POSE WARPING FOR REALTIME ANIMATION

By

Darragh Maloney

INSTITIÚID TEICNEOLAÍOCHTA
AN LEABHARLANN
LEITIR CEANAINN

Institiúid Teicneolaíochta Leitir Ceanainn
Letterkenny Institute of Technology
lyit

SUBMITTED IN PARTIAL FULFILLMENT OF THE
REQUIREMENTS FOR THE DEGREE OF
MASTER OF COMPUTER SCIENCE
AT
LETTERKENNY INSTITUTE OF TECHNOLOGY
DONEGAL, IRELAND
JUNE 2007

© Copyright by Darragh Maloney, 2007

LETTERKENNY INSTITUTE OF TECHNOLOGY
DEPARTMENT OF
COMPUTER SCIENCE

The undersigned hereby certify that they have read and recommend to the Faculty of Science for acceptance a thesis entitled **"Pose Warping for Realtime Animation"** by **Darragh Maloney** in partial fulfillment of the requirements for the degree of **Master of Computer Science**.

lyit | Institiúid Teicneolaíochta Leitir Ceanainn | Letterkenny Institute of Technology

Dated: June 2007

Supervisor: ______________________
Dr. Mark Leeney

Readers: ______________________

____

INSTITIÚID TEICNEOLAÍOCHTA
AN LEABHARLANN
LEITIR CEANAINN

INSTITIÚID TEICNEOLAÍOCHTA
AN LEABHARLANN
LEITIR CEANAINN

LETTERKENNY INSTITUTE OF TECHNOLOGY

Date: **June 2007**

Author: **Darragh Maloney**

Title: **Pose Warping for Realtime Animation**

Department: **Computer Science**

Degree: **M.Sc.** Convocation: **November** Year: **2007**

Permission is herewith granted to Letterkenny Institute of Technology to circulate and to have copied for non-commercial purposes, at its discretion, the above title upon the request of individuals or institutions.

Signature of Author

THE AUTHOR RESERVES OTHER PUBLICATION RIGHTS, AND NEITHER THE THESIS NOR EXTENSIVE EXTRACTS FROM IT MAY BE PRINTED OR OTHERWISE REPRODUCED WITHOUT THE AUTHOR'S WRITTEN PERMISSION.

THE AUTHOR ATTESTS THAT PERMISSION HAS BEEN OBTAINED FOR THE USE OF ANY COPYRIGHTED MATERIAL APPEARING IN THIS THESIS (OTHER THAN BRIEF EXCERPTS REQUIRING ONLY PROPER ACKNOWLEDGEMENT IN SCHOLARLY WRITING) AND THAT ALL SUCH USE IS CLEARLY ACKNOWLEDGED.

lyit Institiúid Teicneolaíochta Leitir Ceanainn Letterkenny Institute of Technology

Abstract

3D computer games with animated characters are restricted to the animations provided by an animator. This thesis explores a method for having a character perform different animations using only simple base animations and discrete poses incorporating a digital signal processing approach. Treating the animations as a series of digital signals allows digital signal processing techniques to be applied to create new motions. This facilitates a decreased animator workload while allowing a character to interact better with its environment. These techniques involve treating an animation as a continuous signal, sampling it and shifting it about another signal to combine an animation and a pose. To eliminate dead poses the use of filters on animations with a view to creating new animations with the aid of timewarping is also explored

lyit | Institiúid Teicneolaíochta Leitir Ceanainn
Letterkenny Institute of Technology

Acknowledgements

I would like to acknowledge the help many people who helped me along the way to completing my masters. Firstly, Mark Leeney, my supervisor, for his guidance, patience and help, especially with my thesis and the many mathematical issues encountered.

Secondly, John OKane, my mentor during my time at Instinct Technology, to whom I am indebted for all he taught me about games programming and animation. I must also thank Ronan Pearce, Chris Gregan and Patrick McColgan, as well as the rest of the staff at Instinct Technology, not just for their technical assistance, but also for providing a fun atmosphere to work.

To Instinct Technologies, Letterkenny Institute of Technology and Enterprise Ireland, thanks are due for their provision of funding and facilities, without which this work would not have been possibly.

Many thanks must go to my parents for their encouragement and support of my continued education.

Lastly, I must acknowledge Maria O Callaghan, who, until she finished her thesis very shortly before me, provided great motivation to finish.

lyit Institiúid Teicneolaíochta Leitir Ceanainn Letterkenny Institute of Technology

Abbreviations

FPS Frames Per Second

IK Inverse Kinematics

DSP Digital Signal Processing

LERP Linear Interpolation

SLERP Spherical Linear Interpolation

ADC Analogue to Digital Converter

DFT Discreet Fourier Transform

DTFT Discreet Time Fourier Transform

PC Personal Computer

STL Standard Template Library

CCD Cyclic Coordinate Descent

Institiúid Teicneolaíochta Leitir Ceanainn
Letterkenny Institute of Technology
lyit

Table of Contents

Abstract **iv**

Acknowledgements **v**

Abbreviations **vi**

Table of Contents **vii**

List of Figures **ix**

1 Introduction **1**
- 1.1 Organisation 2

2 Animation **3**
- 2.1 Introduction 3
- 2.2 Technical Animation 3
- 2.3 Motion Warping 5
- 2.4 Modeling for Animation 6
  - 2.4.1 Polygons 6
  - 2.4.2 Patches 7
- 2.5 The Character 9
- 2.6 Applying the Mesh 12
- 2.7 Conclusion 14

3 Animation Technologies **16**
- 3.1 Introduction 16
- 3.2 A Physical Approach 16
- 3.3 An Inverse Kinematics Approach 18
- 3.4 Motion Blending 19
- 3.5 Conclusion 20

Institiúid Teicneolaíochta Leitir Ceanainn
Letterkenny Institute of Technology

4 Essential Mathematics 22
4.1 Introduction . . . 22
4.2 Coordinate Spaces . . . 22
4.3 Rotation of a Point . . . 23
4.3.1 2D rotation about the origin . . . 23
4.3.2 3D rotation about a cardinal axis . . . 24
4.3.3 3D rotation about an arbitrary axis through the origin . . . 25
4.4 Translation of a Point . . . 25
4.5 Translating Bones . . . 26
4.6 Eulerian Angles . . . 28
4.7 Quaternions . . . 30
4.7.1 Quaternion Algebra . . . 30
4.7.2 Quaternion Multiplication . . . 31
4.7.3 Quaternion Inverse . . . 32
4.7.4 Rotating Points with Quaternions . . . 32
4.8 Conclusion . . . 33

5 Digital Signal Processing 34
5.1 Introduction . . . 34
5.2 Sampling a Signal . . . 36
5.2.1 Aliasing . . . 38
5.2.2 The Frequency Domain . . . 39
5.2.3 Animation as Digital Signals . . . 40
5.3 LERP and SLERP . . . 41
5.3.1 Other Sampling Rate Constraints . . . 42
5.4 Conclusion . . . 43

6 Implementation 44
6.1 Introduction . . . 44
6.2 Sampling an Animation . . . 45
6.2.1 Why is Sampling Necessary? . . . 45
6.2.2 Finding a Sampling Rate that Works . . . 46
6.2.3 Code Reference . . . 47
6.3 Converting a Quaternion Animation to an Eulerian Animation . . . 48
6.3.1 Using this Conversion with an Animation . . . 48
6.3.2 Code Reference . . . 49
6.4 Filtering an animation . . . 50
6.4.1 Implementing a Low Pass Filter . . . 52
6.4.2 Filtering in Real Time . . . 52
6.4.3 Code Reference . . . 54

Institiúid Teicneolaíochta Leitir Ceanainn
Letterkenny Institute of Technology

6.5 Time Warping an Animation . . . 55
6.5.1 Timewarping Algorithms . . . 56
6.5.2 The Implemented Time Warping Algorithm . . . 56
6.5.3 Bending Work . . . 58
6.6 The Grid Revisited . . . 64
6.6.1 Code Reference . . . 67
6.7 Conclusion . . . 70

7 Results and Conclusions **71**
7.1 Introduction . . . 71
7.2 Timewarping Results . . . 71
7.2.1 Timewarping a pose . . . 74
7.3 Motion Warping . . . 76
7.4 Pose Specific Motion Warping . . . 77
7.5 Conclusion . . . 78
7.6 Future Work . . . 79

Bibliography **80**

A Code Diagrams **83**

B C++ Code **88**

lyit | Institiúid Teicneolaíochta Leitir Ceanainn
Letterkenny Institute of Technology

List of Figures

2.1 A polygon wireframe of a cube. The yellow dots are the vertices and the blue lines are edges. 7
2.2 A rendered version of the same cube. The faces are more easily seen. . . . 8
2.3 A sphere showing how a curved surface can be approximated. 8
2.4 A sphere made from patches. The purple points are the control points for controlling the shape. Notice it takes less points to make a sphere from patches than from polygons. Patches are better suited to curved surfaces. . . 9
2.5 A cardinal spline passes through its control points (yellow points). The angle of entry and exit to these points can be altered by adjusting the tangent controls. 9
2.6 Wireframe representation of a character made in Maya. 10
2.7 A smoothed version of the same character. The yellow dots are vertices. . . 11
2.8 A representation of a character's skeleton. 12
2.9 The straight edge on the inside of the elbow is the problem. 13
2.10 Associating the middle vertices with both bones with an even weight. . . . 13
2.11 The result of a weighted bend. 14
2.12 A real world example of a skeleton being used to deform a mesh are the characters in Torc Interactive's Dreadnought. 15

3.1 The animated Luxo lamp. 17

4.1 2D rotation of a point about the origin 24
4.2 Euler Axis 29

Institiúid Teicneolaíochta Leitir Ceanainn
Letterkenny Institute of Technology
lyit

5.1 A sine wave with a period of 0.25s and an amplitude of 1 in the time domain. 35
5.2 A sine wave with a period of 0.125s and an amplitude of 0.33 in the time domain. 35
5.3 The sum of the previous two graphs in the time domain. 35
5.4 The frequency domain equivalent of Figure 5.3. 36
5.5 An aperiodic discrete signal taken from two animations blended together. . 37
5.6 A periodic discrete signal taken from a walk animation. 37
5.7 A sine wave (blue) and a sampled version (pink) of the same sine wave. . . 38
5.8 A sine wave sampled at twice the Nyquist frequency and at $1/10^{th}$ the Nyquist frequency. 39
5.9 LERP - the points on the curve are not evenly spaced. 41
5.10 SLERP - the points on the curve are evenly spaced. 42

6.1 A signal produced using a low sampling rate. 46
6.2 A signal produced using a high sampling rate. 47
6.3 A quaternion to Eulerian conversion shown for the left thigh bone. There is a conversion discrepancy with the Z component of the Eulerian representation, shown in yellow. The right thigh bone Z rotations are similar. 49
6.4 How the signal showing the Z rotations of the left thigh should look. 50
6.5 The filtering architecture. 51
6.6 Four low passes of the Y component of a thigh bone. The signal gets smoother with each iteration. 53
6.7 A graphical representation of the grid created by dynamic programming to implement timewarping. 57
6.8 2 signals illustrating what angles and lengths are compared to calculate a work value for a node on the grid. 58
6.9 Translate the 2 segments so they sit on the origin. 60
6.10 Rotate the second segment so it lies on the x-axis. Rotate the second segment by the same amount to preserve the angle. 60

6.11 Calculate the angle Θ using trigonometry. If necessary, adjust it to account for the quadrant in which point 'a' lies. 61
6.12 As the sampling rate decreases the angle at y2 approaches a limit of 180 degrees. 61
6.13 When the y values are scaled up they have a greater bearing on the angle between the 2 line segments. 63
6.14 A screengrab from excel where the values of a grid were printed. The least cost path through the grid is shown in green. 65
6.15 The diagram shows a 50/50 merge of 2 points on a diagonal move, how a B-Spline is used when moving down, and an average when moving across. It should be noted that only the y values of the signals are involved in the numeric operations shown. 66

7.1 Blending a walk and a run that are slightly out of synchronization with each other. 72
7.2 Blending a walk and a run that are out of synchronization with each other produces an un-useable result. 72
7.3 Timewarping a walk to synchronize with a run. 73
7.4 Blending a timewarped walk with a run to produce a jog. 73
7.5 Timewarping a run to synchronize with a walk. 74
7.6 Blending a timewarped run with a walk to produce a jog. 74
7.7 Timewarping a pose to fit a walk animation. 75
7.8 Blending a timewarped pose with a walk. 75
7.9 Blending a timewarped pose with a walk. 76
7.10 The walk signal is centered on the pose signal with no timewarping. 77

lyit | Institiúid Teicneolaíochta Leitir Ceanainn
Letterkenny Institute of Technology

Chapter 1

Introduction

Modern computer game environments have progressed to simulate real world environments to a very high level of detail, not just in terms of look (for example Far Cry or Half Life 2) but also in terms of physical interaction. With packages such as Havok Physics or PhysX by Ageia, physical interactions between objects mimic real life to the highest detail.

The same can be said of animation. This is most obvious in films like Disney's Toy Story or Monster House from Sony Pictures. These films use the same software for character animation as computer games - namely Maya and 3D Studio Max. The main difference is that with films an animation can be tailor made for a situation and this will not have to change, but with games the animations respond dynamically to changes in the virtual world.

Allowing many different motions for a character helps to supplement the aspect of realism in games. However, to do this successfully requires a lot of time and effort from a skilled animator. A compromise is to make a key set of animations and construct the game so these animations always fit, for example, having every ledge at the same height so only one jump animation is needed, instead of many different jump animations for ledges of different heights.

lyit | Institiúid Teicneolaíochta Leitir Ceanainn
Letterkenny Institute of Technology

The goal of the present work is to develop a system enabling new animations to be created at run time by providing a single pose to warp with an animation. For example, instead of having all obstacles that the character can walk under at the same height - facilitating one stooped animation, obstacles can be at different heights, with each obstacle having its own associated stooped pose. This pose will be warped with the walk animation at run time, removing the need to make several different stooped animations. Given the complex structure of a human based character, such a system offers the opportunity to eliminate a large quantity of pre-animated work and also offer a more natural finished product.

1.1 Organisation

Chapter 2 discusses some animation fundamentals as well as the structure of a character and how characters are constructed. In Chapter 3, different approaches to animation are discussed along with different techniques based on the same skeletal approach used in this thesis. Chapter 4 provides a discussion of the mathematics involved in skeletal animation. The relationship between the signal processing involved in this work and classical digital signal processing is outlined in chapter 5. Chapter 6 provides a detailed description of the procedures implemented to achieve animation warping. Finally, Chapter 7 discusses the results of the implementation and details the conclusions drawn.

lyit Institiúid Teicneolaíochta Leitir Ceanainn Letterkenny Institute of Technology

Chapter 2

Animation

2.1 Introduction

Animation is the optical illusion of motion created by the consecutive display of static images. This holds true from the early Disney cartoons to today's modern computer generated films and computer games. An everyday example where this is very apparent is .gif files, commonly used on websites, or as avatars on web based forums. These loop a series of still images to give an impression of motion.

Generally, when talking about animation, it's assumed the subject is a person, animal, robot or some other creature. However, animation also deals with anything that has motion: vehicles, doors, non character objects, fluids etc. Often these are background details in a scene, adding to the detail. This chapter will deal with character animation involving bipeds (people).

lyit | Institiúid Teicneolaíochta Leitir Ceanainn
Letterkenny Institute of Technology

2.2 Technical Animation

Firstly, some technical background. The still images in an animation are called frames. On film, 24 frames per second (fps) is enough to give the illusion of motion, due to the persistence of vision. Before video, most animation was drawn to work at 24fps, so this

was 24 frames hand drawn for each second of film. Examples are the early Disney films Snow White and Pinocchio. By drawing each frame with a slight progression from the previous one, motion is conveyed. There are two methods in doing this: Pose-to-Pose animation and Straight-Ahead animation.

Straight-Ahead animation is the type mentioned above, you start with one frame, and draw each frame that comes after it in sequence. It is similar in concept to the animation seen in films like Wallace and Gromit, where the character's pose in incremented for the next frame. Pose-to-Pose animation is where the action to be animated is broken down into the main poses for that action, then the in-between frames are drawn. As an example, imagine a character jumping. First they compress, then spring up, lift off, come down and compress again. The main poses here could be compressing, lifting off, the highest point of the jump, landing and compressing after landing. Sketching these gives 6 frames, but if the jump lasts a second, we need another 18 frames. These frames are the in-between frames. MPEG video encoding and compression uses keyframing in this manner.

Pose-to-Pose animation is used in traditional hand drawn animation. It is also known as keyframing with the poses being the keyframes. Because the frames were hand drawn, the lead artist would draw the key frames, having others draw the in-between frames.

In modern computer animation, the same process is still in use. After a character has been modeled, keyframes are set, but in this case the computer draws the in-between frames. While animation for film is made at 24 fps, computer games can run as high as 90 fps, so that means a lot of in between frames are needed. Storing this many frames would take a lot of memory. Also, with PC games, where the hardware varies from machine to machine, the frame rate will vary. As a result, it cannot be assumed that an animation will run at a certain frame rate, say the 90fps mentioned above. Storing an animation as a series of keyframes and in-between frames with the assumption that it will be played at a certain rate of frames per second will cause the animation to be played faster or slower depending on the hardware used. The solution is to only store the keyframes, with each keyframe having an associated time. In-between frames are achieved by interpolating between the

keyframes. If the animation is played at 10 fps or 100 fps, the key frames will always be played at the correct time, and the length of the animation will not change. Under this system, different hardware setups only cause a different number of in-between frames, they will not alter how fast the animation is played.

2.3 Motion Warping

Where do the animations in computer games come from? There are two sources. Either through motion capture, or from an artist using a program such as 3D Studio Max or Maya. For motion capture, an actor wears a suit with sensors. The positions of these sensors can be tracked in 3D space over time by the motion capture rig. It allows the actor to perform motions that can be recorded into a computer. This motion data can be applied to a computer character with the character performing the same motion as the actor did during the recording. This can lead to much more realistic animations as the motion capture can pick up on small movements (like the hips rotating) that an animator either may not know about, or may not think it worth their while incorporating. However, when you have an animation from motion capture, it is hard to edit it, so if you don't get the recording right it often must be re-recorded, which can be expensive to do.

The other method is to have an animator use Maya or 3D Studio Max to make the animations required. This is cheaper than subcontracting motion capture or buying the equipment, but a lot more time consuming. For every action needed, the artist must make an animation for it, thereby building up a library of animations over time. As a result of this, it is accepted that objects in a game environment will conform to a uniform size to suit the corresponding animations: ledges are all the same height, boxes are all the same size and weight, doors all have the same dimensions etc. This means that one 'climb-onto-ledge' animation will suffice, instead of having to make a different 'climb-onto-ledge' animation for every ledge. Similarly for doors, as there are no small doors there is no need for a 'stoop-through-door' animation. This can lead to a somewhat monotonous game environment.

Institiúid Teicneolaíochta Leitir Ceanainn
Letterkenny Institute of Technology

2.4 Modeling for Animation

When creating a model for animation, character traits need to be specified; hero or villain, important or insignificant, aggressive or meek? Often the role of a character will influence the choice of the character's appearance. However, this is mostly for an artist to determine. Looking at a character from a technical side, it is desirable that computations associated with the character are fast and require as little storage space as necessary. There may be limitations to the software used for creating the character, or in the software/game in which the character will be used that will affect their appearance.

When making a model for animation there are a number of different ways to model the character's surface. The two main methods employed are polygons and patches. Each method has its own strengths and weaknesses. The desired appearance of the character will often be the biggest factor in choosing a method to model it [2].

2.4.1 Polygons

Polygons consist of 3 different parts, vertices, edges and faces. A vertex is a point in 3D space, an edge is a line between 2 vertices and a face is the space enclosed by 3 or more edges. Generally, a polygon will end up as triangles. This is because 3 vertices define a plane. Trying to have 4 vertices define a polygon when the 4 vertices are not on the same plane creates a problem which can be solved by using 2 triangle polygons.

Using polygons is the most popular method for modeling in 3D and all other methods at some point are reduced to polygons before rendering. The advantage of polygons is that they can be used to model any type of surface, whereas patches suit only certain types of models. Despite this, accurately modeling a curved surface requires lots of polygons which can be slow to render and have a high memory demand.

The diagrams shown Figures 2.2 and 2.3, show only simple primary shapes. To make more complex shapes, faces are extruded, edges are beveled, split, scaled, moved, and vertices can be added and removed. See the polygon wireframe of a human character in

Institiúid Teicneolaíochta Leitir Ceanainn | Letterkenny Institute of Technology

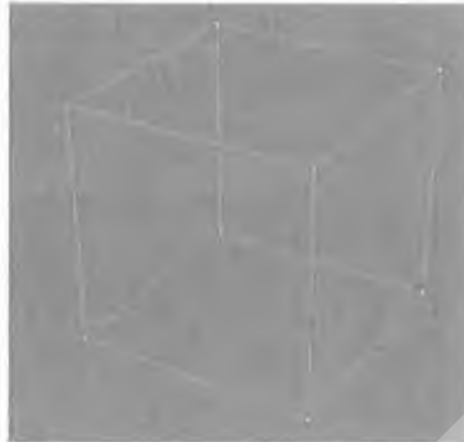


Figure 2.1: A polygon wireframe of a cube. The yellow dots are the vertices and the blue lines are edges.

Figure 2.6 for an example.

2.4.2 Patches

Patches use curves based on splines to define shapes. A shape can be defined with less information but will still get resolved down to polygons before being displayed. There are various different splines used:

Linear - 1st degree The control points are linked by a straight line.

Cardinal 2nd degree The curve passes through the control points, and each control point has control tangents used to influence the angle of approach/leaving angle of the spline. Two extra control points are needed that define the curve at the beginning and the end and, generally the curve will not pass through these.

Bézier Splines - 3rd degree and higher Splines with a degree greater than or equal to three. Bézier splines only pass through the first and last points, and do so with a slope equal to that of the tangent to the line joining the first two points, and at a

lyit Institiúid Teicneolaíochta Leitir Ceanainn Letterkenny Institute of Technology

Figure 2.2: A rendered version of the same cube. The faces are more easily seen.

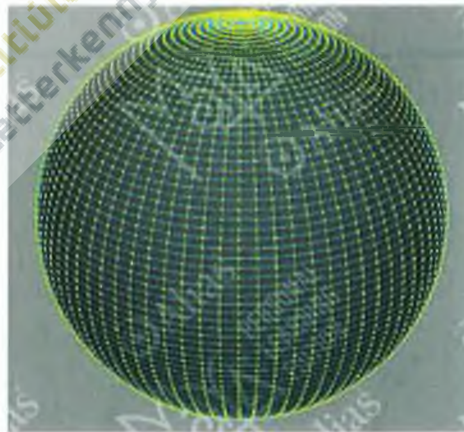


Figure 2.3: A sphere showing how a curved surface can be approximated.

lyit | Institiúid Teicneolaíochta Leitir Ceanainn
Letterkenny Institute of Technology

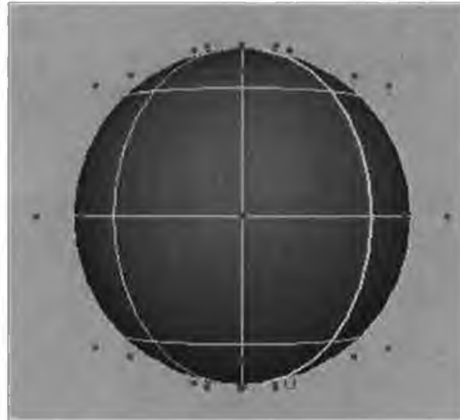


Figure 2.4: A sphere made from patches. The purple points are the control points for controlling the shape. Notice it takes less points to make a sphere from patches than from polygons. Patches are better suited to curved surfaces.

Figure 2.5: A cardinal spline passes through its control points (yellow points). The angle of entry and exit to these points can be altered by adjusting the tangent controls.

tangent to the line joining the last two points. Bézier splines do not have local control, moving one point effects the whole curve. If the degree of a Bézier is greater than three, calculating the spline becomes exponentially expensive, with regard to the number of control points. Instead, it is cheaper to base the spline on a series of 3rd degree Bézier splines. A B-spline is a type of Bézier spline.

Institiúid Teicneolaíochta Leitir Ceanainn | Letterkenny Institute of Technology

2.5 The Character

The wireframe character in Figure 2.6 was created in Maya. It has no texture, just volume. Texture is created by an artist and applied as a texture map, in effect filling the polygons

and producing the impression of a solid. The texture process is beyond the scope of this research. It was created from a polygon cube, with extrusions and bevels and other operations to give it its shape.

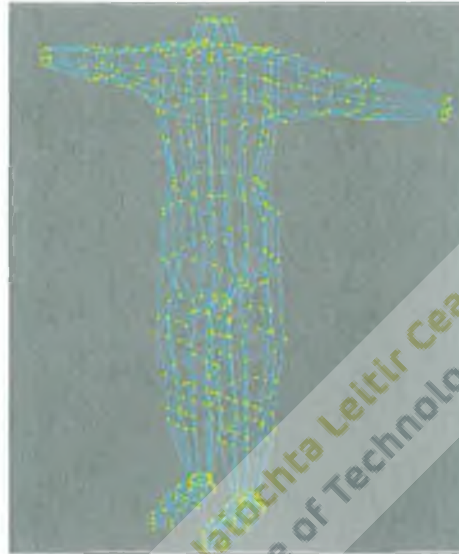


Figure 2.6: Wireframe representation of a character made in Maya.

The character looks blocky and there are quite a few vertices, as shown by the yellow dots. The character can be made more realistic by manually adding more vertices and hence polygons, or automatically, by smoothing the character, as shown in Figure 2.7.

The number of vertices in the second character has increased greatly. To animate this character, using keyframe animation, means having several 'heavy' keyframes, each holding the position of all the vertices. Frames in between keyframes are arrived at by interpolating the two nearest keyframes. This means an interpolation operation for each vertex, which is a lot of interpolation. Another solution may be to use the blocky or low vertex character for interpolation and then perform a smoothing operation before rendering. This will reduce the overhead for interpolation, and reduce the memory overhead for the keyframes. However, even with this approach there are still a lot of vertices being stored for each keyframe.

Institiúid Teicneolaíochta Leitir Ceanainn
Letterkenny Institute of Technology

Figure 2.7: A smoothed version of the same character. The yellow dots are vertices.

A cheaper approach, and one that better suits animation, is to animate a skeleton and then apply the high resolution mesh (like seen above) onto it when the pose for that frame has been calculated. This skeleton pose comes either straight from a keyframe, or from interpolation of two keyframes. In effect the skeleton will act as a deformer for the character's mesh.

The wireframe diagram shown below in Figure 2.8 gives an idea of the bones involved in a character. Without the head, neck and hands, there are 17 bones in this modeled body. The head and hands have been left out of this example as they are often features of animation by themselves. Although shown as polygons, in practice the bones are just lines, represented by two points in 3D space. How they are stored and manipulated is dealt with in Chapter 4. Two points per bone, by 17 bones is 34 points to be held in memory. This is much more efficient than holding a blocky character in memory. Its also more efficient when it comes to interpolation to get a pose. There are now 34 points to interpolate, not a whole character mesh.

The animation warping carried out in this research is based on the skeleton model. It is

Institiúid Teicneolaíochta Leitir Ceanainn
Letterkenny Institute of Technology
lyit

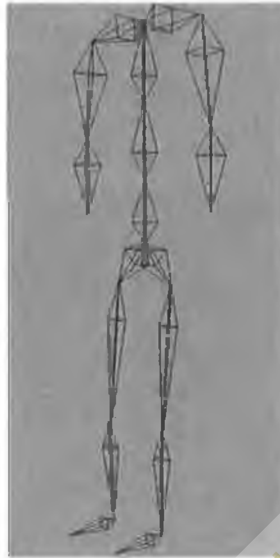


Figure 2.8: A representation of a character's skeleton.

the skeleton that is warped, with the mesh being added at a later point.

2.6 Applying the Mesh

The vertices in the mesh all have a weighted association with one or more bones on the skeleton. If the vertices on a mesh are only associated with one bone, the deformations imposed by the skeleton will lead to tearing in places. For example, take the elbow shown in Figure 2.9.

The forearm and upper arm are shown as the two triangles, with the dots representing the vertices. When the elbow bends, the desired result is for the skin on the acute side to fold up and the skin on the other side to stretch. However, bending the forearm (white triangle) upwards, leads to the situation shown in Figure 2.9, which does not look acceptable.

To eliminate this problem, a weighted association between the bone and the vertices is used. Each vertex is associated with a bone with a certain weight. If the weight is 1, then it is only associated with that bone. If it is less than one, it is associated with more than one

lyit | Institiúid Teicneolaíochta Leitir Ceanainn | Letterkenny Institute of Technology

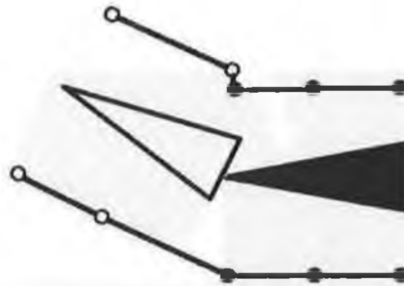


Figure 2.9: The straight edge on the inside of the elbow is the problem.

bone. As in [1], a useful approach is $x = \sum_{i=0}^{n} M_i d_i w_i$ where:

- x is the global position of the vertex.
- M_i is the global matrix for bone i (see 4 for greater discussion),
- d_i is the distance between the a common point on the bone and the vertex,
- w_i is the weight and
- n is the number of bones the vertex is associated with.

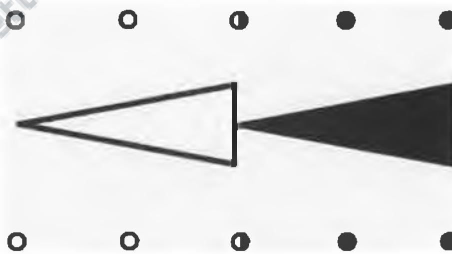


Figure 2.10: Associating the middle vertices with both bones with an even weight.

If the middle vertices in the example above are associated with both bones using this method (not just the upper arm), as shown, the result of the arm bending in the same manner

lyit | Institiúid Teicneolaíochta Leitir Ceanainn Letterkenny Institute of Technology

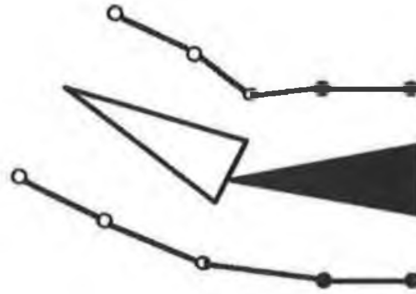


Figure 2.11: The result of a weighted bend.

as before is much more acceptable. Now the vertex at the elbow does move, shrinking on the acute side and stretching on the other side.

One issue with this method occurs when performing an operation like rotating a hand while keeping the forearm fixed. The joint between them will collapse. In practical terms, this is taken into account when setting up the character, resulting in it not being a significant issue. A resolution of this problem is achievable but currently too expensive, especially when taking into account that it does not tend to show up at run time. This method is explained in more detail in [1].

2.7 Conclusion

Real-time character animation is not based on animating a mesh, but instead animating a skeleton and applying a mesh afterwards. In the next chapter, other approaches to animation (such as a physical approach or inverse kinematics based approach), are discussed.

lyit | Institiúid Teicneolaíochta Leitir Ceanainn | Letterkenny Institute of Technology

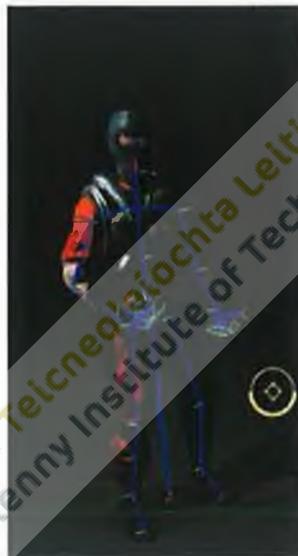


Figure 2.12: A real world example of a skeleton being used to deform a mesh are the characters in Torc Interactive's Dreadnought.

lyit Institiúid Teicneolaíochta Leitir Ceanainn Letterkenny Institute of Technology

Chapter 3

Animation Technologies

3.1 Introduction

The keyframe-based animation framework upon which this research is based is not the only framework for processing computer animation. Other frameworks center around different methods for specifying the animation. A spacetime system uses a physical approach, but there are also hierarchical approaches built upon inverse kinematics (IK) to determine animations. This chapter discusses these other animation frameworks, as well as animation warping approaches that are similar to this work.

lyit | Institiúid Teicneolaíochta Leitir Ceanainn | Letterkenny Institute of Technology

3.2 A Physical Approach

Animation deals with the movement of objects in a virtual environment. Most of the time, these object's appear to obey the same physical laws as in reality, with exceptions being made to express artistic input (Wile Coyote running off a cliff and hovering in the air until he looks down is an example). A physical approach to animation takes away control of a character or object from an animator and surrenders it to the laws of physics. An animation of an objects motion may be determined, not by an animator specifying each frame, rather by computing the motion according to the laws of physics.

In the paper Spacetime Constraints [3], Witkin and Kass describe such a physically based system. Certain attributes are specified to define and enable this physically based system - the action of the character, how this action should be performed, the structure of the character and the environment in which the action is to take place.

Using a Luxo lamp as a character, see Figure 3.1, the main action is jumping and the performance attributes include 'being as energy efficient as possible' or landing as softly as possible'. The structure is a chain of four joints with certain weights and sizes, and the environment includes information about the springs on the lamp and the surfaces on which the action is carried out.

Figure 3.1: The animated Luxo lamp.

Institiúid Teicneolaíochta Leitir Ceanainn
Letterkenny Institute of Technology

The jumping action of a Luxo lamp can be defined by equations of motion, derived from Newton's Laws relating to forces. The force is split into horizontal (f_H) and vertical (f_V) components as the jumping Luxo lamp can be treated as a projectile. The horizontal distance to be jumped s_H can be found by specifying the take off and landing points. The path of the lamp from take off to landing can change depending on the requirements of the jump (e.g. as little energy needed, or as little vertical displacement as possible), and the specifications of the lamp (e.g. the weight of the lamp).

To perform the jump using as little energy as possible, the force required to move the lamp must be minimized. If the lamp only moves across the floor, it will encounter a frictional force. This can be avoided if the lamp jumps - which looks more 'natural'. The sum of f_H and f_V gives the total force required to make the lamp jump a distance s. If this force is to be as small as possible, the problem becomes a minimization problem.

The mass can be assumed constant, leaving displacement, initial velocity and time to be decided. A combination of these needs to be found such that the force will be a minimum. The solution described uses Sequential Quadratic Programming. A solution is needed for each joint on the lamp - where the forces involved are the spring forces for each joint. In the context of animating a lamp, these joint forces can be derived. However, in the context of animating a person, the joint forces are not so well defined or understood, and will change from character to character, as will the characters' mass, and several other physical attributes that would affect any physically-based calculations. See [4].

