INNOVATIONS REPORT

MUSIC-DRIVEN FLOCKING-SYSTEM

GABRIEL ARNOLD
c1416194
BACVA
NATIONAL CENTRE FOR COMPUTER ANIMATION
BOURNEMOUTH UNIVERSITY
2006

CONTENTS

1. INTRODUCTION

1.1 Abstract

This report outlines the design and implementation of a Music-driven Flocking System.

Section 1 introduces the initial aims of the project including what is intended to be the final product of the project. It also looks at what challenges are foreseen.

Section 2 documents research on Flocking and Music Analysis. It investigates why flocking occurs in nature and how it is simulated in computer graphics. The Music Analysis research also looks at a plug-in for Maya created by Andrew Mitchell [MiM_05] that performs music analysis.

Section 3 presents solutions for the challenges that were foreseen, and also gives a comprehensive description of how they were implemented in Maya to make the Music-Driven Flocking System.

Section 4 concludes the report, giving a summary and a critical analysis of the project. Future work is also outlined.

1.2 Aim

The aim of this project is to bring together two concepts: Flocking and music-driven animation.

In the Innovations Unit Guide, there was a quote which particularly stood out to me:

> *"Where do new ideas come from? The answer is simple: differences. While there are many theories of creativity, the only tenet they all share is that creativity comes from unlikely juxtapositions."*
>
> [Neg_95]

The idea of bringing together two concepts to make something new was inspiring. The

phenomenon of flocking is fascinating, in nature, as well as in computer graphics. The notion of using the beat, rhythm, volume and other aspects of music to drive motion and deformation in abstract animations is also compelling. To bring the together the two concepts may not necessarily be the most unlikely of juxtapositions, as they both provoke imagery of motion of similar kinds, but they are a juxtaposition that has the potential for very intriguing outcomes and an innovative animation.

1.3 Product

The product of this project will be an animation displaying what the Music-Driven Flocking tool is capable of.
A decision was made that the type of flock to be simulated would be a school (or schools) of fish: This was due to the fact that other types of flock tend to have less exaggerated movements. The ability of more exaggerated movement would help highlight that the school are being driven by music.
A Down-Tempo piece of music will be used for the animation because although having chosen to simulate a school of fish allowed for more amplified movement, it was deemed that it would be impossible to uphold a natural looking school with the speed of an Up Tempo piece of music.
With this in consideration, the approach to making the Music-Driven Flocking tool was more Ad Hoc than generic: Though it will be possible to use any piece of music to drive the flocking system it may not look natural. I am focusing on making the tool work well for the particular piece of music that I have chosen.

It is important that the flock resembles a school of fish, but of course in nature, schools of fish don't move to music, so it is acceptable if it doesn't look completely organic.

1.4 Challenges

Essentially, there are 5 main challenges in this project:

- To make a flocking-system.
- To find a way to obtain and manipulate data from a music file.
- To find a way to obtain music data on a per frame basis (such as Fourier Frequency Analysis Spectrum, and beat detection.)
- To find a way to use this data to influence the flocking system.
- To make an animation to show the capabilities of the tool.

Undoubtedly, the most challenging aspect of this project will be exactly how to use the music to influence the movement of the flock: Though schools of fish pulsate in a way reminiscent in an abstract way of music, music most often fluctuates in a more erratic and exaggerated way, even if the music is down tempo. So the challenge is to limit how erratic the fluctuation is, without losing the sense that the music is influencing the school.

2. BACKGROUND AND PAST WORK

2.1 Flocking in Nature

In nature, animals flock for several reasons:
For protection as there is more safety in numbers and also to confuse predators. [Blu_01]
If a member strays away from the flock they are the one likely to get attacked.

Photograph of flocking birds,
by José L Gómez de Francisco (2001).

"It took days to get just the right picture of this extraordinary phenomenon – thousands of starlings in an enormous plume, like one huge organism, pouring down into a reedbed roost against the orange light of a setting sun. Starlings probably gather in such huge numbers for the same reason that fish shoal, to confuse predators by sheer numbers and also to exchange information about good feeding places for the following morning."

Rosamund Kidman Cox,
Caption for the above photograph,
from BBC Light on the Earth book.

"For the hunted there are few places to hide. Schooling Mackerel: they have already sensed the sonar beams of approaching dolphin. They're only defence is to gather into a ball. Any individual that stayed out of the shoal would be quickly picked off – within it there is at least some chance of survival."

Sir David Attenborough,
BBC Blue Planet Series.

How do predators interact with shoaling fish?

