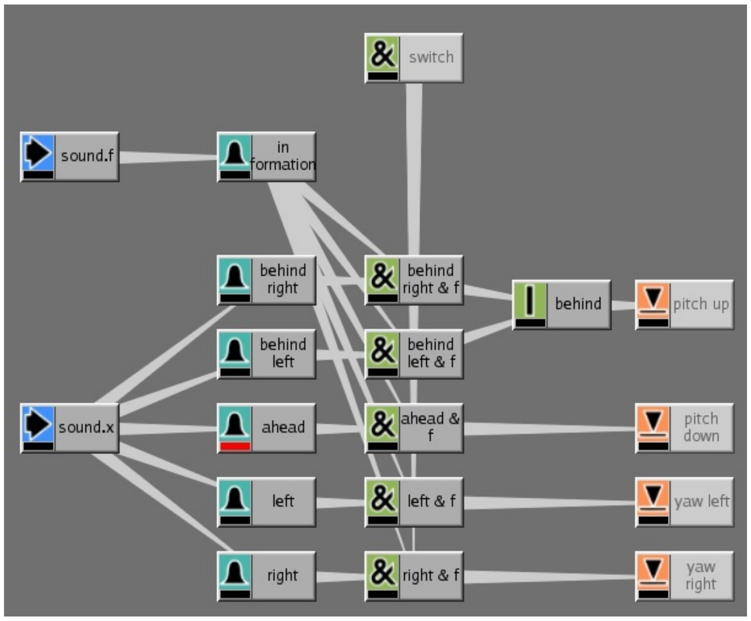


Figure 4.12: Macro for ‘follow’ behaviour

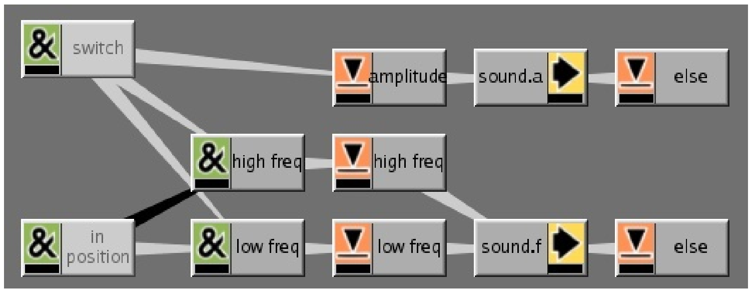


Figure 4.13: Macro for ‘sound’ behaviour of the Follower agent

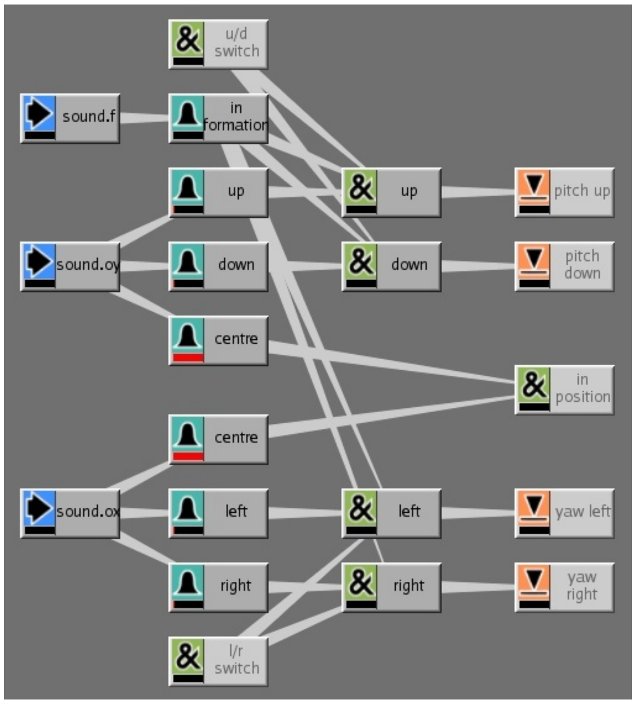


Figure 4.14: Macro for 'orient' behaviour

The updated orient macro (Fig. 4.14) includes 'centre' fuzz nodes to determine when the agent is 'in position' behind its target, in order to trigger a change in the frequency of the sound emitted to match the Leader agent. The behaviour was also given separate triggers for the different rotations it performs, so that when the agent is close enough to align in one axis but not the other, the macro can still be used.

The final behaviour added to the follower agents to keep them following the leader in a column is encapsulated in the 'balance' macro (Fig. 4.15). This part of the brain combines the movements of the other behaviours with a desire to keep the angel pointing directly downwards. This has the effect of reducing the severity of the movements made by a follower jostling for position, and makes the falling angels appear more in control as they travel downwards inside the column. It also means that any agent that gets lost and has no sounds to follow will behave in a predictable way, continuing to fall directly down until it is told otherwise.

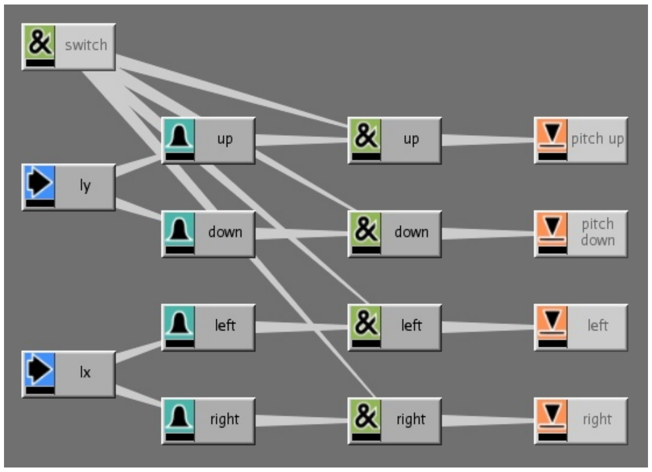


Figure 4.15: Macro for 'balance' behaviour

With the agent aligned along the z-axis the lx and ly channels measure the x and y components in world space of a unit vector pointing along the y-axis of agent space. For the agent to be pointing directly down both of these channels must equal 0, so the brain uses pitch and yaw rotations to reach these values. This macro is only simple because the agent has not been allowed to roll yet; once a roll rotation is introduced the lx, ly and lz values are no longer mutually exclusive and more complicated patterns of rotation are required to reorient the agent.

<u>Spiralling</u>

The first method considered for a spiralling behaviour involved using sound to keep the falling column on one side of the agent. If the agent started outside of the column with its stomach facing the centre, as it flew forward and the sounds of the falling agents moved behind it, it could pitch down so that the sounds were directly underneath it again. Rotating to keep the column on the same side of the agent would cause the agent to move in circles around the column, falling downwards with the other angels and forming a spiral.

This method was first tested with a single spiralling agent and a target agent. The target agent was a sphere which moved at a fixed rate in the y-axis. The spiralling agent took the sound.x and sound.y values of the target and used them to keep the target underneath its wings as it moved forwards at a fixed speed. This set-up resulted in an agent which could successfully move with the target whilst performing a spiralling motion, but was heavily dependent on the starting positions of both agents. If the circle flown by the spiralling agent became lined up with the target's movement then the angel would fly below the target and pull round so that it flew directly upwards against gravity.

In an attempt to fix this problem an input taking into account the y-component of the spiralling agent's heading was added to its brain. When this value rose above 0 then the agent was travelling upwards against gravity and started rotating downwards to correct itself. This adjustment meant that the spiralling agent fell into a correct motion wherever it started near to the target, but if the spiraller did pass below the target then it would twist in an undesirably complicated pattern until it was able to return to spiralling.

At this point it was decided that to create a spiralling agent it would be necessary to find out exactly what rotations a *Massive* agent would have to perform in order to move around the shape of a regular helix. The parametric equation of a helix in Cartesian coordinates is given as

$$x = a\cos(t)$$
$$y = a\sin(t)$$
$$z = bt$$

[14]

for some parameter t, where a is the radius of the helix, and 2πb is the distance travelled in the z-axis after a full rotation. To replicate the desired motion these equations were changed from describing a helix spiralling around the positive z-axis to describing a helix around the negative y-axis by swapping the y and z equations and negating b.

This equation is not directly useful in *Massive* however as agent positions can only be affected by speeds and rotations rather than placement at positions in world-space, so a program was written in C++, OpenGL and the GraphicsLib library[12] to simulate a cuboid following the path dictated by a helix equation and calculating the rotations carried out on the shape's three axes as it follows the tangent of the spiral. The rotations were found to be constant values in all three rotation axes, which varied depending on the radius and pitch of the helix. When these rotations were implemented in the brain of a *Massive* agent (Fig. 4.16) travelling at a fixed speed the resulting motion followed the path of a regular helix, but pointing in an arbitrary direction that sent the angel flying the wrong way.

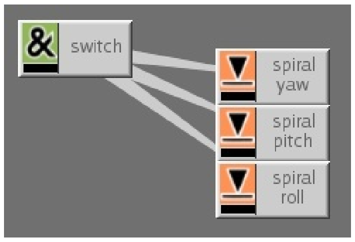


Figure 4.16: Macro for 'spiral' behaviour

To achieve a spiral in the desired direction it was clear that the initial heading of the agent needed to be changed. A test agent was created with controls to manually rotate the agent before flicking a switch to start the spiralling rotations. The lx, ly and lz channels were monitored to determine the rotated heading of the agent in each test. This approach was not fast or accurate enough to provide satisfactory results however and it was decided to calculate the required initial heading to use the helix equation results for spiralling downwards.

The first task was to calculate the central line about which an arbitrary helix rotates. This was achieved by adapting the Helix program to recreate the *Massive* helix, starting the cuboid in the orientation of the agent and moving it using the same rotations that the first program produced. The x, y and z coordinates of the cuboid after each calculation were outputted by this new program and brought into a spreadsheet, where separate graphs were produced for the x, y and z coordinates of the cuboid's position as it followed the shape of the helix (Fig. 4.17). Provided that enough position data was collected it was observed that the centre-line could be calculated using the regression lines of these graphs. If each regression line is a linear equation

$$y = ax + b$$

then the centre-line is the combination of these linear equations into a 3D line, given by

$$l = \begin{pmatrix} a_x \\ a_y \\ a_z \end{pmatrix} t + \begin{pmatrix} b_x \\ b_y \\ b_z \end{pmatrix}$$

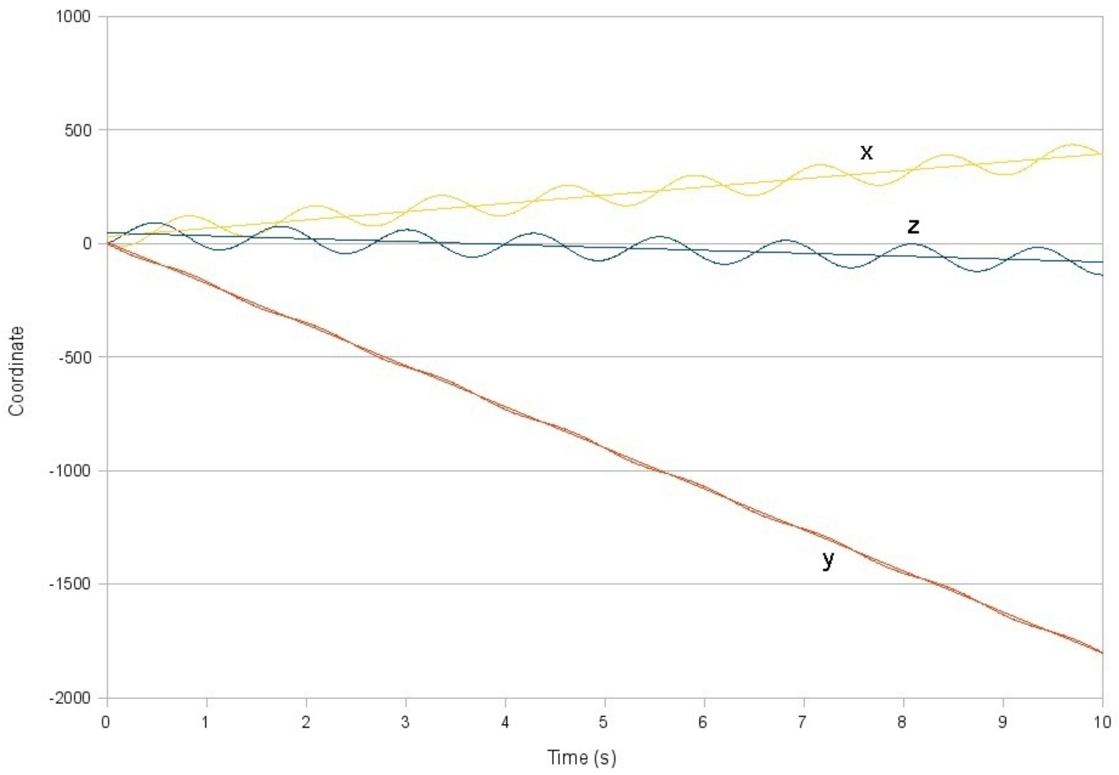


Figure 4.17: Graphs showing the components of the position along a helix over time

To simplify the calculation of the centre-line for an arbitrary helix the three lines of regression were calculated as part of the Helix program using the following equation:

$$y = b_1 x + b_0$$

where

$$b_1 = \frac{\sum (x_i - \bar{x})(y_i - \bar{y})}{\sum (x_i - \bar{x})^2}$$

and

$$b_0 = b_1 \bar{x} - \bar{y}$$

[15]

Once the centre-line is found, the program calculates the rotation required to move from its current direction to the desired downward vector, and applies it to the old initial agent heading to give the correct heading for downwards spiralling. To ensure that the centre-line also passes through the origin the radius of the spiral can be calculated as the shortest distance from any point on the helix to the centre-line, and the agent offset from the centre of the column by that amount. Because the Spiraller agent was still orientated along the y-axis, the lx, ly and lz channels represented the x, y and z components of a unit vector in the direction of the agent's heading. This meant that they could be used to control the rotations required to reach the calculated start heading that would lead to a spiral travelling directly downwards. The setup behaviour used by the Leader and Faller agents was adapted into two separate macros, 'setup yaw' and 'setup pitch'.