3.3 An Inverse Kinematics Approach

The normal approach to character animation can be described as kinematic - angles are assigned to each bone, and these place the feet and hands in certain positions. Inverse kinematics (IK) places the hands and feet at certain positions, and then calculates the angles to assign to the bones. Why use IK to control motion? If the path of a character traverses uneven ground and its walking motion has been authored assuming level ground, the character feet will hover above the ground in places, while sinking into the ground in other places.

In [4] Chung and Hahn discusses the use of IK to adapt animations at run time to incorporate the characters' environment. In order to appear realistic, the IK method described is based on studies from animation, biomechanics, human gait experiments and psychology. This is important, as while producing a motion from IK that doesn't violate any constraints on joint angles is reasonably straightforward, a simple solution is likely to appear unconvincing and somewhat robotic.

Particular attention is paid to the gait of the character. Planning a route through the environment starts by assuming the ground is flat, then looking for any physical obstacles on the straight path. If obstacles are small they can be stepped on or over, but if they are too big for this the path must go round them. Footprints are then placed according to the

lyit Institiúid Teicneolaíochta Leitir Ceanainn Letterkenny Institute of Technology

character step length, changes in direction along the path, and how uneven the terrain is.

With the footsteps in place, the next step is to make the character use them, but still retain a realistic looking motion. At any time there will be a stance leg and a swinging leg. The calculation of the arc of the swinging leg checks for any obstacles that must be avoided. The resulting path is a 'least energy spent' path. The path of the foot will follow a Bézier curve over any detected obstacles, accelerating around the midpoint of the swing. Conversely, the pelvis, while also following a Bézier curve, slows while the swinging leg passes through its midpoint.

An adaptation of such an IK method could perhaps be used to control a character's overall motion. Instead of using IK to make the swinging arms avoid obstacles - certain obstacles could be tagged so a hand will reach for them, handrails being one example. The collision avoidance mechanism could be adapted to control an IK chain with the head as an end effector, forcing the character to stoop or bend under obstacles.

3.4 Motion Blending

The previous sections have discussed approaches to automating animation using methods that didn't involve blending. Blending is an approach widely used in computer animation. Blending is most commonly used when joining two animations. For instance, coming from a crouch animation to a walk animation. The two animations can be blended (assuming the walk cycles are synchronized) to give a crouch-to-walk animation. Blending in this manner removes the need to author transitional animations.

In [5], Sloan, Rose and Cohen describe a system for creating new motions based on a set of example motions. Interpolation between motions is carried out at runtime to create new motions, with adjectives used to indicate the interpolation factor between motions. For example, interpolating a 'sad walk' with an 'injured walk' can give a sad injured walk. But by changing the interpolation factor, a bias can be given to either the sad walk or the injured walk. The system works by placing animations in an abstract multidimensional

Institiúid Teicneolaíochta Leitir Ceanainn
Letterkenny Institute of Technology

space. Animations are classified by adjectives, with a dimension for each adjective. Tagging important parts of the motions, such as when the feet are on the ground, speeds up timewarping. Entering a request like 'reasonably hurt' defines a point in the abstract space - in the 'hurt' dimension, as well as in a 'normal' dimension. The system will interpolate, or blend the hurt walk with a normal walk.

This is different from the solution devised in this research, as while their motions come from a blend of two motions, our motions come from an "addition" of a motion and a pose. While the degree of interpolation between two animations in their work is variable, i.e. very hurt, reasonably hurt, marginally hurt, the degree of addition between an animation and a pose in this work is not. This can be seen in section 7.3.

Indeed the problem of blending a pose and a motion is quite different from blending two motions. This is shown in Chapters 6 and 7, where the timewarping is based upon the procedure outlined by Bruderlin and Williams in their paper Motion Signal Processing [6]. The same timewarping procedure is used by Kovar and Gleicher in their paper Flexible Automatic Motion Blending with Registration Curves [7]. Both describe systems of blending animations, but where the earlier Motion Signal Processing paper describes a system where the motions to be blended have the same course, there is no such requirement in Kovar's/Gleicher's work. The variance in the course of the animations leads to problems involving the character leaning to the side when moving round a corner, and taken to an extreme can reverse the direction of the animation. Their solution involves lining up the animations, not just by timewarping, but by translation, so that they are both traveling in the same direction.

Institiúid Teicneolaíochta Leitir Ceanainn | Letterkenny Institute of Technology

3.5 Conclusion

Undoubtedly there is a large body of work already in the motion warping field. However, most of it describes warping multiple animations, not an animation and a pose. The result of this work is a system for blending animations, which is then adapted and improved to warp

poses with animations. In the next chapter we consider some of the essential mathematics required for this work.

lyit | Institiúid Teicneolaíochta Leitir Ceanainn Letterkenny Institute of Technology

Chapter 4

Essential Mathematics

4.1 Introduction

A skeleton used for animation can be viewed as an abstract entity. In the normal running of a game, the skeleton is not visible. In the development of a game it can be enabled to assist debugging, but artists, when creating animations work with a character with volume - not with the skeleton. This animation is then abstracted to a skeleton form to enhance performance and reduce the memory required for storing that animation. The skeletal animation consists of a series of positions and rotations held in matrices. This chapter discusses the mathematical side of such an implementation. A more detailed description of the mathematics discussed can be found in Dunn and Parberry [12] and calculus regarding IK can be found in [13].

Lyit | Institiúid Teicneolaíochta Leitir Ceanainn
Letterkenny Institute of Technology

4.2 Coordinate Spaces

In the context of a skeleton, there are three associated coordinate spaces:

World Space - This is the space into which the skeleton is placed. As an example, think of a football game. The world space for a character is the football pitch. The character's position will be translated about this area.

Object Space - This is the coordinate space of the skeleton. If a translation or rotation (the skeleton is turning) is applied to the skeleton, the whole skeleton will be moved. Other objects in world space are not affected by this.

Local Space - Each individual bone has its own local space. It allows rotations to be applied to individual bones without affecting their parent bones. An ankle rotating in isolation from its parent leg is an example of this. Local Space is also referred to as bone space.

4.3 Rotation of a Point

In order to rotate a point, three things are required, the point, the angle of rotation and an axis to rotate the point about. When rotating in 2D, the axis of rotation is the z-axis. Rotation in 3D can be about the z-axis, or any of the cardinal axes, or about any axis defined in terms of the x,y and z-axis.

4.3.1 2D rotation about the origin

Looking at Figure 4.1, it shows two points $p = (1, 0)$ and $q = (0, 1)$. Rotating these points by an angle of θ gives $p = (\cos\theta, \sin\theta)$ and $q = (-\sin\theta, \cos\theta)$. This can also be shown in matrix form as

$$\begin{pmatrix} \cos\theta & \sin\theta \\ -\sin\theta & \cos\theta \end{pmatrix}$$

lyit | Institiúid Teicneolaíochta Leitir Ceanainn
Letterkenny Institute of Technology

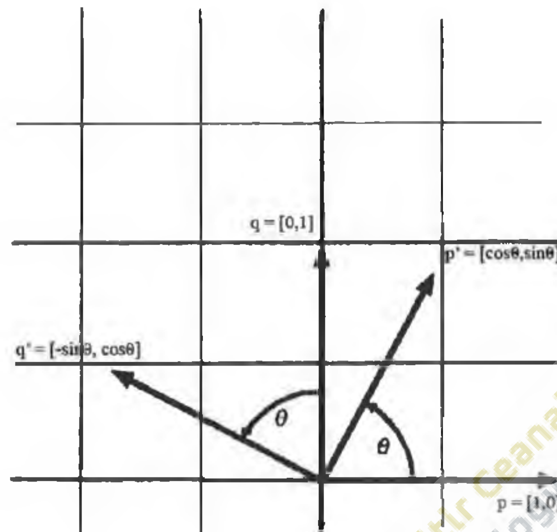


Figure 4.1: 2D rotation of a point about the origin

4.3.2 3D rotation about a cardinal axis

Looking at this in 3D, this can be interpreted as a rotation on the xy plane about the z-axis. Adjusting the matrix to allow for this gives

$$\begin{pmatrix} \cos\theta & \sin\theta & 0 \\ -\sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

In this matrix the top row represents the x-axis, the middle row represents the y-axis and the bottom row represents the z-axis. Keeping this in mind, it follows that a rotation in the yz plane about the x-axis is given by

$$\begin{pmatrix} 1 & 0 & 0 \\ 0 & \cos\theta & \sin\theta \\ 0 & -\sin\theta & \cos\theta \end{pmatrix}$$

lyit | Institiúid Teicneolaíochta Leitir Ceanainn
Letterkenny Institute of Technology

and a rotation in the xz plane about the y-axis is given by

$$\begin{pmatrix} \cos\theta & 0 & -\sin\theta \\ 0 & 1 & 0 \\ \sin\theta & 0 & \cos\theta \end{pmatrix}$$

4.3.3 3D rotation about an arbitrary axis through the origin

Rotations on bones are carried out in the bone's local space. A result of this is that when rotating about an arbitrary axis the angle of rotation will be a local space angle. Because there is no translation involved in such a rotation, the axis of rotation will be through the local space origin.

Combining the three cardinal axis rotation matrices from section 4.3.2, the following formula can be derived to rotate a point about an arbitrary axis that passes through the origin [12]:

$$\begin{pmatrix} n_x^2(1-\cos\theta)+\cos\theta & n_xn_y(1-\cos\theta)+n_z\sin\theta & n_xn_z(1-\cos\theta)-n_y\sin\theta \\ n_xn_y(1-\cos\theta)-n_z\sin\theta & n_y^2(1-\cos\theta)+\cos\theta & n_yn_z(1-\cos\theta)+n_x\sin\theta \\ n_xn_z(1-\cos\theta)+n_y\sin\theta & n_yn_z(1-\cos\theta)-n_x\sin\theta & n_z^2(1-\cos\theta)+\cos\theta \end{pmatrix}$$

Here, n is the axis of rotation, with n_x, n_y and n_z being the x,y and z coordinates of the axis.

4.4 Translation of a Point

To translate a point is to offset a point by a scalar amount. This is different to rotation as seen so far. Rotation involved moving a point through an arc about the origin, and as such, the distance between the point and the origin remained unchanged. With translation, this distance can change.

Institiúid Teicneolaíochta Leitir Ceanainn | Letterkenny Institute of Technology lyit

Taking a point (x, y) it can be translated by adding a scalar to x and y to give $(x + \Delta x, y + \Delta y)$. This can be represented in a matrix:

$$(x\; y\; 1) \times \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ \Delta x & \Delta y & 1 \end{pmatrix} = (x + \Delta x, y + \Delta y)$$

The 1 in the point is there to facilitate matrix multiplication, as the number of columns of the left hand side of the multiplication must be the same as the number of rows on the right hand side of the multiplication.

While this may seem a bit cumbersome to use a 3×3 matrix for a translation, if this matrix is multiplied with the matrix for rotation of a point about the origin it yields:

$$\begin{pmatrix} \cos\theta & \sin\theta & 0 \\ -\sin\theta & \cos\theta & 0 \\ \Delta x & \Delta y & 1 \end{pmatrix}$$

This is the same as rotating the point and then translating it. It can be expanded on by combining the matrix for rotation about an arbitrary axis in 3D with a 3D translation matrix to give:

$$\begin{pmatrix} n_x^2(1-\cos\theta)+\cos\theta & n_xn_y(1-\cos\theta)+n_z\sin\theta & n_xn_z(1-\cos\theta)-n_y\sin\theta & 0 \\ n_xn_y(1-\cos\theta)-n_z\sin\theta & n_y^2(1-\cos\theta)+\cos\theta & n_yn_z(1-\cos\theta)+n_x\sin\theta & 0 \\ n_xn_z(1-\cos\theta)+n_y\sin\theta & n_yn_z(1-\cos\theta)-n_x\sin\theta & n_z^2(1-\cos\theta)+\cos\theta & 0 \\ \Delta x & \Delta y & \Delta z & 1 \end{pmatrix}$$

where the point to be multiplied is given in the form of $(x, y, z, 1)$.

4.5 Translating Bones

A skeleton can be thought of as a series of connected lines. Each of these lines can be defined as two points. Given two connected lines, A and B, moving in two dimensions, if

Institiúid Teicneolaíochta Leitir Ceanainn
Letterkenny Institute of Technology
lyit

a rotation is applied to A, the position of B will change:

$$A = (0,0)\ to\ (1,0), B = (1,0)\ to\ (2,0)$$

Applying a rotation of ϕ to A will give $A = (0,0)\ to\ (cos\phi, \sin\phi)$. This will mean the start point of B will now have changed to $(cos\phi, \sin\phi)$. But what about the endpoint of B? The length of B is one, so the end point of B will be at a distance of 1 from the endpoint of A. Two different things can happen to B:

1. The rotation was applied to A. B will have no rotation, remaining parallel to the x-axis, but will be translated to the end point of A.

2. B will be translated to the endpoint of A and will assume the same rotation as A, remaining parallel to A.

Assuming the two lines A and B represent an arm. If the arm is straight, and then rotated at the shoulder joint, it's desirable to have the forearm - in this case B, assume the same rotation as the upper arm - A. If the lines are stored as two points, there is nothing to infer that the angle between A and B should be maintained after A is rotated. A better representation of the lines - or bones, is to use a matrix. The matrix can hold the rotation of the line relative to the parent. The matrix can also hold the translation its bone must go through to be positioned correctly in object space. Because A is the parent bone in this case, its object space is the same as its local space and hence there is no translation necessary.

$$\begin{pmatrix} a & b & 0 \\ c & d & 0 \\ x & y & 1 \end{pmatrix}$$

a, b, c and d hold the rotation of the line and x and y hold the translation into the object space of the parent bone. Putting this to work in this example gives:

$$A = \begin{pmatrix} \cos\phi & \sin\phi & 0 \\ -\sin\phi & \cos\phi & 0 \\ 0.0 & 0.0 & 1 \end{pmatrix} \quad B = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 1.0 & 0.0 & 1 \end{pmatrix}$$

The translation part of B has the same length as A, namely 1. The translation part of a line can be thought of as holding the length of its parent line. This means that to define the last child line, a separate matrix is needed. This is called a nub. It does not need to contain any angle for rotation, as it has no child bones to be rotated.

$$\begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 1.0 & 0.0 & 1 \end{pmatrix} \times \begin{pmatrix} \cos\phi & \sin\phi & 0 \\ -\sin\phi & \cos\phi & 0 \\ 0.0 & 0.0 & 1 \end{pmatrix} = \begin{pmatrix} \cos\phi & \sin\phi & 0 \\ -\sin\phi & \cos\phi & 0 \\ \cos\phi & \sin\phi & 1 \end{pmatrix}$$

In this result, the $\cos\phi$ and $\sin\phi$ in the 3rd row of the matrix give the translation to apply to B, while giving the length of A, which is still 1. The line A is now $(0,0)$ to $(\cos\phi, \sin\phi)$ as expected. As mentioned above, to give the length of B a nub bone is used. In this case, the nub will be:

$$\begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 1 & 0 & 1 \end{pmatrix}$$

Multiplying this with the object space matrix for B will give the matrix for the object space nub, and in doing so will define the end of B in object space:

$$\begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 1 & 0 & 1 \end{pmatrix} \times \begin{pmatrix} \cos\phi & \sin\phi & 0 \\ -\sin\phi & \cos\phi & 0 \\ \cos\phi & \sin\phi & 1 \end{pmatrix} = \begin{pmatrix} \cos\phi & \sin\phi & 0 \\ -\sin\phi & \cos\phi & 0 \\ 2\cos\phi & 2\sin\phi & 1 \end{pmatrix}$$

As with A, the endpoint of B can be read from the translation part of the matrix. B now runs from the end point of A, $(\cos\phi, \sin\phi)$ to $(2\cos\phi, 2\sin\phi)$.

This example illustrates the parent-child relationship of the bones as well as local space and object space rotation.

4.6 Eulerian Angles

The matrix approach to rotating lines in 3D has its advantages. It is reasonably easy to understand, but not that efficient. It uses nine numbers to express a rotation or orientation.

LYIT Institiúid Teicneolaíochta Leitir Ceanainn
Letterkenny Institute of Technology

Eulerian angles (henceforth known simply as "Eulers") can express angular rotation using three numbers. Instead of x, y and z-axis, Eulers refer to heading(y), pitch(x) and bank(z).

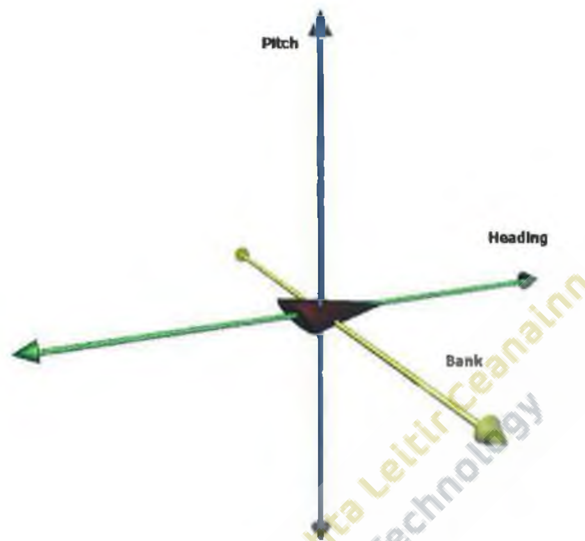


Figure 4.2: Euler Axis

An example of a rotation given in Eulers could be $(50^o, 60^o, 20^o)$. This rotates 50° about the heading, 60° about the pitch and 20° about the bank. After rotating about the heading, there is a pitch of 60° about the x-axis. However, this is the local space x-axis, not the object space x-axis. As such, it was moved with the change in heading. Finally, there is a bank of 20°. The z-axis that the bank occurs around is also a local space axis and has been moved twice as a result of the change in heading and pitch.

The order of rotations is important. Performing the rotations in the opposite order will result in a different orientation.

An issue with Eulers is that a given orientation can be represented by various different tuples of Eulers. The most obvious case of this is adding 360° to any of the elements of the Euler. This will rotate a full circle. Another issue with Eulers is that the three angles are not independent of each other. For example, pitching up 30° is the same as heading 180°, pitching 150° and then banking 180°.

Institiúid Teicneolaíochta Leitir Ceanainn
Letterkenny Institute of Technology
lyit

This can be partially solved by restricting the range of the angles. Limit heading and bank to ± 180 and pitch to ± 90. When functions are returning Eulers, they will always fit this "canonical" format. There is still a singularity that can occur. Eulers have three degrees of freedom. If there is a pitch of $\pm 90^o$, any change in heading will have the same effect as a change in bank. There are now only two degrees of freedom. This is known as gimbal lock. Because of this, Eulers, while more efficient than matrices in terms of computation and memory, are not suitable for representing bones in a skeleton. There is another issue regarding Euler interpolation that crops up later on in the implementation (see Section 6.3.1), and the problem and solution are discussed then.

4.7 Quaternions

Three numbers cannot safely represent an orientation, as seen with Eulers. The proof is quite detailed and advanced, and is not discussed here. Using four numbers, in the form of quaternions, avoids such problems. Quaternions have a lot in their favor when used to represent a 3D orientation.

4.7.1 Quaternion Algebra

A quaternion consists of a scalar and a 3D vector. As the vector is already referred to as $(x,\ y,\ z)$, the scalar part will be w.

$$\mathbf{q} = [w\ (\ x,\ y,\ z\ )]$$

A series of rotations can be expressed as a single rotation about an axis. This axis is not necessarily a cardinal axis. Given a vector $\mathbf{n}$, it can be an axis for a rotation. The length is not important, only the direction. However, it's convenient to scale it to a length of one. The amount of rotation about this axis can be given by a scalar with the positive direction being determined from the way the vector axis is pointed. Thus, the pair $(\theta, \mathbf{n})$ is an axis-angle

Institiúid Teicneolaíochta Leitir Ceanainn | Letterkenny Institute of Technology

rotation. As a quaternion is a scalar and a vector - it can represent an axis-angle rotation. The $(\theta, \mathbf{n})$ pair don't plug straight into a quaternion. The following format is used:

$$\mathbf{q} = [\cos(\theta/2)\ \sin(\theta/2)\mathbf{n}]$$

$$= [\cos(\theta/2)\ (\ \sin(\theta/2)\mathbf{n}_x\ \sin(\theta/2)\mathbf{n}_y\ \sin(\theta/2)\mathbf{n}_z\ ]$$

When negating a quaternion, all the elements are negated:

$$-\mathbf{q} = [-w\ (\ -x,\ -y,\ -z\ )]$$

When the vector is negated, it points in the opposite direction. Because the direction of rotation depends on which way the vector points, after negation, the direction of rotation changes direction. The result of this is a negated quaternion gives the same rotation as its positive version. This can prove troublesome when converting quaternions to Eulers.

4.7.2 Quaternion Multiplication

Quaternions are an extension of complex numbers. The scalar part is real, the vector part is "imaginary". Quaternion multiplication takes this into account:

$$\begin{aligned}
&(w_1 + x_1 i + y_1 j + z_1 k)(w_2 + x_2 i + y_2 j + z_2 k) \\
&= w_1 w_2 + w_1 x_2 i + w_1 y_2 j + w_1 z_2 k \\
&+ x_1 w_2 i + x_1 x_2 i^2 + x_1 y_2 ij + x_1 z_2 ik \\
&+ y_1 w_2 j + y_1 x_2 ij + y_1 y_2 k^2 + y_1 z_2 jk \\
&+ z1 w_2 k + z_1 x_2 ik + z_1 y_2 jk + z_1 z_2 k^2
\end{aligned}$$

$$\begin{aligned}
&= w_1 w_2 + w_1 x_2 i + w_1 y_2 j + w_1 z_2 k \\
&+ x_1 w_2 i + x_1 x_2(-1) + x_1 y_2(-k) + x_1 z_2(-j) \\
&+ y_1 w_2 j + y_1 x_2(-k) + y_1 y_2(-1) + y_1 z_2(i) \\
&+ z1 w_2 k + z_1 x_2(j) + z_1 y_2(-i) + z_1 z_2(-1)
\end{aligned}$$

lyit | Institiúid Teicneolaíochta Leitir Ceanainn
Letterkenny Institute of Technology

$$
\begin{aligned}
&= w_1w_2 - x_1x_2 - y_1y_2 - z_1z_2 \\
&+(w_1x_2 + x_1w_2 + y_1z_2 - z_1y_2)(i) \\
&+(w_1y_2 + y_1w_2 + z_1x_2 - x_1z_2)(j) \\
&+(w_1z_2 + z_1w_2 + x_1y_2 - y_1x_2)(k)
\end{aligned}
$$

Rewriting this in a more recognisable layout:

$$
\begin{bmatrix} w_1w_2 - x_1x_2 - y_1y_2 - z_1z_2 \\ \begin{pmatrix} w_1x_2 + x_1w_2 + y_1z_2 - z_1y_2 \\ w_1y_2 + y_1w_2 + z_1x_2 - x_1z_2 \\ w_1z_2 + z_1w_2 + x_1y_2 - y_1x_2 \end{pmatrix} \end{bmatrix} \Leftrightarrow \begin{bmatrix} w \\ \begin{pmatrix} x \\ y \\ z \end{pmatrix} \end{bmatrix}
$$

4.7.3 Quaternion Inverse

The inverse of a quaternion is calculated by dividing the conjugate of the quaternion by the square of its magnitude. As stated already [4.7.1] the magnitude of quaternions will be 1. In this case the inverse of a quaternion will be the same as its conjugate.

As with complex numbers, the conjugate of a quaternion is calculated by negating the imaginary part - in the case of a quaternion this is the vector part. The conjugate of p is denoted p^*:

$$
p = [\, w\ (\ x,\ y,\ z)] \Rightarrow p^* = [\, w\ (-x, -y, -z)]
$$

4.7.4 Rotating Points with Quaternions

Instead of using a quaternion to hold an axis and an angle of rotation, the quaternion can be used to hold a point in 3D space, by putting the point in the vector and setting the angle to 0, $p = [\, 0\ (\ x,\ y,\ z)]$. This point can be rotated clockwise about the axis of a quaternion by the angle of the quaternion with the following multiplication:

$$
p' = q^{-1}pq
$$

where q is the rotating quaternion and p' is the point after rotation.

Institiúid Teicneolaíochta Leitir Ceanainn
Letterkenny Institute of Technology
lyit

The inverse of a quaternion product is the product of the inverses, when the inverses are multiplied in reverse order:

$$(qr)^{-1} = r^{-1}q^{-1}$$

This comes in useful when a point is to be rotated by a series of quaternions. It means a series of quaternions can be multiplied up before rotating a point - so only one rotation is necessary. Let p be a point in quaternion format, and a and b be quaternions suitable for rotation. Rotating in the order a then b gives:

$$(b^{-1}(a^{-1}pa)b)$$
$$(b^{-1}a^{-1})p(ab)$$
$$(ab)^{-1}p(ab)$$

This means that a series of skeleton bones can be rotated into their parents object space by storing their local space orientation in a quaternion and multiplying it by their parents object space quaternion.

4.8 Conclusion

The quaternion does not replace the matrix completely. It only replaces the rotation part of the matrix, with the translation or length part of the matrix the same as before. Using quaternions, a reduction from nine numbers needed to store orientation to four numbers is achieved. This will reduce the memory and system requirements thus improving performance. In the next chapter we consider the topic of treating animations as a series of rotation signals over time.

lyit Institiúid Teicneolaíochta Leitir Ceanainn
Letterkenny Institute of Technology

Chapter 5

Digital Signal Processing

5.1 Introduction

The movement of a character over time, i.e. an animation, can be thought of as a series of signals changing over time. Each signal represents the rotation of a bone about an axis as the animation progresses. Manipulating these signals will alter the animation, which is a significant goal in the scope of the research. To this end, this chapter discusses the basics of digital signal processing, or DSP for short, and how it relates to the DSP implemented in the project.

One of the major concepts of DSP is being able to represent the same data in both time and frequency domains. The time domain is the representation normally used, where signals are plotted with their amplitude over time. The frequency domain shows the amplitude of the different frequencies in a signal. Figure 5.1 shows a sine wave in the time domain. The period of the signal, 0.25 *seconds* is related to its frequency. It repeats four times a second, and so has a frequency of $4\ Hz$.

In Figure 5.2 the signal has a period of $0.083 seconds$, conversely having a frequency of $12Hz$. Plotting these signals in the frequency domain will give a line at $4Hz$ and at $12Hz$, where the lines give their magnitude. But the information in these frequency domain graphs doesn't tell anything that can't be seen from an intuitive look at the time domain graphs.

lyit | Institiúid Teicneolaíochta Leitir Ceanainn
Letterkenny Institute of Technology

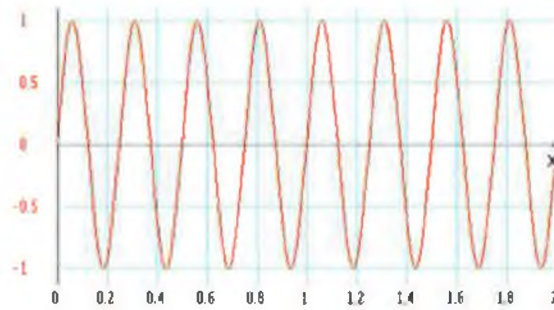


Figure 5.1: A sine wave with a period of 0.25s and an amplitude of 1 in the time domain.

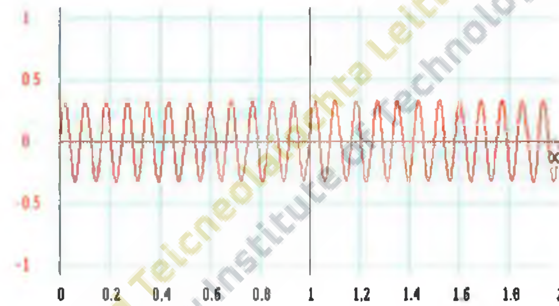


Figure 5.2: A sine wave with a period of 0.125s and an amplitude of 0.33 in the time domain.

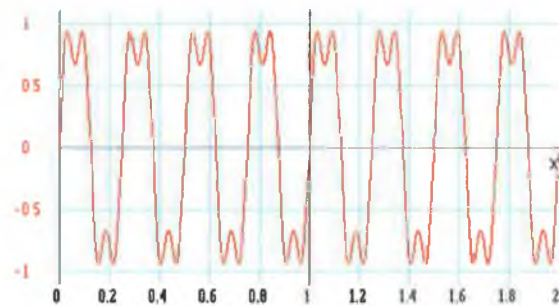


Figure 5.3: The sum of the previous two graphs in the time domain.

lyit | Institiúid Teicneolaíochta Leitir Ceanainn
Letterkenny Institute of Technology

Figure 5.3 shows the sum of the two previous signals in the time domain. The frequency domain version of the signal is not so intuitive. It is shown in Figure 5.4.

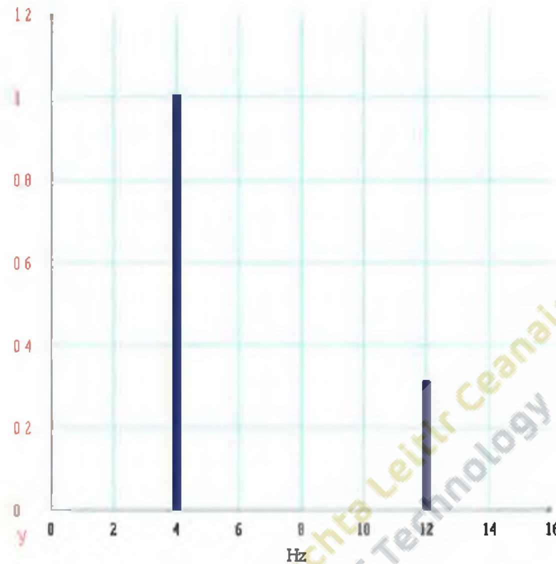


Figure 5.4: The frequency domain equivalent of Figure 5.3.

The graph shows there are two different frequencies in the signal, one at 4 Hz with an amplitude of 1 and another at 12 Hz with an amplitude of 0.33. Here, and in general, the use of the frequency domain displays information in a time signal that may not be obvious [18].

5.2 Sampling a Signal

Most operations in DSP revolve around having a continuous signal and sampling it at discrete times before manipulating the samples. The continuous signal can be periodic or aperiodic [8]. We consider four different types of signal that can occur:

Aperiodic Continuous These are continuous signals that tend towards 0 over time, for example a Gaussian curve.

Institiúid Teicneolaíochta Leitir Ceanainn
Letterkenny Institute of Technology
lyit

Periodic Continuous These are signals that repeat periodically over time, with common examples being sine and cosine curves.

Aperiodic Discrete These are signals only defined at discrete times and that do not repeat over time. The signals used when altering an animation fall into this category.

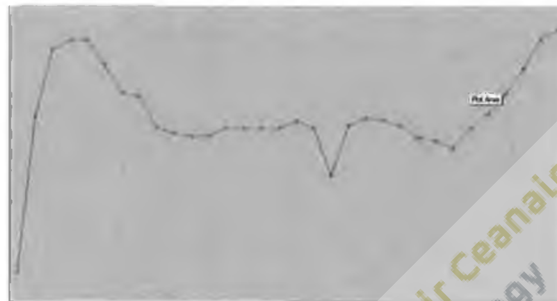


Figure 5.5: An aperiodic discrete signal taken from two animations blended together.

Periodic Discrete Discrete signals that repeat periodically over time. A walk signal from an animation falls into this category. See Figure 5.6.

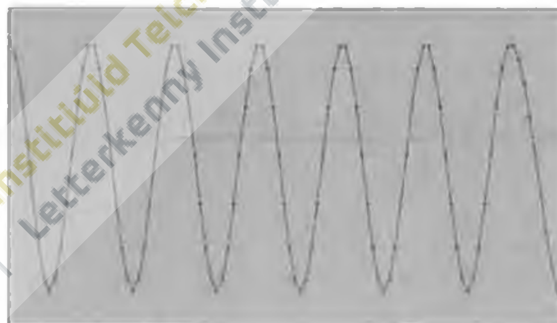


Figure 5.6: A periodic discrete signal taken from a walk animation.

lyit | Institiúid Teicneolaíochta Leitir Ceanainn Letterkenny Institute of Technology

Sampling is used to convert an analogue, or continuous signal, to a digital, or discrete signal. In an electronics setting, an analogue to digital (ADC) converter is used, with the discrete samples stored as a binary value. This conversion introduces noise into the system, known as quantization noise. The more bits in the binary value that holds a sample, the higher the resolution of the ADC and inversely, the smaller the quantization noise.

In Figure 5.7 the sine wave is an analogue signal. Samples are taken every 20°, giving a digital signal consisting of 35 points. The digital signal is now defined only at these 35 points, giving the step signal shown in Figure 5.7. If the number of samples is increased, the digital signal will better approximate the analogue signal.

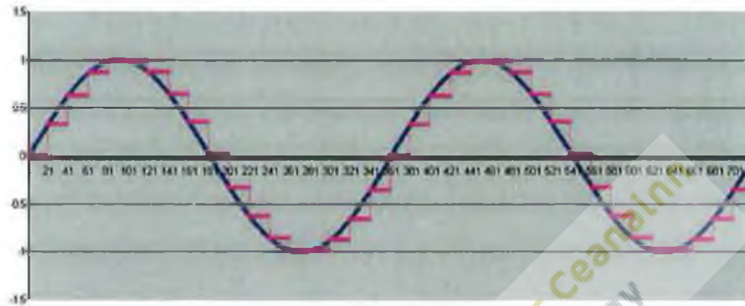


Figure 5.7: A sine wave (blue) and a sampled version (pink) of the same sine wave.

5.2.1 Aliasing

While increasing the number of samples will give a better representation of the original signal, it also requires more memory to store the digital signal, and more operations to process it. However, not taking enough samples will result in not being able to reconstruct the original signal. The "sampling theorem" says that in order to be able to reconstruct a signal from samples, the signal must be sampled at twice the Nyquist frequency, where the Nyquist frequency is the highest frequency in the signal. Sampling at a rate less than this will result in samples representing a difference signal of lower frequency [15].

In Figure 5.8 the dense sine wave is sampled at twice the Nyquist frequency, that is, twice for every period - this is shown by the black dots. If the sampling rate is less than this, as shown by the blue dots, where the signal is being sampled at $1/10^{th}$ the Nyquist frequency, the longer sine wave is stored. This is known as aliasing - where the frequency of the sampling data is different to that of the original signal.

lyit Institiúid Teicneolaíochta Leitir Ceanainn | Letterkenny Institute of Technology

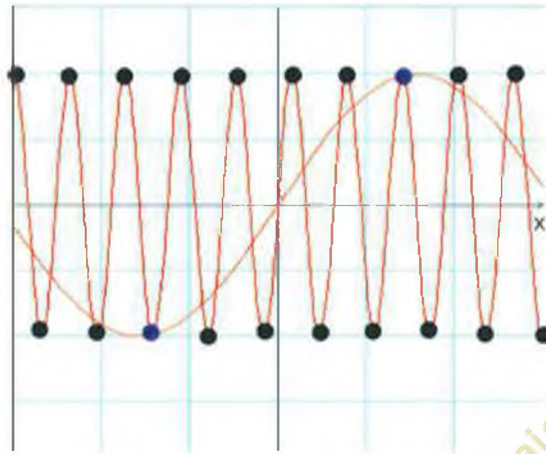


Figure 5.8: A sine wave sampled at twice the Nyquist frequency and at $1/10^{th}$ the Nyquist frequency.