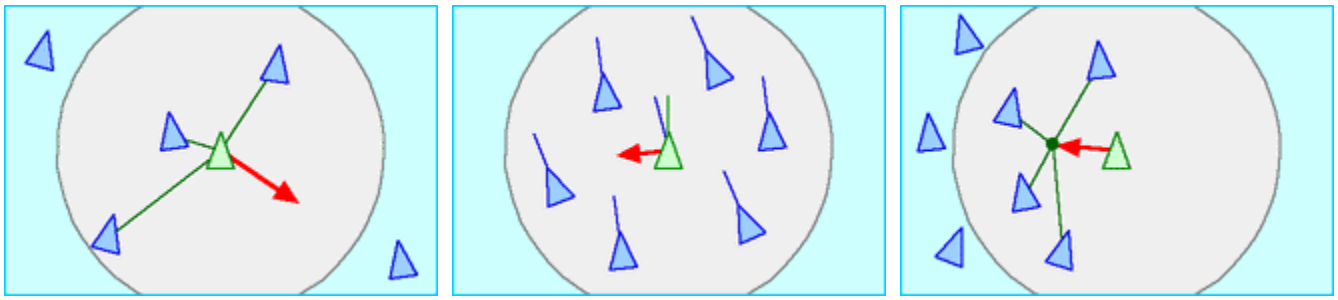
2.2 Flocking in Computer Graphics

2.2.1 Boids

Craig Reynolds wrote a SIGGRAPH paper on the implementation of flocking in computer graphics for the 1987 SIGGRAPH conference [Rey1987]. He called members of the flock Boids (short for bird-oids). The flocking model he invented gave each member of the flock 3 rules to follow:

1) **"Separation**: steer to avoid local boids.
2) **Alignment**: steer towards the average orientation of local boids.
3) **Cohesion**: steer to move toward the average position of local boids."

[Rey1987].



[Rey87]

The principal of Reynold's flocking algorithm is that every frame/timestep each boids new position is calculated by taking the current position and then adding the velocity at the end of the last frame plus the vectors calculated for each of the flocking rules. Conrad Parker gives an explanation of this in pseudocode as follows:

```
"Vector v1, v2, v3
Boid b

FOR EACH BOID b
        v1 = rule1(b)
        v2 = rule2(b)
        v3 = rule3(b)

        b.velocity = b.velocity + v1 + v2 + v3
        b.position = b.position + b.velocity
END"
```

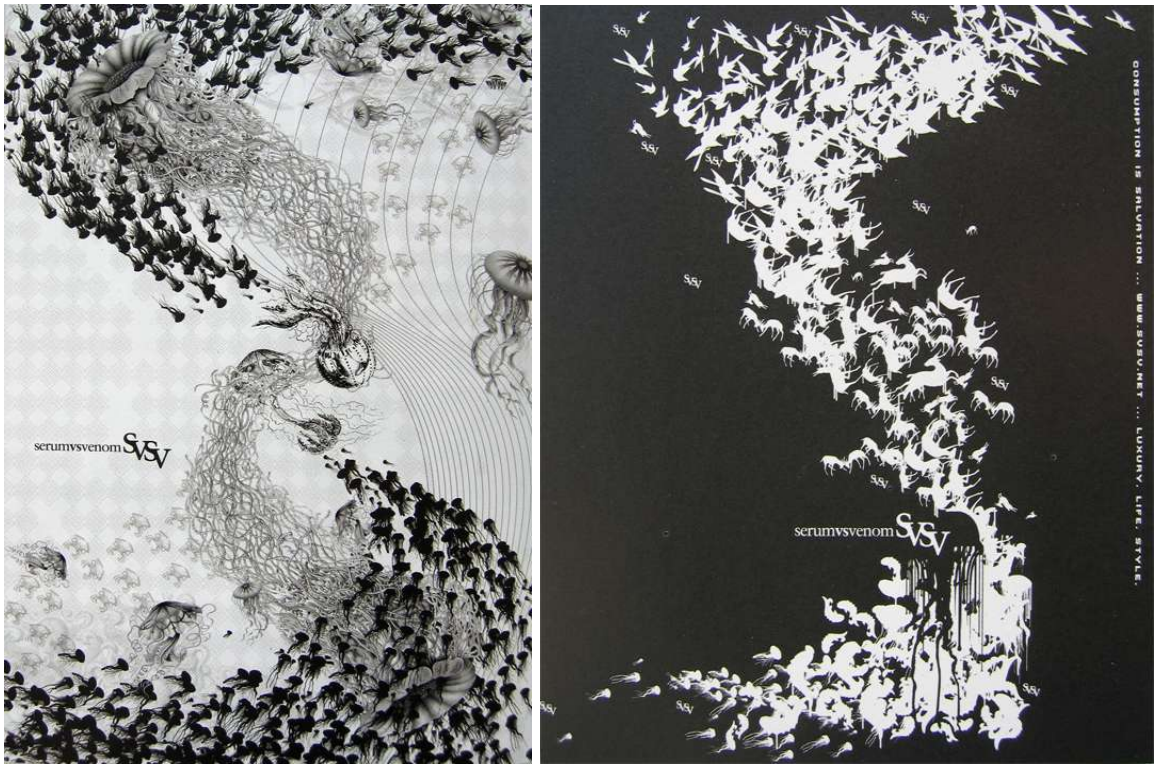
[Par1995]

Where rule1, rule2 & rule3 are the cohesion, alignment and separation rules. Any other required rules can be added in the same way.