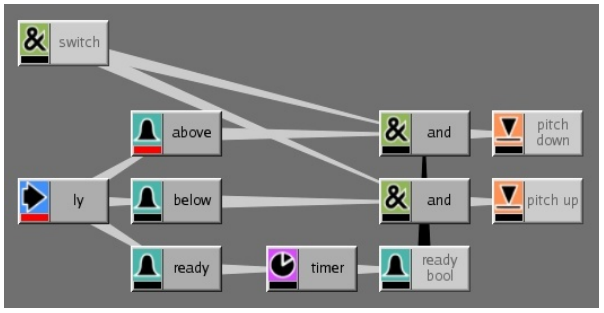


Figure 4.18: Macro for 'setup pitch' behaviour

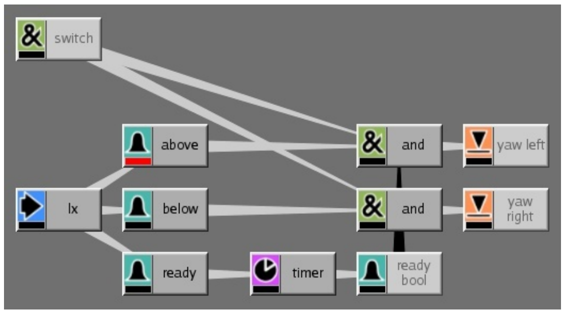


Figure 4.19: Macro for 'setup yaw' behaviour

Each macro works in the same way as the original setup behaviour, with fuzz nodes that determine when the agent is correctly orientated and trigger an appropriate rotation if it is not. They differ however in that they have the ability to rotate the agent in both directions about an axis, so that if the agent overshoots the required value it can rotate in the opposite direction in order to reach it accurately.

Integrating the swoop

In order to coordinate each agent's transition from falling to gliding it was decided to create a beacon agent to trigger the swoop macro in each brain. The beacon was represented as a sphere, and dropped before the angels to a designated height before emitting a loud noise. The angel agents were set up so that moving within a certain distance of the beacon activated a new behaviour named 'prepare', which caused them to roll until they were facing away, with the beacon behind their wings, before starting the swoop action to end up gliding in the direction towards the beacon. This set-up worked except that the time it took to prepare to swoop was decided by the initial direction in which the agent was facing. This meant that the height at which each angel started to swoop varied dramatically. Instead of this system it was decided to have two beacon agents, named 'roll beacon' and 'swoop beacon'. Both beacons emit different frequency sounds, and the angel agents have separate macros for detecting either beacon as they pass behind them on their descent. The roll beacon starts the rotation into the right heading, and the swoop beacon starts the transition from falling to gliding.

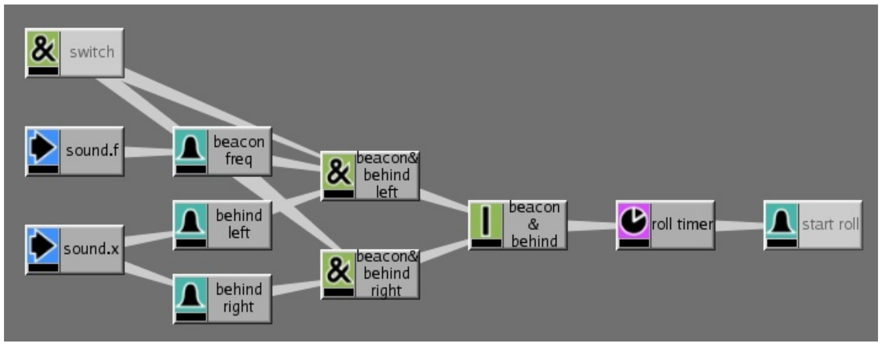


Figure 4.20: Macro for the 'detect roll beacon' behaviour of the Leader and Follower agents

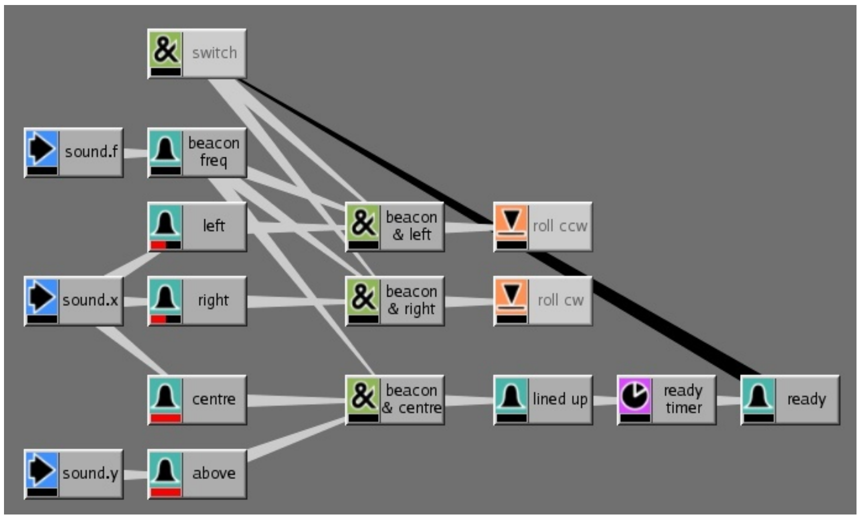


Figure 4.21: Macro for the 'prepare' behaviour of the Leader and Follower agents

The 'detect roll beacon' macro (Fig. 4.20) combines the sound.x and sound.f channels to locate the roll beacon, and wait for it to be passed. When the beacon is more than 90 degrees to the left or right of the agent then it has been passed, and the roll timer is started. The 'detect swoop beacon' works in the same way, triggering the switch belonging to the swoop macro to start the swoop timer. Once the roll timer is started the prepare behaviour (Fig. 4.21) becomes active, using the sound.x channel to dictate in which direction to rotate so that the roll beacon is behind the angel's wings. After these rotations are finished the centre and above fuzz nodes are "true", and the ready timer is started to deactivate the prepare behaviour.

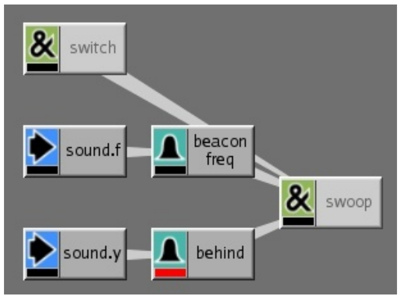


Figure 4.22: Macro for 'detect swoop beacon' behaviour of the Spiraller agent

The detect beacon macros in the brain of the Spiraller agent do exactly the same job as those in the Leader and Follower agents, but because of the different starting orientation of the agent the sound.y channel is used to determine when a beacon has passed behind the agent (Fig 4.22).

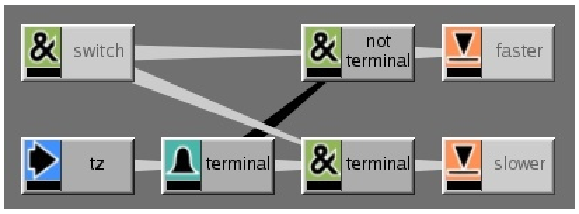


Figure 4.23: Macro for the 'gravity' behaviour of the Leader and Follower agents

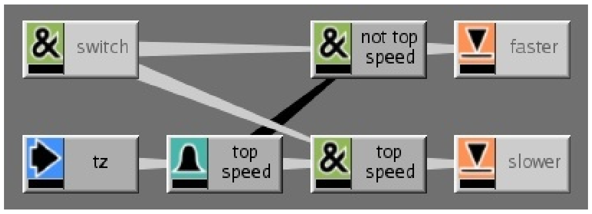


Figure 4.24: Macro for the 'glide' behaviour

Two macros, ‘gravity’ and ‘glide’, are used to control the acceleration of each agent before and after the swooping manoeuvre. The gravity behaviour in the Leader and Faller agents (Fig. 4.23) controls the speed of the angels as they drop, accelerating at 10 m/s^2 until the value of speed tz reaches the terminal velocity of 67 m/s. At this point the acceleration becomes balanced with a decelerating defuzz node in order to maintain the terminal speed for the rest of the drop. The glide behaviour (Fig. 4.24) uses exactly the same method for controlling the speed, but the speed changes at a slower rate, and the top speed of a gliding agent is less than half of the terminal speed.

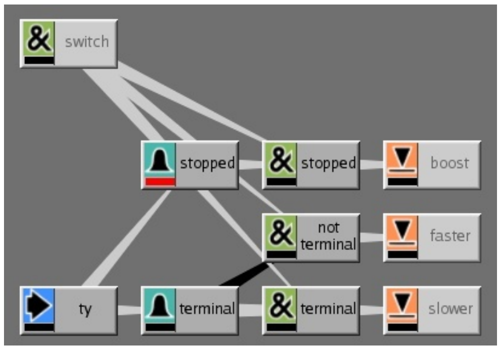


Figure 4.25: Macro for the ‘gravity’ behaviour of the Spiraller agent

The gravity macro for the Spiraller agent (Fig. 4.25) works similarly to that of the other agents, but includes an extra defuzz node named ‘boost’. For the agent to maintain the shape of the spiral it needs to travel at a constant speed, and cannot spend time accelerating in order to reach terminal velocity. The boost node is triggered when the angel first starts to move, and accelerates it rapidly to full speed before relinquishing control to the other defuzz nodes to steady the speed at 67m/s.

Balancing Spirallers

Before the Spiraller agents could replicate the prepare behaviour and line up ready to swoop they first needed to stop spiralling and start falling directly downwards. The balance macro used to keep the Follower agents pointing down could not be used because the spiralling motion requires rotations in all three axes, and the individual rotations required to return them to a specific new heading would be difficult to determine. Instead it was decided a new beacon would be used as a guide, similar to the roll and swoop beacons. The 'centre beacon' emits sound in its own frequency from a position a long way below the path of the angels, but in line with the centre of the column. This beacon can then be used as a target for the Spiraller agents to aim at, causing them to orient into an approximately downwards direction.

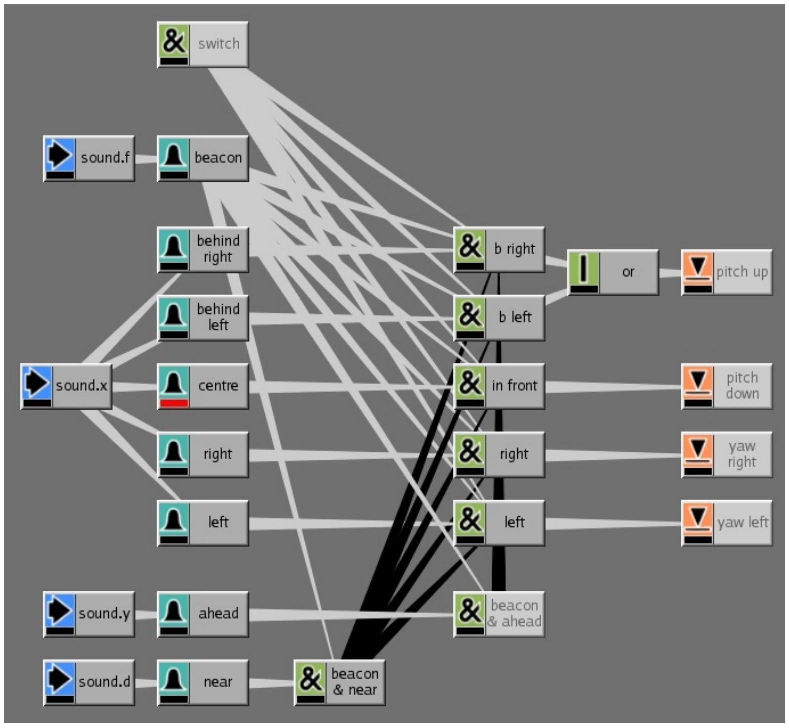


Figure 4.26: Macro for the 'aim at centre' behaviour

The 'aim at centre' macro (Fig. 4.26) uses the same method as the 'follow' behaviour, using the location of the centre beacon determined by the sound.x channel to decide on which way to rotate in order to aim at it. However, because the centre beacon is not directly below the Spiralling agent but below the centre of the column, extra nodes are incorporated to stop them moving towards the inside of the column. Because the angle difference between the line to the beacon and the downwards vector increases as the beacon gets closer to the agent, the 'near' fuzz node is used to test the distance to the beacon, stopping any more rotations when it becomes too small. The 'ahead' fuzz node which evaluates the sound.y channel has a loose definition of when the centre beacon is in front of the Spiralling agent, and stops any more rotations at this point before the agent is pointing directly at the beacon and hence towards the inside of the column. When the rotations stop the 'prepare' behaviour (Fig 4.27) can begin, rolling the agent until the roll beacon is behind its wings, ready to swoop.