5.2.2 The Frequency Domain

Thus far, the signals discussed have all been viewed in the time domain. It has been established that to sample a signal properly, it must be sampled at twice the highest frequency in the signal. But what is the highest frequency in a signal?

The time domain signals can be viewed in the frequency domain, which has the same information, but in a different representation. To convert a discrete time signal to the frequency domain either the Discrete Time Fourier Transform (DTFT) or the Discrete Fourier Transform (DFT) can be used. The DTFT is used for aperiodic signals and the DFT is used for periodic signals.

The idea behind both the DFT and the DTFT is to split up a time domain signal into sine and cosine waves. However, an infinite number of component signals are needed to represent an aperiodic signal, which means the DTFT isn't practical in terms of implementation in a computer program. The solution is to repeat the aperiodic signal over time so it appears to be periodic, instead of aperiodic, and then to use the DFT to convert it to the frequency domain.

The DFT takes a time domain signal of N samples and from it derives two $N/2 + 1$

lyit Institiúid Teicneolaíochta Leitir Ceanainn
Letterkenny Institute of Technology

Cosine signals, and $N/2+1$ Sine signals. The equations used are given in 5.1 and 5.2.

$$ReX[k] = \sum_{i=0}^{N-1} x[i] \cos \frac{2\pi ki}{N} \quad (5.1)$$

$$ImX[k] = -\sum_{i=0}^{N-1} x[i] \sin \frac{2\pi ki}{N} \quad (5.2)$$

Re is the real part of the DFT result, also known as the cosine part, and Im is the imaginary part, or the Sine part. $x[i]$ is the time domain signal being processed by the DFT. k is the index for the frequency domain signal and runs from 0 to $N/2$.

The larger k is, the higher the frequency of the component sine or cosine curve. The goal of using the DFT was to see the frequencies in the original time domain signal, so as to determine the Nyquist frequency and thus the rate at which to sample the signal at. However, the frequency resolution of the DFT depends on the number of samples in the discrete signal. The DFT cannot be used to determine at what rate to sample the original signal. In electronics, the analogue signal is passed through a low pass anti alias filter before sampling. The purpose of this low pass filter is to eliminate any frequencies above a certain threshold, guaranteeing there are no frequencies remaining in the signal above the frequency threshold of the low pass filter. With animation signals however, there are other constraints that have an impact on the sampling frequency that are more important than filter resolution or storage requirements.

5.2.3 Animation as Digital Signals

An animation may be stored as a series of keyframes. Each of these keyframes consists of an array of quaternions where each quaternion holds the rotation applied to a corresponding bone in a skeleton relative to its parent bone (the skeleton is the structure used to animate a character, with the skin being applied at a later stage). Because PC games can run at different frame rates depending on the hardware configuration of the PC the game is being run on, an animation cannot be made from x keyframes and expected to run at x frames

lyit Institiúid Teicneolaíochta Leitir Ceanainn Letterkenny Institute of Technology

per second [1]. With a variable frame rate, it is necessary to be able to resolve an animation into a variable number of frames, determined at run time. To get the frames in between two keyframes, the quaternions of both keyframes are interpolated, either using linear interpolation (LERP) or spherical interpolation (SLERP). This approach results in a series of poses at discrete times. the rotations applied to each bone over time can be taken as a digital signal (section 6.3) forming a discrete aperiodic time domain signal.

5.3 LERP and SLERP

These are two types of quaternion interpolation, both having different attributes. Both methods follow the torque minimal arc between two quaternions. However, LERP has a varying acceleration along its arc over time, see Figure 5.9.

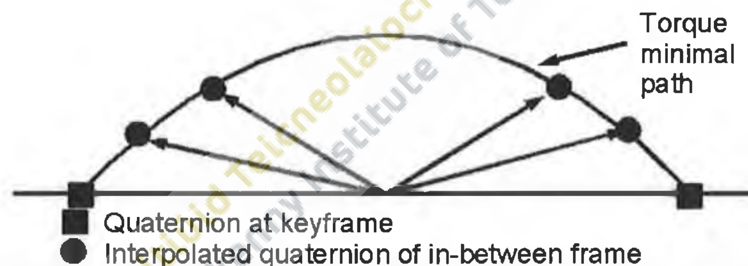


Figure 5.9: LERP - the points on the curve are not evenly spaced.

This manifests itself as the first few in-between frames having less movement than the middle in-between frames, with less movement again for the last in-between frames. SLERP follows the arc with a constant velocity, meaning no such problem exists, as shown in Figure 5.10.

In practice, the varying acceleration of LERP methods isn't a problem, so long as there are a sufficient number of pose quaternions. Although the movement will be non uniform, LERP approximates SLERP to a degree where the negative aspect of LERP isn't visible due

[1]In classic hand drawn animation for film, 12 frames were drawn, and each displayed twice consecutively to give 24 fps.

Institiúid Teicneolaíochta Leitir Ceanainn Letterkenny Institute of Technology

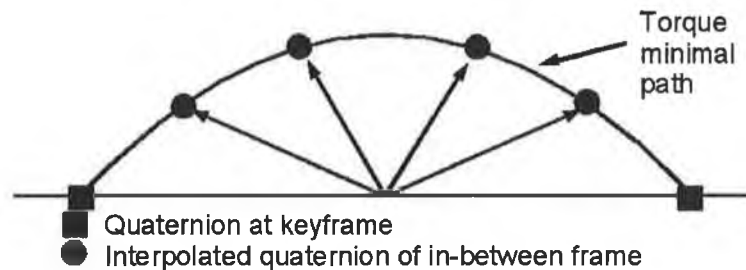


Figure 5.10: SLERP - the points on the curve are evenly spaced.

to high frame rates (60 frames per second and greater). This is a factor in the sampling rate used. With a low sampling rate (less than 10 Hertz) this acceleration and de-acceleration of limbs will start to become visible. While the sampling theorem may evaluate to give a lower sampling rate, it does not take this into account.

If SLERP does not suffer the disadvantage of this varying velocity, why not use it instead? Quaternion LERP is commutative, quaternion SLERP isn't. This commutative property is convenient when blending several animations, as such a blend will involve interpolating between several quaternions for each bone. Using SLERP would require a system to have the blending animations to be in a certain order. LERP is also cheaper to compute than SLERP, though this is to be expected given the nature of the operations [17].

5.3.1 Other Sampling Rate Constraints

There are a number of other system constraints that effect the sampling rate. These are discussed in more detail in Chapter 6, the implementation chapter. They include:

Bending work Part of the timewarping algorithm involves finding the angle between three points in a signal. As such, at least three samples are required, with more samples allowing a better time warp.

Wide spacing of points In the timewarping implementation, an Eulerian angle is converted into three 2D points, with the x value being the time of the Eulerian point in the animation signal, and the y value being either the x, y or z value of the same

Eulerian point. As the y values are in radians, they will be between $\pm\pi$, and will often be close to 0. If the sampling rate forces the x values to be far apart relative to the near 0 values of the y components, any attempt to determine the angle between 3 points will return a value of pi.

Low Pass Filtering To reduce noise in the resulting time warped signal, a low pass filter is used to smooth out the signal. The filter used has a kernel width of five, and so to be effective requires more than five samples, with 15 samples giving acceptable results.

5.4 Conclusion

As will be demonstrated in this thesis, regarding sampling animations with a view to warping, the sampling rate is not as important as the DSP literature would suggest. Using floating point variables to hold samples gives a very high degree of resolution, certainly enough that quantization noise is not noticeable in the finished time warp. The main constraints on the sampling rate are not those imposed by the sampling theorem, but instead, those imposed by the timewarping algorithm, and from the choice of LERP or SLERP for quaternion interpolation. In the next chapter the motion warping implementation is described.

lyit | Institiúid Teicneolaíochta Leitir Ceanainn
Letterkenny Institute of Technology

Chapter 6

Implementation

6.1 Introduction

This chapter discusses an algorithm for implementing animation warping - that is, taking an animation and a pose, and automatically having the animation meet the pose in a realistic manner.

The implementation is in C++, and works in conjunction with Instinct Technology's Instinct Engine. This engine handles importing animations and poses from 3D Studio Max using Instinct Technology's 3D Studio Max exporter. The engine also handles user input at run time as well as the graphics required to display animations in a suitable environment. Throughout the chapter, references are made to the code used in the implementation. At the core is the CMorphData class, used to hold all the various incarnations of the animations involved in the program. Strictly speaking, it's more of a structure than a class, as all of its member variables are declared to be public for ease of access.

The algorithm follows the approach of Bruderlin and Williams in [6], with regard to using a low pass filter to alter an animation, as well as timewarping. Before that however, it is necessary to convert the animation into a format that fits with the approach in [6].

Institiúid Teicneolaíochta Leitir Ceanainn
Letterkenny Institute of Technology

6.2 Sampling an Animation

The Instinct engine works by creating entities and assigning properties to these entities. These entities include lights, sounds and user input along with any other object involved in the environment used at run time. The stickman used for displaying animations is one such entity, and the various animations and poses are some of the stickman's properties.

There are three animation properties of the stickman: the bones, the walk animation and the pose animation. The bones property contains information regarding the length of each bone and the order in which they are connected. The walk animation property contains keyframes for the animation. These keyframes consist of a series of quaternion values - holding the local rotation of each bone for that keyframe. There is also a time associated with each keyframe. Lastly, the pose property is similar to the walk property, in that it contains the quaternion rotations for each bone. However, there is no need to have an associated time, as the pose is a single keyframe.

In order to get the quaternion rotations of an animation, the time of the animation must be set to give the bone rotations at that time. If the time is greater than the length of the animation, it will loop - though generally a function is called to get the length of the animation and this is used as the exit point for a For-Loop. With the time set, the rotations can be accessed on a bone by bone basis. This will give one frame of the animation and is the process by which a sample of the animation is acquired.

6.2.1 Why is Sampling Necessary?

Although the animation is already loaded into the Instinct Engine, it is not in a format that lends itself to motion warping. The main reasons for this is that the four elements of a quaternion are not independent of each other[1] meaning standard shifting and multiplication operations cannot be applied to the components of the samples. Eulerian rotations are

[1]One simple way to help understand this is by noting that as the quaternions are only used to give the rotation of a bone, all the quaternions are unit quaternions meaning their length is 1. If one component of the quaternion increases, another must decrease to maintain the unit length.

Itt | Institiúid Teicneolaíochta Ceatharlach / Letterkenny Institute of Technology

much better suited to the operations required for motion warping as the three values are independent of each other. However, in the conversion from quaternions, successive Eulerian rotations in an animation are not independent of each other. This is discussed in greater detail in section 6.3.

Sampling the entire walk animation leads to an Eulerian version of the animation held locally in the motion warping application as opposed to as a property of the stickman.

6.2.2 Finding a Sampling Rate that Works

The choice of a sampling rate has an impact on various aspects of the motion warping development. Picking a low sampling rate, for example 10 Hertz, will be quicker to compute and will use less memory. However, it may lead to noise in the animation as well as a visible acceleration in the bones of the character for the frames made from interpolating between the samples, see Figure 6.1. On the other hand, picking a high sampling rate, greater than 20 Hertz, will slow the performance of the warp, but will produce a cleaner, smoother result, see Figure 6.2.

Institiúid Teicneolaíochta Leitir Ceanainn
Letterkenny Institute of Technology

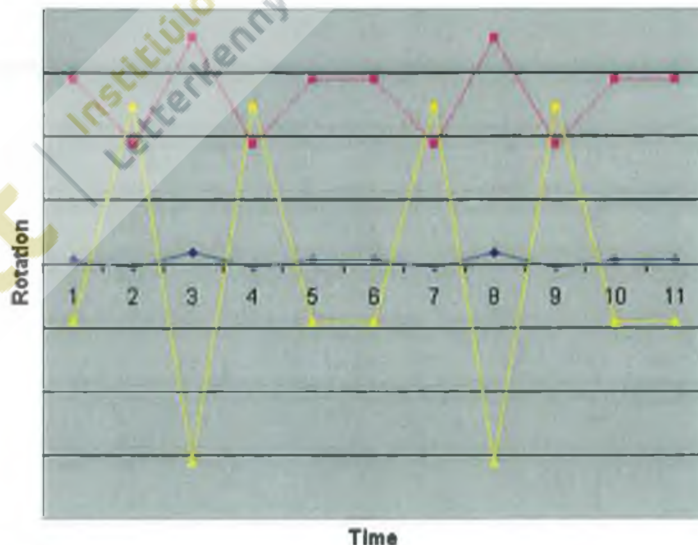


Figure 6.1: A signal produced using a low sampling rate.

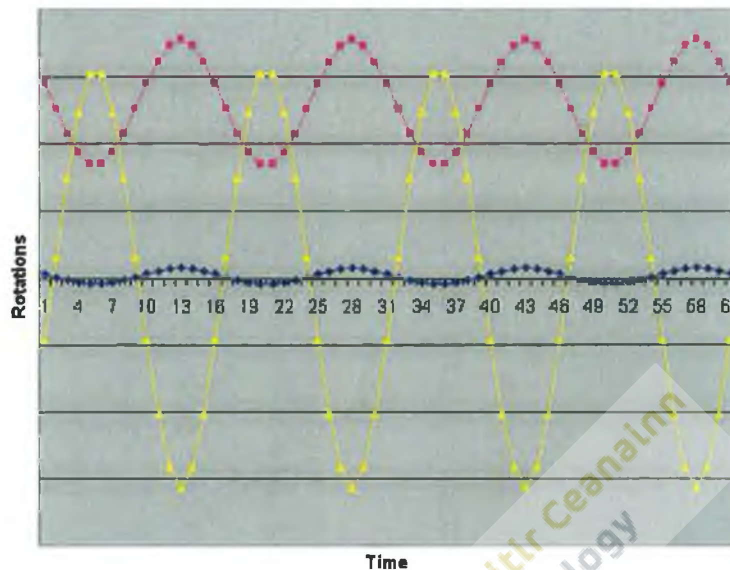


Figure 6.2: A signal produced using a high sampling rate.

The difference can be seen by graphing the rotations for a bone on an axis. The more samples the smoother the signal. The smoother the signal, the less noise in the resulting animation.

Using a sampling rate of 15 Hertz is almost a good compromise, certainly the workload is reduced, but the output is jerky. To rectify this, the warped animation is passed through a low pass filter to smooth out the result and give a much more pleasant warp. For more on this filter, see section 6.4.

lyit Institiúid Teicneolaíochta Leitir Ceanainn Letterkenny Institute of Technology

6.2.3 Code Reference

The sampling function is in the CWarping class and is called SampleAnimation. It takes a CMorphData object and an animation as parameters. The resulting sampled animation is stored as quaternions in the m_pBoneSampleArray. The pRotKeys hold the quaternion rotations while the pPosKeys hold the vector positions. The time of each sample is stored for both.

6.3 Converting a Quaternion Animation to an Eulerian Animation

As quaternions and Eulerian angles are both used to give angular displacement, it is possible to convert from one format to the other. To convert from a quaternion to an Eulerian representation the following equations are used:

$$pitch = \arcsin(-2(yz + wx)) \tag{6.1}$$

$$heading = \arctan 2(xz - wy, 1/2 - x^2 - y^2) \tag{6.2}$$

$$bank = \arctan 2(xy - wz, 1/2 - x^2 - z^2) \tag{6.3}$$

However, when converting, it should be noted that if the Eulerian rotation has entered Gimbal lock, its bank should be set to 0, and the following equation is used for heading:

$$heading = \arctan 2(-xz - wy, 1/2 - y^2 - z^2) \tag{6.4}$$

6.3.1 Using this Conversion with an Animation

A property of quaternions is that a negative quaternion and a positive quaternion will give the same animation. However, they will produce different Eulerian representations when converted using the equations in section 6.3. This problem occurs with the thigh bones of the skeleton.

As it is, converting the signal shown in Figure 6.3 back into quaternions will return the original quaternion signal and so the original walk animation will be reconstructed. However, when the signal of Figure 6.3 is smoothed out by a filter, or passed through the timewarping algorithm, the jump from $\simeq -\pi$ to $\simeq \pi$ becomes a problem. The desired walk signal for the left thigh is shown in Figure 6.4.

Here the signal oscillates around π, but if it oscillated around $-\pi$ the rotation is still the same. The jump from $-\pi$ to π is what causes the problem. The jump is removed by

Institiúid Teicneolaíochta Cheatharlach | Letterkenny Institute of Technology

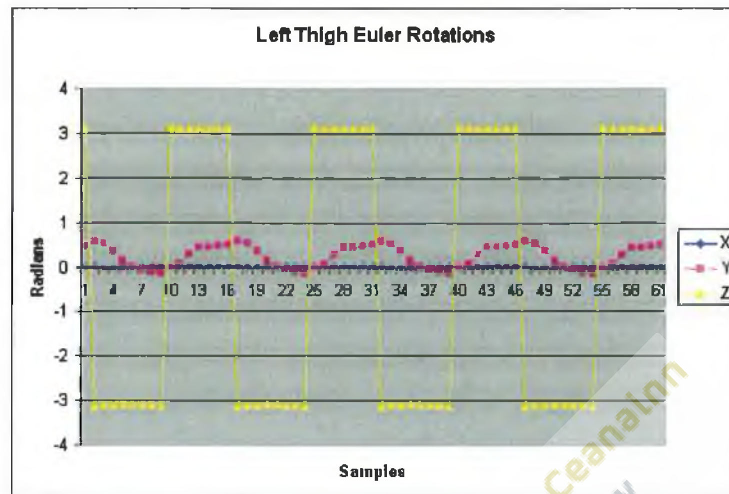


Figure 6.3: A quaternion to Eulerian conversion shown for the left thigh bone. There is a conversion discrepancy with the Z component of the Eulerian representation, shown in yellow. The right thigh bone Z rotations are similar.

comparing each sample with the previous sample and seeing if the difference is greater than $\simeq 2\pi$ radians. If it is, there is a jump in the signal. The jump is eliminated by multiplying the sample by -1.

6.3.2 Code Reference

The method CWarping::convertAnimToEuler handles converting the quaternion signals of an animation to Eulerian signals. It takes in a CMorphData object - which contains the quaternion version of the animation in m_pBoneSampleArray. It makes a conversion using the formula in 6.3 by calling the convertToEuler function. The resulting Eulerian signals are stored in m_pBandPassArray[0]. Then they are corrected for flips between $-\pi$ and π.

Institiúid Teicneolaíochta Leitir Ceanainn
Letterkenny Institute of Technology
lyit

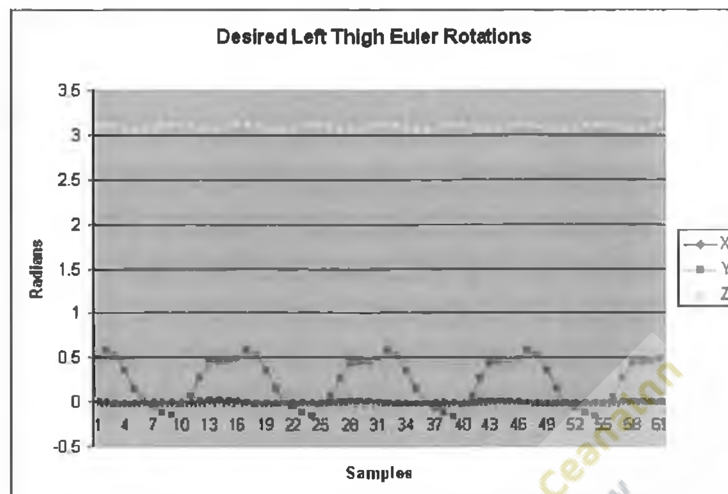


Figure 6.4: How the signal showing the Z rotations of the left thigh should look.

6.4 Filtering an animation

Following the approach in [6], a system was developed to incorporate multiresolution filtering. This can be thought of as an equaliser, as commonly used in audio processing, but used for animation in this case. With Bruderlin and Williams, the purpose of the filter is to be able to adjust animations before timewarping to give a better result. The goals of implementing such a filter for warping an animation with a pose are similar. However, the actual use of the filter in this project is to enhance the resulting warped animation after the timewarping.

In digital audio signals low-pass filters are used to remove high frequency noise in the signal. In the same vein, an animation can be passed through a low pass filter to remove high frequencies. This results in a level of detail being removed from the animation, with the resulting animation appearing somewhat restricted. The more detail each successive low pass filter removes, the more restricted the resulting animation.

In audio signals, the low frequencies contain the bass, the general sound, and the high

Institiúid Teicneolaíochta Cheatharlach / Institute of Technology Carlow

frequencies contain the treble, or the detail. Motion is somewhat similar - the middle frequencies contain the general motion, with the high frequencies containing the detail of the motion.

Passing the animation through successive low pass filters results in several low pass versions of the animation, each with less detail than the low pass before it. Subtracting consecutive low passes from each other gives band passes or wavelets (see Figure 6.5). These band passes can be summed to recreate the original animation. However, if a band pass is scaled before it is added, it will alter the animation. Scaling the lower pass bands positively exaggerates the general shape of the animation; scaling them in a negative direction will restrict the general shape of the animation. Along with this, scaling the higher band passes will cause the character to appear twitchy and nervous. This filtering method can be used to remove noise from the final timewarped signal.

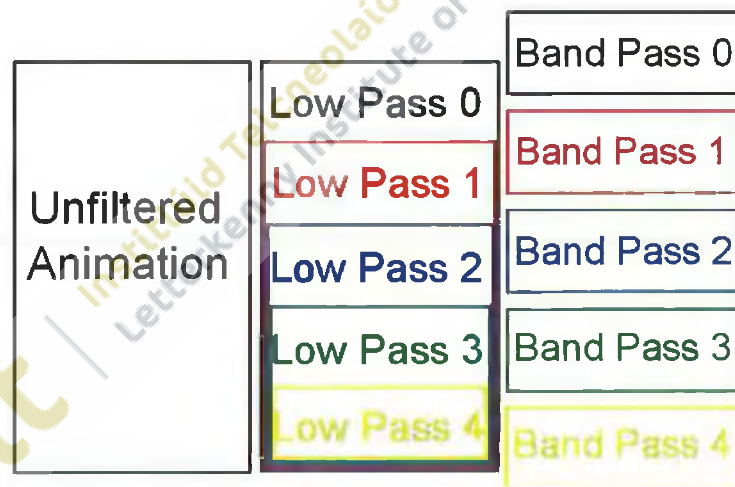


Figure 6.5: The filtering architecture.

6.4.1 Implementing a Low Pass Filter

As already discussed, in the electronics domain, a low pass filter is used to remove high frequencies before sampling a signal. The design of these filters centers around changing resistor values in an op-amp circuit. The values chosen for these resistors give the filter kernel. Its job is to decide how much from each part of a signal gets passed through the low pass filter. The kernel in the animation filter calculates how much of each sample in the animation gets passed to the lower pass bands. As suggested in [6], the filter kernel used is (c, b, a, b, c) with $a = 3/8$, $b = 1/4$ and $c = 1/16$.

The number of lowpass bands, n, is related to the number of samples, m in the animation signal as follows:

$$2^n \leq m \leq 2^{n+1}$$

The filter is convolved with the animation to give the first lowpass. For each successive low pass, the filter kernel is expanded by padding it with 0's between a, b and c:

$$kernel\ 0: \ (c, b, a, b, c)$$

$$kernel\ 1: \ (c, 0, b, 0, a, 0, b, 0, c)$$

$$kernel\ 2: \ (c, 0, 0, b, 0, 0, a, 0, 0, b, 0, 0, c) \ \ etc...$$

When the convolution requires a point that is outside the range of the m points of the animation, the end point is used. On the first iteration this will happen on the first and second convolution operations, as well as on the last and second last operations. An example of this filter in operation is shown in Figure 6.6

6.4.2 Filtering in Real Time

While the implementation of multiresolution filtering affords an ability to alter an animation outside of the normal channels (blending and timewarping), it didn't fit in with the real time goals of the motion warping procedure.

Institiúid Teicneolaíochta Leitir Ceanainn
Letterkenny Institute of Technology
lyit

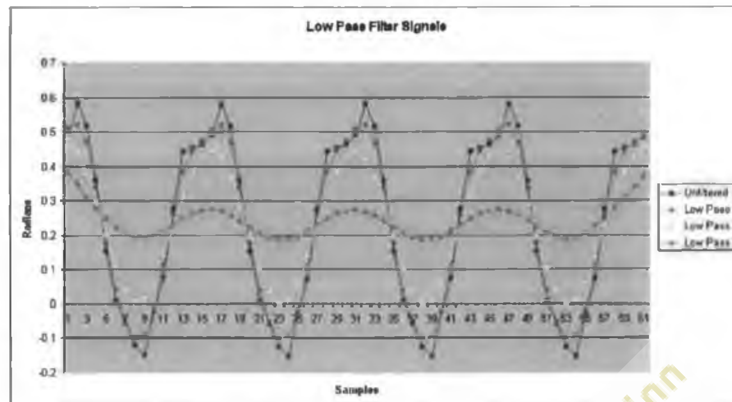


Figure 6.6: Four low passes of the Y component of a thigh bone. The signal gets smoother with each iteration.

In a similar illustration to that by Steven Collins of Havok (www.havok.com) when discussing the resources available for animation in a computer game at the Eurographics conference 2005 in Trinity College, Dublin, the following figures illustrate the number of operations required to implement filtering. Take the case of calculating a motion warp over 2 seconds; at 15Hz, this means 30 samples, and 4 frequency bands. With a kernel width of 5, there are 5 additions needed for each value in each frequency band. So 4 frequency bands, with 30 samples each, with 5 additions per sample gives 600 operations. But then there are 3 values in each sample, as the samples are Eulerian, giving 1800 operations. On their own, this number of operations is not something that would generally cause any problem. But to play a warped animation, it's ideal that the whole animation be calculated and stored in memory as early as possible, so the frames that come from interpolations between keyframes can be calculated at run time - a calculation time of around 5 frames being preferred. Running at 60fps this means a calculation time of $0.0166 * 5 = 0.0833$seconds. The time warping has to be carried out in this same time - it's also an expensive operation, as highlighted in section 6.5.2. 0.0833 seconds is not a great deal of time in the context of a next generation computer game when all the other processes in a game are taken into

consideration as well. This however, is both machine and game dependent.

Another point about multiresolution filtering is that, while it does afford the ability to alter animations when using timewarping and blending, it seems this property isn't needed. The results from filtering appear to be no better than putting the animation and pose in to the timewarp solution[2]. In situations where the mix of timewarping and blending fail to give a good motion warp, this animation altering property provides another avenue to explore.

6.4.3 Code Reference

The filter function, CWarping::calculateLowPass, takes a CMorphData object. It works with the m_pSignalArray, originally using m_pSignalArray[0], as this is where the Eulerian version of the animation is stored. The CMorphData constructor calculates the number of low passes from the number of samples in the animation. This is held in CMorphData.m_noFrequencyBands. The m_pSignalArray uses m_noFrequencyBands to initialize enough memory to hold all the lowpass bands. When the function finishes, m_pSignalArray[0] holds and unfiltered Eulerian version of the animation, while m_pSignalArray[m_noFrequencyBands - 1] holds the lowest lowpass version of the animation.

The CWarping::eulerBandPass function will subtract successive lowpass bands to create bandpasses. It takes a CMorphData object which has m_pBandPassArray initialized in the same way as m_pSignalArray, with enough memory to hold the bandpasses. At this point the band passes can be scaled to alter the animation.

The bandpasses are added using CWarping::eulerSumPassBands. This initializes the

[2]In fact filtering doesn't suit a pose. Filtering removes detail and restrains the motion of an animation - but with a pose there is no detail to remove or motion to restrain. The result of this is the highest pass band contains the pose, with the remaining pass bands all evaluating to 0. To get round this during development, instead of a pose, an animation made from a linear interpolation of the pose and the animation was used and while this could be filtered it had inherent errors making it a bad animation to timewarp with.

Institiúid Teicneolaíochta Cheatharlach Ceanainn
Letterkenny Institute of Technology

CWarping.m_pEDisplayArray. This is an array to hold an Eulerian version of the recombined animation. Calling CWarping.fillQDisplayArray will convert m_pEDisplayArray from Eulerian values to quaternion values and store the result in CWarping.m_pQDisplayArray. This array is accessed when rotations for the animations are required to display the animation on screen.

6.5 Time Warping an Animation

Blending two animations to produce a third animation sounds like an appropriate solution to the problem of creating a lot of separate animations. For instance, why create a jog animation when a blend of a run and a walk will give a jog? To discuss this question, it's first necessary to define what a blend is in terms of animation and illustrate the difference between a blend and a warp.

A blend is taking a frame of one animation and combining it with a frame from a second animation to create a frame of a third animation. The problem with this is if the second animation doesn't synchronize with the first animation regarding the general motion of the limbs, the result can be very restricted or static. A good example is that of foot plants, where the walking/running combination can cancel each other out resulting in a still pose for the character's legs (if the running character has a leg at the highest point of its motion while the walk animation has a leg on the ground, the blend gives the in between, resulting in a hovering motion). The character can then be seen to slide across the ground with no walking motion. What's needed is a different blend that first of all synchronizes the two animations. This is called timewarping. The goal of timewarping is to put one animation into a suitable position before blending to give results that won't be 'canceled out'.

Institiúid Teicneolaíochta Leitir Ceanainn | Letterkenny Institute of Technology

6.5.1 Timewarping Algorithms

Timewarping is not just restricted to the field of animation. It's used in speech recognition - where a word may be pronounced at a slower rate than a test case and so the word is timewarped to a form where it may match the test case. Timewarping of this form is documented in [9]. This approach involves looking at the audio signal in the frequency domain. As noted when discussing the DFT in chapter 4, such a conversion adds a lot of computational expense, an undesirable trait when trying to implement real-time timewarping.

A different approach to time warping, and one that deals with an animation and a pose (as opposed to two animations) is discussed in the Motion Warping paper by Witkin and Popovic [10]. However, with this approach, the timewarping shifts are given beforehand, with the focus being on a blend between an animation and pose - the timewarping of the animation is already calculated.

In [6], there is a section discussing time warping. It was chosen as a blueprint to follow as it didn't require any run time input from a user or any frequency domain conversion.

6.5.2 The Implemented Time Warping Algorithm

As in [6], this approach involves looking at the signals of 2 animations and treating the signals as shaped pieces of wire. The goal is to find the least work required to match them in shape by simulating the physical work required to bend and stretch the pieces of wire. Using this on Eulerian animation signals, each time warp component signal consists of (x, y) pairs, where x is the time of the sample and y is one of either the bank, pitch or heading. Each signal with regard to pitch, bank and heading is timewarped separately.

A 'grid' is dynamically programmed to work out the least cost combination of a signal from the walk animation with the pose. In the diagram (see Figure 6.7) the pose is plotted across the top with each point across the grid holding a value of the pose. The walk animation is plotted in a similar fashion down the side. The walk and pose can be switched with no impact on the result. Starting at $(0, 0)$ (the top left corner) a 'work' or 'cost' value

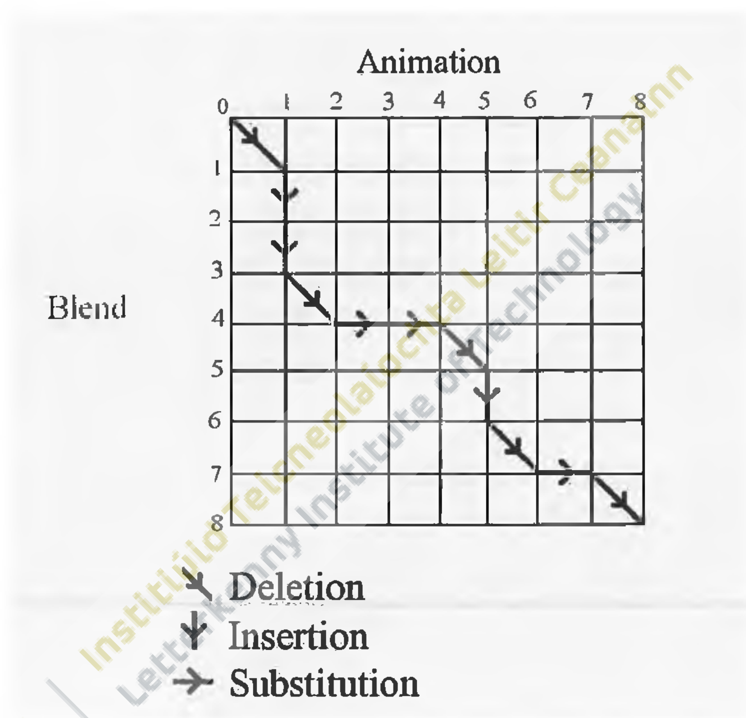
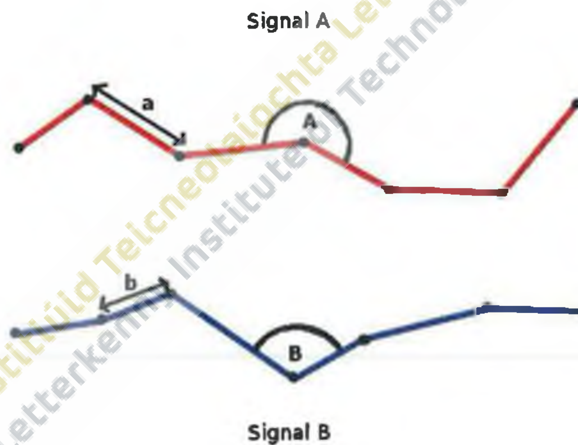


Figure 6.7: A graphical representation of the grid created by dynamic programming to implement timewarping.

lyit | Institiúid Teicneolaíochta Leitir Ceanainn
Letterkenny Institute of Technology

is assigned to each node. $(0,0)$ has a value of 0. The work value in the successive nodes is a combination of a bending work value and a stretching work value. This combination is a weighted sum, changing the weights can change work values, in turn changing the path through the grid and hence the time warp. The path through the grid must follow certain rules: three different types of moves are allowed - across, down, or diagonal. A move across cannot be followed by a move down, and a move down cannot be followed by a move across, without first having a diagonal move in between.

6.5.3 Bending Work



lyit | Institiúid Teicneolaíochta Leitir Ceanainn
Letterkenny Institute of Technology

Figure 6.8: 2 signals illustrating what angles and lengths are compared to calculate a work value for a node on the grid.

The bending work value corresponds to the difference in angle between two successive line segments on one signal (animation signal) with two successive line segments on the other signal (pose signal). As such, this requires three points, so the bending element of the work value of a node does not come into effect on the grid until elements $(2, y)$ and $(x, 2)$.

In [6], the timewarping algorithm is based on methods documented in a paper by Sederberg and Greenwood [11] on 2D shape blending. Where Sederberg discusses taking the angle between 2 line segments, he does so with the aim of morphing from one signal to the other. This leads to setting up a Bézier spline between the 3 points, so an interpolation value can be used to show how far along this spline the shape blend has gone at a point. Implementing such a spline puts a drain on resources, and isn't strictly necessary, as the only values needed are those when the interpolation factor is 0 and 1, ie. the start and end of the spline. Finding the difference between the angles when the interpolation factor is 1 and 0, will give the angle between the 2 line segments of a signal. Doing this for both signals allows a comparison of their angles.