Furthermore, Reynolds explains that boids should only be influences by boids in their

local neighbourhood: "*The neighborhood is characterized by a distance (measured from the center of the boid) and an angle, measured from the boid's direction of flight. Flockmates outside this local neighborhood are ignored."*[Rey1987].

2.2.2 2D Art



Press Ads for the serumvsvenom brand,
designed by the Keystone Design Union,
printed in Creative Review Magazine, Nov 2005.

The images above were printed in Creative Review Magazine. The graphic style is an inspiration, and may influence the aesthetics for the animation for this project. Trying to render in a realistic style rather than a graphic style may be a hindrance because it isn't natural to have fish moving in unison to music, so the viewer may be distracted by this. However, if a graphic render style is used, this won't be a problem, as the abstract notion of fish moving to music won't be confused by a realistic look. The decision of which style to use will, however be made once the Music Driven Flocking System has been fully implemented.

2.3 Music Analysis

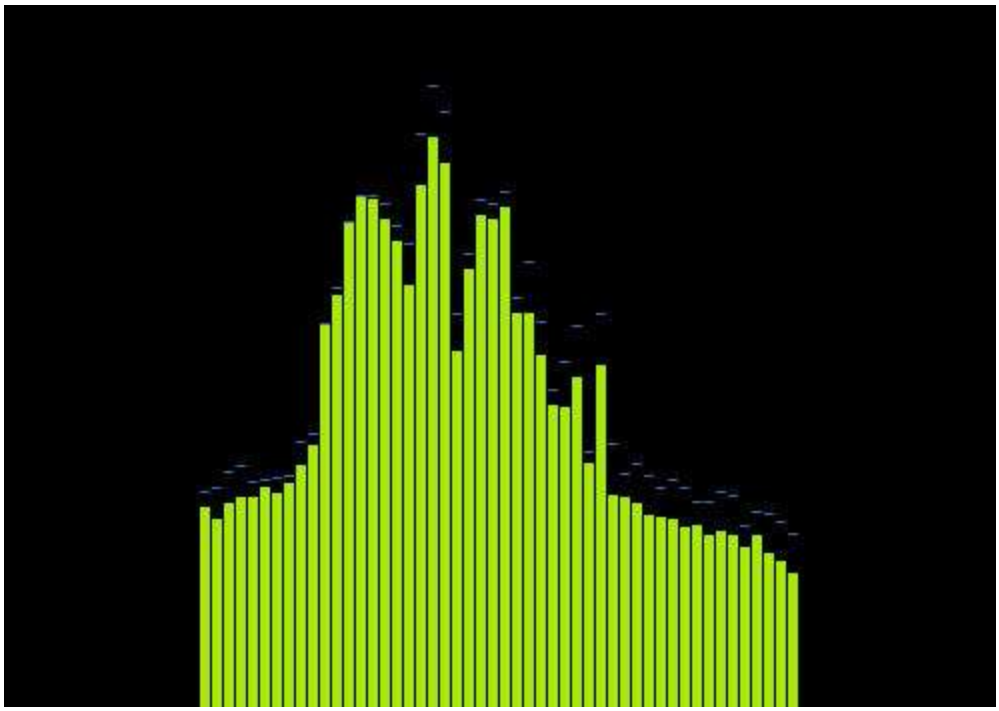
There are several ways that a sound file and more specifically a music file can be analysed, the most obvious being to take its amplitude. Music analysis is extensively used in media player visualisations and graphic equaliser dislays on HiFis: the usual type of analysis for these applications is Fourier Analysis, described below. Other ways of analysing music files includes performing beat detection and calculating the speed of the music (in beats per minute).

2.3.1 Fourier Analysis

> *"Fourier Analysis is the process of taking an aperiodic signal (one which does not repeat itself) such as a stream of music and approximating it using various sinusoidal waves of different frequencies."*
>
> [Mit_05]

These sinusoidal waves are also of varying amplitudes, and when added on top of each other the original sound is recreated. The greater the number of frequencies used, the more accurate the sound will be. The process of adding together a spectrum of sinudoidal waves of different frequencies in order to replicate an original sound is known as the **Fourier Transform** and is used, amongst other things, to digitise music. It is also useful for visualising the amount of bass, mid-tone and treble there is in a sound, as seen in graphic equalisers. In real-time applications such as the graphic visualisations in Windows Media Player an optimised version of the Fourier Transform is used – it is called the **Fast Fourier Transform** (**FFT**).
Interestingly, the Fourier Transform can be in computer graphics as well as an anti-aliasing technique.

Below is an image of a screen-capture of a Fast Fourier Transform spectrum as seen in Windows Media Player.



2.3.2 Beat Detection

Beat detection is based on the volume going over a specified threshold

2.3.3 Andrew Mitchell's Music-in-Maya plug-in. [MiM_05]

To analyse data from a music file for use in Maya, Andrew Mitchell's Music-In-Maya plugin will be used [MiM_05]. This plugin offers the ability to load a music file into Maya, then do the following tasks:

- Analyse the data and create a Fast Fourier Transform spectrum
- Perform beat detection.
- Calculate the speed of the music (beats per minute).
- Give the volume of the music.