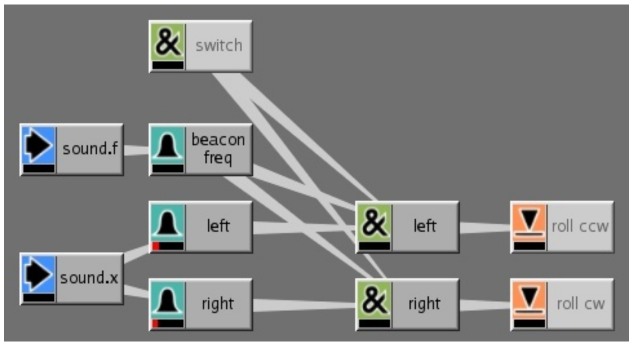


Figure 4.27: Macro for the 'prepare' behaviour of the Spiraller agent

Beacons

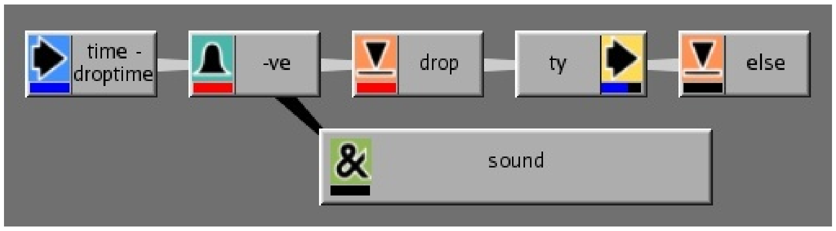


Figure 4.28: Brain of a beacon agent

Each of the roll, swoop and centre beacons share the same brain (Fig 4.28), differing only in the distance they drop before being in position and the frequency of the sound that they emit. The distance is determined by an agent variable named ‘droptime’ and the speed in the ‘drop’ defuzz node which is outputted to ty. When the *Massive* simulation starts the built-in ‘time’ variable is at 0, and the expression ‘time - droptime’ is negative. The ‘-ve’ fuzz node evaluates true whilst the simulation time is less than the value of droptime, causing the agent to move downwards in the y-axis. Once the expression turns positive the fuzz node evaluates as false, the beacon is in position and the sound is activated.

Leader agent

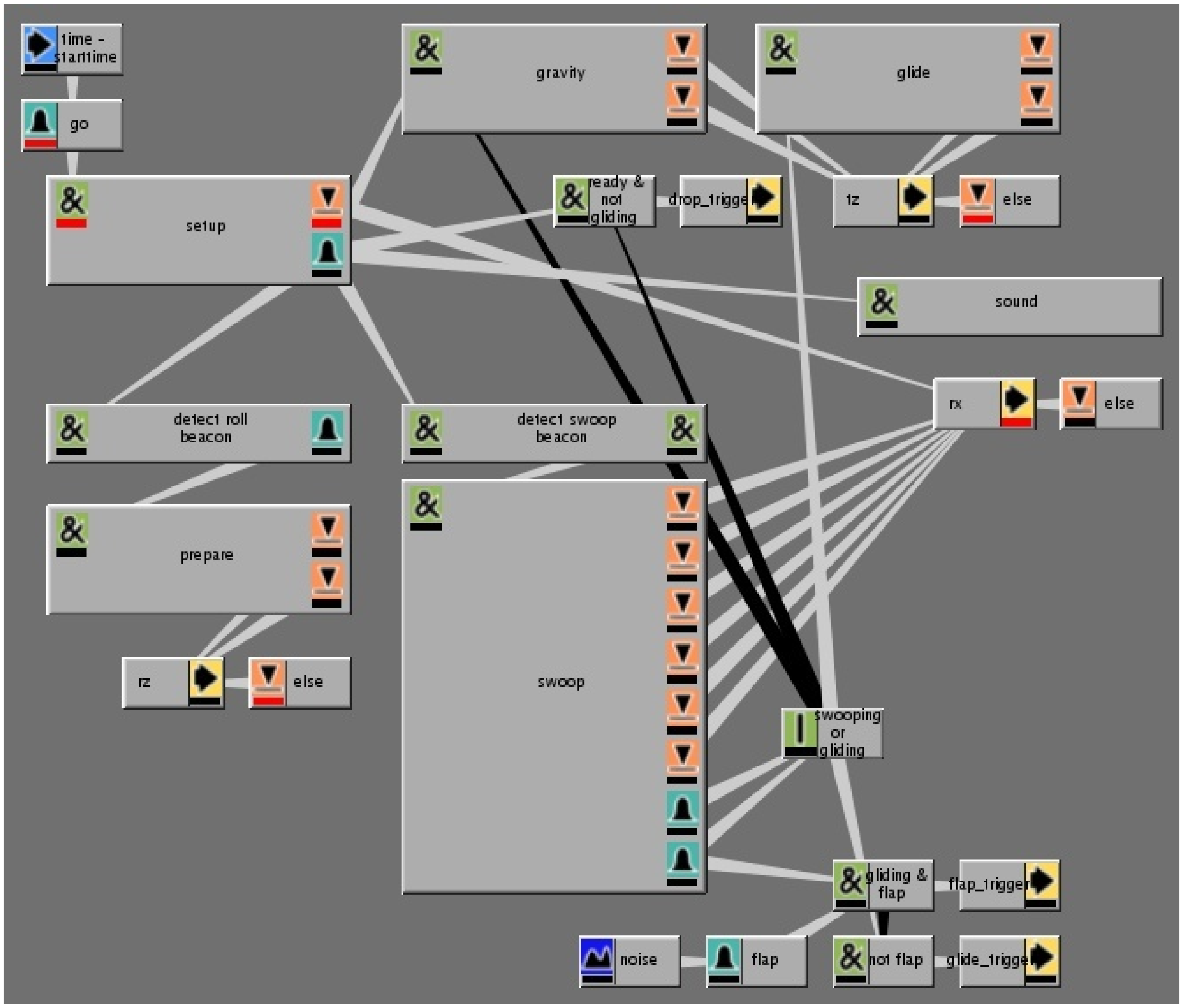


Figure 4.29: Brain of a Leader agent

Each angel has an agent variable named 'starttime' with a value in seconds determining at what point in the *Massive* simulation it should start its descent. The Leader brain (Fig 4.29) uses the built-in time variable to wait for this point, before triggering the setup macro. When the setup has finished the Leader can begin its drop, and the gravity and sound macros are triggered. The end of the setup also causes the detect roll beacon and detect swoop beacon behaviours to be enabled. The angel drops straight down until the roll beacon is passed, and the prepare behaviour is started. This causes the Leader to roll into position with the beacon behind it, before the swoop beacon is detected and the swoop macro is triggered. As well as applying rotations to transition the angel from falling to gliding, this macro switches off the acceleration due to gravity and enables the glide behaviour to control of speed the agent. The Leader angel then continues gliding off into the distance. The 'trigger' output nodes in the brain control the animations of an agent, and are explained in the Animation section.

Follower agent

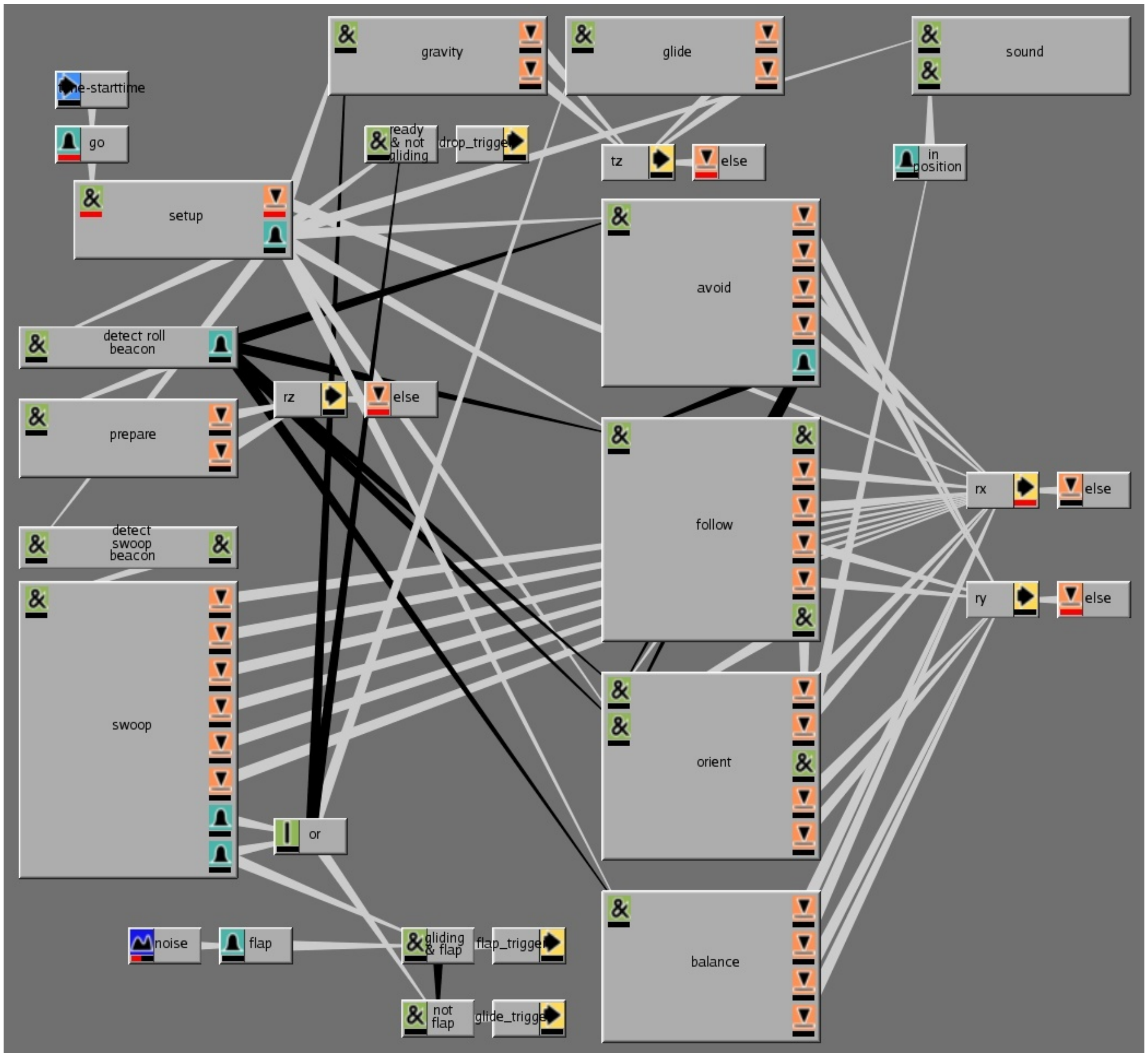


Figure 4.30: Brain of a Follower agent

The brain of a Follower agent (Fig. 4.30) starts in exactly the same way as the Leader, waiting for the start time before performing the setup behaviour and starting to drop. Similarly the end of the setup macro starts the gravity, sound, detect roll beacon and detect swoop beacon behaviours. Instead of just dropping though the end of the setup macro also activates the avoid, follow, orient and balance behaviours so that the Follower angels move around inside the column, trying to follow the nearest Leader agent whilst avoiding collisions which each other. If the Follower finds itself behind a Leader agent and heading in the same direction, the 'in position' fuzz node will evaluate to true, and a switch in the sound macro will change the agents sound frequency to match that of a Leader agent. The Follower continues to position itself within the column until the roll beacon is passed. The detect roll beacon macro uses black node connections so that when the prepare behaviour starts the avoid, follow, orient and balance macros are all disabled, and the agent stays fixed in its position relative to all other agents from that point onwards. After the Followers start to roll their behaviour is identical to that of the leader, the swoop beacon triggers the swoop action, which in turn disables the gravity macro and enables the glide macro to control the agents speed.