Instead of implementing this overly complicated method, the 2 line segments of each signal are brought into a local space. One segment is rotated to lie on the x-axis. The second segment is rotated accordingly. Then depending on the sector the end point of this second segment lies, the angle between it and the x axis is calculated. This is carried out for both signals, allowing a comparison to be made between the angles of both signals. This is illustrated in Figures ??, 6.10 and 6.11.

When in use it was discovered that this method returned 180 degrees the majority of the time, which was odd, as it could be seen from the signals that the angles between points were not 180 degrees. The reason for this lay in the values involved. To illustrate why, suppose the sampling rate is 10Hz, meaning the samples are spaced 0.1 seconds apart. The natural unit for angles in C++ is radians, giving angle values a range between -2π and 2π, or -3.14 to 3.14. Quite often a bone won't have a great deal of motion in this time, perhaps oscillating in a range of -0.0001 to 0.0001 radians (of course, it can be much bigger, but such small rotations are common). Getting the difference in angle between 2 line segments with such values will always approximate 180 degrees as the points are relatively far apart, given their magnitude.

One possible solution may be to sample the signals at a higher rate, meaning the points won't be spaced so far apart and so their magnitude would have a greater effect when

Institiúid Teicneolaíochta Leitir Ceanainn / Letterkenny Institute of Technology

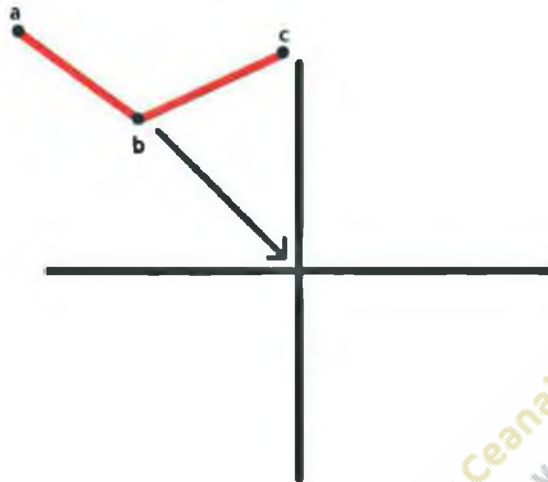


Figure 6.9: Translate the 2 segments so they sit on the origin.

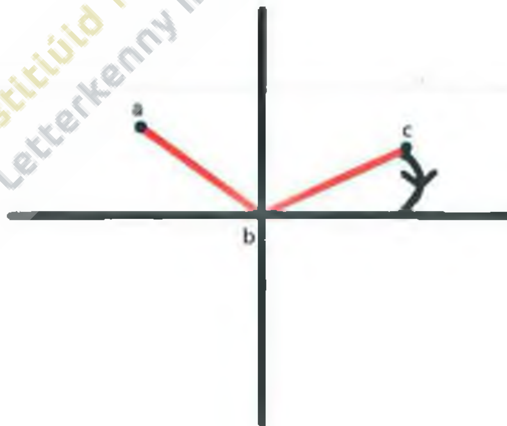


Figure 6.10: Rotate the second segment so it lies on the x-axis. Rotate the second segment by the same amount to preserve the angle.

lyit | Institiúid Teicneolaíochta Leitir Ceanainn
Letterkenny Institute of Technology

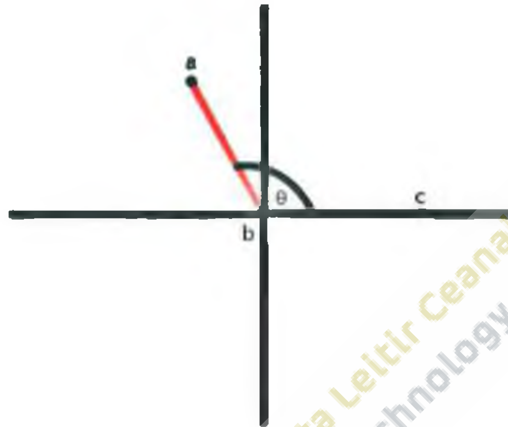


Figure 6.11: Calculate the angle Θ using trigonometry. If necessary, adjust it to account for the quadrant in which point 'a' lies.

lyit | Institiúid Teicneolaíochta Leitir Ceanainn
Letterkenny Institute of Technology

Figure 6.12: As the sampling rate decreases the angle at y2 approaches a limit of 180 degrees.

calculating angles. This is an undesirable solution for two reasons. The most obvious is that it will mean more samples, in turn meaning more processing and hence a slower algorithm. The second reason is that as the sampling rate increases the angular difference between two points will decrease (taken to the limit will lead to every consecutive pair of points being co-linear), leading back to the case where all the angles computed will return 180 degrees.

A second solution, and indeed the solution used, is to scale up the rotation values before calculating an angle. If the angle of rotation of the 3 points used (3 points to define 2 segments) has a magnitude of less than 1, the values are all scaled up by increasing powers of 10 until one of the 3 points (or possibly 2 or all 3 points) has a value greater than 1. This resolves the problem of the points being relatively far apart by increasing them so they can influence the angle between the 2 segments. Changing the values of the data on which the time warp is based may give the impression that the resulting timewarp will be deformed. However, if a point is scaled up by 1000, it can be observed that all the other points in the signal will require a similar scale value, meaning the bending work of a node hasn't changed relative to all the other nodes[3].

Having an angle for each signal, these are used to calculate how much work is needed to bend one signal to the shape of the other signal. In [11] the following formula is used:

$$k_b(\Delta\theta + m_b\Delta\theta^*)^{e_b} \tag{6.5}$$

where k_b is a constant to indicate bending stiffness, $\Delta\theta$ is the change in angle that the point must undergo to match up with the new shape, $m_b\Delta\theta^*$ is an additional angle to be added if the Bézier spline was not monotonic and lastly, e_b is a user definable constant to do with elasticity. As the bending application in this case isn't shape blending where the positions in between the start and end are important, the formula can be reduced significantly. As the Bézier spline has been removed, there is no need to account for monotonicity, so that can go. k_b and e_b are both constants and are set at the start of the bending function. In the case

[3]There is normally a certain uniformity in all the samples of a given signal, if one is $x^(- 5)$ the rest of the samples will also be $x^(- 5)$ where x is a variable within a signal.

Institiúid Teicneolaíochta Leitir Ceanainn
Letterkenny Institute of Technology

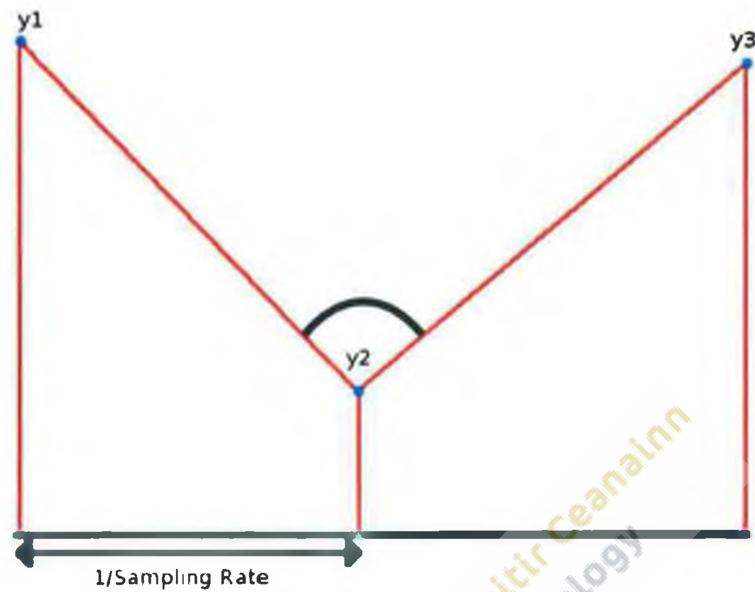


Figure 6.13: When the y values are scaled up they have a greater bearing on the angle between the 2 line segments.

of getting a bending value where there aren't 3 points to make 2 segments, but one vertex or two vertices, the angle is taken to be 180 degrees.

Calculating the stretching component of a node's work value is much more straightforward. It's based on the work required to stretch one segment of a signal to the length of the corresponding segment on the other signal. The length of each segment is calculated using the formula for the distance between two points. If there is only one vertex the length of the segment is 0. Again, in [11] the following formula is used:

$$k_s \frac{|L_1 - L_0|^{e_s}}{(1 - c_s)min(L_0, L_1) + c_s max(L_0, L_1)} \tag{6.6}$$

where k_s is a constant involving a theoretical cross sectional area of the wire - it is set to 1, but can be altered to change the influence of stretching in the work value of a node. L_0 and L_1 are the lengths of each of the segments, c_s is a constant that imposes a penalty when one of the segments has a length of 0. e_s is an elasticity constant similar to e_b is with bending. The bending and stretching values are added to give a possible work value for a node.

6.6 The Grid Revisited

It was stated previously in section 6.5.2 that a work value is assigned to each node in the grid. As the aim of the grid is to find the lowest cost route from the top left corner to the bottom right corner, it is necessary to ensure that the work value of each node is the smallest work value available to that node. The combination of moves to access a node (down, across or diagonal) leads to 7 possible combinations to get the least work for that node - across across, across diagonal, diagonal across, diagonal diagonal, diagonal down, down diagonal and down down (remember, it takes 3 points to get the bending work). Obviously, there will be some restrictions, for example, if the node in question is at the top or the side, it can only be accessed by going across (top), or down (side). The possible costs for a node 'A' are evaluated for each possible route to A and the smallest cost is selected. This cost is then added to the cost of the middle node of the three nodes used to get the bending part of the cost for 'A'. In the implementation, this cost is worked out when the grid is being created dynamically.

When each node in the grid has been assigned a cost, the grid is traversed backwards from the point $(x-1,\ y-1)$ to $(0,0)$, where x is the number of samples of the pose and y is the number of samples of the animation[4]. The route taken to traverse the grid controls the timewarp. The algorithm for finding this route is as follows:

- The CurrentNode is the node on the grid currently in use.
- Start by setting the current node to $(x-1, y-1)$ - the last node on the grid.
- Push the CurrentNode to the OptimalPath list.
- Repeat:

  {

  - If CurrentNode.x or CurrentNode.y has depleted to 0, force the next node along the side or across the top.

[4] The -1 is because the arrays used to hold the signals run from 0 to x-1/y-1, not from 1-x/y

- If this is the first move, or if the previous move was diagonal check the cost values of the following three points:
  $(CurrentNode.x - 1, CurrentNode.y)$,
  $(CurrentNode.x - 1, CurrentNode.y - 1)$,
  $(CurrentNode.x, CurrentNode.y - 1)$
- If the previous move was across, check the cost values of the following 2 points:
  $(CurrentNode.x - 1, CurrentNode.y)$,
  $(CurrentNode.x - 1, CurrentNode.y - 1)$,
- If the previous move was up, check the cost values of the following 2 points:
  $(CurrentNode.x - 1, CurrentNode.y - 1)$,
  $(CurrentNode.x, CurrentNode.y - 1)$
- Set the CurrentNode to whichever of the nodes has the lowest cost.
- Push the CurrentNode to the OptimalPath.

}
while the CurrentNode is not$(0, 0)$

The result of this is the OptimalPath list contains a series of nodes that will trace out the path through the grid. An example of such a path is shown in Figure 6.14.

Figure 6.14: A screengrab from excel where the values of a grid were printed. The least cost path through the grid is shown in green.

The purpose of this path through the grid is it gives a plan for how to construct the timewarped signal. If there is a diagonal move on the path, the corresponding point on the signal being timewarped is stored in a separate array.

LyIT Institiúid Teicneolaíochta Leitir Ceanainn Letterkenny Institute of Technology

If it's an across move, the point to be stored is an average of the pose points corresponding to each consecutive across move.

Finally, on a downwards move, the points are created by taking a B-spline round the pose point - i.e. the point immediately before the pose, the relevant point on the pose, and the point immediately after that. A value is extracted from the B-spline for every downwards move[5] and stored for the timewarped signal. This is shown in Figure 6.15. For more detail on B-splines see [16]

The signal resulting from this timewarp contains times for each sample. However, these are not the times used, instead, the original sample times are mapped to the new samples, thus timewarping the blended signal.

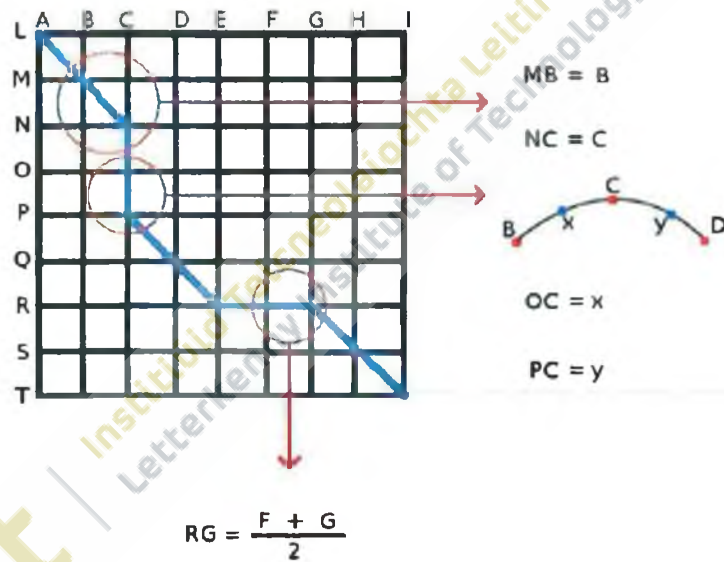


Figure 6.15: The diagram shows a 50/50 merge of 2 points on a diagonal move, how a B-Spline is used when moving down, and an average when moving across. It should be noted that only the y values of the signals are involved in the numeric operations shown.

[5]Creating a B-spline from animation points suffers from the same 'small numbers' problem as getting the angle between two segments of a signal. This leads to the spline always approximating a straight line, when often that it approximate a curve. It is resolved in the same way as it's solved in the bending function.

6.6.1 Code Reference

The CSederberg class (named after the author of [11]), located in the graph.cpp file, computes the time warp. It is passed five parameters, three CMorphData objects - referred to locally as a, b, and result, and a start time (morphStartTime) and an end time (morphEndTime) for the timewarp.

As the grid will compare all the samples in 'a' with all the samples in 'b', some memory allocation is required. Firstly m_NoAcrossPoints holds the number of samples in 'a' and m_NoDownPoints holds the number of samples in 'b'. These are equal in value, but there is no guarantee this will always be the case.

The CSederberg class is able to timewarp the band passes of two animations. This is implemented in a series of nested loops. The outer loop cycles through the band passes. Inside this, the next loop cycles through the x, y and z strands of each band pass. Inside this again is a loop on each bone in the character. Lastly, inside this is a loop on the samples in each signal.

In order to timewarp each pair of corresponding signals, they are first copied into the m_pAcross and m_pDown arrays. The CSederberg::findOptimalPath function is called to find a path through the grid. Before calling CSederberg::plotPath to draw up the grid, two PATH nodes are created. A PATH node is a structure to hold the information required by each node on the grid. It can be thought of as a doubly linked list node, as it has a pointer to its parent - PATH * pParent, and child PATH * pNextNode. Also in a PATH node are:

bool north Used when recording the path through the grid. Set to true if the preceding node on the path lies north of this node.

bool west Used when recording the path through the grid. Set to true if the preceding node on the path lies west of this node.

float cost Used to store the cost associated with the node

MYPOINT coordinates Each node has coordinates to make it possible to locate that node

in the grid without having to search through all the elements in the grid.

MYPOINT I Holds the time (I.x) and rotation (I.y) values of the 'a' signal plotted across the top of the grid.

MYPOINT J Holds the time (J.x) and rotation (J.y) values of the 'b' signal plotted down the side of the gird.

The parent node, m_pGridPath is set with invalid information to differentiate it from the regular path nodes. Its pNextNode pointer points to m_pGridEnd. The grid will be inserted between these two PATH nodes.

Creating the grid as a linked list is relatively straightforward, but because the elements of a linked list do not occupy a contiguous block of memory, it is not possible to refer to nodes by their coordinates. Coordinate reference for nodes is desirable as without it some form of searching algorithm is needed, and these are generally slow. Taking advantage of knowing how many samples will be plotted along the top of the grid, and down the side of the grid, a contiguous block of memory can be allocated to hold a grid of size $(acrossPoints * downPoints)$ - this is done in CSederberg::plotPath. Building on this, the function CSederberg::findNode takes the coordinates of a node, multiply's the y value by the number of across points and then adds x. From this value the address of the node (x, y) is returned.

CSederberg::plotPath builds the grid a node at a time, going across in rows. It sets the I and J values and the coordinates of each node, before inserting it in the grid by setting its pNextNode to point to m_pGridEnd, and its pParent to point to m_pGridEnd.pParent.pParent, i.e. the last node currently in the grid before the node being inserted. Lastly, CSederberg::calculateWork is called on the new node to get the cost value for that node.

CSederberg::calculateWork has 2 variables, bendWeight and stretchWeight that can be set to ensure the results from either the bendingWork function or the stretchingWork function are not so great as to make the other insignificant. The calculateWork method looks at the position of the node in the grid and then calculates the work for that node from the

Institiúid Teicneolaíochta Leitir Ceanainn
Letterkenny Institute of Technology

information in the node and the nodes around it. It will then set the north and west elements in the node to indicate the cheapest node of the three immediate nodes before it.

Returning to the plotPath function and starting with the last node in the grid - bottom right, the coordinates of this node are pushed onto the m_vOptimalPath vector (in this instance, the vector is an instance of the STL vector class, of type MYPOINT). Looking at the north and west elements of each node pushed onto this vector indicates the next node to be pushed on to the vector. Thus plotPath plots a path from the bottom right corner of the grid to the start at the top left corner.

When plotPath has entered a path into the m_vOptimalPath, control returns to the findOptimalPath function. It steps backwards through the vector [6]. From the coordinates in the coordinate values in the vector, there are three possible cases:

A diagonal move In this case the shift value is added to the relevant point in the m_pDown array, placing the result in m_pPathResult vector.

One or more moves down A count is taken for all the successive moves down. A B-Spline is created using three points: m_pAcross[currentNode.coordinates.x - 1], m_pAcross[currentNode.coordinates.x] and m_pAcross[currentNode.coordinates.x + 1]. A point for every move down is calculated from the B-Spline.

One or move moves across The number of successive moves across are counted. For each move the y or rotation value of of that sample in m_pAcross recorded and the average of these points is returned.

Returning to the CSederberg constructor, the time warped signal is in the m_pPathResult vector. It is copied into the corresponding space in the CMorphData result object. The memory created for the grid is deallocated. The time values of the new timewarped signal

[6]The vector holds points starting with the last point on the path, so the first step on the path through the grid lies in the last element of the vector.

Institiúid Teicneolaíochta Leitir Ceanainn Letterkenny Institute of Technology

are copied over from one of the original signals. The timewarp process then repeats for the next bone etc.

6.7 Conclusion

In the next chapter the results of this work are considered and some general conclusions are drawn.

lyit | Institiúid Teicneolaíochta Leitir Ceanainn
Letterkenny Institute of Technology

Chapter 7

Results and Conclusions

7.1 Introduction

This chapter discusses how effectively the timewarping procedure works and how suited it is to warping a pose within a real time environment. To aid this discussion, graphs are used to illustrate results. In each case - unless otherwise stated - the graphs show the Eulerian rotations about the x-axis of the hip bone.

Institiúid Teicneolaíochta Leitir Ceanainn
Letterkenny Institute of Technology
lyit

7.2 Timewarping Results

As discussed in section 6.5, timewarping is necessary to synchronize two animations before blending them. Figure 7.1 shows two different animations, a run animation and a walk animation. It also shows a blend of the two animations. This blend is a 50/50 mix. It gives a reasonable result - the phase is similar, as is the amplitude.

A 50/50 blend will not give a good result if the animations are not synchronized first. In fact, if they are out of phase, it's possible they will cancel each other out, resulting in a lifeless pose. This is illustrated in Figure 7.2.

The yellow signal is the blend. It doesn't reflect either signal - it is out of phase with both the walk and run signals. Its amplitude is also less than the walk signal - which does

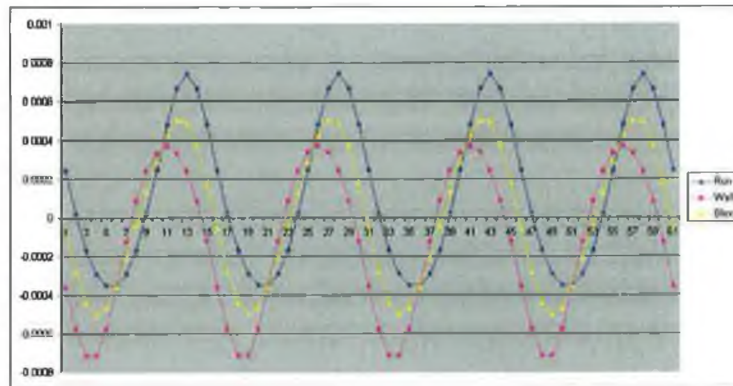


Figure 7.1: Blending a walk and a run that are slightly out of synchronization with each other.

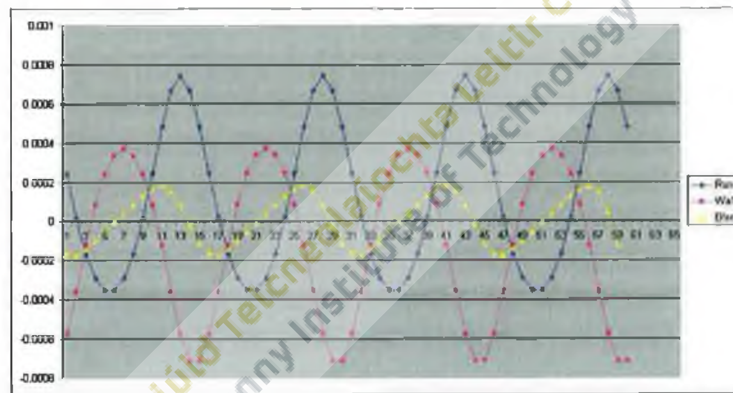


Figure 7.2: Blending a walk and a run that are out of synchronization with each other produces an un-useable result.

lyit Institiúid Teicneolaíochta Leitir Ceanainn
Letterkenny Institute of Technology

not serve to give the expected 'jog' motion, instead giving a constrained walk motion.

Timewarping synchronizes the two animations so they can be blended. The timewarping algorithm described in 6.5 produces such a synchronization, as shown in Figure 7.3. Here, the walk signal is timewarped so it synchronizes with the run signal. The blend of this timewarped walk and the run signal is shown in Figure 7.4 as the green signal. As expected for a jogging signal, it sits between the run and (timewarped) walk signals.

It is also possible to warp the run to synchronize with the walk, by swapping which animation is plotted across the top of the grid. The results are shown in Figure 7.5 and

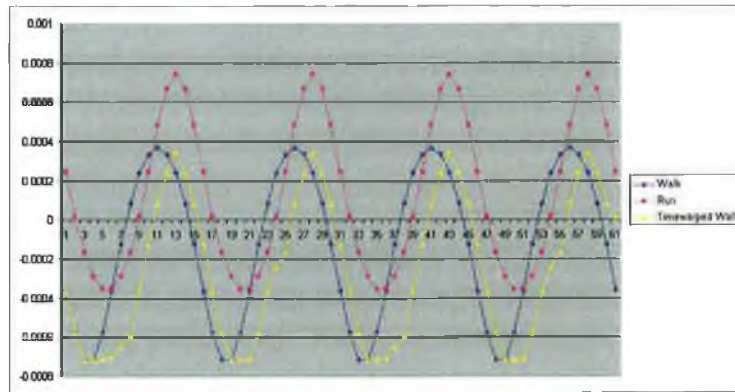


Figure 7.3: Timewarping a walk to synchronize with a run.

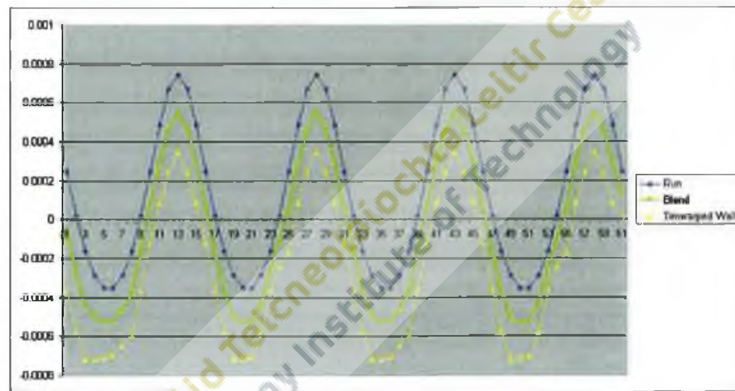


Figure 7.4: Blending a timewarped walk with a run to produce a jog.

lyit Institiúid Teicneolaíochta Leitir Ceanainn Letterkenny Institute of Technology

Figure 7.6.

The capacity to implement the timewarp of a run and a walk with the aim of getting a useable result on screen has not been considered in this project. This is because, up until now, there has only been a need to work with the rotations of bones to be able to successively warp a pose with an animation. To create a jog animation from warping a run and walk will involve the world-space position of the character. The world-space position of the character is related to their velocity, a velocity which will be different for a run and a walk. The velocity of a jog will be in between. Without anything to account for this, the resulting jog animation is played with the velocity of the walk animation, resulting in the

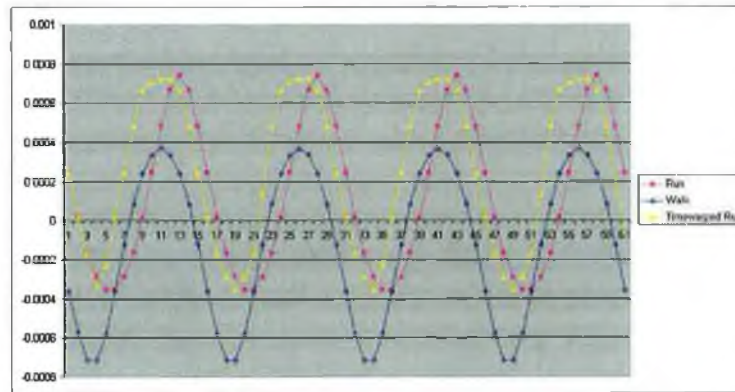


Figure 7.5: Timewarping a run to synchronize with a walk.

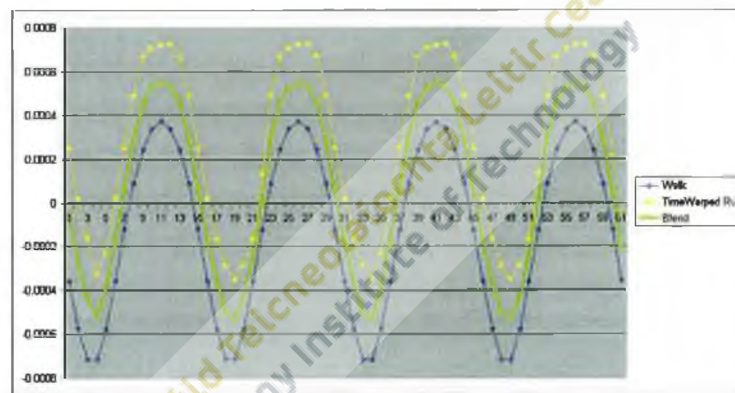


Figure 7.6: Blending a timewarped run with a walk to produce a jog.

lyit | Institiúid Teicneolaíochta Leitir Ceanainn | Letterkenny Institute of Technology

feet gliding across the ground.

7.2.1 Timewarping a pose

The signals from a pose are all flat lines when graphed over time - like a DC component of a current - they don't have any phase. This lack of phase doesn't fit with the idea of shifting a signal so it synchronizes with another signal. When a pose signal is shifted to the left or right, the result is the same signal. This is shown in Figure 7.7.

Progressing as before and assuming the pose has been timewarped, a 50/50 blend will

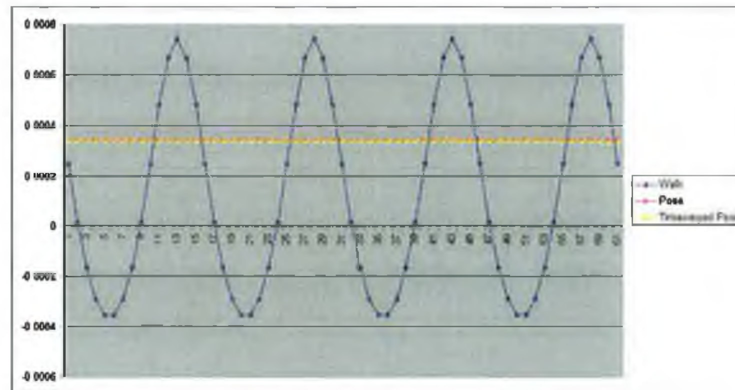


Figure 7.7: Timewarping a pose to fit a walk animation.

give the result shown in Figure 7.8. The blended signal has the correct phase, but its amplitude is significantly reduced. This leads to a constrained motion from the character, meaning the character never reaches the pose and at the same time does not move his legs enough to reproduce a useable walking motion. The flat nature of pose signals does not lend itself to 50/50 blends. A possible solution to this constrained issue is to filter the animation and scale up the bass bands to give the resulting motion a greater amplitude in its signals, hence increasing the motion of the character. But as discussed in section 6.4, filtering adds a significant overhead. A simpler, cheaper solution is discussed in the next section - section 7.3.

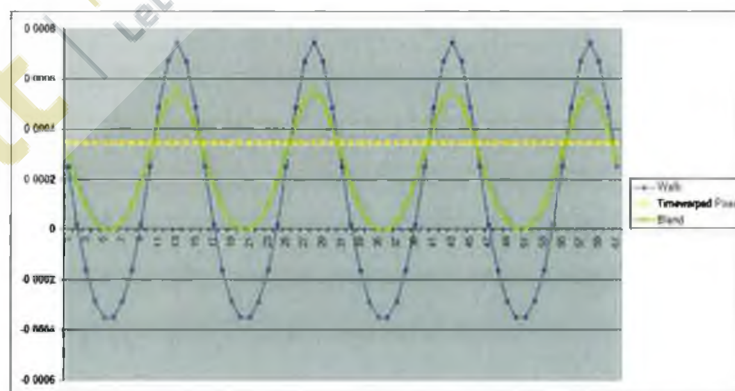


Figure 7.8: Blending a timewarped pose with a walk.

lyit Institiúid Teicneolaíochta Leitir Ceanainn Letterkenny Institute of Technology

7.3 Motion Warping

A 50/50 blend doesn't reach the pose or contain enough of a walking element to be really useful. The reason the pose is not met, or the walk isn't acceptable is because there isn't enough of each signal (pose and walk) in the resulting blended signal. To accommodate this, instead of a 50/50 blend, the walk signal is shifted to oscillate about the pose signal. This is shown in Figure 7.9, where the walk signal has been timewarped before shifting it about the pose.

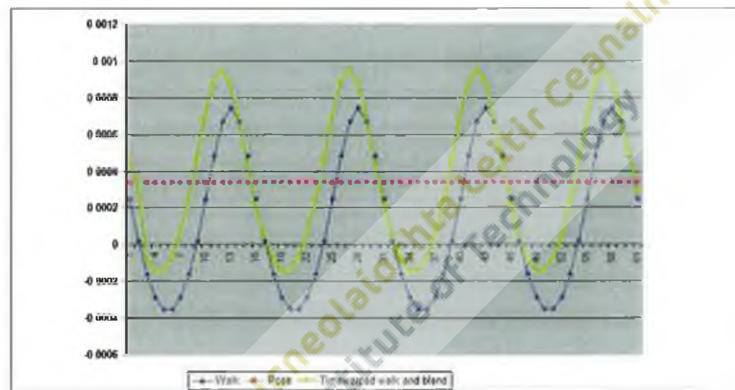


Figure 7.9: Blending a timewarped pose with a walk.

This shift makes the pose central to the resulting animation, which still retains the motion from the walk. The results from this are acceptable, with just a slight element of footskate [1] - introduced by the walk animation being timewarped, which could be fixed up with some inverse kinematics. It works for a variety of poses, for example stooping, crouching and raising the characters arms.

One point to note is, when a crouching pose is warped with a walk the resulting crouching walk animation is not planted on the ground. The position of the crouching walk comes straight from the walk animation, which assumes the character is upright. This could be solved by incorporating a positional strand of both the pose and animation into the warp.

[1]Footskate is where the characters foot 'skates' across the ground, instead of planting solidly

lyit Institiúid Teicneolaíochta Leitir Ceanainn Letterkenny Institute of Technology

However, as the goal is to produce something that may be used in the computer games industry, it is fair to assume a character controller will be used.

Normally, a character controller is a type of bounding box encapsulating the character, allowing the character to respond to the physics of the environment it is in, such as collisions, or falling off ledges. It can also be used to control the character's position in world space - part of which includes keeping the character on the ground.

7.4 Pose Specific Motion Warping

While the work presented so far produces reasonable results, it is too slow to implement in practice. However, observing that the aim of timewarping is to synchronize two animations, and that the DC nature of a pose cannot be synchronized in the normal sense, timewarping doesn't serve any particular purpose in this instance. Removing timewarping gives the result shown in Figure 7.10.

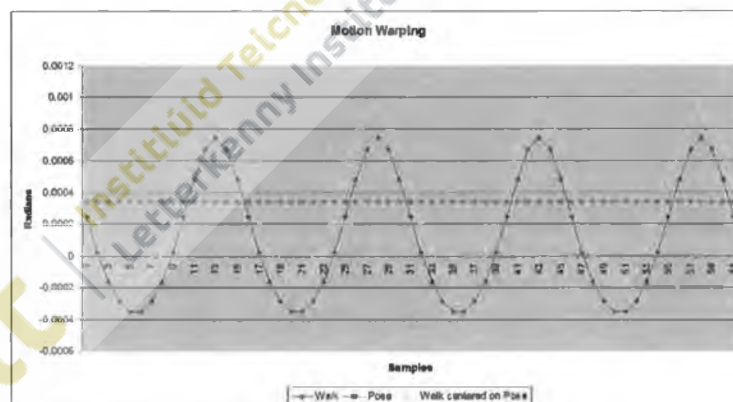


Figure 7.10: The walk signal is centered on the pose signal with no timewarping.

This produces a better result than timewarping and then shifting, as there is no footskate in the resulting animation. At the same time, the result has both the movement of the original walk, and the rotation needed to meet the pose. Removing the timewarping also removes most of the work required to warp the pose and the animation. Calculating the

lyit Institiúid Teicneolaíochta Leitir Ceanainn Letterkenny Institute of Technology

grid is quite expensive. As an illustration of this, at 35 bones per character, and with 3 signals per bone, 135 grids must be calculated. A 4 second interval with 15 samples per second, means $4 \times 15 \times 135 = 8100$ memory allocations. For each of these, a cost must be calculated. It can be seen that it's not a cheap method, especially when the goal is to have it completed in the space of 4-5 frames. Comparing this to doing a shift on a signal - get the average value of the walk signal, find the difference between the first point of the walk and pose signals and then add this difference to every point on the walk signal. There is no need to allocate temporary memory, work out the bending or stretching costs or find grid paths. This means the process ought to be cheap enough to develop for use in a real time environment.