It also makes it easy to connect the data from the above tasks to attributes of Maya nodes and objects. A detailed description is given on the making of and the use of the plug-in in Andrew Mitchell's Major Project report [Mit_05].

Fmod [Fmod_05]

The plug-in uses an audio library called Fmod in order to implement features such as creating a Fast Fourier Transform spectrum. The Fmod library is for object oriented languages such as C++, which makes it apt for use in developing a plug-in for Maya (because the Maya API is written in C++).

Caching

In order to enable rendering, the author of the plug-in made it possible to cache the data, because without this feature, it was only possible to access the data during playback, because it is calculated real-time: i.e. if you skipped to a certain frame, or scrubbed through the time line, the values would not be calculated, because the computation only occurs during playback. To cache the data, it is stored in main memory during playback, and then when the stop button is pressed, it is saved to a **Cache File**, which can be loaded into Maya - it is then possible to skip to any frame or even scrub through the timeline and the data will be as it should be.

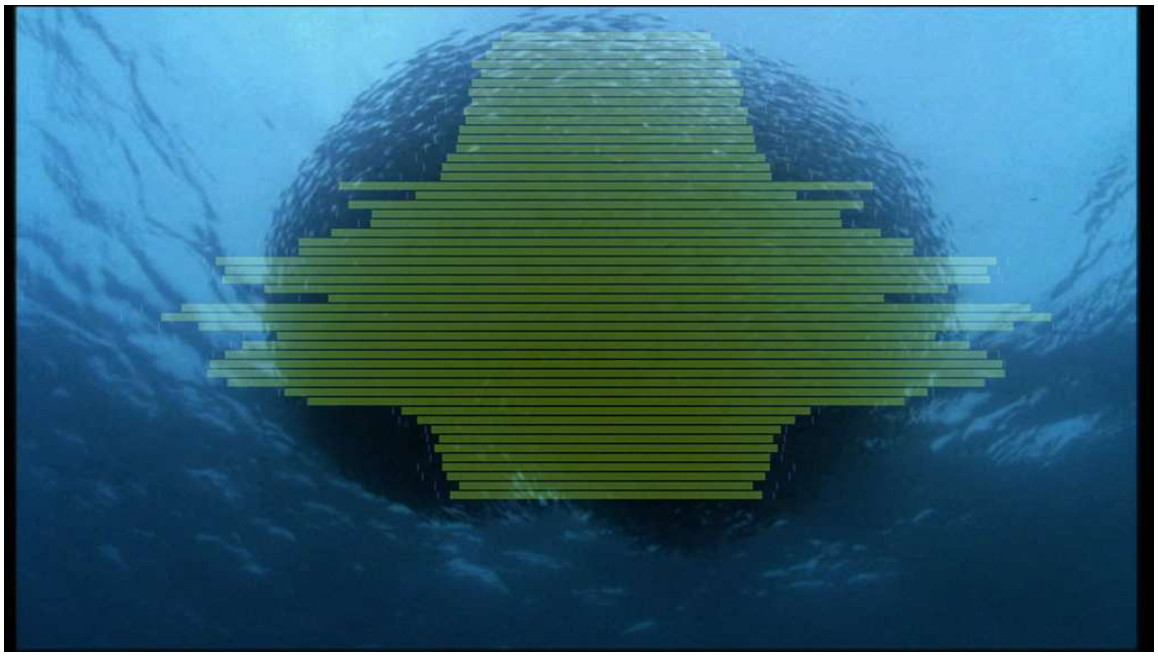
3. IMPLIMENTATION

3.1 Solving the Challenges

How will the music influence the movement of the flock? There were 2 main initial ideas:

3.1.1 Idea 1

The first idea would be to have a school of fish in a defensive ball (as in the picture below) and the shape of the ball would be influenced by the shape of the **FFT** spectrum mirrored vertically and then put on its side (as below), with bass frequencies at the bottom and treble frequencies at the top. One issue with this idea is that the **FFT** spectrum moves in quite an erratic way, but this problem could be solved by restricting the speed at which the amplitude for each frequency can move back inwards. By restricting the speed only for inwards movement will allow the sudden impact of a beat to be seen but will inhibit constant erratic movement.



3.1.2 Idea 2

The second idea is designed to deal with music with frequent beats. It would be interesting if on the sound of a beat, the school of fish separated suddenly, as if avoiding a predator. However, if the beats are frequent it could look very unnatural to have the school separating, then cohering and then separating again on the next beat, within a very short space of time. So instead of having one school reacting to every beat, there could be several schools reacting every so often.

3.2 Flocking System Documentation

With possible solutions to the foreseen challenges, the task of creating the flocking system could now be undertaken.

C++ or MEL?

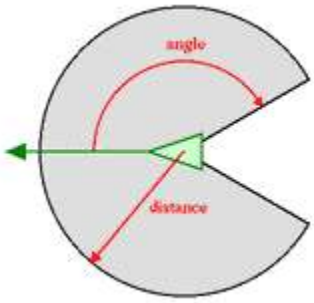
Though a more optimized flocking system could be made with C++, MEL was used because it took away the task of bringing the data into Maya and it made it easy to connect data from the music file to the flocking system. The main interest in this project is the challenge of driving the flocking system with music, so that's what the most time should be spent on.