Spiraller agent

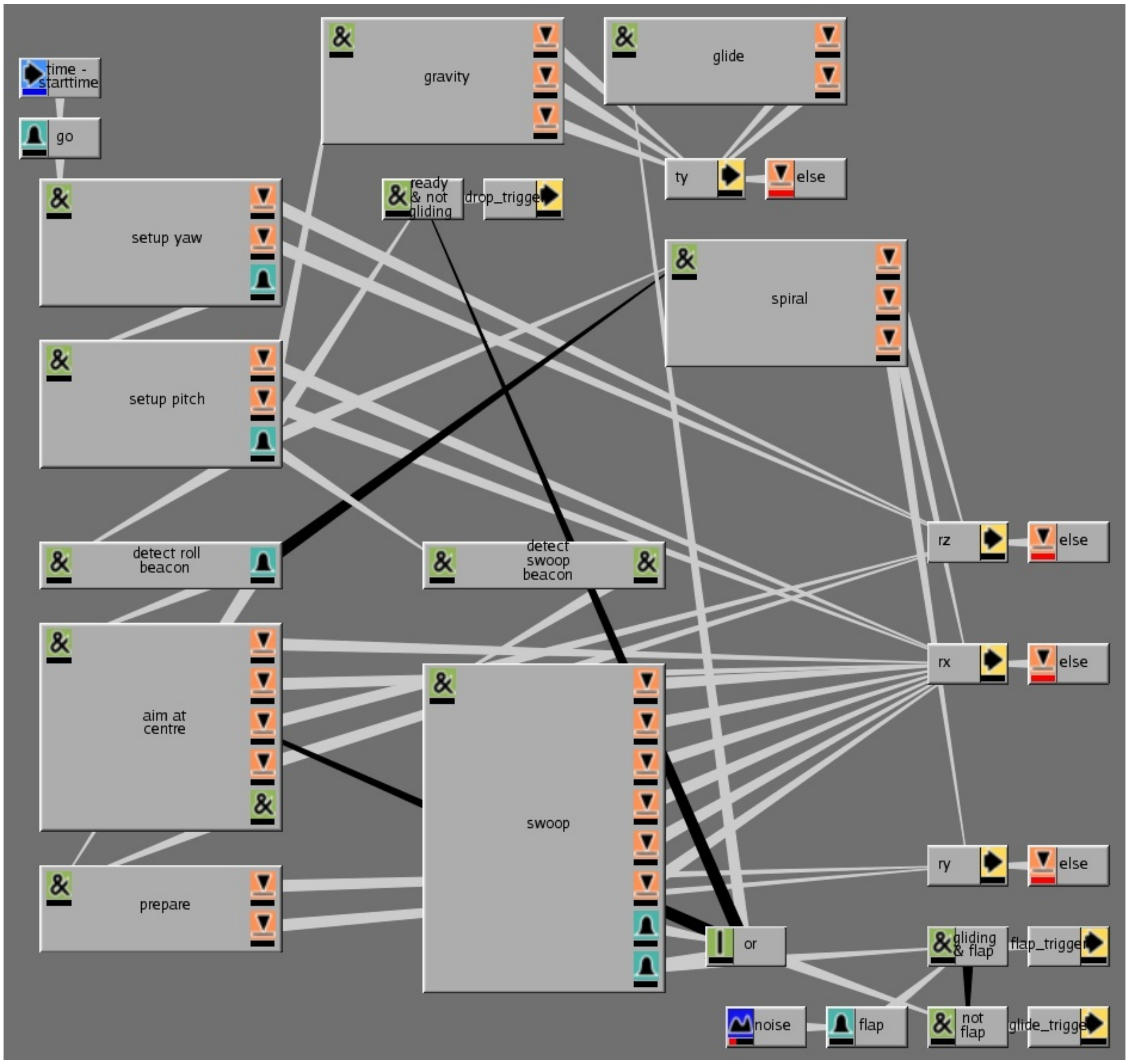


Figure 4.31: Brain of a Spiraller agent

The Spiraller agent's brain (Fig 4.31) starts with the setup yaw macro once the 'starttime' value is reached. After the yaw rotations are complete the setup pitch macro rotates the angel into the right heading to ensure that the helix it creates spirals directly downwards. The end of the set-up triggers the spiral behaviour and the gravity macro, which boosts the angel to terminal velocity within a few frames so that the radius of the helix is fixed as the angel rotates around the falling column. The detect roll beacon and detect swoop beacon behaviours are also enabled, and the angel spirals downwards until the roll beacon is passed. A black connection from the detect roll beacon macro stops the agent from spiralling once the roll beacon is behind the agent, and the aim at centre behaviour is enabled. The agent rotates out of its spiralling position until it is heading downwards, and when the centre beacon is approximately ahead of the angel the prepare macro is triggered, rolling the angel into position to perform the swoop manoeuvre. Similarly to the other agents, when the swoop beacon passes the agent the swoop behaviour is started and the speed control is switched from the gravity to the glide macro, leaving the angel to glide off into the distance.

Placement

The initial placement of each agent within the scene is crucial for their behaviours to produce the intended visual result. The Spiraller agent is the most dependent on its positioning, as the helix that they follow is fixed and must encapsulate the entire column in a tight circle, without causing collisions with any other agents. To achieve this the Spiraller agents are placed along on a curve in the shape of a circle around the origin, with a radius equal to that of the helix. Using trial and error the angle of each agent is offset by a fixed amount from the path of the circle until the circular motion of each Spiraller is centred on the origin. For large numbers of spiralling agents a random height offset is also added to each starting position, to reduce the risk of two agents following the same spiral at the same time and intersecting each other. In order to form a column when they descend the Follower agents are placed inside a circle, sized to fit just inside the curve of the Spirallers so that they are close to each other but not colliding.

At first the Leader agents were placed randomly inside the Follower's circle, but this created problems when the Follower agents followed them away towards one side of the column, causing it to skew. This was fixed by creating another circle in which to place the Leaders, with a small radius in the centre of the column.

Finally, a single centre beacon agent is placed at the centre of the three circles, and the roll and swoop beacons are placed together a short distance outside of the circles, in the direction that the angels will swoop.

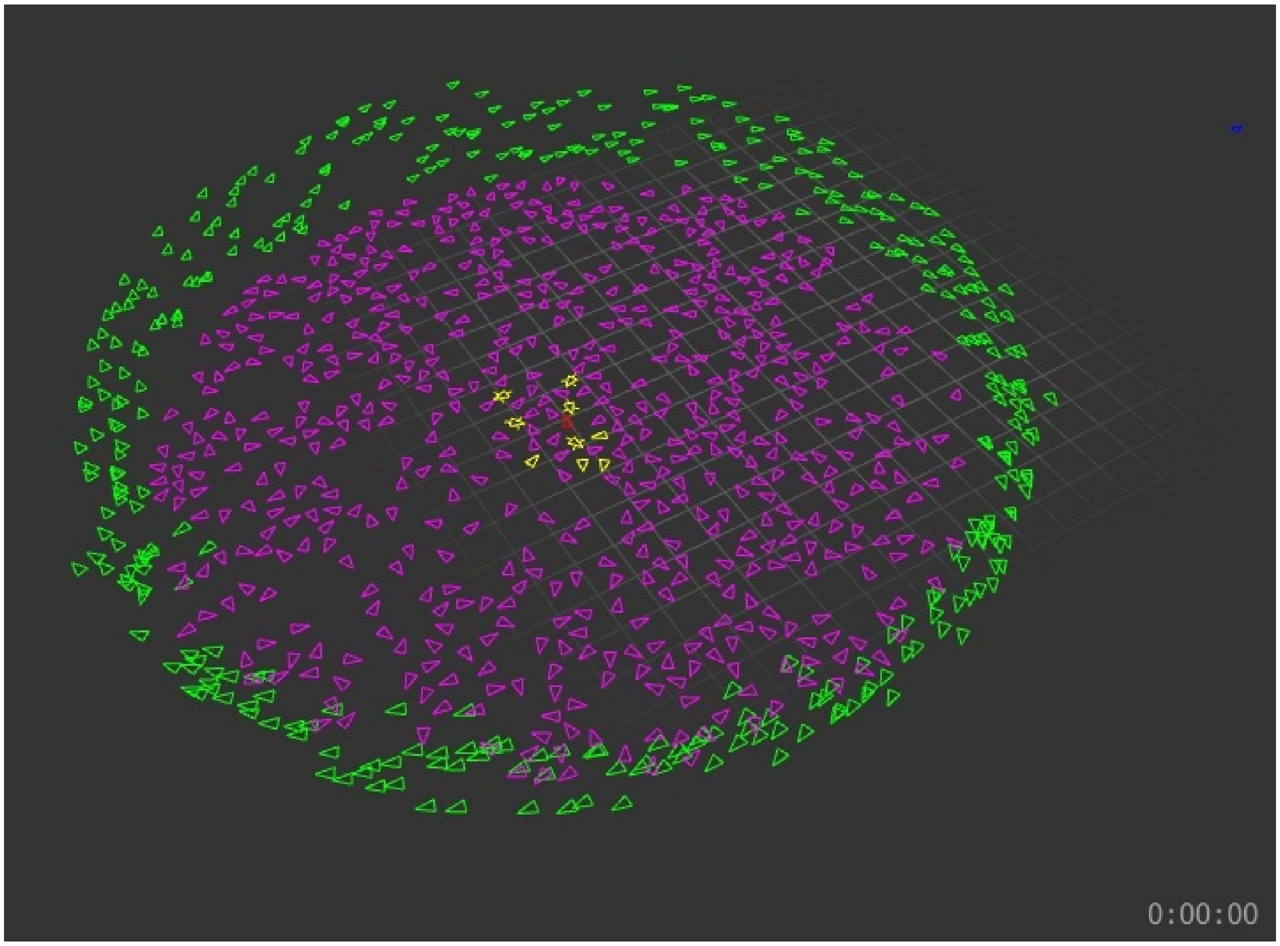


Figure 4.32: The locators of every agent for the final render. The Spirallers are green, the Followers are pink and the Leaders are yellow. The red locator is for the centre beacon and the roll and swoop beacons are placed at the blue marker

For the final render 360 Spiraller agents, 600 Followers, 9 leader agents, and 1 of each of the three beacons were used. Utilising the 'starttime' variables, these angels were set to start dropping over a period of 16.8 seconds. The Leader agent start times ranged from 0 to 15 seconds, whilst the Follower agents' start times ranged from 0.2 to 15 seconds, giving time for a Leader agent to drop first. As the spiralling angels start at terminal speed, the first Spiraller agents began after 2 seconds and finished at 16.8 seconds, giving the other angels a head start.

The 'droptime' agent variables for the roll and swoop beacons were set to 2 and 4 seconds respectively. Because they drop at a speed equalling the terminal speed of the angels, this meant that each angel would have slightly more than 2 seconds after passing the roll beacon in order to prepare for its swooping manoeuvre. The centre beacon was given a drop time of 6 seconds at a faster drop speed than the other beacons, ensuring it was far enough below the column to approximate a point directly beneath each Spiraller.

Modelling

To simplify the modelling of the angels a human character model, “Michael 4”[16], and a sword model[17] (Fig. 4.33) were acquired as a starting point to work from. As well as geometry the human model also came with textures and rigged for inverse kinematics in *DAZ Studio* software[18]. This left only the armour and the wings to be modelled in *Maya*[19]. As reference for the armour it was decided to use the Roman Lorica Segmentata[19] (Fig. 4.34), as it combines a simple structure without too much detail with an impressive appearance.

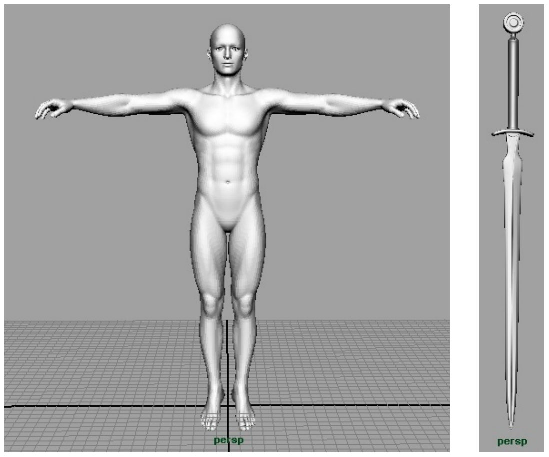


Figure 4.33: Human and sword models

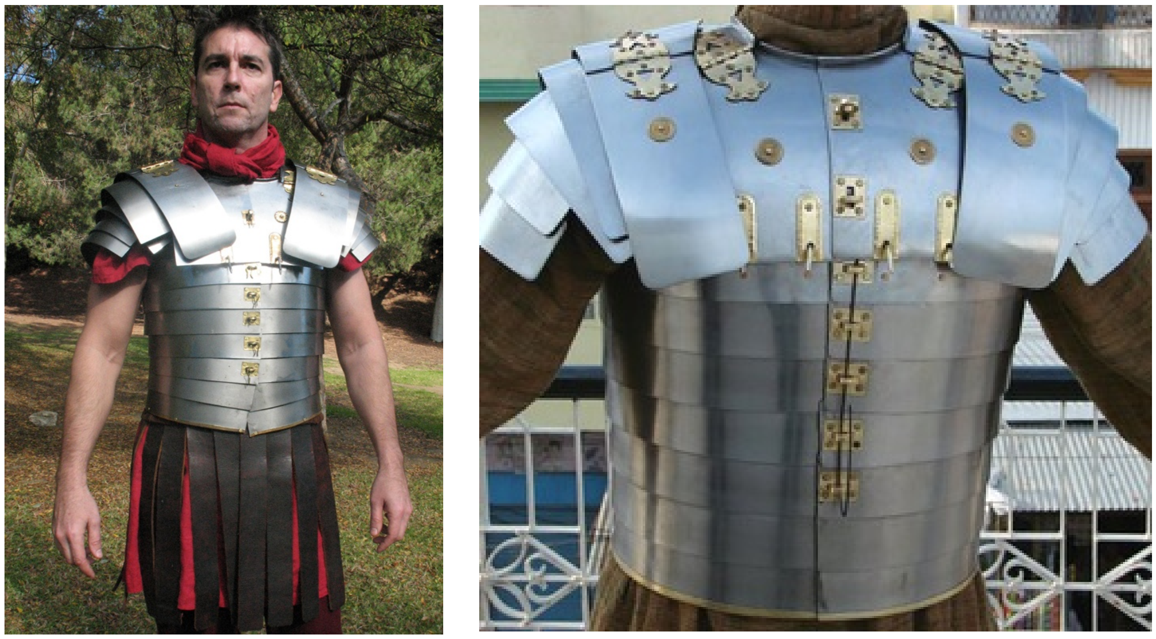


Figure 4.34: Reference images of Lorica Segmentata armour[19]