7.5 Conclusion

While timewarping is necessary for warping two animations, it is not necessary when warping a pose and an animation. It offers no advantage over warping the two signals as described in section 7.4.

A possible improvement in performance could be achieved by removing the quaternion to Eulerian (and back) angle conversion.

A real time implementation of pose specific motion warping in a game can lead to more variations in character animations, without the need to create case specific animations before hand - just case specific poses. Instead of having all the obstacles arranged so that a character must crouch under at the same height to suit a single crouched walk animation, they can be placed at different heights (or even variable heights) with a pose attached to control the crouched walk to pass under the ledge.

Other applications can include emotional animations. Create a walk - create a sad pose, warp them to create a sad walk. Mix an enthusiastic pose with a walk to create an enthusiastic walk - and so on.

While the filtering was not used, due to its relatively high overhead regarding real time

lyit Institiúid Teicneolaíochta Leitir Ceanainn Letterkenny Institute of Technology

use, it may have applications in off line use. As an example, take a crowd at a football match. Having one 'hands up cheering' animation means every character in the crowd will be doing the same thing. So several different cheering animations are needed, to stop the crowd looking automated. However, if the 'hands up cheering' animation was filtered, it could be applied with different scale factors to different characters to make them cheer with different intensities.

7.6 Future Work

The animation/pose warping method described is a starting point to using animations not previously authored, at runtime. There are two main issues to consider when implementing this animation/pose approach. Firstly, how it is handled in a game, i.e. at what point does the control of a character switch from the player to the warp and will the warp retain control of the character until the warp is finished, or is it possible to break from a warp because of player input. This is an issue for a games AI.

One of the advantages of such an animation/pose warping system is that game levels don't have to bend to a set of predefined animations. The second issue with implementing this system in a game involves collisions with the world. Unless the poses are very carefully chosen, walking surfaces will have to remain mostly flat, to avoid visual glitches with foot placement. In a way, the game environment is once again bending to the constraints of the animation system. If an IK system was incorporated with the warping system, the dependency of the game environment on the animation system decreases. [4] describes a suitable IK system. The animation warping would occur, and then an IK pass would work out foot placements. The CCD IK system described by Welman in [19] is possibly suitable, as it deals with a bone structure. However, the Jacobian IK method described is slow to converge and as such, is not suited for use in a real time environment.

Institiúid Teicneolaíochta Leitir Ceanainn | Letterkenny Institute of Technology

Bibliography

[1] J. Weber, "Run-Time Skin Deformation", Game Developers Conference 2000, http://www.gamasutra.com/features/gdcarchive/2000/weber.doc; accessed January 15th, 2006.

[2] G. Maestri *Digital Character Animation 2*, vol. 1 - Essential Techniques, New Riders Publishing, 201 West 103rd Street, Indianapolis, 1999.

[3] A. Witkin and M. Kass, "Spacetime Constraints," *Computer Graphics*, vol. 22, no. 4, pp. 159–168, Aug 1988.

[4] S. Chung and J.K. Hahn, "Animation of Human Walking in Virtual Environments" Institute for Computer Graphics, School of Engineering and Applied Science, The George Washington University.

[5] P. Sloan, C.F. Rose III and M.F. Cohen, "Shape and Animation by Example" Technical Report MSR-TR-2000-79, Microsoft Research, Microsoft Corporation, One Microsoft Way, Redmond, WA 98052.

[6] A. Bruderlin and L. Williams "Motion Signal Processing" *Proceedings of the 22 Annual Conference on Computer Graphics and Interactive Techniques*, pp. 97–104, 1995.

LyIT Institiúid Teicneolaíochta Leitir Ceanainn Letterkenny Institute of Technology

[7] L. Kovar and M. Gleicher "Flexible Automatic Motion Blending with Registration Curves" *Eurographics/SIGGRAPH symposium on Computer Animation*, July 2003. `http://www.cs.wisc.edu/graphics/Gallery/kovar.vol/RegistrationCurves/`; accessed February 10th, 2006.

[8] S. W. Smith "The Scientist and Engineers Guide to Digital Signal Processing" California Technical Publishing, P.O. Box 502407, San Diego, CA, 1997.

[9] S. Goldenstein and J. Gomes "Time Warping of Audio Signals" *Computer Graphics International*, vol. 22, no. 4, pp. 52–57, 1999. `citeseer.ist.psu.edu/goldenstein99time.html`; accessed April 9th, 2006.

[10] A. Witkin and Z. Popović "Motion Warping" *Computer Graphics Proceedings* 1995. `http://www.cs.washington.edu/homes/zoran/warpage/warpage.pdf`; accessed January 2005.

[11] T. W. Sederberg and E. Greenwood "A Physically Based Approach to 2D Shape Blending" *Computer Graphics (SIGGRAPH '92 Proceedings)*, vol. 26, pp. 26–34, 1992.

[12] F. Dunn and I. Parberry *3D Math Primer for Graphics and Game Development*, Worldware Publishing Inc, 2320 Los Rios Boulevard, Plano, Texas, 2002.

[13] S. Lang *Calculus of Several Variables*, Springer-Verlag New York Inc.

[14] H. Goldstein *Classical Mechanics*, Addison-Wesley Publishing Company, 11th Printing, p. 118, 1974.

[15] E. C. Ifeachor and B. W. Jervis *Digital Signal Processing - A Practical Approach*, 2nd Edition, Prentice Hall, Pearson Education Limited, Edinburgh Gate, Harlow, Essex CM20 2JE, 2001.

Institiúid Teicneolaíochta Leitir Ceanainn
Letterkenny Institute of Technology
lyit

[16] H. Donald and M. P. Baker *Computer Graphics C Version*, 2nd Edition, chapter 10, section 9, page 334. Pearson Education Limited, Edinburgh Gate, Harlow, Essex CM20 2JE, 2006.

[17] J. Blow "Understanding Slerp, Then Not Using It", The Inner Product, April 2004, `http://number-none.com/product/Understanding\%20Slerp,\%20Then\%20Not\%20Using\%20It/`; accessed March 3rd, 2005.

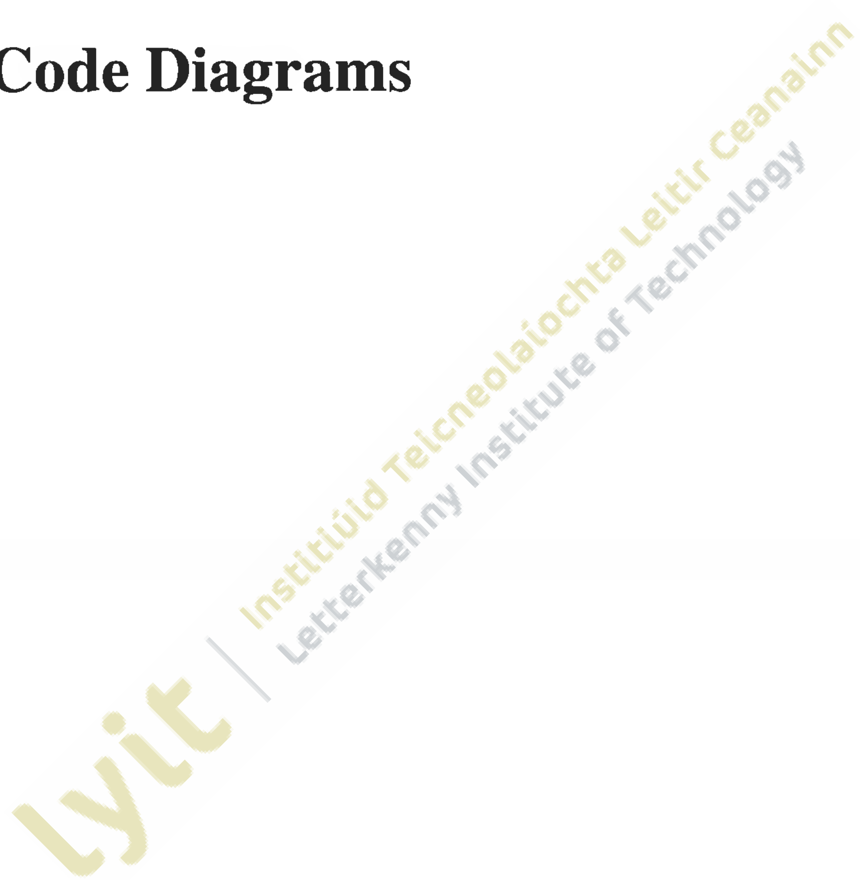
[18] A. Sverdlov "A Very Basic Introduction to Time/Frequency Domains", Particle, March 10, 2004, `http://www.theparticle.com/cs/bc/mcs/signalnotes.pdf`; accessed August 19th, 2006.

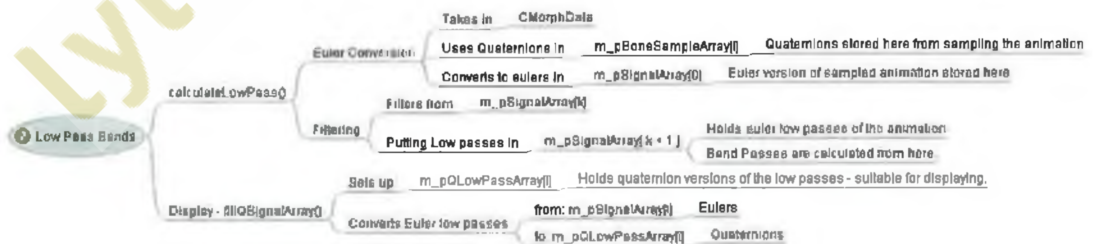
[19] C. Welman "Inverse Kinematics and Geometric Constraints for Articulated Figure Manipulation", British Columbia, Canada, Simon Fraser University, 1993. (M.Sc Thesis).

lyit | Institiúid Teicneolaíochta Leitir Ceanainn
Letterkenny Institute of Technology

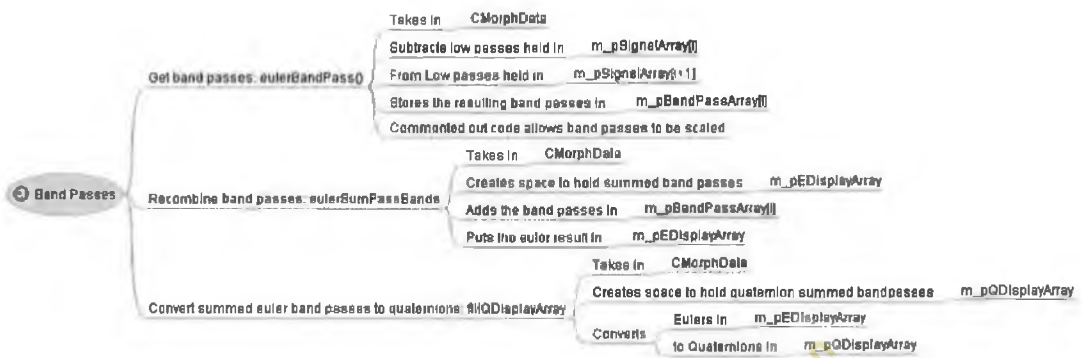
Appendix A

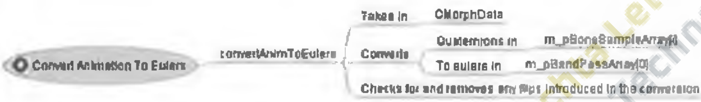
Code Diagrams



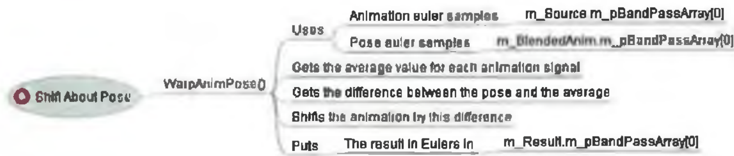


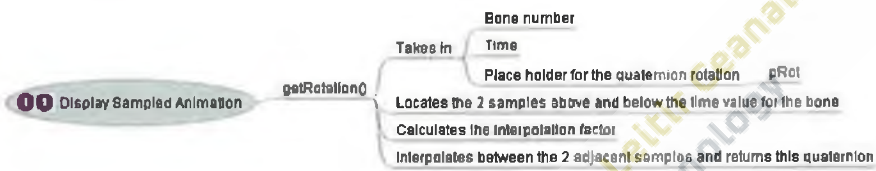
lyit | Institiúid Teicneolaíochta Leitir Ceanainn
Letterkenny Institute of Technology

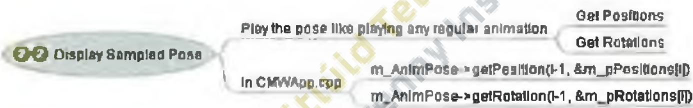




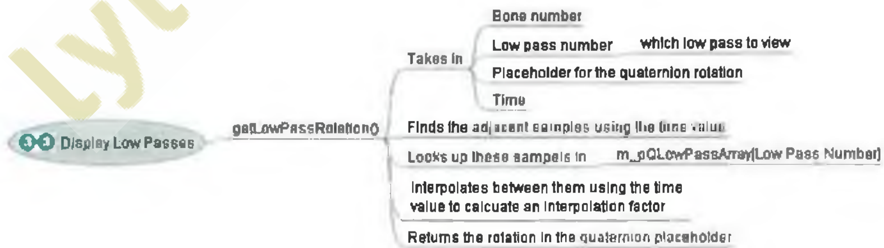
Institiúid Teicneolaíochta Leitir Ceanainn
Letterkenny Institute of Technology
lyit

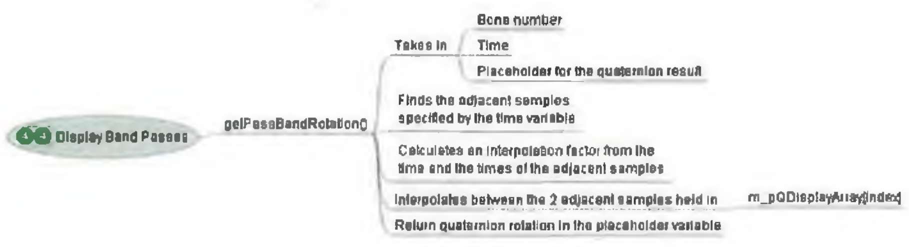


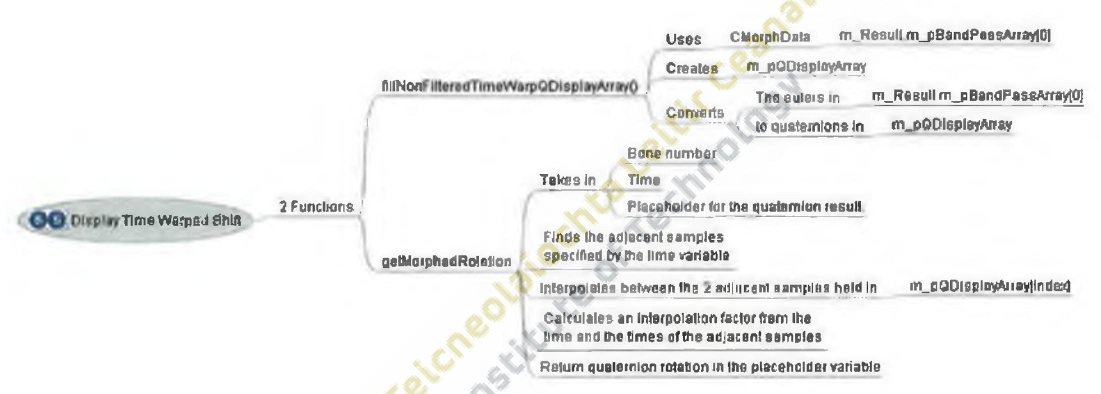


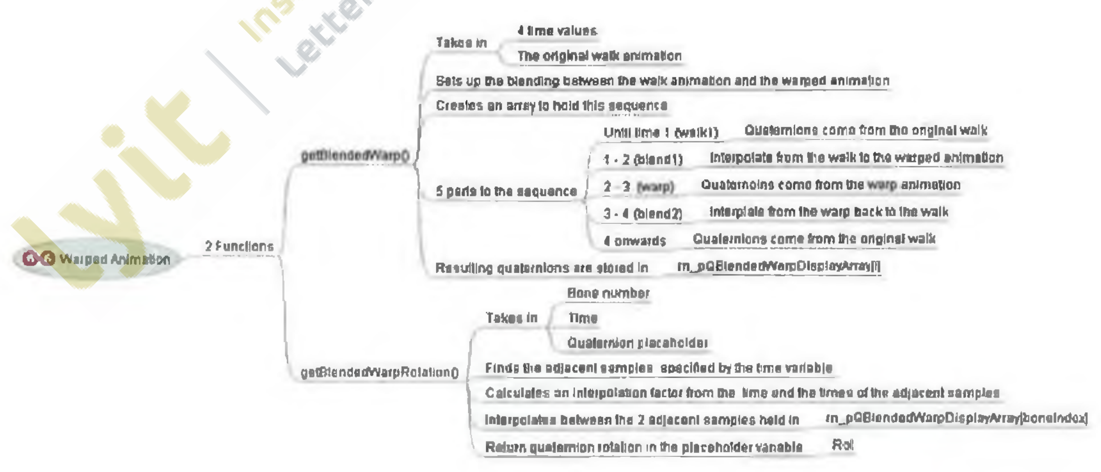


lyit | Institiúid Teicneolaíochta Leitir Ceanainn
Letterkenny Institute of Technology









lyit | Institiúid Teicneolaíochta Leitir Ceanainn
Letterkenny Institute of Technology

Appendix B

C++ Code

Institiúid Teicneolaíochta Leitir Ceanainn
lyit | Letterkenny Institute of Technology

c:\DarraghBuild\src\common.h

```
#ifndef _COMMON_H_
#define _COMMON_H_

// include files common to all the other files in the project.
#include <ieCore/System.h>
#include <ieCore/Utils/CEntityComponentRef.h>
#include <ieModels/IAnimation.h>
#include <ieMaths/Quaternion.h>
#include <ieMaths/Vector.h>
#include <fstream>
#include <math.h>
#include <vector>

namespace IE
{

// holds positions of an animation, along with the time of each
// position
struct POS_KEY
{
    float   time;
    VECTOR  pos;
};

// holds rotations of an animation, along with the time of each
// rotation
struct ROT_KEY
{
    float       time;
    QUATERNION  rot;
};

// bone samples use this structure. Each sample can hold the
// rotation and position of a bone at each sample time.
struct BONE_SAMPLES
{
    ROT_KEY*    pRotKeys;
    POS_KEY*    pPosKeys;
};

struct EULER
{
    float x;
    float y;
    float z;
};

// The euler version of a sample of an animation.
struct SIGNAL_BONE
```

Institiúid Teicneolaíochta Leitir Ceanainn
Letterkenny Institute of Technology
lyit

lyit | Institiúid Teicneolaíochta Leitir Ceanainn
Letterkenny Institute of Technology

```
{
    float *       time;
    EULER *       pSigEuler;
};

struct SIGNAL
{
    SIGNAL_BONE * SignalBone;
};

} // end namespace


struct MYPOINT
{
    float x;
    float y;
};

#endif
```

lyit | Institiúid Teicneolaíochta Leitir Ceanainn
Letterkenny Institute of Technology

Institiúid Teicneolaíochta Leitir Ceanainn
Letterkenny Institute of Technology

c:\DarraghBuild\src\glfunctions.h

```
#include <windows.h>
#include <glut.h>
#include <stdlib.h>
#include <stdio.h>

// initilization
void reshape(int w, int h);
void writeNumber(int number, int offset_x, int offset_y);
void writeWord(const char *word, int offset_x, int offset_y);
void mouse(int button, int state, int x, int y);
void updateDisplay(void);
```

lyit | Institiúid Teicneolaíochta Leitir Ceanainn Letterkenny Institute of Technology

lyit | Institiúid Teicneolaíochta Leitir Ceanainn
Letterkenny Institute of Technology

```
#ifndef _CWARPING_H_
#define _CWARPING_H_



#include "CMorphData.h"
#include "common.h"


namespace IE
{

struct QSIGNAL_BONE
{
    float *       time;
    QUATERNION * pSigQuaternion;
};
struct QSIGNAL
{
    QSIGNAL_BONE * SignalBone;
};
class CWarping
{
public:

    float                 m_WarpStartTime;
    float                 m_WarpEndTime;
    // constructor
    CWarping()
    :   m_Time(0.0f),
        m_pRootPositions(0),
        m_pBoneSampleArray(0)
    {}

    // destructor
    ~CWarping()
    {}

    CMorphData m_Result;
    CMorphData m_Source;
    CMorphData m_BlendedAnim;
    CMorphData m_BlendedWarp;

    // initilization function, takes an animation and a target pose to warp the animation to.
    ieResult init(CEntityComponentRef<Models::IAnimation> &source_anim,
                  CEntityComponentRef<Models::IAnimation> &target_anim);

    void shutdown();
```

lyit | Institiúid Teicneolaíochta Leitir Ceanainn
Letterkenny Institute of Technology

```
    // sets the time. Used for getting the right part of the animation back to apply to the bones to animate them.
    ieResult setTime(float time, CEntityComponentRef<Models::IAnimation> &Animation);

    // converts a quaternion to an euler
    ieResult convertToEuler(const QUATERNION & q,
                            EULER & result );

    // converts an euler to a quaternion
    ieResult convertToQuaternion(const EULER &      e,
                                 QUATERNION & result);

    // returns the time
    float getTime()
    {
        return m_Time;
    }

    // returns the rotation of a bone referenced by 'index' into the quaternion variable provided.
    // The time has already been set using 'set time'
    ieResult getRotation(ieUInt16    index,
                         QUATERNION* pRot,
                         float       time);

    ieResult getLowPassRotation(ieUInt16     boneIndex,
                                QUATERNION & Rot,
                                ieUInt16     lowPassIndex,
                                float        time);

    ieResult getPassBandRotation(ieUInt16    index,
                         QUATERNION* pRot,
                         float       time);
    ieResult getMorphedRotation(ieUInt16    index,
                         QUATERNION* pRot,
                         float       time);

    // returns the position of a bone referenced by 'index' into the Vector provided.
    // The time has already been set using 'set time'
    ieResult getPosition(ieUInt16 index,
                         VECTOR * pPos,
                         float    time);

    ieResult SampleAnimation(CMorphData &data, CEntityComponentRef<Models::IAnimation> &Animation);

    // Returns a pointer to all the sampled information about an animation
    ieResult getBoneSampleArray(BONE_SAMPLES * Result)
    {
        Result = m_pBoneSampleArray;
        return IE_S_OK;
    }
```

lyit | Institiúid Teicneolaíochta Leitir Ceanainn Letterkenny Institute of Technology

```
    ieResult calculateLowPass(CMorphData &data);
    ieResult eulerBandPass(CMorphData &data);
    ieResult eulerSumPassBands(CMorphData &data);
    ieResult fillQDisplayArray(CMorphData &data);
    ieResult eulerSumMorphed();
    ieResult fillMorphedQDisplayArray();
    ieResult fillQSignalArray(CMorphData &data);
    ieResult convertAnimToEulers(CMorphData &anim);
    ieResult fillNonFilteredTimeWarpQDisplayArray();
    ieResult lowPassTimeWarpedSignal();
    ieResult getBlendedWarp(float walk1, float blend1, float warp, float blend2,
                            CEntityComponentRef<Models::IAnimation> &Animation );
    ieResult getBlendedWarpRotation(ieUInt16 boneIndex,
                                    QUATERNION & Rot,
                                    float time);
    ieResult WarpAnimPose();

private:

    float                   m_Time;

    float                   m_WarpPeriodLength;
    float                   m_warpWeight;


    ieInt16                 m_NumSamples;

    BONE_SAMPLES *          m_pBoneSampleArray;
    SIGNAL *                m_pSignalArray;

    QSIGNAL *               m_pQLowPassArray;
    BONE_SAMPLES *          m_pQDisplayArray;
    BONE_SAMPLES *          m_pQBlendedWarpDisplayArray;
    BONE_SAMPLES *          m_pQ1PassDisplayArray;
    VECTOR *                m_pRootPositions;
    SIGNAL_BONE *           m_pEDisplayArray;
};

} //namespace IE
#endif
```

lyit | Institiúid Teicneolaíochta Leitir Ceanainn
Letterkenny Institute of Technology

```
#include "CWarping.h"

#include <iostream>
#include <ieCore/Memory.h>
#include <iemaths/mathsutility.h>
#include "graph.h"

const int SAMPLING_RATE = 15;

//using namespace std;
 namespace IE
{

void CWarping::shutdown()
{
    {
        //This is used when when the series of low pass filters is used
        //The data is split into band passes, and after scaling they are summed
        //with the result put into.

        // can only be deleted if eulerSumPassBands is called, otherwise
        // m_pEDsiplayArray hasn't been initialized.
        //Delete euler display array data
        int i;
        /*for (i = 0; i < m_Result.m_noBoneTracks; ++i)
        {
            ieDeleteDataArray(m_pEDisplayArray[i].pSigEuler);
            ieDeleteDataArray(m_pEDisplayArray[i].time);
        }
        ieDeleteDataArray(m_pEDisplayArray);
        */

        //Delete quaternion display array data
        for (i = 0; i < m_Result.m_noBoneTracks; ++i)
        {
            ieDeleteDataArray(m_pQDisplayArray[i].pRotKeys);
        }
        ieDeleteDataArray(m_pQDisplayArray);
    }

    m_Result.shutdown();
    m_Source.shutdown();
    m_BlendedAnim.shutdown();
}

/////////////////////////////////////////////////////////////////////////////////////////
//
//  CWarping::SampleAnimation
```

Institiúid Teicneolaíochta Leitir Ceanainn
Letterkenny Institute of Technology
lyit

```
//
//          Takes in an animation, discretely samples it, recording the quaternion
//          rotations, the vector positions and the time in the animation they were
//          recorded at in the m_pBoneSampleArray member variable of a CMorphData object
//
/////////////////////////////////////////////////////////////////////////////////////////

ieResult CWarping::SampleAnimation(CMorphData &data, CEntityComponentRef<Models::IAnimation> &Animation)
{
    int i;
    float time_between_samples;
    time_between_samples = (float) 1/data.m_samplingRate;

    // This fills up the quaternion rotation array
    for (i = 0; i < data.m_noBoneTracks; i++)
    {
        // writes the sampled info for each bone to the bone sample array.
        ieInt16 j;
        for (j = 0; j < data.m_noSamples /*- 1*/; j++)
        {
            // Increment time on the animation
            float sample_time = ( j * time_between_samples ) + m_WarpStartTime;
            Animation->setTime(sample_time);

            // Set the key time
            data.m_pBoneSampleArray[i].pRotKeys[j].time = sample_time;
            data.m_pBoneSampleArray[i].pPosKeys[j].time = sample_time;

            // Set the key rotation
            Animation->getRotation(i, &(data.m_pBoneSampleArray[i].pRotKeys[j].rot) );

            // Set the key position
            Animation->getPosition(i, &(data.m_pBoneSampleArray[i].pPosKeys[j].pos) );
        }
    }
    return IE_S_OK;
}


/////////////////////////////////////////////////////////////////////////////////////////
//
//  CWarping::init
//
//          Called when a CWarping object is created. Calls for an animation to be
//          sampled, and then calls for the sampled animation to be put through a low
//          pass filter.
//
/////////////////////////////////////////////////////////////////////////////////////////
```

lyit | Institiúid Teicneolaíochta Leitir Ceanainn
Letterkenny Institute of Technology

```
ieResult CWarping::init(CEntityComponentRef<Models::IAnimation> &SourceAnim,
                        CEntityComponentRef<Models::IAnimation> &TargetAnim)
{
    IE_TRACE

    m_WarpStartTime = 2.0f;
    m_WarpEndTime = 6.0f;
    m_WarpPeriodLength = 4.0f;
    m_warpWeight= 0.0f;

    // Initialze the CMorphData objects
    m_Source.initA(SourceAnim, m_WarpPeriodLength);
    m_BlendedAnim.initA(TargetAnim, m_WarpPeriodLength);
    m_Result.initB(m_WarpPeriodLength, SAMPLING_RATE, SourceAnim->getNumBones());

    // Sample both animations
    SampleAnimation(m_Source, SourceAnim);
    // sample the pose and store the results in m_BlendedAnim
    SampleAnimation(m_BlendedAnim, TargetAnim);

    // Convert the sample animations from quaternions to eulers
    // if the animations are being put through the low pass filters, this is not necessary.
    convertAnimToEulers(m_Source);
    convertAnimToEulers(m_BlendedAnim);

    // Pass the animations through a series of low pass filters.
    // calculateLowPass(m_Source);
    // calculateLowPass(m_BlendedAnim);

    // This shifts the animation signals so the oscillate about the pose signals.
     WarpAnimPose();


    // Initialize a CSederberg object. This will time warp 2 signals. If the 2 signals have
    // been low passed and band passed, it will timewarp the band passes - a call to
    // eulerSumMorphed is requred to sum all the timewarped pass bands.

    // CSederberg(m_Source, m_BlendedAnim, m_Result, m_WarpStartTime, m_WarpEndTime);
    // eulerSumMorphed();

    // when just doing a time warp with no filtering, there is no need to sum pass bands
    // so, eulerSumMorphed is dropped in favour of fillNonFilteredTimeWarpQDisplayArray
    // fillNonFilteredTimeWarpQDisplayArray();

    // use this functoin if the output is noisey
    calculateResultLowPass(m_Result);

    // Set up the blendedWarp animation
    // This sets the times to blend from the walk animation to the warped animation and back
```

Institiúid Teicneolaíochta Leitir Ceanainn
Letterkenny Institute of Technology
lyit

```
    getBlendedWarp(2.0f, 3.5f, 4.5f, 6.0f, SourceAnim );

    return IE_S_OK;
}

//////////////////////////////////////////////////////////////////////////////////////////
//
//  CWarping::setTime
//
//          Sets the time for an animation. If the time is greater than the length of
//          the animation, it will loop round. The m_Time member variable is set to the
//          calculated time.
//
//////////////////////////////////////////////////////////////////////////////////////////

ieResult CWarping::setTime(float time, CEntityComponentRef<Models::IAnimation> &Animation)
{
    m_Time = time;
    float AnimationLength = Animation->getLength();

    if(m_Time > AnimationLength)
    {
        // divide floating point time by floating point animation length, cast it to an int.
        // multiply by animation length, to give an int number of animation loops, subtract
        // this from the time, and get the float left over.
        m_Time = m_Time - ( ( static_cast<ieInt16>( m_Time / AnimationLength ) ) * AnimationLength);
    }

    return IE_S_OK;
}


//////////////////////////////////////////////////////////////////////////////////////////
//
//  CWarping::getRotation
//
//          Takes a bone, a time and returns the rotation applied to that bone at that
//          time, as recorded by the sampling function.
//
//////////////////////////////////////////////////////////////////////////////////////////

ieResult CWarping::getRotation(ieUInt16 index,
                               QUATERNION* pRot,
                               float time)
{
    // binary search to get the correct samples for interpolation.
    ieInt16 low = -1;
    ieInt16 high = m_Source.m_noSamples;
    ieInt16 element;
```

Institiúid Teicneolaíochta Leitir Ceanainn | Letterkenny Institute of Technology lyit

```
    while(( element = ( high - low ) / 2 ) > 0 )
    {
        if(m_Source.m_pBoneSampleArray[index].pRotKeys[low + element].time < time)
        {
            low = low + element;
        }
        else
        {
            high = low + element;
        }
    }
    if(low == -1)
      {
          *pRot = m_Source.m_pBoneSampleArray[index].pRotKeys[0].rot;
          return IE_S_OK;
      }
    if(high == m_Source.m_noSamples)
      {
          *pRot = m_Source.m_pBoneSampleArray[index].pRotKeys[m_Source.m_noSamples - 1].rot;
          return IE_S_OK;
      }

    // first, get the length of the current sample.
    float sample_length;
    sample_length = m_Source.m_pBoneSampleArray[index].pRotKeys[low + 1].time - m_Source.m_pBoneSampleArray[index].pRotKeys[low].time;

    // second, get the time in this interval the quaternion we requrie is at.
    float current_time_in_sample;
    current_time_in_sample = time - m_Source.m_pBoneSampleArray[index].pRotKeys[low].time;

    // divide, to get the interpolation factor - with a check for dividing by 0.
    float interpolation_factor;
    if(sample_length > 0)
    {
        interpolation_factor = current_time_in_sample/sample_length;
    }
    else
    {
        interpolation_factor = 0;
    }
    // interpolate
    QuatLerp(pRot, &m_Source.m_pBoneSampleArray[index].pRotKeys[low].rot, &m_Source.m_pBoneSampleArray[index].pRotKeys[low + 1].rot, ↙
    interpolation_factor);

    return IE_S_OK;
}


////////////////////////////////////////////////////////////////////////////////////////////////////////////
```

Institiúid Teicneolaíochta Leitir Ceanainn
lyit | Letterkenny Institute of Technology

```
//
//  CWarping::getLowPassRotation
//
//          Takes a bone, a time and a low pass index and returns the rotation applied
//          to that bone at that time, as calculated by the low pass filter.
//
//////////////////////////////////////////////////////////////////////////////////////////

ieResult CWarping::getLowPassRotation(ieUInt16 boneIndex,
                                QUATERNION & Rot,
                                ieUInt16 lowPassIndex,
                                float time)
{
    // binary search to get the correct samples for interpolation.
    ieInt16 low = -1;
    ieInt16 high = m_Source.m_noSamples;
    ieInt16 element;
    while(( element = ( high - low ) / 2 ) > 0 )
    {
        if( m_Source.m_pSignalArray[lowPassIndex].SignalBone[boneIndex].time[low + element] < time )
        {
            low = low + element;
        }
        else
        {
            high = low + element;
        }
    }

    if(low == -1)
       {
           Rot = m_pQLowPassArray[lowPassIndex].SignalBone[boneIndex].pSigQuaternion[0];
       }
    if(high == m_Source.m_noSamples)
       {
           Rot = m_pQLowPassArray[lowPassIndex].SignalBone[boneIndex].pSigQuaternion[m_Source.m_noSamples - 1];
       }

    // first, get the length of the current sample.
    float sample_length;

    sample_length = m_pQLowPassArray[lowPassIndex].SignalBone[boneIndex].time[low + 1] - m_pQLowPassArray[lowPassIndex].SignalBone
    [boneIndex].time[low];

    // second, get the time in this interval the quaternion we requrie is at.
    float current_time_in_sample;
    current_time_in_sample = time - m_pQLowPassArray[lowPassIndex].SignalBone[boneIndex].time[low];

    // divide, to get the interpolation factor - with a check for dividing by 0.
```