Maya Particles

Using Maya's particle system is ideal for making a flocking system, because it provides direct and easy access to attributes on a per particle basis: you can query or set attribute values such as velocity and position, using a Dynamic Expression. A Dynamic Expression is a type of expression used specifically and exclusively for use with particles: it calculates the expression for each particle every frame.

The Rules:

There is a good explanation of the boids algorithm written in pseudocode by Conrad Parker [Par_02]. It helped me in writing my own boids algorithm, but it is only a very basic algorithm so I added a few things to make it more complex and therefore simulating a real flock more accurately.
For example, Parker's algorithm says that, the boids perceive all other boid apart from themselves, whereas, really they should only be able to perceive the boids within their local neighbourhood, which is defined by a certain angle and distance of perception.



[Rey_87]

In the dynamic expression, arrays were created to store the position and velocity of each boid, so that each boid could check which boids are in its local neighbourhood. They would then check the average position and average velocity of the boids in their local neighbourhood in order to calculate the cohesion and alignment rules (respectively).

Pseudocode:

Below is an explanation for each of the rules that were used for my flocking system, written in pseudocode. MEL code for each Rule can be found in the Appendix [**A.1**].
As stated before, because these rules are implemented within a Dynamic Expression, the code is performed for each boid, every frame.
For clarity the boid currently being computed will be called currentBoid, any OTHER boid will be called Boid. The name of the final vector for each rule will be the name of the rule in block capitals. (e.g. COHESION). Each rule will be multiplied by a factor which will be controlled by a Slider – these factors will be given the name of their rule plus "Slider", (e.g. cohesionSlider)

Cohesion Pseudocode

Aim: For the currentBoid, find the average position of the Boids that lie within its local neighbourhood (not including itself)(Remember Boid defines all boids other than currentBoid).

```
for each (Boid)
{
      if(Boid is in currentBoid's local neighborhood)
      {
            add Boid.position to Sum;
            increment the Counter by 1;
```

```
	}

}

AveragePosition = Sum / Counter;
```

The Cohesion vector that steers the currentBoid towards what it perceives to be the centre of the flock is the vector from its position to the position that it perceives to be the average.

```
COHESION = (the vector from currentBoid.position to AveragePosition) x
cohesionSlider;
```

Seperation Pseudocode

Aim: For the currentBoid, see if there are any boids that lie within its local neighbourhood (not including itself). If there are, then steer away from them

```
for each (Boid)
{
	if (the distance between Boid the currentBoid is within the
	currentBoid's perception)
	{
		SEPERATION += (the vector from currentBoid.position to
							Boid) x
							cohesionSlider;
	}
}
```

Alignment Pseudocode

Aim: For the currentBoid, find the average velocity (orientation) of the Boids that lie within its local neighbourhood (not including itself).

```
for each (Boid)
{
	if(Boid is in currentBoid's local neighborhood)
	{
		add Boid.velocity to Sum;
		increment the Counter by 1;
	}

}

AverageVelocity = Sum / Counter;
```

```
COHESION = (the vector from currentBoid.velocity to AverageVelocity)
                                                    *alignmentSlider;
```

Boundary Pseudocode

The Boundary Rule tries to keep the boids within a spherical boundary: they can get out of it but even if they do, they will soon steer back inside the boundary.

```
if (currentBoid can perceive the boundary)
{
BOUNDARY = (the vector from the boundary to currentBoid.position)
                                                *boundarySlider;
}
```

Avoidance (of Predators) Pseudocode

The Avoidance Rule tries to avoid any attacking predator.

```
If (the predator is in the currentBoid's perception
{
            AVOIDANCE -= the vector from the current boid to the
      predator  *avoidanceSlider
}
```

Flocking Rule Pseudocode

THIS IS WHERE THE FLOCKING HAPPENS. Note that with Maya's Particles, if the velocity is updated, the position is automatically updated, so it is not necessary to update the position.

```
currentBoid.velocity += ALIGNMENT + BOUNDARY + COHESION + SEPERATION +
AVOIDANCE;
```

Speed Limit Pseudocode

The speed limit rule is to stop the boids from going unnaturally fast. This rule is not added to the current velocity like the other rules, as it is designed to restrict the velocity.

```
if (the currentBoid's speed is greater than the speed limit)
{
     currentBoid.velocity =(currentBoid.velocity as a unit vector) *
                                                          speedLimit;
}
```

Optimization

Currently several of the Rules use a for-loop to go through each Boid. Reducing this to one will make the code less readable but is justified by making a huge difference on computation time. Optimization is important, because a larger number of boids will look more impressive.

3.3 Music Analysis Implementation

With the flocking system implemented and nicely simulating a school of fish, it was time to get the flocking system to be driven by music.