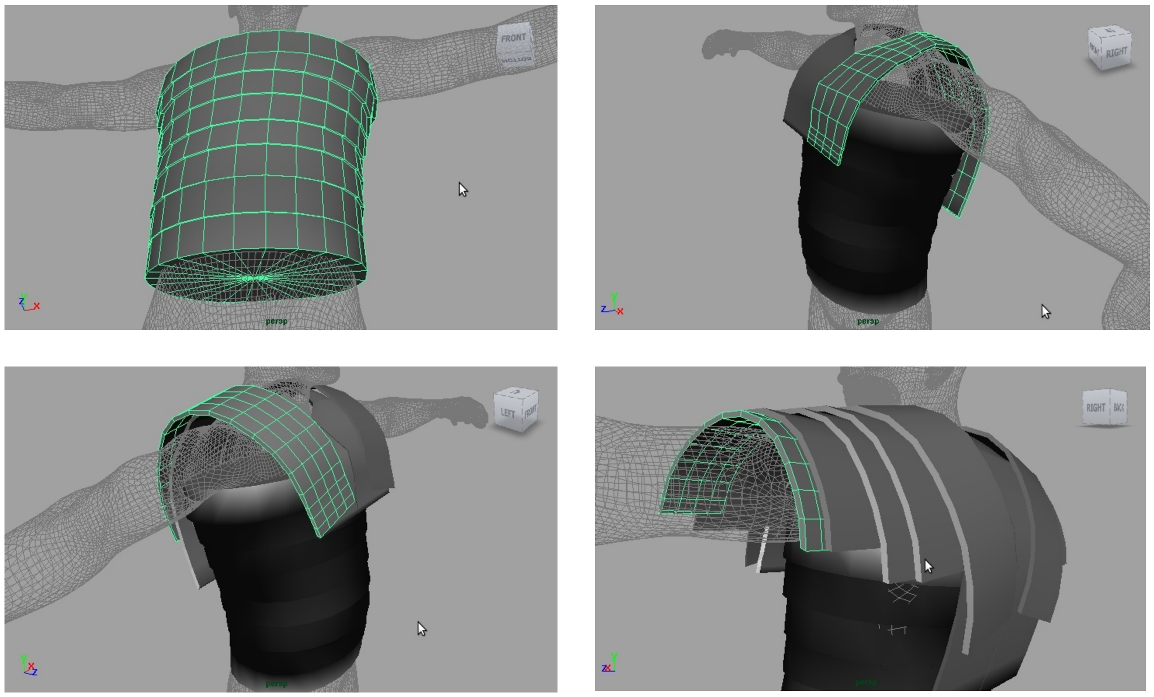


Figure 4.35: Stages of the armour modelling

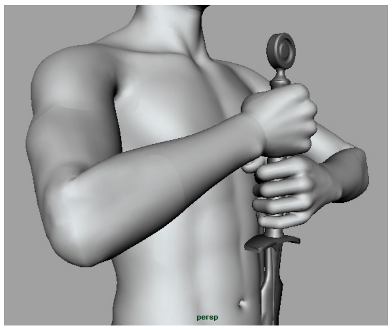


Figure 4.36: Reposed arms and hands

At this point in the modelling it was decided that since the arms carrying a sword would not move very much during flight it would be better to model the armour with them in place, rather than outstretched. To this end the body geometry was taken into *Daz Studio* and re-posed using the inverse kinematics rig to match the angel's position in flight. The hands were brought out in front of the chest and fingers wrapped around the handle of the sword (Fig. 4.36), and the geometry in its new pose was exported as a .obj file to import back into *Maya*.

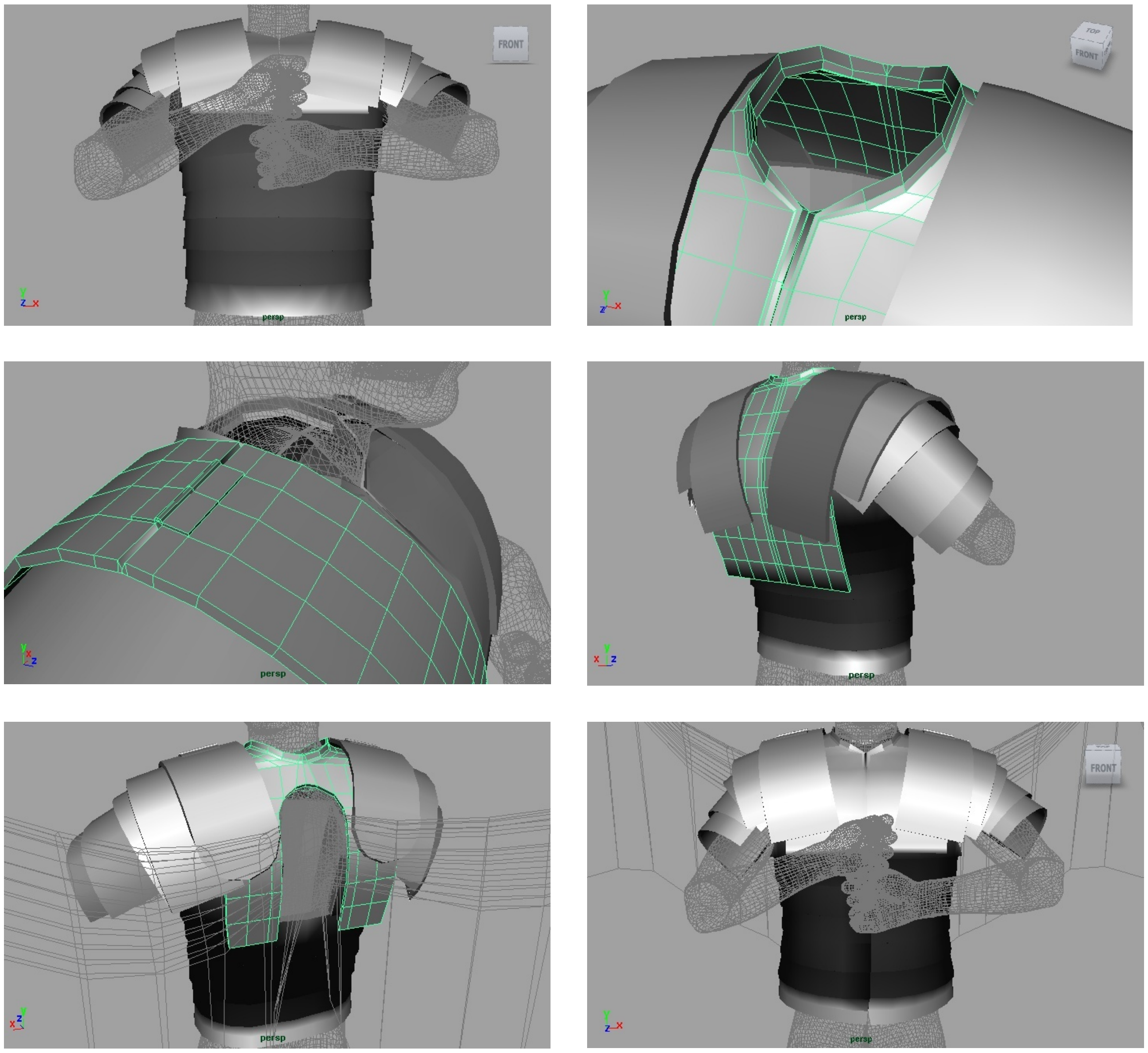


Figure 4.37: Stages of the armour modelling

To cover the bottom half of the angel it was not immediately obvious what should be modelled. With the characters moving so fast the material needed to be rigid to avoid the need for cloth simulation, but also had to appear to allow the angel to move well enough to fight. This resulted in a pair of short metal shorts, incorporating the theme of the Lorica Segmentata with metal plates covering the thighs.

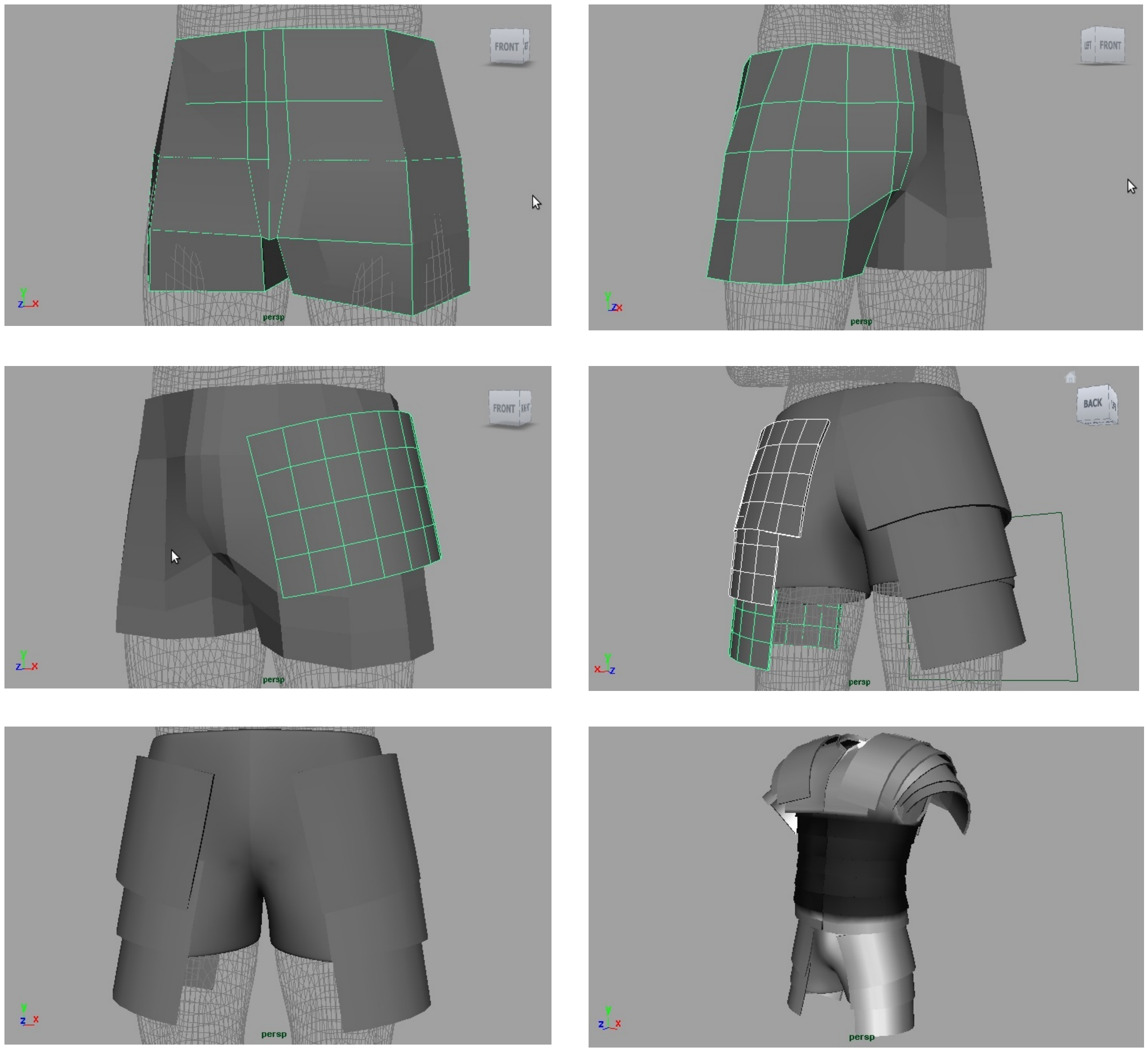


Figure 4.38: Stages of the armour modelling

Rather than modelling individual feathers to produce the detailed outline of the angel's wings, it was decided to combine simpler geometry with a texture that was transparent wherever there weren't feathers. Images of ospreys[21] (Fig. 4.39) were used as reference for the shape and profile of the wings. Once finished the wing and body models were combined into a single mesh to make it easier to bring into *Massive*, and ensure that they couldn't separate and create visible gaps.