Institiúid Teicneolaíochta Leitir Ceanainn
lyit | Letterkenny Institute of Technology

```
    float interpolation_factor;
    if(sample_length > 0)
    {
        interpolation_factor = current_time_in_sample/sample_length;
    }
    else
    {
        interpolation_factor = 0;
    }

    // interpolate
    QuatLerp(& Rot, &m_pQLowPassArray[lowPassIndex].SignalBone[boneIndex].pSigQuaternion[low],&m_pQLowPassArray[lowPassIndex].
    SignalBone[boneIndex].pSigQuaternion[low + 1], interpolation_factor);

    return IE_S_OK;
}


///////////////////////////////////////////////////////////////////////////////////////////
//
//  CWarping::getMorphedRotation
//
//          Takes a bone, a time and a low pass index and returns the rotation applied
//          to that bone at that time, as calculated by the low pass filter.
//
//          This function displayes the timewarped animation - which has the slight
//          bit of footskate.
//
///////////////////////////////////////////////////////////////////////////////////////////

ieResult CWarping::getMorphedRotation(ieUInt16 index,
                               QUATERNION* pRot,
                               float time)
{
    float timeInWarp = time - m_WarpStartTime;
    float lengthOfWarp = m_WarpEndTime - m_WarpStartTime;
    float fractionDone = timeInWarp / lengthOfWarp;

    int NoSamples = (int)( lengthOfWarp * SAMPLING_RATE) + 2;
    float indexRequired = NoSamples * fractionDone;
    indexRequired = floor(indexRequired);
    int intIndexRequired = (int)indexRequired;


    // first, get the length of the current sample.
    float sample_length;
    sample_length = m_pQDisplayArray[index].pRotKeys[intIndexRequired + 1].time - m_pQDisplayArray[index].pRotKeys[intIndexRequired].
    time;
```

Institiúid Teicneolaíochta Leitir Ceanainn Letterkenny Institute of Technology lyit

```
    // second, get the time in this interval the quaternion we requrie is at.
    float current_time_in_sample;
    current_time_in_sample = time - m_pQDisplayArray[index].pRotKeys[intIndexRequired].time;

    // divide, to get the interpolation factor - with a check for dividing by 0.
    float interpolation_factor;
    if(sample_length > 0)
    {
        interpolation_factor = current_time_in_sample/sample_length;
    }
    else
    {
        interpolation_factor = 0;
    }

    // interpolate
    QuatLerp(pRot, &m_pQDisplayArray[index].pRotKeys[intIndexRequired].rot,&m_pQDisplayArray[index].pRotKeys[intIndexRequired + 1].rot↙
    , interpolation_factor);

    return IE_S_OK;
}


////////////////////////////////////////////////////////////////////////////////////////
//
//  CWarping::getPassBandRotation
//
//          Takes a bone, a time and returns the rotation applied to that bone at that
//          time, as recorded by the sampling function.
//
////////////////////////////////////////////////////////////////////////////////////////

ieResult CWarping::getPassBandRotation(ieUInt16 index,
                                QUATERNION* pRot,
                                float time)
{
    // binary search to get the correct samples for interpolation.
    ieInt16 low = -1;
    ieInt16 high = 8*SAMPLING_RATE;
    ieInt16 element;
    while(( element = ( high - low ) / 2 ) > 0 )
    {
        if(m_pQDisplayArray[index].pRotKeys[low + element].time < time)
        {
            low = low + element;
        }
        else
        {
            high = low + element;
```

Institiúid Teicneolaíochta Leitir Ceanainn | Letterkenny Institute of Technology lyit

```
        }
    }
    if(low == -1)
       {
           *pRot = m_pQDisplayArray[index].pRotKeys[0].rot;
       }
    if(high == ( 8 * SAMPLING_RATE ) )
       {
           *pRot = m_pQDisplayArray[index].pRotKeys[( 8 * SAMPLING_RATE ) - 1].rot;
       }

    // first, get the length of the current sample.
    float sample_length;
    sample_length = m_pQDisplayArray[index].pRotKeys[low + 1].time - m_pQDisplayArray[index].pRotKeys[low].time;

    // second, get the time in this interval the quaternion we requrie is at.
    float current_time_in_sample;
    current_time_in_sample = time - m_pQDisplayArray[index].pRotKeys[low].time;

    // divide, to get the interpolation factor - with a check for dividing by 0.
    float interpolation_factor;
    if(sample_length > 0)
    {
        interpolation_factor = current_time_in_sample/sample_length;
    }
    else
    {
        interpolation_factor = 0;
    }

    // interpolate
    QuatLerp(pRot, &m_pQDisplayArray[index].pRotKeys[low].rot,&m_pQDisplayArray[index].pRotKeys[low + 1].rot, interpolation_factor);

    return IE_S_OK;
}

/////////////////////////////////////////////////////////////////////////////////////////
//
//  CWarping::getPosition
//
//          Takes a bone, a time and returns the position of that bone at that
//          time, as recorded by the sampling function.
//
/////////////////////////////////////////////////////////////////////////////////////////

ieResult CWarping::getPosition(ieUInt16 index,
                               VECTOR* pPos,
                               float time)
{
```

Institiúid Teicneolaíochta Leitir Ceanainn | Letterkenny Institute of Technology lyit

```
    // binary search to get the correct samples for interpolation.
    ieInt16 low = -1;
    ieInt16 high = m_NumSamples;
    ieInt16 element;
    while((element = (high - low)/2) > 0)
    {
        if( m_pBoneSampleArray[index].pPosKeys[low + element].time < time )
        {
            low = low + element;
        }
        else
        {
            high = low + element;
        }
    }
    if(low == -1)
       {
          *pPos = m_pBoneSampleArray[index].pPosKeys[0].pos;
          return IE_S_OK;
       }
    if(high == m_NumSamples)
       {
          *pPos = m_pBoneSampleArray[index].pPosKeys[m_NumSamples - 1].pos;
          return IE_S_OK;
       }

    // first, get the length of the current sample.
    float sample_length;
    sample_length = m_pBoneSampleArray[index].pPosKeys[low + 1].time - m_pBoneSampleArray[index].pPosKeys[low].time;

    // second, get the time in this interval the quaternion we requrie is at.
    float current_time_in_sample;
    current_time_in_sample = m_Time - m_pBoneSampleArray[index].pPosKeys[low].time;

    // divide, to get the interpolation factor - with a check for dividing by 0.
    float interpolation_factor;
    if(sample_length > 0)
    {
        interpolation_factor = current_time_in_sample/sample_length;
    }
    else
    {
        interpolation_factor = 0;
    }

    // This is used for the numbers displayed on the uppper left corner of the screen
    VECTOR delta_position;
    delta_position.x = (m_pBoneSampleArray[index].pPosKeys[low + 1].pos.x - m_pBoneSampleArray[index].pPosKeys[low].pos.x)*
    interpolation_factor;
```

Institiúid Teicneolaíochta Leitir Ceanainn
Letterkenny Institute of Technology
lyit

```
    delta_position.y = (m_pBoneSampleArray[index].pPosKeys[low + 1].pos.y - m_pBoneSampleArray[index].pPosKeys[low].pos.y)*
    interpolation_factor;
    delta_position.z = (m_pBoneSampleArray[index].pPosKeys[low + 1].pos.z - m_pBoneSampleArray[index].pPosKeys[low].pos.z)*
    interpolation_factor;

    delta_position.x = delta_position.x + m_pBoneSampleArray[index].pPosKeys[low].pos.x;
    delta_position.y = delta_position.y + m_pBoneSampleArray[index].pPosKeys[low].pos.y;
    delta_position.z = delta_position.z + m_pBoneSampleArray[index].pPosKeys[low].pos.z;
    *pPos = delta_position;

    return IE_S_OK;
}


////////////////////////////////////////////////////////////////////////////////////////
//
//  CWarping::convertToEuler
//
//          Takes a quaternion and converts it to an euler, both in local space.
//
////////////////////////////////////////////////////////////////////////////////////////

ieResult CWarping::convertToEuler(const QUATERNION & q, EULER & result)
{
    // Extract sin(pitch)
    float sp = -2.0f * (q.y*q.z - q.w*q.x);

    // check for gimbel lock, giving light tolerance for numerical imprecision
    if(fabs(sp) > 0.9999f)
    {
        // Looking straight up or down
        result.x = 1.5707963f * sp; // Pi/2 x sp

        // Compute heading, slam bank to zero
        result.y = atan2(-q.x*q.z + q.w*q.y, 0.5f - q.y*q.y - q.z*q.z);
        result.z = 0.0f;
    }
    else
    {
        // compute angles. We don't have to use the "safe" asin
        // function because we already checked for range errors
        // when checking for gimbel lock
        //pitch
        result.x = asin(sp);

        // heading
        result.y = atan2(q.x*q.z + q.w*q.y, 0.5f - q.x*q.x - q.y*q.y);

        // bank
```

Institiúid Teicneolaíochta Leitir Ceanainn | Letterkenny Institute of Technology lyit

```
        result.z = atan2(q.x*q.y + q.w*q.z, 0.5f - q.x*q.x - q.z*q.z);
    }

    return IE_S_OK;
}


//////////////////////////////////////////////////////////////////////////////////////////
//
//  CWarping::convertToQuaternion
//
//          Takes an euler and converts it to a quaternion, both in local space.
//
//////////////////////////////////////////////////////////////////////////////////////////

ieResult CWarping::convertToQuaternion( const EULER & e,
                                        QUATERNION & result )
{
    result.w = ( cos( e.y / 2 ) * cos( e.x / 2 ) * cos( e.z / 2 ) ) + ( sin( e.y / 2 ) * sin( e.x / 2 ) * sin( e.z / 2 ) );
    result.x = ( cos( e.y / 2 ) * sin( e.x / 2 ) * cos( e.z / 2 ) ) + ( sin( e.y / 2 ) * cos( e.x / 2 ) * sin( e.z / 2 ) );
    result.y = ( sin( e.y / 2 ) * cos( e.x / 2 ) * cos( e.z / 2 ) ) - ( cos( e.y / 2 ) * sin( e.x / 2 ) * sin( e.z / 2 ) );
    result.z = ( cos( e.y / 2 ) * cos( e.x / 2 ) * sin( e.z / 2 ) ) - ( sin( e.y / 2 ) * sin( e.x / 2 ) * cos( e.z / 2 ) );

    return IE_S_OK;
}


//////////////////////////////////////////////////////////////////////////////////////////
//
//  CWarping::calculateLowPass
//
//          Looks at the sampled information, specifically the number of samples per bone
//          and calculates how many low pass filters to apply. It then passes all the
//          bone information though a series of low pass filters, and records the results
//          in the m_QSignal arrays.
//
//////////////////////////////////////////////////////////////////////////////////////////

ieResult CWarping::calculateLowPass(CMorphData &data)
{
    // constants for the filter kernal
    float a = 0.375;
    float b = 0.25;
    float c = 0.0625;

    // do for every point in a signal
    // Fills up the first signal with Eulers
    // Copies the existing bone sample array into the first signal array.
```

Institiúid Teicneolaíochta Leitir Ceanainn | Letterkenny Institute of Technology lyit

```
for ( int i = 0; i < data.m_noBoneTracks; i++ )
{

    float previous_bank = 0.0f;
    float previous_pitch = 0.0f;
    float previous_heading = 0.0f;

    // do time copy here.
    for ( int j = 0; j < data.m_noSamples; j++ )
    {
        // converts the original samples to eulers and puts them in the unfiltered signal in the array.
        convertToEuler(data.m_pBoneSampleArray[i].pRotKeys[j].rot, data.m_pSignalArray[0].SignalBone[i].pSigEuler[j]);

        float pitch;
        float heading;
        float bank;

        pitch = data.m_pSignalArray[0].SignalBone[i].pSigEuler[j].x;
        // if pitch is within 0.01 of 0.0 then return true.
        if ( FloatAlmostEquals( pitch , 0.0f , 0.000001f ) )
        {
            pitch = 0.0f;
        }
        if ( j == 0 )
        {
            previous_pitch = pitch;
        }
        else
        {
            if ( fabs( pitch - previous_pitch ) > 5.6f )
            {
                pitch = - pitch;
            }
        }

        data.m_pSignalArray[0].SignalBone[i].pSigEuler[j].x = pitch;

        heading = data.m_pSignalArray[0].SignalBone[i].pSigEuler[j].y;
        if ( FloatAlmostEquals( heading, 0.0, 0.000001f ) )
        {
            heading = 0.0f;
        }

        if ( j == 0 )
        {
            previous_heading = heading;
        }
        else
```

Institiúid Teicneolaíochta Leitir Ceanainn
Letterkenny Institute of Technology
lyit

```
        {
            if ( fabs( heading - previous_heading ) > 5.6f )
            {
                heading = - heading;
            }

        }

        data.m_pSignalArray[0].SignalBone[i].pSigEuler[j].y = heading;

        bank = data.m_pSignalArray[0].SignalBone[i].pSigEuler[j].z;
        if( FloatAlmostEquals( bank, 0.0, 0.00001f ) )
        {
            bank = 0.0f;
        }

        if ( j == 0 )
        {
            previous_bank = bank;
        }
        else
        {
            if ( fabs( bank - previous_bank ) > 5.6f )
            {
                bank = - bank;
            }
        }
        data.m_pSignalArray[0].SignalBone[i].pSigEuler[j].z = bank;
        data.m_pSignalArray[0].SignalBone[i].time[j] = data.m_pBoneSampleArray[i].pRotKeys[j].time;

        // set pitch
        pitch = data.m_pSignalArray[0].SignalBone[i].pSigEuler[j].x;
        if ( j == 0 )
        {
            previous_pitch = pitch;
        }
        else
        {
            // If the difference between 2 points is greater than 180 degrees, this is too big of a rotation
            // for one frame. So, find the difference between the current frame and the last frame
            // and add this difference to the current frame, making it the same as the last frame.
            // record this difference for later, and record that the value has been modified.

            if ( fabs( previous_pitch - pitch ) > 3.14f ) // this is the threshold of the angle to fix up.
            {
                data.m_pSignalArray[0].SignalBone[i].pSigEuler[j].x = previous_pitch;
                previous_pitch = pitch;
            }
        }
```

Institiúid Teicneolaíochta Leitir Ceanainn
lyit | Letterkenny Institute of Technology

```
            // set heading
            heading = data.m_pSignalArray[0].SignalBone[i].pSigEuler[j].y;

            if ( j == 0 )
            {
                previous_heading = heading;
            }
            else
            {
                if ( fabs( previous_heading - heading ) > 3.14f )
                {
                    data.m_pSignalArray[0].SignalBone[i].pSigEuler[j].y = previous_heading;
                    previous_heading = heading;
                }
            }

            bank = data.m_pSignalArray[0].SignalBone[i].pSigEuler[j].z;
            if ( j == 0 )
            {
                previous_bank = bank;
            }
            else
            {
                if ( fabs( previous_bank - bank ) > 3.14f )
                {
                    data.m_pSignalArray[0].SignalBone[i].pSigEuler[j].z = previous_bank;
                    previous_bank = bank;
                }
            }
        }
    }

    for ( int k = 0; k < data.m_noFrequencyBands - 1; k++ )
    {
        SIGNAL* p_cur_signal = &data.m_pSignalArray[k];
        for( int j = 0; j < data.m_noBoneTracks; j++ )
        {
            SIGNAL_BONE* p_cur_signal_bone = &p_cur_signal->SignalBone[j];
            for ( int i = 0; i < data.m_noSamples; i++ )
            {
                bool done = false;

                // Need to check to ensure that the function is not looking for values using a negative index

                if( ( i < 2 * pow( 2.0f, k ) ) && ( i < pow( 2.0f, k ) ) )
                {
                    data.m_pSignalArray[ k + 1 ].SignalBone[j].pSigEuler[i].x =
                        c * ( p_cur_signal_bone->pSigEuler[0].x ) +
```

Institiúid Teicneolaíochta Leitir Ceanainn | Letterkenny Institute of Technology lyit

```
                b * ( p_cur_signal_bone->pSigEuler[0].x ) +
                a * ( p_cur_signal_bone->pSigEuler[i].x )+
                b * ( p_cur_signal_bone->pSigEuler[i + static_cast<int>( pow( 2.0f, k ) ) ].x ) +
                c * ( p_cur_signal_bone->pSigEuler[i + static_cast<int>( 2 * pow( 2.0f, k ) ) ].x  );

            data.m_pSignalArray[ k + 1 ].SignalBone[j].pSigEuler[i].y =
                c * ( p_cur_signal_bone->pSigEuler[0].y ) +
                b * ( p_cur_signal_bone->pSigEuler[0].y ) +
                a * ( p_cur_signal_bone->pSigEuler[i].y )+
                b * ( p_cur_signal_bone->pSigEuler[i + static_cast<int>( pow( 2.0f, k ) ) ].y ) +
                c * ( p_cur_signal_bone->pSigEuler[i + static_cast<int>( 2 * pow( 2.0f, k ) ) ].y  );

            data.m_pSignalArray[ k + 1 ].SignalBone[j].pSigEuler[i].z =
                c * ( p_cur_signal_bone->pSigEuler[0].z ) +
                b * ( p_cur_signal_bone->pSigEuler[0].z ) +
                a * ( p_cur_signal_bone->pSigEuler[i].z )+
                b * ( p_cur_signal_bone->pSigEuler[i + static_cast<int>( pow( 2.0f, k ) ) ].z ) +
                c * ( p_cur_signal_bone->pSigEuler[i + static_cast<int>( 2 * pow( 2.0f, k ) ) ].z  );
            done = true;
        }
        else
        {
            if( i < 2 * pow( 2.0f, k ) )
            {
                if( i + ( 2 * pow( 2.0f, k ) ) > data.m_noSamples - 1 )
                {
                    data.m_pSignalArray[ k + 1 ].SignalBone[j].pSigEuler[i].x =
                        c * ( p_cur_signal_bone->pSigEuler[0].x ) +
                        b * ( p_cur_signal_bone->pSigEuler[i - static_cast<int>( pow( 2.0f, k ) ) ].x ) +
                        a * ( p_cur_signal_bone->pSigEuler[i].x ) +
                        b * ( p_cur_signal_bone->pSigEuler[i + static_cast<int>( pow( 2.0f, k ) ) ].x ) +
                        c * ( p_cur_signal_bone->pSigEuler[ data.m_noSamples - 1].x );

                    data.m_pSignalArray[ k + 1 ].SignalBone[j].pSigEuler[i].y =
                        c * ( p_cur_signal_bone->pSigEuler[0].y ) +
                        b * ( p_cur_signal_bone->pSigEuler[i - static_cast<int>( pow( 2.0f, k ) ) ].y ) +
                        a * ( p_cur_signal_bone->pSigEuler[i].y ) +
                        b * ( p_cur_signal_bone->pSigEuler[i + static_cast<int>( pow( 2.0f, k ) ) ].y ) +
                        c * ( p_cur_signal_bone->pSigEuler[ data.m_noSamples - 1].y );
                    done = true;

                    data.m_pSignalArray[ k + 1 ].SignalBone[j].pSigEuler[i].z =
                        c * ( p_cur_signal_bone->pSigEuler[0].z ) +
                        b * ( p_cur_signal_bone->pSigEuler[i - static_cast<int>( pow( 2.0f, k ) ) ].z ) +
                        a * ( p_cur_signal_bone->pSigEuler[i].z ) +
                        b * ( p_cur_signal_bone->pSigEuler[i + static_cast<int>( pow( 2.0f, k ) ) ].z ) +
                        c * ( p_cur_signal_bone->pSigEuler[ data.m_noSamples - 1].x );
                }
                else
```

Institiúid Teicneolaíochta Ceatharlach
Carlow Institute of Technology

```
                {
                data.m_pSignalArray[ k + 1 ].SignalBone[j].pSigEuler[i].x =
                    c * ( p_cur_signal_bone->pSigEuler[0].x ) +
                    b * ( p_cur_signal_bone->pSigEuler[i - static_cast<int>( pow( 2.0f, k ) ) ].x ) +
                    a * ( p_cur_signal_bone->pSigEuler[i].x )+
                    b * ( p_cur_signal_bone->pSigEuler[i + static_cast<int>( pow( 2.0f, k ) ) ].x ) +
                    c * ( p_cur_signal_bone->pSigEuler[i + static_cast<int>( 2 * pow( 2.0f, k ) ) ].x  );

                data.m_pSignalArray[ k + 1 ].SignalBone[j].pSigEuler[i].y =
                    c * ( p_cur_signal_bone->pSigEuler[0].y ) +
                    b * ( p_cur_signal_bone->pSigEuler[i - static_cast<int>( pow( 2.0f, k ) ) ].y ) +
                    a * ( p_cur_signal_bone->pSigEuler[i].y )+
                    b * ( p_cur_signal_bone->pSigEuler[i + static_cast<int>( pow( 2.0f, k ) ) ].y ) +
                    c * ( p_cur_signal_bone->pSigEuler[i + static_cast<int>( 2 * pow( 2.0f, k ) ) ].y  );

                data.m_pSignalArray[ k + 1 ].SignalBone[j].pSigEuler[i].z =
                    c * ( p_cur_signal_bone->pSigEuler[0].z ) +
                    b * ( p_cur_signal_bone->pSigEuler[i - static_cast<int>( pow( 2.0f, k ) ) ].z ) +
                    a * ( p_cur_signal_bone->pSigEuler[i].z )+
                    b * ( p_cur_signal_bone->pSigEuler[i + static_cast<int>( pow( 2.0f, k ) ) ].z ) +
                    c * ( p_cur_signal_bone->pSigEuler[i + static_cast<int>( 2 * pow( 2.0f, k ) ) ].z  );
                done = true;
                }
            }

        }
        // now, need to check that the index is not greater than the number of samples stored
        if( ( i + ( pow( 2.0f, k ) ) >= data.m_noSamples -1) && ( i + ( 2 * pow( 2.0f, k ) ) >= data.m_noSamples ) && done == ↙
false )
            {
                data.m_pSignalArray[ k + 1 ].SignalBone[j].pSigEuler[i].x =
                    c * ( p_cur_signal_bone->pSigEuler[ i - static_cast<int>( 2 * pow( 2.0f, k ) ) ].x  ) +
                    b * ( p_cur_signal_bone->pSigEuler[ i - static_cast<int>( pow( 2.0f, k ) ) ].x ) +
                    a * ( p_cur_signal_bone->pSigEuler[ i ].x ) +
                    b * ( p_cur_signal_bone->pSigEuler[ data.m_noSamples - 1].x ) +
                    c * ( p_cur_signal_bone->pSigEuler[ data.m_noSamples - 1].x );
                done = true;

                data.m_pSignalArray[ k + 1 ].SignalBone[j].pSigEuler[i].y =
                    c * ( p_cur_signal_bone->pSigEuler[ i - static_cast<int>( 2 * pow( 2.0f, k ) ) ].y  ) +
                    b * ( p_cur_signal_bone->pSigEuler[ i - static_cast<int>( pow( 2.0f, k ) ) ].y ) +
                    a * ( p_cur_signal_bone->pSigEuler[ i ].y ) +
                    b * ( p_cur_signal_bone->pSigEuler[ data.m_noSamples - 1].y ) +
                    c * ( p_cur_signal_bone->pSigEuler[ data.m_noSamples - 1].y );
                done = true;

                data.m_pSignalArray[ k + 1 ].SignalBone[j].pSigEuler[i].z =
                    c * ( p_cur_signal_bone->pSigEuler[ i - static_cast<int>( 2 * pow( 2.0f, k ) ) ].z  ) +
```

Institiúid Teicneolaíochta Leitir Ceanainn / Letterkenny Institute of Technology

```
                    b * ( p_cur_signal_bone->pSigEuler[ i - static_cast<int>( pow( 2.0f, k ) ) ].z ) +
                    a * ( p_cur_signal_bone->pSigEuler[ i ].z ) +
                    b * ( p_cur_signal_bone->pSigEuler[ data.m_noSamples - 1].z ) +
                    c * ( p_cur_signal_bone->pSigEuler[ data.m_noSamples - 1].z );
                done = true;
            }
        else
        {
            if ( ( i + ( 2 * pow( 2.0f, k ) ) >= data.m_noSamples  ) && done == false )
            {
            data.m_pSignalArray[ k + 1 ].SignalBone[j].pSigEuler[i].x =
                c * ( p_cur_signal_bone->pSigEuler[ i - static_cast<int>( 2 * pow( 2.0f, k ) ) ].x  ) +
                b * ( p_cur_signal_bone->pSigEuler[ i - static_cast<int>( pow( 2.0f, k ) ) ].x ) +
                a * ( p_cur_signal_bone->pSigEuler[ i ].x ) +
                b * ( p_cur_signal_bone->pSigEuler[ data.m_noSamples - 1 ].x ) +
                c * ( p_cur_signal_bone->pSigEuler[ data.m_noSamples - 1 ].x );

            data.m_pSignalArray[ k + 1 ].SignalBone[j].pSigEuler[i].y =
                c * ( p_cur_signal_bone->pSigEuler[ i - static_cast<int>( 2 * pow( 2.0f, k ) ) ].y  ) +
                b * ( p_cur_signal_bone->pSigEuler[ i - static_cast<int>( pow( 2.0f, k ) ) ].y ) +
                a * ( p_cur_signal_bone->pSigEuler[ i ].y ) +
                b * ( p_cur_signal_bone->pSigEuler[ data.m_noSamples - 1 ].y ) +
                c * ( p_cur_signal_bone->pSigEuler[ data.m_noSamples - 1 ].y );

            data.m_pSignalArray[ k + 1 ].SignalBone[j].pSigEuler[i].z =
                c * ( p_cur_signal_bone->pSigEuler[ i - static_cast<int>( 2 * pow( 2.0f, k ) ) ].z  ) +
                b * ( p_cur_signal_bone->pSigEuler[ i - static_cast<int>( pow( 2.0f, k ) ) ].z ) +
                a * ( p_cur_signal_bone->pSigEuler[ i ].z ) +
                b * ( p_cur_signal_bone->pSigEuler[ data.m_noSamples - 1 ].z ) +
                c * ( p_cur_signal_bone->pSigEuler[ data.m_noSamples - 1 ].z );
            done = true;
            }
        }
        if ( done == false )
        {
            data.m_pSignalArray[ k + 1 ].SignalBone[j].pSigEuler[i].x =
                c * ( p_cur_signal_bone->pSigEuler[i - static_cast<int>( 2 * pow( 2.0f, k ) ) ].x  ) +
                b * ( p_cur_signal_bone->pSigEuler[i - static_cast<int>( pow( 2.0f, k ) ) ].x ) +
                a * ( p_cur_signal_bone->pSigEuler[i].x ) +
                b * ( p_cur_signal_bone->pSigEuler[i + static_cast<int>( pow( 2.0f, k ) ) ].x ) +
                c * ( p_cur_signal_bone->pSigEuler[i + static_cast<int>( 2 * pow( 2.0f, k ) ) ].x  );

            data.m_pSignalArray[ k + 1 ].SignalBone[j].pSigEuler[i].y =
                c * ( p_cur_signal_bone->pSigEuler[i - static_cast<int>( 2 * pow( 2.0f, k ) ) ].y  ) +
                b * ( p_cur_signal_bone->pSigEuler[i - static_cast<int>( pow( 2.0f, k ) ) ].y ) +
                a * ( p_cur_signal_bone->pSigEuler[i].y ) +
                b * ( p_cur_signal_bone->pSigEuler[i + static_cast<int>( pow( 2.0f, k ) ) ].y ) +
                c * ( p_cur_signal_bone->pSigEuler[i + static_cast<int>( 2 * pow( 2.0f, k ) ) ].y  );
```

itit | Institiúid Teicneolaíochta Leitir Ceanainn | Letterkenny Institute of Technology

```
                    data.m_pSignalArray[ k + 1 ].SignalBone[j].pSigEuler[i].z =
                        c * ( p_cur_signal_bone->pSigEuler[i - static_cast<int>( 2 * pow( 2.0f, k ) ) ].z  ) +
                        b * ( p_cur_signal_bone->pSigEuler[i - static_cast<int>( pow( 2.0f, k ) ) ].z ) +
                        a * ( p_cur_signal_bone->pSigEuler[i].z ) +
                        b * ( p_cur_signal_bone->pSigEuler[i + static_cast<int>( pow( 2.0f, k ) ) ].z ) +
                        c * ( p_cur_signal_bone->pSigEuler[i + static_cast<int>( 2 * pow( 2.0f, k ) ) ].z  );
                }
                data.m_pSignalArray[ k + 1 ].SignalBone[j].time[i] = data.m_pBoneSampleArray[j].pRotKeys[i].time;
            }
        }
    }

 // Fill the quaternion version of the array only if displaying the low pass bands
 // fillQSignalArray(data);
    eulerBandPass(data);

 return IE_S_OK;
}


////////////////////////////////////////////////////////////////////////////////////////////
//
//  CWarping::eulerBandPass
//
//          Calculates the band pass values from the low pass values by subtracting the
//          low pass bands from the lower pass bands - the result being the pass band.
//
////////////////////////////////////////////////////////////////////////////////////////////

ieResult CWarping::eulerBandPass(CMorphData &data)
{
    for(int i = 0; i< data.m_noFrequencyBands - 1; i++)
    {
        for(int j = 0; j < data.m_noBoneTracks; j++)
        {
            for (int k = 0; k < data.m_noSamples; k++)
            {
                // To get the band pass, subtract the lower passes from the higher passes
                data.m_pBandPassArray[i].SignalBone[j].pSigEuler[k].x = data.m_pSignalArray[i].SignalBone[j].pSigEuler[k].x - data.
    m_pSignalArray[i+1].SignalBone[j].pSigEuler[k].x;
                data.m_pBandPassArray[i].SignalBone[j].pSigEuler[k].y = data.m_pSignalArray[i].SignalBone[j].pSigEuler[k].y - data.
    m_pSignalArray[i+1].SignalBone[j].pSigEuler[k].y;
                data.m_pBandPassArray[i].SignalBone[j].pSigEuler[k].z = data.m_pSignalArray[i].SignalBone[j].pSigEuler[k].z - data.
    m_pSignalArray[i+1].SignalBone[j].pSigEuler[k].z;
                // set the time for the sample
                data.m_pBandPassArray[i].SignalBone[j].time[k] = data.m_pSignalArray[0].SignalBone[j].time[k];
            }
        }
    }
```

Institiúid Teicneolaíochta Leitir Ceanainn | Letterkenny Institute of Technology

```
    // Copy the lowest low pass band to the band pass array. This is not done above as there is nothing to subtract
    // from it. However, it in itself is a band pass.

    for (int j = 0; j < data.m_noBoneTracks; j++ )
    {
        for (int k = 0; k < data.m_noSamples; k++ )
        {
            data.m_pBandPassArray[data.m_noFrequencyBands - 1].SignalBone[j].pSigEuler[k].x = data.m_pSignalArray[data.
    m_noFrequencyBands - 1].SignalBone[j].pSigEuler[k].x;
            data.m_pBandPassArray[data.m_noFrequencyBands - 1].SignalBone[j].pSigEuler[k].y = data.m_pSignalArray[data.
    m_noFrequencyBands - 1].SignalBone[j].pSigEuler[k].y;
            data.m_pBandPassArray[data.m_noFrequencyBands - 1].SignalBone[j].pSigEuler[k].z = data.m_pSignalArray[data.
    m_noFrequencyBands - 1].SignalBone[j].pSigEuler[k].z;
        }
    }

    // trying out a scaling of one of the bands to see what effect it has on the resulting animation.

    /*for (int j = 0; j < m_NoBoneTracks; j++)
    {
        for (int k = 0; k < m_NumSamples; k++)
        {
            m_pBandPassArray[0].SignalBone[j].pSigEuler[k].x = m_pBandPassArray[0].SignalBone[j].pSigEuler[k].x * 1;
            m_pBandPassArray[0].SignalBone[j].pSigEuler[k].y = m_pBandPassArray[0].SignalBone[j].pSigEuler[k].y * 1;
            m_pBandPassArray[0].SignalBone[j].pSigEuler[k].z = m_pBandPassArray[0].SignalBone[j].pSigEuler[k].z * 1;

            m_pBandPassArray[1].SignalBone[j].pSigEuler[k].x = m_pBandPassArray[1].SignalBone[j].pSigEuler[k].x * 1;
            m_pBandPassArray[1].SignalBone[j].pSigEuler[k].y = m_pBandPassArray[1].SignalBone[j].pSigEuler[k].y * 1;
            m_pBandPassArray[1].SignalBone[j].pSigEuler[k].z = m_pBandPassArray[1].SignalBone[j].pSigEuler[k].z * 1;

            m_pBandPassArray[2].SignalBone[j].pSigEuler[k].x = m_pBandPassArray[2].SignalBone[j].pSigEuler[k].x * 1;
            m_pBandPassArray[2].SignalBone[j].pSigEuler[k].y = m_pBandPassArray[2].SignalBone[j].pSigEuler[k].y * 1;
            m_pBandPassArray[2].SignalBone[j].pSigEuler[k].z = m_pBandPassArray[2].SignalBone[j].pSigEuler[k].z * 1;

            m_pBandPassArray[3].SignalBone[j].pSigEuler[k].x = m_pBandPassArray[3].SignalBone[j].pSigEuler[k].x * 1;
            m_pBandPassArray[3].SignalBone[j].pSigEuler[k].y = m_pBandPassArray[3].SignalBone[j].pSigEuler[k].y * 1;
            m_pBandPassArray[3].SignalBone[j].pSigEuler[k].z = m_pBandPassArray[3].SignalBone[j].pSigEuler[k].z * 1;

            m_pBandPassArray[6].SignalBone[j].pSigEuler[k].x = m_pBandPassArray[6].SignalBone[j].pSigEuler[k].x * 3;
            m_pBandPassArray[6].SignalBone[j].pSigEuler[k].y = m_pBandPassArray[6].SignalBone[j].pSigEuler[k].y * 3;
            m_pBandPassArray[6].SignalBone[j].pSigEuler[k].z = m_pBandPassArray[6].SignalBone[j].pSigEuler[k].z * 3;
        }
    }*/

    eulerSumPassBands(data);
    return IE_S_OK;
}
```