3.3.1 Using the Music In Maya plug-in

A GUI is provided with the plug-in to make it easy to connect the data to attributes of objects. The features that will be used are the Beat Detection function and the Fast Fourier Transform Spectrum function.

3.3.2 Manipulating the Fast Fourier Transform Spectrum

The results of the FFT spectrum were very erratic so experiments to stabilise the data were undertaken.
Following the use of particles for the flocking system, particles were used to experiment with manipulating the FFT spectrum because Dynamic Expressions make dealing with arrays of data easy, especially when you want to set and query attributes such as velocity and position.
The plug-in cannot connect the music data to particles' attributes, so the FFT spectrum must be connected to attributes (such as scale) of an array of objects (such as spheres)

first. Given that the number of objects is the same as the number of particles, the particles can then query the scale of their corresponding object.
Unfortunately, these experiments to stabilise the FFT spectrum were not successful, so concentration moved on to beat detection.

3.3.3 Using Beat Detection to drive predators

Not much time was spent on resolving the problems in stabilising the FFT spectrum, because of the fact that the flocking system already had predator avoidance implemented, meaning that Idea 2, (outlined in section **3.1.2**), could be put into fruition using the beat detection function in conjunction with predator avoidance.
To recap and clarify Idea 2:
There will be a number of shoals of fish and there will be a number of predators (such as sharks or dolphins). Each Predator will have a target shoal randomly allocated. They will head slowly towards their target. On the sound of a beat, one of the predators will go fast and attack their target shoal. The shoal will move quickly to avoid the predator. Once the predator reaches its target it will randomly choose another target and on the next beat it will be one of the other predators' turn to attack. The theory is that instead of having one school reacting to every beat, there could be several schools reacting every so often so that the influence of the music is easier to see.

3.4 Integrating the music with the flocking system

The final step was integrating the music with the flocking system. Another dynamic expression was created [**A.2**] to make a group of predators attack a set of locations in a random order, each taking a turn to attack on the beat. To integrate these predators with shoals of fish, the Dynamic Expressions for the shoals of fish had to be changed. The expressions were changed so that they would recognise predators. This stage took longer than expected to complete, because there was a problem with the expression that took a while to Debug.
Once these problems were sorted, it was possible to create my finished product. In the finished product there is 5 shoals of fish and they are randomly attacked on the beat of the music by 3 predators – the music used is by Cinematic Orchestra and is called "Every day", from the Album with the same title.

4. CONCLUSION

The main aim of this project was to make a flocking system and use music to directly influence its movement. Along with this, was the objective of creating an animation to show the capability of the tool. Both of these objectives were achieved. However, the intention was that the animation produced would be aesthetically stimulating - with modelled fish that would flap their fins quickly when moving fast and less so when moving slowly. An interesting, maybe even innovative, graphic render style was aspired, but unfortunately time did not allow the project to get to such a stage. The influence of the music on the fish is interesting, but the general movement of the predators looks rather primitive and not like how a shark or dolphin would swim.
The full potential of this project is intriguing, so work on it will without doubt proceed. It also took a while to get the final animation to look as desired, because with quite a lot of particles, it could not run real time. If I had time to optimise my code, this may have helped this stage of the project.
Another improvement, other than those listed above would be to change the algorithm for the predators so that when they are not attacking, they are wandering (rather than just going towards their target slowly).

4.1 Future Work

Working with Maya's particles has been an absorbing learning curve: there is so much that you can do with a particle system: simulating crowds, traffic, flocks, weather conditions such as snow or a tornado would name just a few of the things that you can do. This project has opened the door for me to explore many of these possibilities and has undoubtedly also improved my problem solving skills:
Unlike most programming/scripting languages, MEL doesn't have a debugging tool, so debugging is often very time-consuming. This led me to learn an important lesson about problem solving. If having trouble solving a particular problem, it is often a good idea to try to find a different route. Such a lesson is reminiscent of French lessons at school, where the teacher would often say that if you can't think of how to translate a sentence, try saying that sentence in a different way using words that you do know.

In addition to further work with Maya Particles, it would be good to learn how to create a flocking system in a language such as Java so that I could have some kind of flocking on my website.

5. GLOSSARY

FFT (Fast Fourier Transform): See 2.3.1 for description.
Cache File: A file that stores data so that it can be used real time. Cache Files are used when computing the data is too slow to view in real time.