Figure 4.39: Reference images of osprey wings[21]

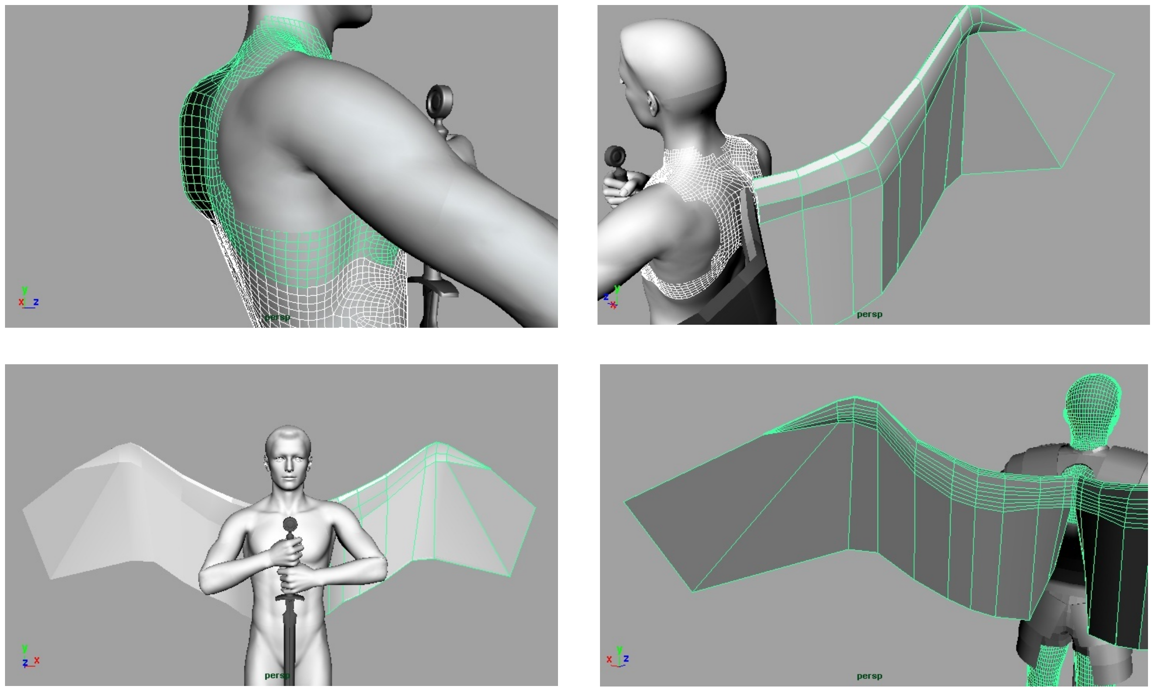


Figure 4.40: Stages of the wing modelling

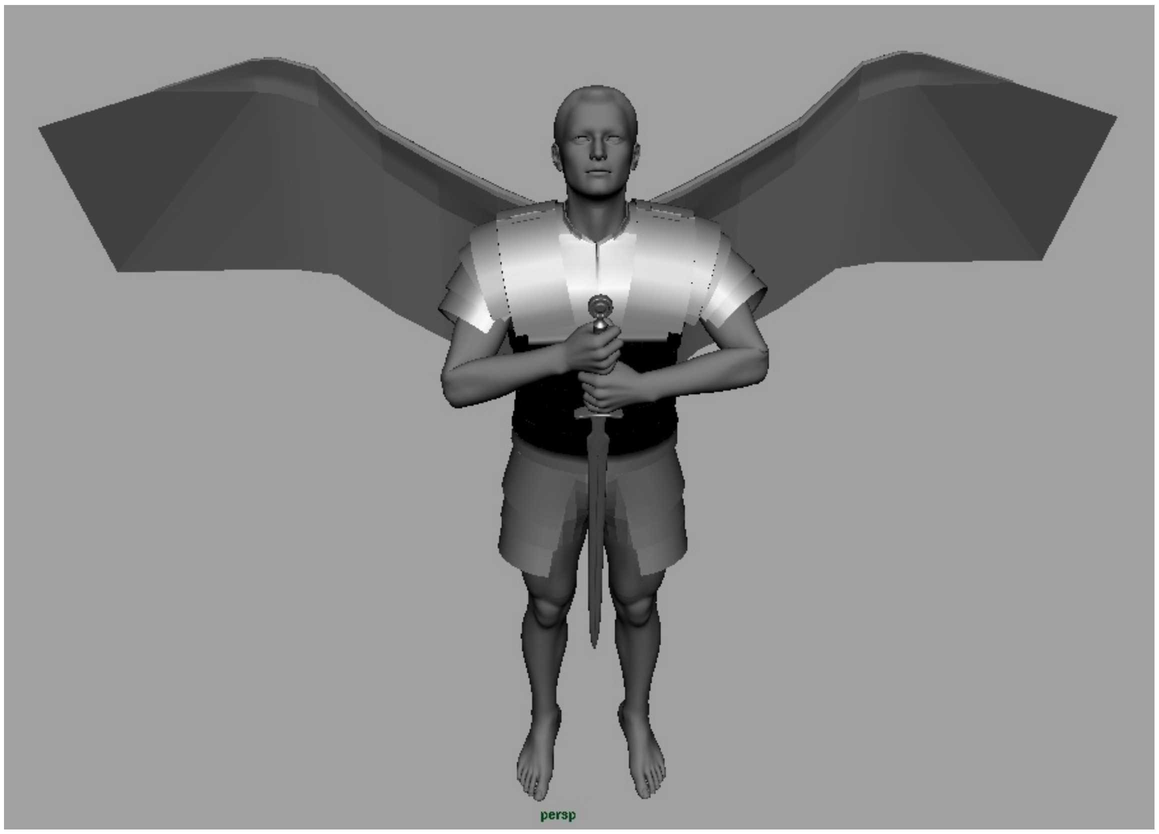


Figure 4.41: Completed angel model

As the desired scene requires a large number of angels to be rendered, it is unnecessary to use such a highly detailed model for each one. For characters that weren't very close to the camera two lower polygon models were created. The first was the mid-poly model created from the high-detail geometry by removing individual edge loops to create a model with fewer and larger polygons. The lowest detail model was created from scratch to match the outline of the other models when viewed from a distance.

As the desired scene requires a large number of angels to be rendered, it is unnecessary to use such a highly detailed model for each one. For characters that weren't very close to the camera two lower polygon models were created. The first was the mid-poly model created from the high-detail geometry by removing individual edge loops to create a model with fewer and larger polygons, reducing the total polygon count from approximately 55000 to 29000. The lowest detail model was created from scratch to match the outline of the other models when viewed from a distance. The low detail model used a total of 811 polygons in *Maya*, but after problems displaying non-planar polygons in *Massive* the model was triangulated, increasing this number to 1561 polygons.

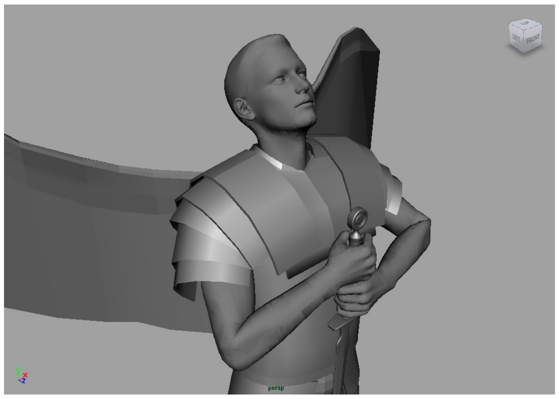


Figure 4.42: Mid-poly model

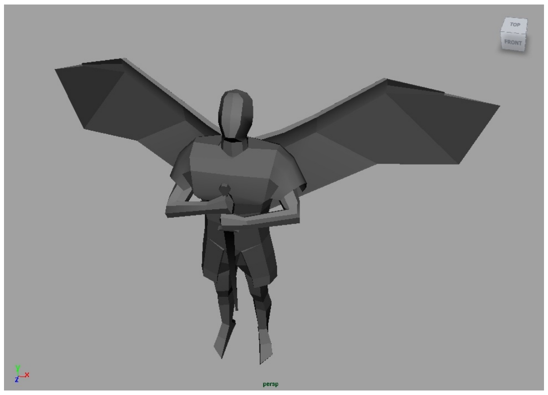


Figure 4.43: Low-poly model

Texturing

Once the modelling was complete, textures were needed to add detail to the surfaces. As the human model had its own already and the metal was expected to use the *Renderman*[22] metal shaders, only the wings needed a new texture creating. Images of feathers[23] (Fig. 4.44) were acquired to combine into a wing, with an alpha background so that the outlines of the feathers would form the visible edge of the wings (Fig 4.45).

Figure 4.44: Images of feathers for the wing texture[23]

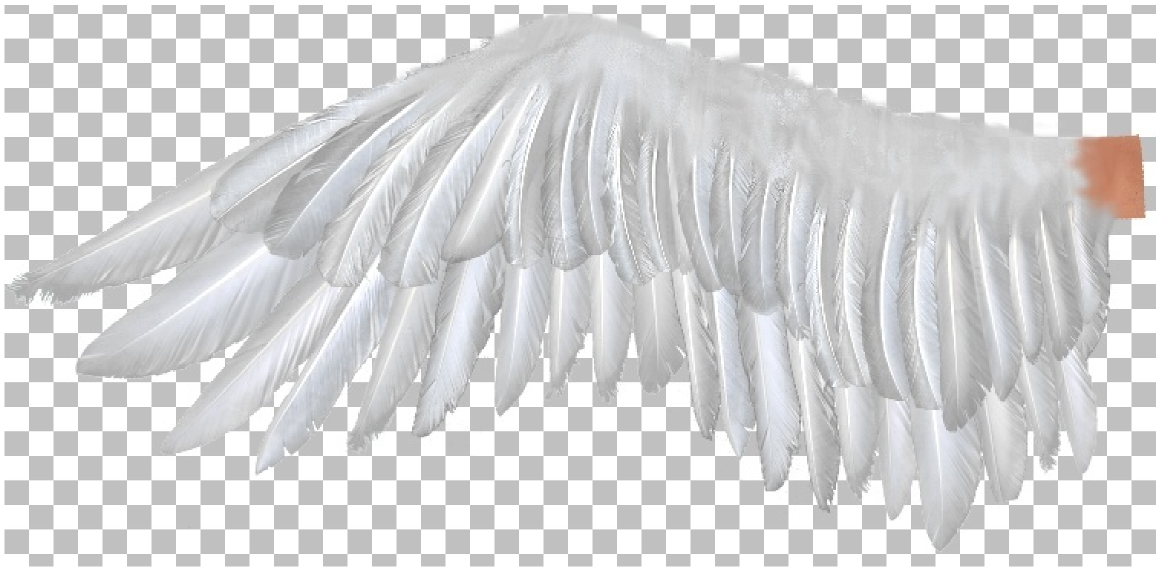


Figure 4.45: Wing texture on a transparency grid

The textures for the male body[16] model came as separate face, torso, limbs and eye images in 512 x 512 or 1024 x 1024 resolutions. Since the body and wing models were joined to form a single mesh for use in *Massive*, similarly all of the body textures and the wing texture also had to be combined into a single 2048 x 2048 pixel map. Once the joint map was created the UV texture coordinates of the mesh were modified in *Maya* to line up with the new texture.

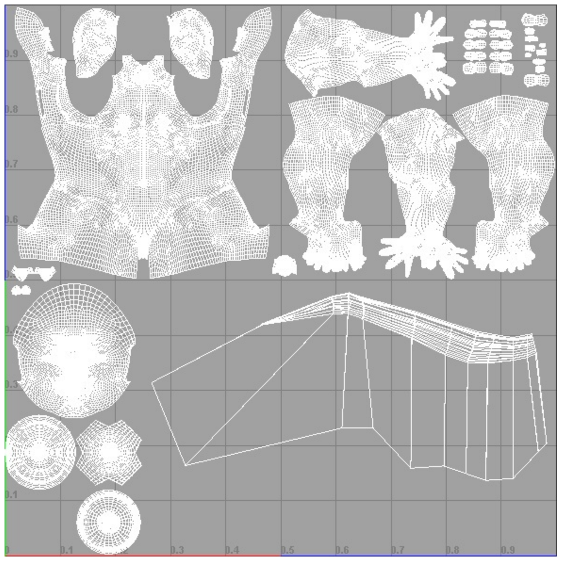


Figure 4.46: UV-coordinates of the high-poly model

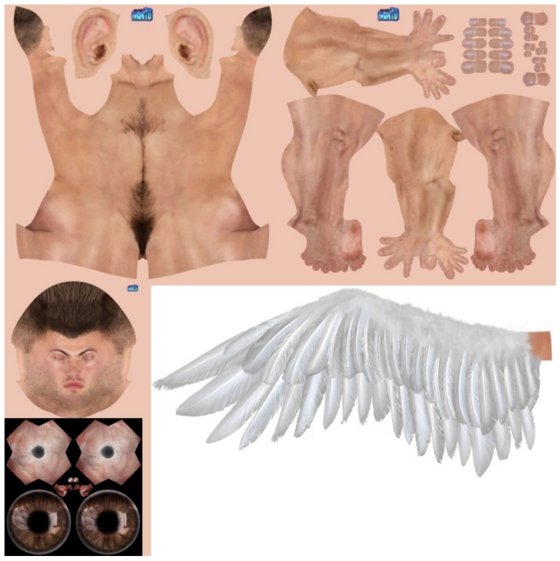


Figure 4.47: Combined skin and wing texture map, with alpha as pure white

To keep the lower poly models quick to render, the same texture map was used, but at 1024 x 1024 resolution for the mid-poly model and 512x512 pixels for the low-poly model.