Institiúid Teicneolaíochta Leitir Ceanainn
Letterkenny Institute of Technology
lyit

```
///////////////////////////////////////////////////////////////////////////////////////////
//
//  CWarping::eulerSumPassBands
//
//          The pass bands are held seperately. This function adds them up, in eulers,
//          to give a representation of the animation in eulers. Needed to display
//          an animation altered by scaling band passes
//
///////////////////////////////////////////////////////////////////////////////////////////

ieResult CWarping::eulerSumPassBands(CMorphData &data)
{
    // Create an array to hold the summed band passes in Eulers - later to be
    // converted to quaternions for display.
    m_pEDisplayArray = ieNewDataArray(SIGNAL_BONE, data.m_noBoneTracks);
    for(int j = 0; j < data.m_noBoneTracks; j++)
    {
        m_pEDisplayArray[j].pSigEuler = ieNewDataArray(EULER, data.m_noSamples);
        m_pEDisplayArray[j].time = ieNewDataArray(float, data.m_noSamples);
    }

    for(int j = 0; j < data.m_noBoneTracks; j++)
    {
        for(int k = 0; k < data.m_noSamples; k++)
        {
            for(int i = 0; i < data.m_noFrequencyBands; i++)
            {
                if(i == 0)
                {
                    // if i == 0, we just need to add on the last/lowest pass band.
                    m_pEDisplayArray[j].pSigEuler[k].x = data.m_pBandPassArray[data.m_noFrequencyBands - 1].SignalBone[j].pSigEuler[k]
    .x*10;
                    m_pEDisplayArray[j].pSigEuler[k].y = data.m_pBandPassArray[data.m_noFrequencyBands - 1].SignalBone[j].pSigEuler[k]
    .y*10;
                    m_pEDisplayArray[j].pSigEuler[k].z = data.m_pBandPassArray[data.m_noFrequencyBands - 1].SignalBone[j].pSigEuler[k]
    .z*10;
                    // time
                    m_pEDisplayArray[j].time[k] = data.m_pBandPassArray[0].SignalBone[j].time[k];
                }
                else
                {
                    m_pEDisplayArray[j].pSigEuler[k].x = m_pEDisplayArray[j].pSigEuler[k].x + data.m_pBandPassArray[data.
    m_noFrequencyBands - 1 - i].SignalBone[j].pSigEuler[k].x;
                    m_pEDisplayArray[j].pSigEuler[k].y = m_pEDisplayArray[j].pSigEuler[k].y + data.m_pBandPassArray[data.
    m_noFrequencyBands - 1 - i].SignalBone[j].pSigEuler[k].y;
                    m_pEDisplayArray[j].pSigEuler[k].z = m_pEDisplayArray[j].pSigEuler[k].z + data.m_pBandPassArray[data.
    m_noFrequencyBands - 1 - i].SignalBone[j].pSigEuler[k].z;
```

Institiúid Teicneolaíochta Leitir Ceanainn
Letterkenny Institute of Technology
lyit

```
                }
            }
        }
    }
    fillQDisplayArray(data);
    return IE_S_OK;
}


//////////////////////////////////////////////////////////////////////////////////////////
//
//  CWarping::fillQDisplayArray
//
//          Takes the added up band pass signal which is in eulers and converts it to
//          a quaternion signal, suitable for the engine to display on screen.
//
//////////////////////////////////////////////////////////////////////////////////////////

ieResult CWarping::fillQDisplayArray(CMorphData &data)
{
    // Set up an array to hold the quaternions. A 2D array, Bones * Samples
    m_pQDisplayArray = ieNewDataArray(BONE_SAMPLES, data.m_noBoneTracks);
    for(int j = 0; j < data.m_noBoneTracks; j++)
    {
        m_pQDisplayArray[j].pRotKeys = ieNewDataArray(ROT_KEY, data.m_noSamples);
    }

    for(int j = 0; j < data.m_noBoneTracks; j++)
    {
        for (int k = 0; k < data.m_noSamples; k++)
        {
            convertToQuaternion( m_pEDisplayArray[j].pSigEuler[k] , m_pQDisplayArray[j].pRotKeys[k].rot );
            m_pQDisplayArray[j].pRotKeys[k].time = m_pEDisplayArray[j].time[k];
        }
    }
    return IE_S_OK;
}


//////////////////////////////////////////////////////////////////////////////////////////
//
//  CWarping::eulerSumMorphed
//      Takes the seperately timewarped euler signals and recombines them to form 1 euler
//      signal.
//
//////////////////////////////////////////////////////////////////////////////////////////

ieResult CWarping::eulerSumMorphed()
{
```

Institiúid Teicneolaíochta Leitir Ceanainn
Letterkenny Institute of Technology
lyit

```
    // Create an array to hold the summed band passes in Eulers - later to be
    // converted to quaternions for display.
    m_pEDisplayArray = ieNewDataArray(SIGNAL_BONE, m_Result.m_noBoneTracks);
    for(int j = 0; j < m_Result.m_noBoneTracks; j++)
    {
        m_pEDisplayArray[j].pSigEuler = ieNewDataArray(EULER, m_Result.m_noSamples);
        m_pEDisplayArray[j].time = ieNewDataArray(float, m_Result.m_noSamples);
    }

    for(int j = 0; j < m_Result.m_noBoneTracks; j++)
    {
        for(int k = 0; k < m_Result.m_noSamples; k++)
        {
            for(int i = 0; i < m_Result.m_noFrequencyBands; i++)
            {
                if(i == 0)
                {
                    // if i == 0, we just need to add on the last/lowest pass band.
                    m_pEDisplayArray[j].pSigEuler[k].x = m_Result.m_pBandPassArray[m_Result.m_noFrequencyBands - 1].SignalBone[j].
    pSigEuler[k].x;
                    m_pEDisplayArray[j].pSigEuler[k].y = m_Result.m_pBandPassArray[m_Result.m_noFrequencyBands - 1].SignalBone[j].
    pSigEuler[k].y;
                    m_pEDisplayArray[j].pSigEuler[k].z = m_Result.m_pBandPassArray[m_Result.m_noFrequencyBands - 1].SignalBone[j].
    pSigEuler[k].z;
                    // time
                    m_pEDisplayArray[j].time[k] = m_Result.m_pBandPassArray[0].SignalBone[j].time[k];
                }
                else
                {
                    m_pEDisplayArray[j].pSigEuler[k].x = m_pEDisplayArray[j].pSigEuler[k].x + m_Result.m_pBandPassArray[m_Result.
    m_noFrequencyBands - 1 - i].SignalBone[j].pSigEuler[k].x;
                    m_pEDisplayArray[j].pSigEuler[k].y = m_pEDisplayArray[j].pSigEuler[k].y + m_Result.m_pBandPassArray[m_Result.
    m_noFrequencyBands - 1 - i].SignalBone[j].pSigEuler[k].y;
                    m_pEDisplayArray[j].pSigEuler[k].z = m_pEDisplayArray[j].pSigEuler[k].z + m_Result.m_pBandPassArray[m_Result.
    m_noFrequencyBands - 1 - i].SignalBone[j].pSigEuler[k].z;
                }
            }
        }
    }
    fillMorphedQDisplayArray();
    return IE_S_OK;
}


////////////////////////////////////////////////////////////////////////////////////////
//
//  CWarping::fillQDisplayArray
//
//          Takes the added up band pass signal which is in eulers and converts it to
```

Institiúid Teicneolaíochta Leitir Ceanainn
Letterkenny Institute of Technology

```
//          a quaternion signal, suitable for the engine to display on screen.
//
/////////////////////////////////////////////////////////////////////////////////////////

ieResult CWarping::fillMorphedQDisplayArray()
{
    // Set up an array to hold the quaternions. A 2D array, Bones * Samples
    m_pQDisplayArray = ieNewDataArray(BONE_SAMPLES, m_Result.m_noBoneTracks);
    for(int j = 0; j < m_Result.m_noBoneTracks; j++)
    {
        m_pQDisplayArray[j].pRotKeys = ieNewDataArray(ROT_KEY, m_Result.m_noSamples);
    }

    for(int j = 0; j < m_Result.m_noBoneTracks; j++)
    {
        for (int k = 0; k < m_Result.m_noSamples; k++)
        {
            convertToQuaternion( m_pEDisplayArray[j].pSigEuler[k] , m_pQDisplayArray[j].pRotKeys[k].rot );
            m_pQDisplayArray[j].pRotKeys[k].time = m_pEDisplayArray[j].time[k];
        }
    }

    return IE_S_OK;
}


/////////////////////////////////////////////////////////////////////////////////////////
//
//  CWarping::fillQSignalArray
//
//          Fills the array holding the quaternions after they have been passed through
//          the filter. Each index of the array holds the results of a lower pass filter
//          Used when displaying low pass animations on screen.
//
/////////////////////////////////////////////////////////////////////////////////////////

ieResult CWarping::fillQSignalArray(CMorphData &data)
{
    // Create the QLowPass array - used to hold quaternions to run the low pass display from
    m_pQLowPassArray = ieNewArray(QSIGNAL, ( data.m_noFrequencyBands + 1 ) );
    for ( int i = 0; i < data.m_noFrequencyBands; i++ )
    {
        m_pQLowPassArray[i].SignalBone = ieNewArray(QSIGNAL_BONE, data.m_noBoneTracks );
    }
    for ( int i = 0; i <data.m_noFrequencyBands; i++ )
    {
        for (int j = 0; j < data.m_noBoneTracks; j++ )
        {
            m_pQLowPassArray[i].SignalBone[j].pSigQuaternion = ieNewArray(QUATERNION, data.m_noSamples);
```

lyit | Institiúid Teicneolaíochta Leitir Ceanainn | Letterkenny Institute of Technology

```
            m_pQLowPassArray[i].SignalBone[j].time = ieNewArray(float,data.m_noSamples);
        }
    }

    // Convert the euler low pass bands to quaternions and store in m_pQLowPassArray
    for (int i = 0; i < data.m_noFrequencyBands; i++)
    {
        for (int j = 0; j < data.m_noBoneTracks; j++)
        {
            for(int k = 0; k < data.m_noSamples; k++ )
            {
                convertToQuaternion( data.m_pSignalArray[i].SignalBone[j].pSigEuler[k], m_pQLowPassArray[i].SignalBone[j].
    pSigQuaternion[k] );
                m_pQLowPassArray[i].SignalBone[j].time[k] = data.m_pSignalArray[i].SignalBone[j].time[k];
            }
        }
    }

    // Need to dot product the quaternions to remove stray rotations.
    float result;
    for (int i = 0; i <data.m_noFrequencyBands; i++ )
    {
        for ( int j = 0; j < data.m_noBoneTracks; j++ )
        {
            for (int k = 1; k < data.m_noSamples; k++ )
            {
                result = QuatDotProduct(&m_pQLowPassArray[i].SignalBone[j].pSigQuaternion[k], &m_pQLowPassArray[i].SignalBone[j].
    pSigQuaternion[k-1]);
                if ( result < 1 )
                {
                    // need to negate the quaternion
                    m_pQLowPassArray[i].SignalBone[j].pSigQuaternion[k].w = -m_pQLowPassArray[i].SignalBone[j].pSigQuaternion[k].w;
                    m_pQLowPassArray[i].SignalBone[j].pSigQuaternion[k].x = -m_pQLowPassArray[i].SignalBone[j].pSigQuaternion[k].x;
                    m_pQLowPassArray[i].SignalBone[j].pSigQuaternion[k].y = -m_pQLowPassArray[i].SignalBone[j].pSigQuaternion[k].y;
                    m_pQLowPassArray[i].SignalBone[j].pSigQuaternion[k].z = -m_pQLowPassArray[i].SignalBone[j].pSigQuaternion[k].z;
                }
            }
        }
    }

    //calculateBandPass();
    return IE_S_OK;
}


//////////////////////////////////////////////////////////////////////////////////////////
//
// CWarping::convertAnimToEulers
//      Takes the quaternion signal (made up of the samples in the sampling function) and
```

Institiúid Teicneolaíochta Leitir Ceanainn
Letterkenny Institute of Technology
lyit

```
//      converts them to Eulerian signals. It takes account of the fact a negative
//      quaternion is the same rotation as a positive quaternion.
//
////////////////////////////////////////////////////////////////////////////////////////

ieResult CWarping::convertAnimToEulers(CMorphData &anim)
{
    // do for every point in a signal
    // Fills up the band pass array with eulers.
    // Copies the existing bone sample array into the first signal array.
    for ( int i = 0; i < anim.m_noBoneTracks; i++ )
    {
        // need to know the angles of the previous sample to be able to detect if a rotation
        // close to 2 pi has occured, as this will cause a flip.
        float previous_bank = 0.0f;
        float previous_pitch = 0.0f;
        float previous_heading = 0.0f;

        // do time copy here.
        for ( int j = 0; j < anim.m_noSamples; j++ )
        {
            anim.m_pBandPassArray[0].SignalBone[i].time[j] = anim.m_pBoneSampleArray[i].pRotKeys[j].time;
            // converts the original samples to eulers and puts them in the unfiltered signal in the array.
            convertToEuler(anim.m_pBoneSampleArray[i].pRotKeys[j].rot, anim.m_pBandPassArray[0].SignalBone[i].pSigEuler[j]);

            float pitch;
            float heading;
            float bank;

            pitch = anim.m_pBandPassArray[0].SignalBone[i].pSigEuler[j].x;
            // if pitch is within 0.01 of 0.0 then return true.
            if ( FloatAlmostEquals( pitch , 0.0f , 0.000001f ) )
            {
                pitch = 0.0f;
            }
            if ( j == 0 )
            {
                previous_pitch = pitch;
            }
            else
            {
                if ( fabs( pitch - previous_pitch ) > 1.65f )
                {
                    pitch = - pitch;
                }
            }

            anim.m_pBandPassArray[0].SignalBone[i].pSigEuler[j].x = pitch;
```

lyit | Institiúid Teicneolaíochta Leitir Ceanainn Letterkenny Institute of Technology

```
heading = anim.m_pBandPassArray[0].SignalBone[i].pSigEuler[j].y;
if ( FloatAlmostEquals( heading, 0.0, 0.000001f ) )
{
    heading = 0.0f;
}

if ( j == 0 )
{
    previous_heading = heading;
}
else
{
    if ( fabs( heading - previous_heading ) > 1.65f )
    {
        heading = - heading;
    }
}

anim.m_pBandPassArray[0].SignalBone[i].pSigEuler[j].y = heading;

bank = anim.m_pBandPassArray[0].SignalBone[i].pSigEuler[j].z;
if( FloatAlmostEquals( bank, 0.0, 0.00001f ) )
{
    bank = 0.0f;
}

if ( j == 0 )
{
    previous_bank = bank;
}
else
{
    if ( fabs( bank - previous_bank ) > 1.65f )
    {
        bank = - bank;
    }
}
anim.m_pBandPassArray[0].SignalBone[i].pSigEuler[j].z = bank;

// set pitch
pitch = anim.m_pBandPassArray[0].SignalBone[i].pSigEuler[j].x;
if ( j == 0 )
{
    previous_pitch = pitch;
}
else
{
    previous_pitch = anim.m_pBandPassArray[0].SignalBone[i].pSigEuler[j-1].x;
    // If the difference between 2 points is greater than 180 degrees, this is too big of a rotation
```

Institiúid Teicneolaíochta Leitir Ceanainn | Letterkenny Institute of Technology lyit

```
        // for one frame. So, find the difference between the current frame and the last frame
        // and add this difference to the current frame, making it the same as the last frame.
        // record this difference for later, and record that the value has been modified.

        if ( fabs( previous_pitch - pitch ) > 1.65f ) // this is the threshold of the angle to fix up.
        {
            anim.m_pBandPassArray[0].SignalBone[i].pSigEuler[j].x = previous_pitch;
            previous_pitch = pitch;
        }
    }
    // if the angle is less than 5 x E-6 set it to 0 - angle is too small, ends up as noise.
    if( ( anim.m_pBandPassArray[0].SignalBone[i].pSigEuler[j].x < 0.000005 )
      && ( anim.m_pBandPassArray[0].SignalBone[i].pSigEuler[j].x > -0.000005 ) )
    {
        anim.m_pBandPassArray[0].SignalBone[i].pSigEuler[j].x = 0.000f;
    }
    // set heading
    heading = anim.m_pBandPassArray[0].SignalBone[i].pSigEuler[j].y;

    if ( j == 0 )
    {
        previous_heading = heading;
    }
    else
    {   previous_heading = anim.m_pBandPassArray[0].SignalBone[i].pSigEuler[j-1].y;
        if ( fabs( previous_heading - heading ) > 1.65f )
        {
            anim.m_pBandPassArray[0].SignalBone[i].pSigEuler[j].y = previous_heading;
            previous_heading = heading;
        }
    }
    // if the angle is less than 5 x E-6 set it to 0 - angle is too small, ends up as noise.
    if( ( anim.m_pBandPassArray[0].SignalBone[i].pSigEuler[j].y < 0.000005 )
      && ( anim.m_pBandPassArray[0].SignalBone[i].pSigEuler[j].y > -0.000005 ) )
    {
        anim.m_pBandPassArray[0].SignalBone[i].pSigEuler[j].y = 0.000f;
    }
    // set bank
    bank = anim.m_pBandPassArray[0].SignalBone[i].pSigEuler[j].z;
    if ( j == 0 )
    {
        previous_bank = bank;
    }
    else
    {
        previous_bank = anim.m_pBandPassArray[0].SignalBone[i].pSigEuler[j-1].z;
        if ( fabs( previous_bank - bank ) > 1.65f )
        {
            anim.m_pBandPassArray[0].SignalBone[i].pSigEuler[j].z = previous_bank;
```

Institiúid Teicneolaíochta Leitir Ceanainn | Letterkenny Institute of Technology

```
                    previous_bank = bank;
                }
            }
            // if the angle is less than 5 x E-6 set it to 0 - angle is too small, ends up as noise.
            if( ( anim.m_pBandPassArray[0].SignalBone[i].pSigEuler[j].z < 0.000005 )
              && ( anim.m_pBandPassArray[0].SignalBone[i].pSigEuler[j].z > -0.000005 ) )
            {
                anim.m_pBandPassArray[0].SignalBone[i].pSigEuler[j].z = 0.000f;
            }
        }
    }

    // need to remove any z axis rotation from the thighs - this rotation is only introduced to counteract
    // the roation in the pelvis - which has been squashed in the conversion.
    // also need to fix the Z rotation on the pelvis, if left open the legs will circulate around
    // the vertical axis

    for( int j = 0; j < anim.m_noSamples; j++ )
    {

        // the more the pelvis z deviates from its average z rotation of 1.57... the more the
        // thighs deviate to compensate, by a scale of 2 in the opposite direction.
        float deviation;
        deviation = 1.571153f - anim.m_pBandPassArray[0].SignalBone[1].pSigEuler[j].z;
        anim.m_pBandPassArray[0].SignalBone[21].pSigEuler[j].z = 3.14159265f + ( 1 * deviation );//EWSPelvis.z;
        anim.m_pBandPassArray[0].SignalBone[28].pSigEuler[j].z = 3.14159265f + ( 1 * deviation );//EWSPelvis.z;
    }

    // need to set the number of frequency bands to 1
    anim.m_noFrequencyBands = 1;

    return IE_S_OK;
}


/////////////////////////////////////////////////////////////////////////////////////////
//
//  CWarping::fillNonfilteredTimeWarpQDisplayArray
//      Takes the euler timewarped signal returned from CSederberg and converts it to
//      quaternions so it can be displayed on screen
//
/////////////////////////////////////////////////////////////////////////////////////////

ieResult CWarping::fillNonFilteredTimeWarpQDisplayArray()
{
    // Set up an array to hold the quaternions. A 2D array, Bones * Samples
    m_pQDisplayArray = ieNewDataArray(BONE_SAMPLES, m_Result.m_noBoneTracks);
    for(int j = 0; j < m_Result.m_noBoneTracks; j++)
    {
```

Institiúid Teicneolaíochta Leitir Ceanainn | Letterkenny Institute of Technology lyit

```
        m_pQDisplayArray[j].pRotKeys = ieNewDataArray(ROT_KEY, m_Result.m_noSamples);
    }

    // convert the euler signals to quaternions and store them in the m_pQDisplayArray
    for(int j = 0; j < m_Result.m_noBoneTracks; j++)
    {
        for (int k = 0; k < m_Result.m_noSamples; k++)
        {
            convertToQuaternion( m_Result.m_pBandPassArray[0].SignalBone[j].pSigEuler[k] , m_pQDisplayArray[j].pRotKeys[k].rot );
            m_pQDisplayArray[j].pRotKeys[k].time = m_Result.m_pBandPassArray[0].SignalBone[j].time[k];
        }
    }

    return IE_S_OK;
}


////////////////////////////////////////////////////////////////////////////////////////
//
//  CWarping::getBlendedWarp
//      Sets up the warped animation to display on screen. Takes both the walk
//      animation and the warped animation. Walk1 is the time to start blending between
//      the walk and the warp. The blend finsishes at time blend1. Now the animation
//      all comes from the warp - until the 'warp' time. Then the warp blends back
//      to the walk. The blend finsishes at time 'blend2' and now the animation is just
//      the walk again.
//
////////////////////////////////////////////////////////////////////////////////////////

ieResult CWarping::getBlendedWarp(float walk1, float blend1, float warp, float blend2,
                                  CEntityComponentRef<Models::IAnimation> &Animation )
{
    // Make an array to hold the quaternions of the animation to display on screen.
    // Calculate the size of the array.
    float length = Animation->getLength();
    int size_of_quat_array;
    size_of_quat_array = (int)( length * m_Result.m_samplingRate );

    // Set up an array to hold the quaternions. A 2D array, Bones * Samples
    m_pQBlendedWarpDisplayArray = ieNewDataArray(BONE_SAMPLES, m_Result.m_noBoneTracks);
    for(int p = 0; p < m_Result.m_noBoneTracks; p++)
    {
        m_pQBlendedWarpDisplayArray[p].pRotKeys = ieNewDataArray(ROT_KEY, size_of_quat_array);
    }

    float time_between_samples;
    time_between_samples = (float) 1/m_Result.m_samplingRate;
    int j = 0;
    // This fills up the quaternion rotation array
```

Institiúid Teicneolaíochta Leitir Ceanainn
Letterkenny Institute of Technology
lyit

```
    for (int i = 0; i < m_Result.m_noBoneTracks; i++)
    {
        // writes the sampled info for each bone to the bone sample array.
        float time = 0.0f;
        // reset j
        j = 0;
        for (time = 0; time < length ; time = time + time_between_samples)
        {
            // when time is less than walk1, pull the rotations from the original
            // walk animation.
            if ( time < walk1 )
            {
                getRotation(i, &m_pQBlendedWarpDisplayArray[i].pRotKeys[j].rot, time);
                m_pQBlendedWarpDisplayArray[i].pRotKeys[j].time = time;
            }
            else
            {
                // when time is less than blend 1, pull the rotations from a blend of the
                // original walk and the warped walk.
                if ( time < blend1 )
                {
                    QUATERNION walkQuat;
                    QUATERNION warpedQuat;

                    // get the quaternion from the walk
                    Animation->setTime(time);
                    Animation->getRotation(i, &walkQuat);

                    // get the quaternion from the warp
                    get1PassRotation(i, &warpedQuat, time);

                    // the interpolation factor needs to ramp up as the time goes from the
                    // start of the blend segment to the start of the warped segment
                    float interpolationFactor;
                    interpolationFactor = ( ( time - walk1 ) / ( blend1 - walk1 ) );
                    // Lerp the 2 quaternions
                    QuatLerp(&m_pQBlendedWarpDisplayArray[i].pRotKeys[j].rot, &walkQuat, &warpedQuat, interpolationFactor);
                    m_pQBlendedWarpDisplayArray[i].pRotKeys[j].time = time;
                }
                else
                {
                    // when the time is less than warp - pull the quaternions from the warped animation
                    if( time < warp )
                    {
                        get1PassRotation(i, &m_pQBlendedWarpDisplayArray[i].pRotKeys[j].rot, time);
                        m_pQBlendedWarpDisplayArray[i].pRotKeys[j].time = time;
                    }
                    else
                    {
```

Institiúid Teicneolaíochta Leitir Ceanainn Letterkenny Institute of Technology LyIT

```
                        // if the time is less than blend 2, pull the quaternions from a lerp of the
                        // walk and the warped animations
                        if ( time < blend2 )
                        {
                            QUATERNION walkQuat;
                            QUATERNION warpedQuat;

                            // get the quaternion from the walk
                            Animation->setTime(time);
                            Animation->getRotation(i, &walkQuat);

                            // get the quaternion from the warp
                            get1PassRotation(i, &warpedQuat, time);
                            // the interpolation factor needs to ramp up as the time goes from the
                            // start of the blend segment to the start of the warped segment
                            float interpolationFactor;
                            interpolationFactor = 1.0f - ( ( time - warp ) / ( blend2 - warp ) );
                            // Lerp the 2 quaternions
                            QuatLerp(&m_pQBlendedWarpDisplayArray[i].pRotKeys[j].rot, &walkQuat, &warpedQuat, interpolationFactor);
                            m_pQBlendedWarpDisplayArray[i].pRotKeys[j].time = time;
                        }
                        else
                        {
                            Animation->setTime(time);
                            Animation->getRotation(i, &m_pQBlendedWarpDisplayArray[i].pRotKeys[j].rot );
                            m_pQBlendedWarpDisplayArray[i].pRotKeys[j].time = time;
                        }
                    }
                }
            j++;
            }
        }
    }
    return IE_S_OK;
}


/////////////////////////////////////////////////////////////////////////////////////////
//
//  CWarping::getBlendedWarpRotation
//      Takes a bone number and a time and returns a rotation for the walk-warp-walk
//      sequence animation.
//
/////////////////////////////////////////////////////////////////////////////////////////

ieResult CWarping::getBlendedWarpRotation(ieUInt16 boneIndex,
                                QUATERNION & Rot,
                                float time)
{
```

lyit | Institiúid Teicneolaíochta Leitir Ceanainn
Letterkenny Institute of Technology

```
    // binary search to get the correct samples for interpolation.
    ieInt16 low = -1;
    ieInt16 high = m_Source.m_noSamples;
    ieInt16 element;
    while(( element = ( high - low ) / 2 ) > 0 )
    {
        if( m_pQBlendedWarpDisplayArray[boneIndex].pRotKeys[low + element].time < time )
        {
            low = low + element;
        }
        else
        {
            high = low + element;
        }
    }

    if(low == -1)
       {
           Rot = m_pQBlendedWarpDisplayArray[boneIndex].pRotKeys[0].rot;
       }
    if(high == m_Source.m_noSamples)
       {
           Rot = m_pQBlendedWarpDisplayArray[boneIndex].pRotKeys[0].rot; // MAY WANT TO CHANGE THIS
       }

    // first, get the length of the current sample.
    float sample_length;

    sample_length = m_pQBlendedWarpDisplayArray[boneIndex].pRotKeys[low + 1].time - m_pQBlendedWarpDisplayArray[boneIndex].pRotKeys
    [low].time;

    // second, get the time in this interval the quaternion we requrie is at.
    float current_time_in_sample;
    current_time_in_sample = time - m_pQBlendedWarpDisplayArray[boneIndex].pRotKeys[low].time;

    // divide, to get the interpolation factor - with a check for dividing by 0.
    float interpolation_factor;
    if(sample_length > 0)
    {
        interpolation_factor = current_time_in_sample/sample_length;
    }
    else
    {
        interpolation_factor = 0;
    }

    // interpolate
    QuatLerp(& Rot, &m_pQBlendedWarpDisplayArray[boneIndex].pRotKeys[low].rot, &m_pQBlendedWarpDisplayArray[boneIndex].pRotKeys[low +
    1].rot, interpolation_factor);
```

Institiúid Teicneolaíochta Leitir Ceanainn | Letterkenny Institute of Technology lyit

```
    return IE_S_OK;
}


/////////////////////////////////////////////////////////////////////////////////////////
//
//  CWarping::WarpAnimPose
//      Shifts the animation signal so it centers its oscillations about the pose signal
//
/////////////////////////////////////////////////////////////////////////////////////////

ieResult CWarping::WarpAnimPose()
{
        float averageX;
        float averageY;
        float averageZ;

        float shiftX;
        float shiftY;
        float shiftZ;

    // Find the average value of each (XYZ) animation signal
    for(int j = 0; j < m_Source.m_noBoneTracks; j++ )
    {
        averageX = 0;
        averageY = 0;
        averageZ = 0;

        shiftX = 0;
        shiftY = 0;
        shiftZ = 0;

        for( int i = 0; i < m_Source.m_noSamples; i++ )
        {
            // Get the total of all the valus in each XY and Z component of each signal.
            averageX = averageX + m_Source.m_pBandPassArray[0].SignalBone[j].pSigEuler[i].x;
            averageY = averageY + m_Source.m_pBandPassArray[0].SignalBone[j].pSigEuler[i].y;
            averageZ = averageZ + m_Source.m_pBandPassArray[0].SignalBone[j].pSigEuler[i].z;
        }
        // Divide the totals by the number of points to get the average.
        averageX = averageX/m_Source.m_noSamples;
        averageY = averageY/m_Source.m_noSamples;
        averageZ = averageZ/m_Source.m_noSamples;
        // The distance the signal must be shifted is the distance between the average value and the
        // value of the pose (m_BlendedAnim) points.
        shiftX = m_BlendedAnim.m_pBandPassArray[0].SignalBone[j].pSigEuler[0].x - averageX;
        shiftY = m_BlendedAnim.m_pBandPassArray[0].SignalBone[j].pSigEuler[0].y - averageY;
        shiftZ = m_BlendedAnim.m_pBandPassArray[0].SignalBone[j].pSigEuler[0].z - averageZ;
```

Institiúid Teicneolaíochta Leitir Ceanainn
Letterkenny Institute of Technology
lyit

```
        // Preform the shift here, putting the result in the CMorphData m_Result object.
        for( int i = 0; i < m_Source.m_noSamples; i++ )
        {
            m_Result.m_pBandPassArray[0].SignalBone[j].pSigEuler[i].x = m_Source.m_pBandPassArray[0].SignalBone[j].pSigEuler[i].x +
    shiftX;
            m_Result.m_pBandPassArray[0].SignalBone[j].pSigEuler[i].y = m_Source.m_pBandPassArray[0].SignalBone[j].pSigEuler[i].y +
    shiftY;
            m_Result.m_pBandPassArray[0].SignalBone[j].pSigEuler[i].z = m_Source.m_pBandPassArray[0].SignalBone[j].pSigEuler[i].z +
    shiftZ;

            m_Result.m_pBandPassArray[0].SignalBone[j].time[i] = m_Source.m_pBandPassArray[0].SignalBone[j].time[i];
        }
    }
    return IE_S_OK;
}

} //namespace IE
```

lyit | Institiúid Teicneolaíochta Leitir Ceanainn
Letterkenny Institute of Technology

```
#ifndef _CMORPHDATA_H_
#define _CMORPHDATA_H_

#include "common.h"
namespace IE
{
class CMorphData
{
public:
    CMorphData();
    virtual ~CMorphData();

    void shutdown();
    void initA(CEntityComponentRef<Models::IAnimation> &Animation, float warpLength);
    void initB(float warpLength, int samplingRate, int noBoneTracks);

    float   m_warpLength;
    int     m_samplingRate;
    int     m_noSamples;
    int     m_noBoneTracks;
    int     m_noFrequencyBands;

    BONE_SAMPLES *      m_pBoneSampleArray;
    SIGNAL *            m_pSignalArray;
    SIGNAL *            m_pBandPassArray;
};
} // end namespace IE
#endif
```

lyit | Institiúid Teicneolaíochta Leitir Ceanainn
Letterkenny Institute of Technology

```
#include <ieCore/Memory.h>
#include "CMorphData.h"

// The sampling rate used for sampling an animation. If it is
// change here, it must also be changed in CWarping.cpp
const int SAMPLING_RATE = 15;

namespace IE
{

//////////////////////////////////////////////////////////////////////////////
//
// CMorphData
//      The default constructor for a CMorphData object - rarely used.
//
//////////////////////////////////////////////////////////////////////////////

CMorphData::CMorphData()
:   m_warpLength(0.0f),
    m_samplingRate(0),
    m_noSamples(0),
    m_noBoneTracks(0),
    m_noFrequencyBands(0)
{}


//////////////////////////////////////////////////////////////////////////////
//
// CMorphData::~CMorphData
//      The default destructor - use ::shutdown instead
//
//////////////////////////////////////////////////////////////////////////////

CMorphData::~CMorphData() {}


//////////////////////////////////////////////////////////////////////////////
//
// CMorphData::shutdown
//      De-allocates the memory reserved for a CMorphData object.
//
//////////////////////////////////////////////////////////////////////////////

void CMorphData::shutdown()
{
    //Delete bone samples
    {
        int i;
        for (i = 0; i < m_noBoneTracks; ++i)
```

Institiúid Teicneolaíochta Leitir Ceanainn
lyit | Letterkenny Institute of Technology

```
        {
            ieDeleteDataArray(m_pBoneSampleArray[i].pPosKeys);
            ieDeleteDataArray(m_pBoneSampleArray[i].pRotKeys);
        }
        ieDeleteDataArray(m_pBoneSampleArray);
    }

    //Delete signals
    {
        int i;
        for (i = 0; i < m_noFrequencyBands; ++i)
        {
            int j;
            for (j = 0; j < m_noBoneTracks; j++)
            {
                ieDeleteDataArray(m_pSignalArray[i].SignalBone[j].pSigEuler);
                ieDeleteDataArray(m_pSignalArray[i].SignalBone[j].time);
            }
            ieDeleteDataArray(m_pSignalArray[i].SignalBone);
        }
        ieDeleteDataArray(m_pSignalArray);
    }

    //Delete band passes
    {
        int i;
        for (i = 0; i< m_noFrequencyBands; ++i)
        {
            int j;
            for (j = 0; j < m_noBoneTracks; ++j)
            {
                ieDeleteDataArray(m_pBandPassArray[i].SignalBone[j].pSigEuler);
                ieDeleteDataArray(m_pBandPassArray[i].SignalBone[j].time);
            }
            ieDeleteDataArray(m_pBandPassArray[i].SignalBone);
        }
        ieDeleteDataArray(m_pBandPassArray);
    }
}

////////////////////////////////////////////////////////////////////////////////
//
// CMorphData::initA
//      Sets up a CMorphData object. Reserves memory for frequency bands,
//      animation samples and pass bands. Also sets up the sampling rate and
//      the number of samples. Only requires an animation and the warp length
//
////////////////////////////////////////////////////////////////////////////////
```

Institiúid Teicneolaíochta Leitir Ceanainn
Letterkenny Institute of Technology
lyit

```
void CMorphData::initA(CEntityComponentRef<Models::IAnimation> &Animation, float warpLength)
{
    // Get the length of the animation
    m_warpLength = warpLength;
    // Want to sample at 15 Hz (15 times a second).
    m_samplingRate = SAMPLING_RATE;
    // get the number of samples to be taken.
    m_noSamples = (int)(m_warpLength * m_samplingRate) + 1;
    // get the number of bones in the skeleton
    m_noBoneTracks = Animation->getNumBones();
    // set up the Bone Sample Array
    ieInt16 i;
    m_pBoneSampleArray = ieNewDataArray(BONE_SAMPLES,   m_noBoneTracks);
    for (i = 0; i < m_noBoneTracks; i++)
    {
         m_pBoneSampleArray[i].pRotKeys = ieNewDataArray(ROT_KEY, m_noSamples);
         m_pBoneSampleArray[i].pPosKeys = ieNewDataArray(POS_KEY, m_noSamples);
    }

    // get number of frequency bands
    float n = 0.0f;
    float base = 2.0f;
    float result = 0.0f;
    do
    {
        result = pow( base, n );
        n++;
    }
    while ( result <= m_noSamples );
    m_noFrequencyBands = static_cast<int>(n) - 2;


    // set up the signal array
    m_pSignalArray = ieNewDataArray(SIGNAL, ( m_noFrequencyBands + 1 ) );

    // Make an array in each signal to hold the data for that signal
    for ( i = 0; i < m_noFrequencyBands; i++ )
    {
        m_pSignalArray[i].SignalBone = ieNewDataArray(SIGNAL_BONE, m_noBoneTracks );
    }

    // Make an array in each bone in each signal to hold the euler values for each signal
    for ( i = 0; i < m_noFrequencyBands; i++ )
    {
        for (int j = 0; j < m_noBoneTracks; j++ )
        {
            m_pSignalArray[i].SignalBone[j].pSigEuler = ieNewDataArray(EULER, m_noSamples);
            m_pSignalArray[i].SignalBone[j].time = ieNewDataArray(float, m_noSamples);
```