6. APPENDIX

A.1 My Mel Flocking Algorithm:
(This algorithm goes in a Dynamic Expression)

```
// TURNS OFF CYCLE CHECK WARNING //

cycleCheck -e off;


///////////////
// VARIABLES //
///////////////

vector $particleCentroid=<<particleShape1.centroidX,
particleShape1.centroidY, particleShape1.centroidZ>>;

vector $particlePosition=particleShape1.position;

vector $particleVelocity=particleShape1.velocity;




////////////////////////////////////////////////////////////////
// Store All particle positions and velocities in vector arrays //
////////////////////////////////////////////////////////////////

int $id=particleShape1.particleId;

float $boidPosArr[] =`getParticleAttr -at lastPosition -array true
particleShape1.pt[0:(particleShape1.count)]`;

float $boidVelArr[] =`getParticleAttr -at velocity -array true
particleShape1.pt[0:(particleShape1.count)]`;

// Convert float3 arrays to vector arrays //
```

```
vector $boidPosVecArr[];

vector $boidVelVecArr[];



for($i=0;$i<`size($boidPosArr)`;$i+=3)
{
      $boidPosVecArr[($i/3)]=<< $boidPosArr[$i], $boidPosArr[($i+1)],
$boidPosArr[($i+2)] >>;
      $boidVelVecArr[($i/3)]=<< $boidVelArr[$i], $boidVelArr[($i+1)],
$boidVelArr[($i+2)] >>;
}




//////////////////
// COHESION RULE //
//////////////////

// vector from boid to its Perceived Center (See PseudoCode)//

vector $PerceivedCentre=<<0.0, 0.0, 0.0>>;

int $particleCounter=0;

float $angle=0.0;

// Find the centre of mass as the boid perceives it (i.e. not including
itself and within the perceptionAngle) //

for($i=0;$i<`size($boidPosVecArr)`;$i++)
{
      $angle=angle($boidVelVecArr[$id],($boidPosVecArr[$i]-
$boidPosVecArr[$id]));//)`;
      if ($i!=$id && $angle <`deg_to_rad
(particleShape1.perceptionRadius/2)`)
      {
            $PerceivedCentre+=$boidPosVecArr[$i];
            $particleCounter++;
      }
}

// Average the positions of all of the boids that are within the
current boids perception distance//
// If the particle counter is on zero, make it = 1, so that the are no
divide by zero errors //
if($particleCounter==0)$particleCounter=particleShape1.count;
$PerceivedCentre/=$particleCounter;
```

```
vector $COHESION=($PerceivedCentre-$particlePosition)
*particleShape1.cohesionSlider;

//OLDVERSION// vector $COHESION = ($particleCentroid -
$particlePosition)*particleShape1.cohesionSlider;



////////////////////////////
//SEPERATION RULE //////////
////////////////////////////

vector $SEPERATION=0;

for($i=0;$i<`size($boidPosVecArr)`;$i++)
{
      if ($i!=$id)
      {
            if (`mag($boidPosVecArr[$i]-particleShape1.position)
`<particleShape1.boidPerception)
            {
                  $SEPERATION -= $boidPosVecArr[$i]-
particleShape1.position;
            }
      }

}

$SEPERATION*=particleShape1.seperationSlider;



////////////////////////////
////// BOUNDARY RULE ///////
////////////////////////////

vector $boundaryVec = << particleShape1.boundaryPerception ,
particleShape1.boundaryPerception , particleShape1.boundaryPerception
>>;

// initialise rule to zero
vector $BOUNDARY=<<0,0,0>>;

// distance from indivisual boid to boundary
float $distToBoundary=(particleShape1.boundaryRadius-`mag
(particleShape1.position)`);

// If the boundary is within the boids' range of perception
if ($distToBoundary<particleShape1.boidPerception)
{
```

```
      $BOUNDARY += (( (`unit(particleShape1.position)`) *
particleShape1.boidPerception)   -    particleShape1.position  );
}

$BOUNDARY *= particleShape1.seperationSlider;




////////////////////////////
////// ALIGNMENT RULE //////
////////////////////////////

vector $ALIGNMENT=<<0,0,0>>;

vector $PerceivedVelocity=<<0.0, 0.0, 0.0>>;

for($i=0;$i<`size($boidVelVecArr)`;$i++)
{
      if ($i!=$id)
      {
            $PerceivedVelocity+=$boidVelVecArr[$i];
      }
}

// Average the positions of all of the boids that are within the
current boids perception distance//

$PerceivedVelocity/=((particleShape1.count)-1);

vector $ALIGNMENT=($PerceivedVelocity-$particleVelocity) *
particleShape1.alignmentSlider;

normalize($ALIGNMENT);


//////////////////////////////////////////
//////// (PREDATOR) AVOIDANCE RULE ////////
//////////////////////////////////////////

vector $AVOIDANCE=0;

vector
$predatorPos=<<pSphere2.translateX,pSphere2.translateY,pSphere2.transla
teZ>>;
if (`mag($predatorPos - particleShape1.position)
`<particleShape1.predatorPerception)
{
      $AVOIDANCE -= $predatorPos - particleShape1.position;
}
```

```
$AVOIDANCE*=particleShape1.avoidanceSlider;



////////////////////////////
// FLOCKING RULES GO HERE //
////////////////////////////

particleShape1.velocity += $ALIGNMENT + $BOUNDARY + $COHESION +
$SEPERATION + $AVOIDANCE;


////////////////////////////
/////// LIMIT RULE /////////
////////////////////////////


if(`mag(particleShape1.velocity)` > particleShape1.speedLimit)
{
      particleShape1.velocity = (particleShape1.velocity/`mag
(particleShape1.velocity)`) * particleShape1.speedLimit;
}



////////////////////////////////////////////////
/////// OBJECT AND LOCATOR CONSTRAINTS /////////
////////////////////////////////////////////////

locator1.translateX=$particleCentroid.x;

locator1.translateY=$particleCentroid.y;

locator1.translateZ=$particleCentroid.z;



polySphere1.radius=particleShape1.boundaryRadius;
```