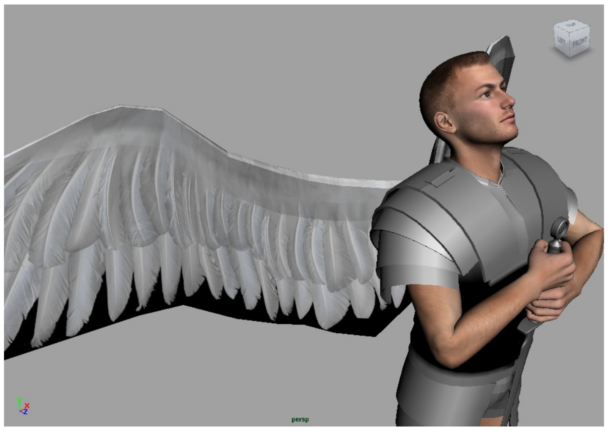


Figure 4.48: Textured high-poly model

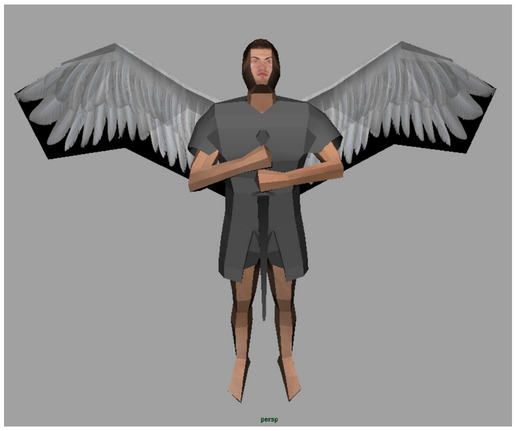


Figure 4.49: Textured low-poly model

<u>Animation</u>

To make the angels more convincing it was decided that their wings should be given simple animations. To achieve this, the angel geometry was first brought into *Massive* to be rigged. Using the body page and Bones window the angel skeleton was matched to the pose of the angel model, and the volumes of influence of each bone in the wings of the skeleton were adjusted so that every point of the wing geometry was constrained to the appropriate bones. As the rest of the body was not going to be animated the volumes of influence were set up only to ensure that all of the points in the model were constrained to the skeleton and would move with the agent when the scene was run. Once the skeleton was matched with the model, the bind pose was exported from *Massive* as a .ma file of joints to be used in *Maya*. These joints were then key frame animated by forward kinematics, before the motion graphs were smoothed using *Maya*'s Graph Editor.

The basic wing poses required were for angels falling at speed, gliding and flapping their wings. These three poses were animated as short loop-able sequences one after another in a single *Maya* scene (Fig. 4.50), with transitions animated between them to move from dropping to gliding, gliding to flapping and flapping back to gliding.

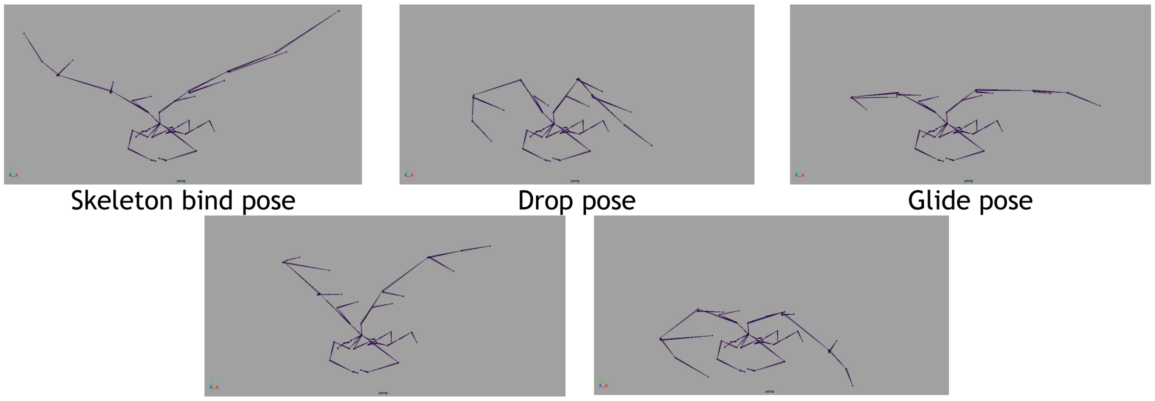
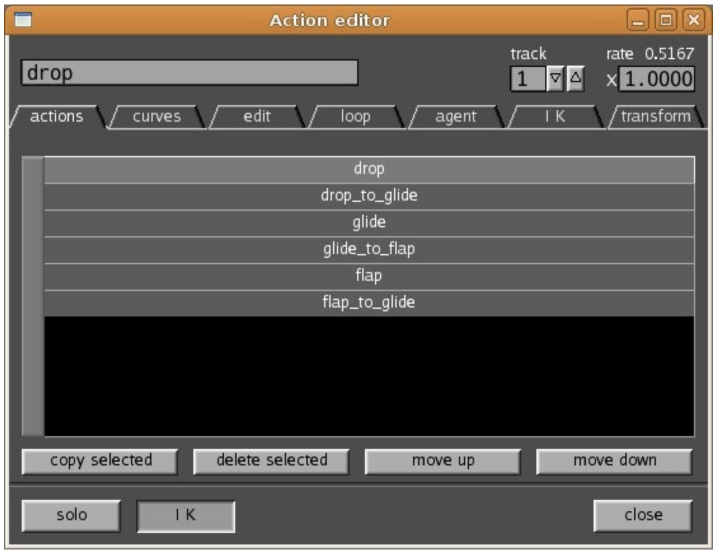

Skeleton bind pose
Drop pose
Glide pose
Poses from wing flap

Figure 4.50: Animation poses in *Maya*

The animations were imported back into *Massive* as a single sequence before being trimmed and named in the Action Editor as six separate actions: drop, drop to glide, glide, glide to flap, flap and flap to guide (Fig. 4.51).



drop
drop_to_glide
glide
glide_to_flap
flap
flap_to_glide

Figure 4.51: Animations in *Massive*'s Action Editor

The "one shot" attribute in the loop tab was set for each transition action to ensure that they were only triggered to move from one loop to another, and wouldn't be played in a loop themselves.

The next task was to add latch and transition curves to each action. Latch curves dictate when one animation can be interrupted in order to begin another, and transition curves specify how many frames to use when blending an action with the animation that preceded it. These two curves are created for an action along with its agent curves after the animation is specified as being "static", "turning", "locomotion", "ramp" or "other". Each type of animation creates a different set of agent curves to specify how the agent's position and rotation have changed in carrying out that action. This caused difficulties, because the wing animations were designed to be played alongside the motion already built into each agent's brain, but the agent curves entirely controlled the movement and rotations of an agent whilst an action was being carried out. This meant that each character was stuck in its starting position when the actions were enabled, and attempts by the brain to move the agent were ignored. This problem was overcome by specifying each animation as "other" before looking in the curves tab for every agent curve created and deleting them.

Because the animations were created to start and end in fixed poses the latch and transition curves were simple to set up and no other blending aids were required.

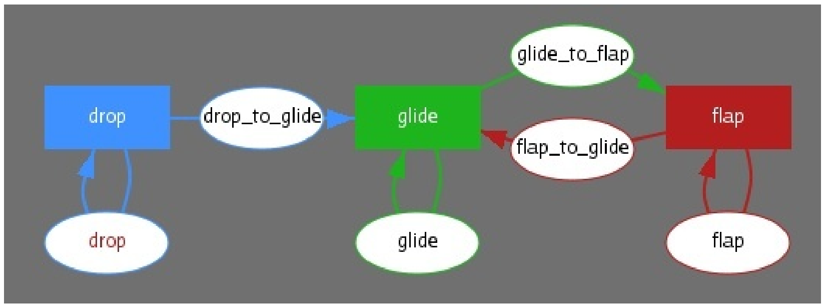


Figure 4.52: Agent motion tree

The *Massive* motion tree for each agent consists of the three key states that an angel can take, with a looped animation connected to each one and transitions to move between them (Fig 4.52). Three trigger channels named "drop_trigger", "glide_trigger" and "flap_trigger" control the tree. Drop is the default action and the only one assigned to the drop trigger. Drop to glide, glide and flap to glide are all controlled by the glide trigger and glide to flap and flap use the flap trigger. These triggers are channels which can be controlled by the brain of an agent. When a trigger is given a value above 0.5 then *Massive* will assess the motion tree to see which action associated with that trigger is available from the current state and carry it out.

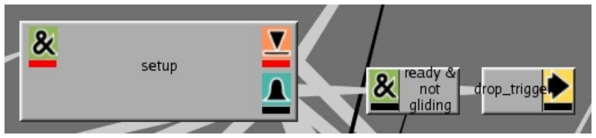


Figure 4.53: Drop trigger connected in an agent brain

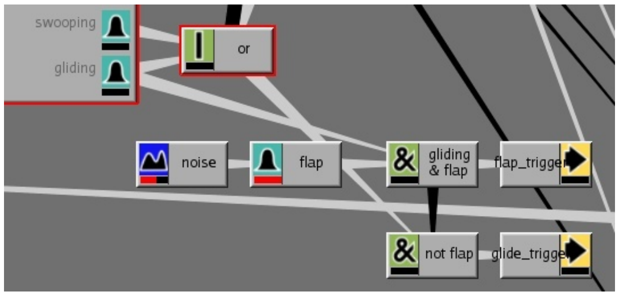


Figure 4.54: Glide and flap triggers connected in an agent brain

The three animation triggers are connected to the brain of each agent in the same way. The drop trigger is connected to the 'ready & not gliding' AND node, which evaluates to true when the setup macro has finished and the angel is falling. This causes the looping drop animation to play until the angel begins to swoop. An OR node is used to determine if the angel is either carrying out the swoop behaviour or has already finished it and has started to glide. When the OR node first evaluates as true, the swoop manoeuvre has begun; the AND node controlling the drop trigger is stopped and the 'not flap' node controlling the glide trigger is activated, causing a transition from the drop pose to the glide pose at the start of the swoop. The flap trigger is only introduced when the swoop behaviour is finished and the agent is gliding. A noise node is connected to the 'flap' fuzz node, which evaluates as true when the random number generated is greater than 0.5. When it is true the flap trigger is activated, causing the angel to flap its wings as it glides, until the noise value drops again and the motion tree transitions the animation back into the glide loop.

Lighting

The default lighting set-up for a *Massive* scene consists of four lights, named 'key', 'sky', 'bounce' and 'ambi' (Fig. 4.55), short for ambient, which are designed to replicate daylight for an outdoors scene. This lighting setup was an appropriate simple set-up for demonstrating the agents, but the ambient light was toned down in intensity from 1 to 0.15 to prevent the characters looking washed out, with the other lights being made brighter to compensate. Also the original lights were altered from directional lights pointing in the same direction to directional lights pointing in multiple directions.

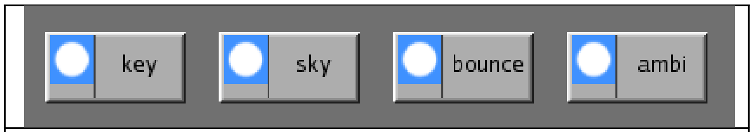


Figure 4.55: Default lights in *Massive*

Shaders

The first shader tested for the body and wings of the angel character was *Renderman*'s texmap shader for projecting a texture onto a surface using uv-coordinates. This shader does not take into account the alpha values of the texture map however and shaded the parts of the wing geometry without feathers black, occluding any objects behind them including the body and armour of the angel. Since none of the other default shaders support alpha in textures a new shader was required to render the wings correctly. The Alpha-channel texturing shader by Daylon Graphics[24] was acquired as a starting point for this new shader, as it combined the simple plastic shader coefficients and functionality with the ability to read texture files including the alpha channel. This shader uses the alpha value of the texture to blend between the colour of the texture and the default colour assigned to the surface, but was modified to use the alpha value as the surface opacity Oi instead, and to incorporate this opacity value into the surface colour Ci as well. The new txalpha shader was brought into *Massive* by adding its parent directory in the Shader Paths window. With this shader the transparency of the texture was corrected and only the feathers can be seen on the wings of the angel. Also with the specular coefficient Ks lowered from the default 0.5 to 0.1 and the roughness value increased the plastic look of the shader was removed and the skin and wings were more convincing.

For the armour and the sword both the metal and shinymetal default shaders were tested. The metal shader gave an accurate representation of metal but was too dark for the steel-look that was desired, with no way to adjust its diffuse colour. The shinymetal shader on the other hand was extremely bright, mimicking chrome, and had too much reflected light to appear realistic. To get more control over the colour of the metal it was decided to use a texture, and since no transparency was required the default texmap shader was used. An image of brushed metal[25] was acquired and modified into a simple texture, with no extra detail added. The roughness value for the specular highlights was set as 0.1 to mimic the metal shader, the coefficient Ks was set to 0.75 and the diffuse coefficient Kd was also set to 0.75 to lower the brightness.

Figure 4.56: Metal texture

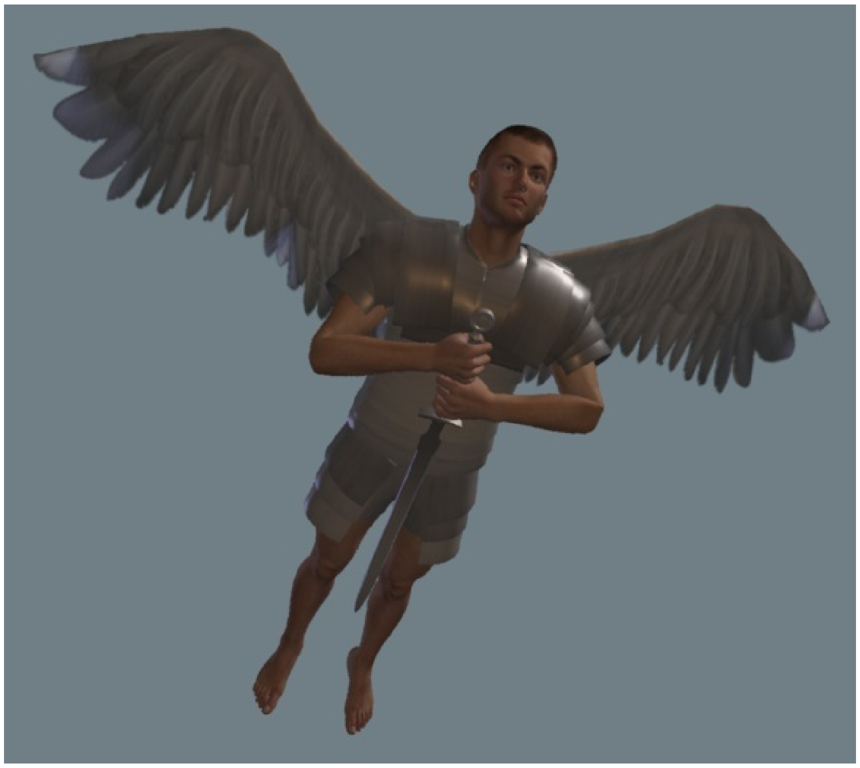


Figure 4.57: Rendered high-poly angel

<u>Cameras</u>

The first camera used to demonstrate the agents was created in *Maya*. The scene was exported from *Massive* as a series of particle cache files, with each particle demonstrating the motion of a single agent. This simple representation of the scene showed the movement of the agents in a way which could be played forwards and backwards without loading times, making it much easier to decide where to place the camera to capture the most important action. The camera was set up to start close to the column in order to see the first angels falling, before tracking back out to a point along the path taken by the angels after they pull up, so that they fly over and past its position.

The second camera was setup in *Massive*. Using the constrain options this camera was attached to a single Follower agent, moving and rotating in the same way as the angel to get a first-person perspective of the inside of the column.

<u>Background</u>

Rather than demonstrate the *Massive* agents against a black background a simple static billboard with a cloudy sky texture[26] was created in *Maya* to place behind the agents. Using the key-framed camera already created in *Maya* a sequence of images was rendered that matched the camera movement in the *Massive* shot.

Unfortunately this method could only be used for the first camera, as the second was dynamically created during the *Massive* simulation, and could not be exported in a format that *Maya* could read.

Render set-ups

Two render set-ups were created in *Massive*, one for each camera, using the 'PRman rgba' render pass. This is a *Renderman* beauty pass including an alpha channel, so that areas without agents are rendered as transparent, and the render can be easily composited with other footage.

Level of detail

Because of the large number of characters in the scene it was important to ensure that only those close to the camera were rendered using the highest detail model, and that the furthest away angels used the lowest detail model. This was achieved using *Renderman*'s Level of Detail function. This function enables multiple models of varying detail to be included in a .rib file, so that the renderer can decide which one to use dynamically at render time depending on the size of the object in the frame. *Massive* does not support Level of Detail itself however, so a separate method[27] was required instead.

Inside *Massive* a new variable "lod" was created for each agent, and used to control option nodes in the body page which selected which detail level of geometry to use.

When the .rib files are created for rendering, they contain lines like the following for each agent:

```
Procedural "DynamicLoad" ["massive.so" "1 /AgentDir/agent_1_1.cdl /AgentDir/frame.1.apf 1
starttime 2.1 lod 0"] [-119.343 119.958 -15.3002 76.9196 -94.8108 118.459]
```

This command loads an agent name "agent_1_1" with the Sim data from "frame.1.apf", and variables "starttime" and "lod" with values 2.1 and 0 respectively. The numbers in square brackets represent the bounding box coordinates of the specified agent. Using a script each of these lines can be replaced with a Level of Detail structure which calls the same command, changing the value of the "lod" variable depending on the required level of detail. The final .rib file looks like the following for each agent:

```
AttributeBegin

Detail [-119.343 119.958 -15.3002 76.9196 -94.8108 118.459]

DetailRange [0 0 2000 2000]

Procedural "DynamicLoad" ["massive.so" "1 /AgentDir/agent_1_1.cdl /AgentDir//frame.1.apf 1
starttime 2.16851 lod 0"] [-119.343 119.958 -15.3002 76.9196 -94.8108 118.459]

DetailRange [2000 2000 42000 42000]

Procedural "DynamicLoad" ["massive.so" "1 /AgentDir/agent_1_1.cdl /AgentDir//frame.1.apf 1
starttime 2.16851 lod 1"] [-119.343 119.958 -15.3002 76.9196 -94.8108 118.459]
```

```
DetailRange [42000 42000 10000000 10000000]

Procedural "DynamicLoad" ["massive.so" "1 /AgentDir/agent_1_1.cdl /AgentDir//frame.1.apf 1
starttime 2.16851 lod 2"] [-119.343 119.958 -15.3002 76.9196 -94.8108 118.459]

AttributeEnd
```

The numbers after the "Detail" command are the coordinates of the bounding box used for testing the size of the object being drawn, and are set equal to the bounding box of the agent. The numbers after the "DetailRange" command represent the amount of the frame taken up by the object being drawn. Rendering these .rib files with the new function included means that when an agent is being rendered a test will be carried out to find out how close it is to the camera. If it is a long way away then the agent will be loaded with an "lod" value of 0, and the low-poly model will be used. Similarly closer agents will be called with level of detail values of 1 or 2 and the mid or high-poly models will be used respectively. This addition to the rendering process dramatically reduces render time.

Since the Level of Detail function overwrites the value of the "lod" variable outside of *Massive* it would be useful to keep it at 0 inside of *Massive* to avoid having to load the high-poly model. The first attempt to achieve this involved using the expression window to set the value with the expression "0". This did not work however, as the expression was applied after the call from the .rib file and undid the work of the Level of Detail function. Instead it was found that setting the range of the variable in *Massive* from 0 to 0 kept the value at 0 for all agents in *Massive*, but allowed for higher values to be used when they were set in the .rib file.

<u>Rendering</u>

The first stage in rendering out the Massive simulation to an image sequence involved creating a preview sequence using the Pics output in the Sim dialog. This feature runs the simulation and outputs a series of screen captures of the Massive View window as .tif files. This sequence can then be played back in real-time, to check that the camera is set up correctly and that the agents behave as expected. If the preview is satisfactory then the Sim Dialog can be used to run the simulation again, creating the .apf and .rib files required for rendering. The .apf or 'Sim' files store the position and animation of each agent as it is dictated by the agent brains, and can be used instead of recalculating the brains in later simulations. Two types of .rib files are outputted by Massive for rendering, one containing the scene information such as lights and camera position, and one containing all of the agents within the scene. To use the Level of Detail function only the agent .rib files need altering, with each agent being replaced as described previously. All agents used in the scene need to be saved before the .rib files are rendered, as the files they are stored in are used as part of the rendering process. Once the .rib files are ready to be rendered, a shell script file named 'render_script.sh', created by Massive along with the .rib files, can be run from the terminal to render each frame into a .tif image file.

5. Conclusion and Further Work

The aim of this project was to create Massive agents that could be used in a shot featuring a large number of angels leaving Heaven to fight a battle. These agents were meant to produce angels that dropped from the sky together in a coordinated column, with some spiralling around the outside, until they gathered enough speed to use their wings and fly towards the camera. Although several of the intended methods for producing these behaviours did not make it into the final agents, the alternative simpler, less robust methods still managed to achieve the desired effect. The render from the perspective of an angel is as confusing as it is entertaining, but the dollying shot is a good demonstration for the shot that was intended. Considering this I would describe the project as a success, but with a lot of scope for further work.

To improve on the agents the following could be investigated:

- dynamically producing the swoop and spiralling manoeuvres, creating a wider variety of movements and allowing interaction between agents whilst they are carried out
- combining the different agents into one agent, that changes its behaviour depending on its situation
- creating more coordinated angels that can move together in a tighter formation before spreading out
- extending the angel behaviours to their first contact with the enemy

To improve on the visual aspect of the scene further work could include:

- multiple angel models and/or multiple textures for a wider variety of angels
- more animations to reflect the movement of the agent, and animations on the entire body
- 3D clouds or a matte painting backdrop, with a volumetric cloud or alpha mask to hide the angels before they drop
- multiple camera angles to place focus on the interesting action, and stretch out the effect
- volumetric lighting, with the sun shining from the clouds behind the angels

References

[1] The Internet Movie Database, 2009. Legion (2010). Available from: http://www.imdb.com/title/tt1038686/ [Accessed 20 August 2009].

[2] The Internet Movie Database, 2009. Paradise Lost (2011). Available from: http://www.imdb.com/title/tt0484138/ [Accessed 20 August 2009].

[3] MTV News, 2008. Scott Derrickson Says His 'Paradise Lost' Film Might Lead To Sympathy For The Devil. Available from: http://www.vh1.com/movies/news/articles/1591447/20080723/story.jhtml [Accessed 20 August 2009].

[4] Massive Software, 2008. Massive (3.51) [computer program].

[5] Dogma, 1999. Film. Directed by Kevin Smith. USA: View Askew Productions.

[6] Constantine, 2005. Film. Directed by Francis Lawrence. USA: Warner Bros. Pictures.

[7] X-Men: The Last Stand, 2006. Directed by Brett Ratner. USA: Twentieth Century-Fox Film Corporation.

[8] Pitch Black, 2000. Directed by David Twohy. USA: Polygram Filmed Entertainment

[9] The Matrix Revolutions, 2003. Directed by Andy Wachowski and Larry Wachowski. USA: Warner Bros. Pictures.

[10] Computer Science for Fun, ca. 2007. How Who's fat just walks away. Queen Mary, University of London. Available from: http://www.cs4fn.org/simulation/adipose.php [Accessed 20 August 2009].

[11] Massive Software. ca. 2008.Ambient agent. (0.9.0). [Massive agent]

[12] Macey, J. ca. 2008. GraphicsLib. (52). [Library]

[13] Mckenzie, C. Flight Dynamics Engineer, Marshall Aerospace Ltd. (personal communication, April 2009).

[14] Devlin, K. 2003. The Double Helix. The Mathematical Association of America. Available from: http://www.maa.org/devlin/devlin_04_03.html [Accessed 20 August 2009].

[15] Dallal, G. E., 2000. Introduction to Simple Linear Regression. Available from: http://www.jerrydallal.com/LHSP/slr.htm [Accessed 20 August 2009].

[16] DAZ 3D. 14 October 2008. Michael 4 Base. (4) [3D rigged model and textures]. Available from:

http://www.daz3d.com/i/3d-models/-/michael-4-base?item=7877&_m=d&refid=362817725 [Accessed 20 August 2009].

[17] 3DaMaze. 15 July 2005. Sword. [3D model]. TurboSquid. Available from: http://www.turbosquid.com/3d-models/free-3ds-mode-sword/269802 [Accessed 20 August 2009].

[18] DAZ 3D Inc. 2009. DAZ Studio 3.0. (3.0.1.135) [computer program].

[19] Autodesk. 2008. Maya 2008 Extension 2. (02 25 03 22) [computer program]

[20] Imperium Ancient Armory, 2007. Newstead Cuirass. Available from: http://www.imperiumancientarmory.com/Newstead%20Lorica.htm [Accessed 20 August 2009].

[21] Bartosik, M. B., 2006, Life on the Osprey time Photo Gallery by Mark B Bartosik at pbase.com. Available from: http://www.pbase.com/mbb/life_on_the_osprey_time [Accessed 20 August 2009].

[22] Pixar. ca. 2008. Renderman. (14.0) [computer program]

[23] CGTextures, [CG Textures] - The worlds largest free texture site, Available from: http://cgtextures.com/ [Accessed 20 August 2009].

[24] Gardener, R., 2003. Alpha-channel texturing shader. Daylon Graphics Ltd. Available from: http://www.daylongraphics.com/products/leveller/shaders/index.php#txalpha [Accessed 20 August 2009].

[25] Masters Appliance Heating & Air, 2009. Home. Available from: http://mastersappliancehvac.com/ [Accessed 20 August 2009].

[26] lightfiretech, 11 October 2006. Sky_Collection_01 – Cloudy. TurboSquid. Available from: http://www.turbosquid.com/FullPreview/Index.cfm/ID/325022 [Accessed 20 August 2009].

[27] Woods, A. 2008. Massive Community Site - Forum - Rendering - LOD – prman. Available from: https://secure.massivesoftware.com/community/modules/newbb/viewtopic.php?topic_id=646&forum=9&post_id=2881& [Accessed 20 August 2009].

Appendices

The following video clips are included in the hand-in as part of the Previous Work section:

Dogma – Bartleby.avi[5]
Dogma – Metatron.avi[5]

X_Men_3-Angel-1.avi[7]
X_Men_3-Angel-2.avi[7]
X_Men_3-Angel-3.avi[7]

Pitch Black 1.avi[8]
Pitch Black 2.avi[8]
Pitch Black 3.avi[8]
Pitch Black 4.avi[8]

Matrix Revolutions 1.avi[9]
Matrix Revolutions 2.avi[9]