A.2 ALGORITHM TO MAKE PREDATORS ATTACK ON THE BEAT

```
////////////////////////////
//// RUNTIME EXPRESSION ///
////////////////////////////

//////////////////
//// VARIABLES ///
//////////////////

string $locatorName=("locator"+(particleShape1.goal_ID-1));
float $locatorPosition[3];
$locatorPosition[0]=`getAttr ($locatorName+".tx")`;
$locatorPosition[1]=`getAttr ($locatorName+".ty")`;
$locatorPosition[2]=`getAttr ($locatorName+".tz")`;
vector $locatorPosVec=<<$locatorPosition[0],$locatorPosition[1],
$locatorPosition[2] >>;
$numPredators=particleShape1.count;
int $key;
// IF PREDATOR GETS CLOSE TO TARGET, CHANGE TARGET AND PASS ATTACK
BATTON TO OTHER PREDATOR //

if($locatorPosVec - particleShape1.position<7)
{     // Change Attacker
      particleShape1.attacker_ID++;
      if(particleShape1.attacker_ID==particleShape1.count)
      {     particleShape1.attacker_ID=0;
      }
      // Change Goal
      int $randGoal=rand(2,6);
      particleShape1.goal_ID=$randGoal;
      // Reduce speed
      $speedLimit=particleShape1.normalSpeedLimit;
      $key=0;

}
// GOAL = predator to goal
vector $GOAL = ($locatorPosVec - particleShape1.position)*4.0;


////////////////////////////
// FLOCKING RULES GO HERE //
////////////////////////////

particleShape1.velocity += $GOAL;

////////////////////////////
/////// SPEED LIMIT ////////
////////////////////////////
```

```
float $speedLimit;

// Get value from Beat Detect Node //
float $beatNodeVal=`getAttr pSphere0.sx`;

// Determine if there is a beat //

// If the volume is over the threshold.&& you are the attacker //
if($beatNodeVal==1 &&
particleShape1.particleId==particleShape1.attacker_ID)
{
      // then add one to the key //
      $key++;
}

// If you're the attacker and there is a beat, then attack //
if(particleShape1.particleId==particleShape1.attacker_ID && $key>=1)
{
      $speedLimit=particleShape1.attackSpeedLimit;
}
// if you are not the attacker Go Slow //
else if (particleShape1.particleId!=particleShape1.attacker_ID)
{
      $speedLimit=particleShape1.normalSpeedLimit;
}

if(`mag(particleShape1.velocity)` > $speedLimit)
{
      particleShape1.velocity = (`unit(particleShape1.velocity)`) *
$speedLimit;
}

// If you are the attacker, where are you? //
if (particleShape1.particleId==particleShape1.attacker_ID)
{
      string $particleName=("particle"+(particleShape1.goal_ID));
      float $locatorPosition[3];
      float $particleTranslatePosX=`getAttr ($particleName+".tx")`;
      float $particleTranslatePosY=`getAttr ($particleName+".ty")`;
      float $particleTranslatePosZ=`getAttr ($particleName+".tz")`;

$goalTranslateVec=<<$particleTranslatePosX,$particleTranslatePosY,$part
icleTranslatePosZ >>;
      float $attackerPosArr[3] =`getParticleAttr -at position -array
true particleShape1.pt[particleShape1.particleId]`;
      $attackerPos= <<$attackerPosArr[0],$attackerPosArr[1],
$attackerPosArr[2]>> - $goalTranslateVec;
}
```

7. REFERENCES

[Rey_87] REYNOLDS, Craig. 1987. *Boids (Flocks, Herds, and Schools: a Distributed Behavioral Model)* [online]. Available from: http://www.red3d.com/cwr/boids/

[Par_02] PARKER, Conrad. 2002. *Boids Pseudocode.* [online]. Available from: http://www.vergenet.net/~conrad/boids/pseudocode.html

[Blu_01] BBC *The Blue Planet*, 2001 [DVD]

[LoE_05] BBC *Light on the Earth,* 2005 [book]
Caption writen by COX, R.K.
Photograph taken by DE FRANCISCO, J.L.G. 2001

[MiM_05] MITCHELL, Andrew. 2005. *Music in Maya* [maya plug-in]
Available from:
http://www.a-mitchell.co.uk/portfolioMain.php

[Mit_05] MITCHELL, Andrew. 2005. *Music Animation Plugin for Maya.* [Major Project Report]. BA CVA. Bournemouth University.

[Neg_95] NEGROPONTE, Nicholas. *Wired UK 1.07,* Nov. 1995

[Fmod_05] FMOD. 2005. [C++ library]. Available from:
http://www.fmod.org/

[KDU_05] KEYSTONE DESIGN UNION, 2005 [magazine] *Creative Review Magazine*.