Institiúid Teicneolaíochta Leitir Ceanainn
Letterkenny Institute of Technology
lyit

```
        }
    }

    // set up the Band Pass Array
    m_pBandPassArray = ieNewDataArray(SIGNAL, m_noFrequencyBands);
    for( i = 0; i < m_noFrequencyBands; i++)
    {
        // In each band pass, a track for each bone
        m_pBandPassArray[i].SignalBone = ieNewDataArray(SIGNAL_BONE, m_noBoneTracks);
    }

    for( i = 0; i < m_noFrequencyBands; i++)
    {
        for(int j = 0; j < m_noBoneTracks; j++)
        {
            // In each bone track, set aside an euler variable for each sample.
            m_pBandPassArray[i].SignalBone[j].pSigEuler = ieNewDataArray(EULER, m_noSamples);
            m_pBandPassArray[i].SignalBone[j].time = ieNewDataArray(float, m_noSamples);
        }
    }
}

///////////////////////////////////////////////////////////////////////////////
//
// CMorphData::initB
//      Sets up a CMorphData object. Reserves memory for frequency bands,
//      animation samples and pass bands. Also sets up the sampling rate and
//      the number of samples. Requires a warp lenght, sampling rate and the
//      number of bones.
//
///////////////////////////////////////////////////////////////////////////////
void CMorphData::initB(float warpLength, int samplingRate, int noBoneTracks)
{
    // Get the length of the animation
    m_warpLength = warpLength;
    // Want to sample at 15 Hz (15 times a second).
    m_samplingRate = SAMPLING_RATE;

    // get the number of samples to be taken.
    m_noSamples = (int)(m_warpLength * m_samplingRate) + 1;

    // get the number of bones in the skeleton
    m_noBoneTracks = noBoneTracks;

    // set up the Bone Sample Array
    ieInt16 i;
    m_pBoneSampleArray = ieNewDataArray(BONE_SAMPLES,   m_noBoneTracks);
    for (i = 0; i < m_noBoneTracks; i++)
    {
```

Institiúid Teicneolaíochta Leitir Ceanainn
lyit | Letterkenny Institute of Technology

```
        m_pBoneSampleArray[i].pRotKeys = ieNewDataArray(ROT_KEY, m_noSamples);
        m_pBoneSampleArray[i].pPosKeys = ieNewDataArray(POS_KEY, m_noSamples);
    }

    // get number of frequency bands
    float n = 0.0f;
    float base = 2.0f;
    float result = 0.0f;
    do
    {
        result = pow( base, n );
        n++;
    }
    while ( result <= m_noSamples );
    m_noFrequencyBands = static_cast<int>(n) - 2;

    // set up the signal array
    m_pSignalArray = ieNewDataArray(SIGNAL, ( m_noFrequencyBands + 1 ) );

    // Make an array in each signal to hold the data for that signal
    for ( i = 0; i < m_noFrequencyBands; i++ )
    {
        m_pSignalArray[i].SignalBone = ieNewDataArray(SIGNAL_BONE, m_noBoneTracks );
    }

    // Make an array in each bone in each signal to hold the euler values for each signal
    for ( i = 0; i < m_noFrequencyBands; i++ )
    {
        for (int j = 0; j < m_noBoneTracks; j++ )
        {
            m_pSignalArray[i].SignalBone[j].pSigEuler = ieNewDataArray(EULER, m_noSamples);
            m_pSignalArray[i].SignalBone[j].time = ieNewDataArray(float, m_noSamples);
        }
    }

    // set up the Band Pass Array
    m_pBandPassArray = ieNewDataArray(SIGNAL, m_noFrequencyBands);
    for( i = 0; i < m_noFrequencyBands; i++)
    {
        // In each band pass, a track for each bone
        m_pBandPassArray[i].SignalBone = ieNewDataArray(SIGNAL_BONE, m_noBoneTracks);
    }

    for( i = 0; i < m_noFrequencyBands; i++)
    {
        for(int j = 0; j < m_noBoneTracks; j++)
        {
            // In each bone track, set aside an euler variable for each sample.
            m_pBandPassArray[i].SignalBone[j].pSigEuler = ieNewDataArray(EULER, m_noSamples);
```

Institiúid Teicneolaíochta Leitir Ceanainn | Letterkenny Institute of Technology lyit

```
                m_pBandPassArray[i].SignalBone[j].time = ieNewDataArray(
            }
        }
}

} // end namespace IE
```

lyit | Institiúid Teicneolaíochta Leitir Ceanainn
Letterkenny Institute of Technology

```
float, m_noSamples);
```

lyit | Institiúid Teicneolaíochta Leitir Ceanainn
Letterkenny Institute of Technology

c:\DarraghBuild\src\bSpline.h

```
#ifndef _BSPLINE_H_
#define _BSPLINE_H_
#include "common.h"

class bSpline
{
public:
    bSpline();
    ~bSpline() {};
    float   CoxDeBoor(int k, int d, float u);
    void    getBSpline();
    void    enterControlPoint(MYPOINT a);
    MYPOINT getUResult(float u);
    int     getNoControlPoints();

private:
    float m_U;
    MYPOINT m_First;
    MYPOINT m_Second;
    std::vector <int> m_Knot;
    std::vector <MYPOINT> m_ControlPoints;
    int m_NoControlPoints;
    int m_D;
};

#endif
```

lyit | Institiúid Teicneolaíochta Leitir Ceanainn
Letterkenny Institute of Technology

lyit | Institiúid Teicneolaíochta Leitir Ceanainn
Letterkenny Institute of Technology

```
#include "bSpline.h"

//////////////////////////////////////////////////////////////////////////////
//
// BSpline Constructor
//      Sets up an empty Bspline by setting the range of influence of each
//      subcurve.
//
//////////////////////////////////////////////////////////////////////////////

bSpline::bSpline()
{
    // we have 4 control points
    // therefore, the knot vector u, is of length 4 - d + 1
    // d is set to 3, to give C2 continunity, and strong local influence
    // hence the knot vector, u, is {0,1,2,3,4,5,6}

    // set the degree of influence each subcurve has on the result.
    m_D = 3;
    m_NoControlPoints = 0;

    // initialize knot vector ( n + d + 1)
    for( int j = 0; j<m_D; j++)
    {
        m_Knot.push_back(j);
    }

    m_U = 0.0f;
}


//////////////////////////////////////////////////////////////////////////////
//
// bSpline::CoxDeBoor
//      The recursive function in a Bspline. Calulates a point on the
//      Bspline curve
//
//////////////////////////////////////////////////////////////////////////////

float bSpline::CoxDeBoor(int k, int d, float u)
{
    if( d == 1 )
    {
        if ( ( u >= m_Knot[k]) && ( u < m_Knot[k+1] ) )
        {
            return 1.0f;
        }
        else
```

Institiúid Teicneolaíochta Leitir Ceanainn | Letterkenny Institute of Technology lyit

```
        {
            return 0.0f;
        }
    }

    return ( ( ( u - m_Knot[k] ) * CoxDeBoor(k, d - 1, u) ) / ( m_Knot[k + d - 1 ] - m_Knot[k] ) ) +
           ( ( ( m_Knot[k + d] - u ) * CoxDeBoor(k + 1, d - 1, u ) ) / ( m_Knot[k + d] - m_Knot[k + 1] ) );
}


////////////////////////////////////////////////////////////////////////////////
//
// bSpline::getBSpline
//        Calculates all the points on a BSpline by calling the CoxDeBoor
//        function for every control point.
//
////////////////////////////////////////////////////////////////////////////////

void bSpline::getBSpline()
{

    for(float u = (float)m_Knot[m_D - 1]; u <= (float)m_Knot[m_NoControlPoints] + 0.2f; u += 0.2f)
    {
        m_Second.x = 0.0f;
        m_Second.y = 0.0f;

        for( int j = 0; j < m_NoControlPoints; j++)
        {
            m_Second.x = m_Second.x + m_ControlPoints[j].x * CoxDeBoor(j, m_D, u);
            m_Second.y = m_Second.y + m_ControlPoints[j].y * CoxDeBoor(j, m_D, u);
        }

        if(u == 2.0f)
        {
            m_First.x = m_Second.x;
            m_First.y = m_Second.y;
        }
        else
        {
            m_First.x = m_Second.x;
            m_First.y = m_Second.y;
        }
    }
}


////////////////////////////////////////////////////////////////////////////////
//
// bSpline::enterControlPoint
```

lyit | Institiúid Teicneolaíochta Leitir Ceanainn
Letterkenny Institute of Technology

```
//      Puts a new control point into the ControlPoints Vector before a BSpline
//      is calculated.
//
////////////////////////////////////////////////////////////////////////////////

void   bSpline::enterControlPoint(MYPOINT a)
{
    m_ControlPoints.push_back(a);
    m_NoControlPoints++;

    // need to adjust the knot vector -> n + d + 1;
    // n = m_NoControlPoints - 1;
    // n + d + 1 = m_NoControlPoints - 1 + d + 1
    // this gives m_NoControlPoints + d
    m_Knot.push_back(m_NoControlPoints + m_D - 1);
}


////////////////////////////////////////////////////////////////////////////////
//
// bSpline::getUResult
//      Takes in a U-value and returns a point on the Bspline corresponding to
//      that U-value.
//
////////////////////////////////////////////////////////////////////////////////

MYPOINT bSpline::getUResult(float u)
{
    MYPOINT pointOnSpline;
    pointOnSpline.x = 0.0f;
    pointOnSpline.y = 0.0f;
    for( int j = 0; j < m_NoControlPoints; j++)
        {
            pointOnSpline.x = pointOnSpline.x + m_ControlPoints[j].x * CoxDeBoor(j, m_D, u);
            pointOnSpline.y = pointOnSpline.y + m_ControlPoints[j].y * CoxDeBoor(j, m_D, u);
        }
    return pointOnSpline;
}


////////////////////////////////////////////////////////////////////////////////
//
// bSpline::getNoControlPoints
//      Returns the number of control points for the BSpline.
//
////////////////////////////////////////////////////////////////////////////////
int bSpline::getNoControlPoints()
{
    return m_NoControlPoints;
```

Institiúid Teicneolaíochta Leitir Ceanainn | Letterkenny Institute of Technology lyit

```
}
```

lyit | Institiúid Teicneolaíochta Leitir Ceanainn Letterkenny Institute of Technology

```
#include "CMorphData.h"
#include "common.h"
#include "bSpline.h"

#ifndef _GRAPH_H_
#define _GRAPH_H_

namespace IE
{
struct PATH
{
    bool north;
    bool west;
    float cost;
    MYPOINT coordinates;
    MYPOINT I;
    MYPOINT J;
    PATH * pNextNode;
    PATH * pParent;
};


enum DIRECTION
{
    diagional = 0,
    west,
    north
};


class CSederberg
{
public:
   CSederberg::CSederberg(CMorphData &, CMorphData &, CMorphData & result, float morphStartTime, float morphEndTime);
    ~CSederberg()
    {
    }
    float        inputPoints();
    float        maximum(float a, float b);
    float        minimum(float a, float b);
    float        crossProduct(MYPOINT a, MYPOINT b);
    float        dotProduct(MYPOINT a, MYPOINT b);



    float        calculateWork(PATH * node);
    void         plotPath();
    void         insertPathNode(PATH * pPreviousNode, PATH * pNewNode);
    float        stretchingWork(MYPOINT a, MYPOINT b, MYPOINT c, MYPOINT d);
```

lyit Institiúid Teicneolaíochta Leitir Ceanainn Letterkenny Institute of Technology

```
    float       bendingWork2(MYPOINT a, MYPOINT b, MYPOINT c, MYPOINT d, MYPOINT e, MYPOINT f);
    PATH *      findNode(int x, int y);
    MYPOINT     average(std::vector<MYPOINT> &value);
    MYPOINT     bSplineEvaluator(std::vector<MYPOINT> &value);
    bSpline     getBSpline(std::vector<MYPOINT> &value);
    void        timeReassignment(CMorphData &a, CMorphData &result);
    void        findOptimalPath();


    int m_AcrossSize;
    int m_DownSize;
    MYPOINT *m_pAcross;
    MYPOINT *m_pDown;

    MYPOINT m_F0;
    MYPOINT m_F1;
    MYPOINT m_B0;
    MYPOINT m_B1;

    MYPOINT m_Q0;
    MYPOINT m_Q1;
    MYPOINT m_Q2;

    float m_d0;
    float m_d1;
    float m_d2;
    int m_NoAcrossPoints;
    int m_NoDownPoints;

    float  m_WarpStartTime;
    float  m_WarpEndTime;

    std::vector<MYPOINT> m_vOptimalPath;
    std::vector<MYPOINT> m_vPathResult;
    std::vector<MYPOINT> m_vFunctionVector;

    PATH * m_pGridPath;
    PATH * m_pGridEnd;
    PATH * m_pNewPathNode ;

    MYPOINT m_Output;
};
} // end namespace IE
#endif
```

lyit | Institiúid Teicneolaíochta Leitir Ceanainn Letterkenny Institute of Technology

```
#include "graph.h"
#include <ieCore/Memory.h>
#include <ieMaths/MathsUtility.h>

namespace IE
{
//////////////////////////////////////////////////////////////////////////////////////
//
// CSederberg::CSederberg
//      Takes 2 corresponding signals and time warps them so they synchronize as best
//      as possible. Takes in 2 CMorphData objects, and places the result in a third
//      CMorphData object.
//
//////////////////////////////////////////////////////////////////////////////////////

CSederberg::CSederberg(CMorphData &a, CMorphData &b, CMorphData &result, float morphStartTime, float morphEndTime)
{
    // Initialize some member variables
    m_WarpStartTime = morphStartTime;
    m_WarpEndTime = morphEndTime;

    m_NoAcrossPoints = b.m_noSamples;
    m_NoDownPoints = a.m_noSamples;

    // for each frequency band
    for (int i = 0; i < a.m_noFrequencyBands; i++)
    {
        // for each euler component
        for ( int XYZ = 0; XYZ < 3; XYZ++ )
        {
            // for each bone
            for (int j = 0; j < a.m_noBoneTracks; j++ )
            {
                // make an array of points for across the top of the grid...
                m_pAcross = new MYPOINT[m_NoAcrossPoints];
                // and down the side of the grid
                m_pDown = new MYPOINT[m_NoDownPoints];

                // copy one signal to the array across the top of the gird
                for (int k = 0;  k < a.m_noSamples; k++ )
                {
                    if( XYZ == 0 )
                    {
                        m_pDown[k].x = a.m_pBandPassArray[i].SignalBone[j].time[k];
                        m_pDown[k].y = a.m_pBandPassArray[i].SignalBone[j].pSigEuler[k].x;
                    }
                    if ( XYZ == 1 )
                    {
                        m_pDown[k].x = a.m_pBandPassArray[i].SignalBone[j].time[k];
```

lyit Institiúid Teicneolaíochta Leitir Ceanainn Letterkenny Institute of Technology

```
                m_pDown[k].y = a.m_pBandPassArray[i].SignalBone[j].pSigEuler[k].y;
            }
            if ( XYZ == 2 )
            {
                m_pDown[k].x = a.m_pBandPassArray[i].SignalBone[j].time[k];
                m_pDown[k].y = a.m_pBandPassArray[i].SignalBone[j].pSigEuler[k].z;
            }
        }

        // and the other to the array down the side of the grid
        for (int l = 0; l < m_NoAcrossPoints; l++ )
        {
            if ( XYZ == 0 )
            {
                m_pAcross[l].x = b.m_pBandPassArray[i].SignalBone[j].time[l];
                m_pAcross[l].y = b.m_pBandPassArray[i].SignalBone[j].pSigEuler[l].x;
            }
            if ( XYZ == 1 )
            {
                m_pAcross[l].x = b.m_pBandPassArray[i].SignalBone[j].time[l];
                m_pAcross[l].y = b.m_pBandPassArray[i].SignalBone[j].pSigEuler[l].y;
            }
            if ( XYZ == 2 )
            {
                m_pAcross[l].x = b.m_pBandPassArray[i].SignalBone[j].time[l];
                m_pAcross[l].y = b.m_pBandPassArray[i].SignalBone[j].pSigEuler[l].z;
            }
        }

        // start creating the grid here
        m_pGridEnd = new PATH;
        m_pGridEnd->pNextNode = 0;
        // create the first node on the grid
        m_pGridPath = new PATH;
        m_pGridPath->north = false; // a non binary value;
        m_pGridPath->west = false;
        m_pGridPath->cost = 0.0f;
        m_pGridPath->coordinates.x = -1;
        m_pGridPath->coordinates.y = -1;
        m_pGridPath->pNextNode = m_pGridEnd;
        findOptimalPath();
        // m_vPathResult holds the merged signals.

        // copy the new timewarped signal to the result CMorphData object.
        for(int m = 0; m < a.m_noSamples; m++)
        {
            result.m_pBandPassArray[i].SignalBone[j].time[m] = m_vPathResult[m].x;
            if( XYZ == 0 )
            {
```

Institiúid Teicneolaíochta Leitir Ceanainn
Letterkenny Institute of Technology
lyit

```
                        result.m_pBandPassArray[i].SignalBone[j].pSigEuler[m].x = ( m_vPathResult[m].y );
                    }
                    if( XYZ == 1 )
                    {
                        result.m_pBandPassArray[i].SignalBone[j].pSigEuler[m].y = ( m_vPathResult[m].y );
                    }
                    if( XYZ == 2 )
                    {
                        result.m_pBandPassArray[i].SignalBone[j].pSigEuler[m].z = ( m_vPathResult[m].y );
                    }
                }
                // delete the across and down arrays
                delete[] m_pAcross;
                m_pAcross = 0;
                delete[] m_pDown;
                m_pDown = 0;
                // delete the start and end nodes of the grid
                delete m_pGridEnd;
                m_pGridEnd = 0;
                delete[] m_pNewPathNode;
                m_pNewPathNode = 0;
                delete m_pGridPath;
                m_pGridPath = 0;
            }
        }
    }
    // reassign the time values to force the timewarp.
    timeReassignment(a, result);
}


////////////////////////////////////////////////////////////////////////////////////////////
//
// CSederberg::crossProduct
//      Takes 2 points and returns their cross product
//
////////////////////////////////////////////////////////////////////////////////////////////

float CSederberg::crossProduct(MYPOINT a, MYPOINT b)
{
    return ( a.x * b.y ) - ( b.x * a.y );
}


////////////////////////////////////////////////////////////////////////////////////////////
//
// CSederberg::dotProduct
//      Takes 2 points and returns their dot product
//
```

Institiúid Teicneolaíochta Leitir Ceanainn | Letterkenny Institute of Technology lyit

```
/////////////////////////////////////////////////////////////////////////////////////////

float CSederberg::dotProduct(MYPOINT a, MYPOINT b)
{
    return ( a.x * b.x ) + ( a.y * b.y );
}


/////////////////////////////////////////////////////////////////////////////////////////
//
// CSederberg::calculateWork
//      Calculates the work at a node on the grid.
//
/////////////////////////////////////////////////////////////////////////////////////////

float CSederberg::calculateWork(PATH * node)
{
    // These weights give a ratio of the 2 cost components - bending and stretching.
    // they can be altered to give different timewarping results.
    float bendWeight = 5.0f;
    float stretchWeight = 1.0f;

    // firstly, check if the coordinates of the point are 0,0, if so, return 0 for the work.
    if( ( node->coordinates.x == 0 ) && ( node->coordinates.y == 0 ) )
    {
        return 2.0f;
    }

    // check both other corners and set their cost to be higher than the nodes north and west of them
    if( ( node->coordinates.x == 0 ) && ( node->coordinates.y == ( m_NoDownPoints - 1 ) ) )
    {
        PATH * temp;
        temp = findNode(0, m_NoDownPoints - 2 );
        return temp->cost * 1.1;
    }

    if( ( node->coordinates.x == ( m_NoAcrossPoints - 1 ) ) && ( node->coordinates.y == 0 ) )
    {
        // should not be able to access this node as it is on the upper right corner.
        return node->pParent->cost * 1.1;
    }

    // if i is 0, point is on the j axis, work is 1 dimensional
    if ( ( node->coordinates.x == 0 ) && ( node->coordinates.y < ( m_NoDownPoints - 1 ) ) )
    {
        // stretching work is between this node and its parent.
        if (node->coordinates.y == 1 )
        {
            node->north = true;
```

Institiúid Teicneolaíochta Leitir Ceanainn | Letterkenny Institute of Technology | lyit

```
            node->west = false;
            PATH * stretchingPoint;
            stretchingPoint = findNode(0,0);
            return ( stretchingWork(node->I, node->J, stretchingPoint->I, stretchingPoint->J) * stretchWeight )
                + stretchingPoint->cost;
        }
        node->north = true;
        node->west = false;
        PATH * bendingPoint;
        PATH * stretchingPoint;

        bendingPoint = findNode(0, node->coordinates.y - 2);
        stretchingPoint = findNode(0,node->coordinates.y - 1);
        return ( ( stretchingWork(node->I, node->J, stretchingPoint->I, stretchingPoint->J) * stretchWeight ) +
               ( bendingWork2(node->I, stretchingPoint->I, bendingPoint->I, node->J, stretchingPoint->J, bendingPoint->J) * bendWeight↙
 )
               + stretchingPoint->cost );//
    }

    // if j is 0, point is on the i axis, work is 1 dimensional
    if ( ( node->coordinates.y == 0 ) && ( node->coordinates.x < ( m_NoAcrossPoints - 1 ) ) )
    {
        // stretching work is between this node and its parent.
        if (node->coordinates.x == 1 )
        {
            node->north = false;
            node->west = true;
            return ( stretchingWork(node->I, node->J, node->pParent->I, node->pParent->J) * stretchWeight )
                + node->pParent->cost;
        }
        node->north = false;
        node->west = true;
        return ( stretchingWork(node->I, node->J, node->pParent->I, node->pParent->J) * stretchWeight ) +
               ( bendingWork2(node->I, node->pParent->I, node->pParent->pParent->I, node->J, node->pParent->J, node->pParent->pParent ↙
    ->J) * bendWeight )
               + node->pParent->cost;
    }

    // if the point is 1,1, there is no bending, only stretching.
    if( ( node->coordinates.x == 1 ) && ( node->coordinates.y == 1 ) )
    {
        node->north = true;
        node->west = true;
        PATH * stretchingPoint;
        stretchingPoint = findNode(0,0);
        return ( stretchingWork(node->I, node->J, stretchingPoint->I, stretchingPoint->J) * stretchWeight )
            + stretchingPoint->cost;
    }
```

Institiúid Teicneolaíochta Leitir Ceanainn
Letterkenny Institute of Technology
lyit

```
    // if the point is (2,2) there is no vertial or horizontal bending
    if ( ( node->coordinates.x == 2 ) && ( node->coordinates.y == 2 ) )
    {
        PATH * bendingPoint;
        PATH * diagonalPoint;

        float upDiagonal;
        float diagonalDiagonal;
        float acrossDiagonal;

        bendingPoint = findNode(0,0);
        diagonalPoint = findNode(1,1);

        upDiagonal = ( stretchingWork(node->I, node->J, diagonalPoint->pNextNode->I, diagonalPoint->pNextNode->J) * stretchWeight )↙
     +
            ( bendingWork2(node->I, diagonalPoint->pNextNode->I, bendingPoint->pNextNode->I, node->J, diagonalPoint->pNextNode->J, ↙
    bendingPoint->pNextNode->J) * bendWeight )
            + diagonalPoint->pNextNode->cost;
        diagonalDiagonal = ( stretchingWork(node->I, node->J, diagonalPoint->I, diagonalPoint->J) * stretchWeight ) +
            ( bendingWork2(node->I, diagonalPoint->I, bendingPoint->I, node->J, diagonalPoint->J, bendingPoint->J) * bendWeight )
            + diagonalPoint->cost;
        acrossDiagonal = ( stretchingWork(node->I, node->J, node->pParent->I, node->pParent->J) * stretchWeight ) +
            ( bendingWork2(node->I, node->pParent->I, diagonalPoint->pParent->I, node->J, node->pParent->J, diagonalPoint->pParent->↙
    J) * bendWeight )
            + node->pParent->cost;

        float min;
        min = upDiagonal;
        node->north = true;
        node->west = false;
        if( diagonalDiagonal < min )
        {
            node->north = true;
            node->west = true;
            min = diagonalDiagonal;
        }
        if( acrossDiagonal < min )
        {
            node->north = false;
            node->west = true;
            min = acrossDiagonal;
        }
        return min;
    }

    // if the point is (m_NoAcrossPoints, 1) then just calculate the diagonal
    if( ( node->coordinates.x == m_NoAcrossPoints ) && ( node->coordinates.y == 1 ) )
    {
        PATH * pDiagonal;
```

lyit | Institiúid Teicneolaíochta Leitir Ceanainn
Letterkenny Institute of Technology

```
        pDiagonal = findNode((int)node->coordinates.x - 1, (int)node->coordinates.y - 1 );

        node->north = true;
        node->west = true;
        return ( stretchingWork(node->I, node->J, pDiagonal->I, pDiagonal->J) * stretchWeight ) +
            ( bendingWork2(node->I, pDiagonal->I, pDiagonal->pParent->I, node->J, pDiagonal->J, pDiagonal->pParent->J) * 
    bendWeight )
            + pDiagonal->cost;
    }

    // if the point is (1, m_NoDownPoints) then just calculate the diagonal
    if ( ( node->coordinates.x == 1 ) && ( node->coordinates.y == m_NoDownPoints ) )
    {
        PATH * pDiagonal;
        PATH * pBendingPoint;

        pDiagonal = findNode((int)node->coordinates.x - 1, (int)node->coordinates.y - 1);
        pBendingPoint = findNode((int)node->coordinates.x - 1, (int)node->coordinates.y - 2);

        node->north = true;
        node->west = true;
        return ( stretchingWork(node->I, node->J, pDiagonal->I, pDiagonal->J)  * stretchWeight ) +
            ( bendingWork2(node->I, pDiagonal->I, pBendingPoint->J, node->J, pDiagonal->J, pBendingPoint->J) * bendWeight )
            + pDiagonal->cost;
    }

    // if the point is (x,1) the vertial stretching is infinate - don't calculate it
    if( node->coordinates.y == 1 )
    {
        float diagionalBack;
        float backDiagional;

        PATH * bendingPoint;
        bendingPoint = findNode((int)node->coordinates.x - 2, (int)node->coordinates.y - 1);

        backDiagional = ( stretchingWork(node->I, node->J, node->pParent->I, node->pParent->J)  * stretchWeight ) +
            ( bendingWork2(node->I, node->pParent->I, bendingPoint->I, node->J, node->pParent->J, bendingPoint->J) * bendWeight )
            + node->pParent->cost;

        diagionalBack = ( stretchingWork(node->I, node->J, bendingPoint->pNextNode->I, bendingPoint->pNextNode->J) * stretchWeight ) +
            ( bendingWork2(node->I, bendingPoint->pNextNode->I, bendingPoint->I, node->J, bendingPoint->pNextNode->J, bendingPoint->J)
     * bendWeight )
            + bendingPoint->pNextNode->cost;
        if ( diagionalBack < backDiagional )
        {
            node->north = true;
            node->west = true;
            return diagionalBack;
        }
```

```
        else
        {
            node->north = false;
            node->west = true;
            return backDiagonal;
        }
    }

    // if the point is (1,y) the horizontal stretching is infinate - don't calculate it
    if( node->coordinates.x == 1)
    {
        float upDiagonal;
        float diagonalUp;

        PATH * bendingPoint;
        PATH * upPoint;
        PATH * diagonalPoint;
        bendingPoint = findNode((int)node->coordinates.x - 1, (int)node->coordinates.y - 2);
        upPoint = findNode((int)node->coordinates.x, (int)node->coordinates.y - 1);
        diagonalPoint = findNode((int)node->coordinates.x - 1, (int)node->coordinates.y - 1);

        upDiagonal = ( stretchingWork(node->I, node->J, upPoint->I, upPoint->J)  * stretchWeight ) +
            ( bendingWork2(node->I, upPoint->I, bendingPoint->I, node->J, upPoint->J, bendingPoint->J) * bendWeight )
            +upPoint->cost;

        diagonalUp = ( stretchingWork(node->I, node->J, diagonalPoint->I, diagonalPoint->J)  * stretchWeight ) +
            ( bendingWork2(node->I, diagonalPoint->I, bendingPoint->I, node->J, diagonalPoint->J, bendingPoint->J) * bendWeight )
            + diagonalPoint->cost;

        if( upDiagonal < diagonalUp )
        {
            node->north = true;
            node->west = false;
            return upDiagonal;
        }
        else
        {
            node->north = true;
            node->west = true;
            return diagonalUp;
        }
    }

    // if the point is (m_NoAcrossPoints, m_NoDownPoints)
    if( ( node->coordinates.x == m_NoAcrossPoints ) && ( node->coordinates.y == m_NoDownPoints ) )
    {
        PATH * bendingPoint;
        PATH * diagonalPoint;
```

Institiúid Teicneolaíochta Leitir Ceanainn
Letterkenny Institute of Technology
lyit

```
    float upUp;
    float upDiagonal;
    float diagonalUp;
    float diagonalDiagonal;
    float diagonalAcross;
    float acrossDiagonal;
    float acrossAcross;

    bendingPoint = findNode((int)node->coordinates.x - 1, (int)node->coordinates.y - 2);
    diagonalPoint = findNode((int)node->coordinates.x - 1, (int)node->coordinates.y - 1);

    upUp = ( stretchingWork(node->I, node->J, diagonalPoint->pNextNode->I, diagonalPoint->pNextNode->J)  * stretchWeight ) +
        ( bendingWork2(node->I, diagonalPoint->pNextNode->I, bendingPoint->pNextNode->I, node->J, diagonalPoint->pNextNode->J, ↙
bendingPoint->pNextNode->J) * bendWeight )
        + diagonalPoint->pNextNode->cost;
    upDiagonal = ( stretchingWork(node->I, node->J, diagonalPoint->pNextNode->I, diagonalPoint->pNextNode->J)  * stretchWeight ↙
 ) +
        ( bendingWork2(node->I, diagonalPoint->pNextNode->I, bendingPoint->I, node->J, diagonalPoint->pNextNode->J, bendingPoint↙
->J) * bendWeight )
        + diagonalPoint->pNextNode->cost;
    diagonalUp = ( stretchingWork(node->I, node->J, diagonalPoint->I, diagonalPoint->J)  * stretchWeight ) +
        ( bendingWork2(node->I, diagonalPoint->I, bendingPoint->I, node->J, diagonalPoint->J, bendingPoint->J) * bendWeight )
        + diagonalPoint->cost;
    diagonalDiagonal = ( stretchingWork(node->I, node->J, diagonalPoint->I, diagonalPoint->J)  * stretchWeight ) +
        ( bendingWork2(node->I, diagonalPoint->I, bendingPoint->pParent->I, node->J, diagonalPoint->J, bendingPoint->pParent->J)↙
 * bendWeight )
        + diagonalPoint->cost;
    diagonalAcross =( stretchingWork(node->I, node->J, diagonalPoint->I, diagonalPoint->J)  * stretchWeight ) +
        ( bendingWork2(node->I, diagonalPoint->I, diagonalPoint->pParent->I, node->J, diagonalPoint->J, diagonalPoint->pParent↙
->J) * bendWeight )
        + diagonalPoint->cost;
    acrossDiagonal =( stretchingWork(node->I, node->J, node->pParent->I, node->pParent->J)  * stretchWeight ) +
        ( bendingWork2(node->I, node->pParent->I, diagonalPoint->pParent->I, node->J, node->pParent->J, diagonalPoint->pParent->↙
J) * bendWeight )
        + node->pParent->cost;
    acrossAcross = ( stretchingWork(node->I, node->J, node->pParent->I, node->pParent->J)  * stretchWeight ) +
        ( bendingWork2(node->I, node->pParent->I, node->pParent->pParent->I, node->J, node->pParent->J, node->pParent->pParent->J)↙
 * bendWeight )
        + node->pParent->cost;

    float min = upUp;
    node->north = true;
    node->west = false;
    if( upDiagonal < min )
    {
        node->north = true;
        node->west = false;
        min = upDiagonal;
    }
```

```
        if( diagonalUp < min )
        {
            node->north = true;
            node->west = true;
            min = diagonalUp;
        }
        if( diagonalDiagonal < min )
        {
            node->north = true;
            node->west = true;
            min = diagonalDiagonal;
        }
        if( diagonalAcross < min )
        {
            node->north = true;
            node->west = true;
            min = diagonalAcross;
        }
        if( acrossDiagonal < min )
        {
            node->north = false;
            node->west = true;
            min = acrossDiagonal;
        }
        if( acrossAcross < min )
        {
            node->north = false;
            node->west = true;
            min = acrossAcross;
        }
        return min;
    }

    // if the point is (2, m_NoDownPoints) calculate back and diagional
    if( ( node->coordinates.x == 2) && ( node->coordinates.y == m_NoDownPoints ) )
    {
        PATH * bendingPoint;
        PATH * diagionalPoint;

        float   diagionalUp;
        float   diagionalDiagional;
        float   diagionalAcross;
        float   acrossDiagional;

        bendingPoint = findNode(1, m_NoDownPoints - 2);
        diagionalPoint = findNode(1, m_NoDownPoints - 1);

        diagionalUp = ( stretchingWork(node->I, node->J, diagionalPoint->I, diagionalPoint->J)  * stretchWeight ) +
            ( bendingWork2(node->I, diagionalPoint->I, bendingPoint->I, node->J, diagionalPoint->J, bendingPoint->J) * bendWeight )
```

Institiúid Teicneolaíochta Leitir Ceanainn
Letterkenny Institute of Technology
lyit

```
            + diagonalPoint->cost;
        diagonalDiagonal = ( stretchingWork(node->I, node->J, diagonalPoint->I, diagonalPoint->J)  * stretchWeight ) +
            ( bendingWork2(node->I, diagonalPoint->I, bendingPoint->pParent->I, node->J, diagonalPoint->J, bendingPoint->pParent->J)↙
 * bendWeight )
            + diagonalPoint->cost;
        diagonalAcross = ( stretchingWork(node->I, node->J, diagonalPoint->I, diagonalPoint->J)  * stretchWeight ) +
            ( bendingWork2(node->I, diagonalPoint->I, diagonalPoint->pParent->I, node->J, diagonalPoint->J, diagonalPoint->pParent↙
->J) * bendWeight )
            + diagonalPoint->cost;
        acrossDiagonal = ( stretchingWork(node->I, node->J, node->pParent->I, node->pParent->J)  * stretchWeight ) +
            ( bendingWork2(node->I, node->pParent->I, diagonalPoint->pParent->I, node->J, node->pParent->J, diagonalPoint->pParent->↙
J) * bendWeight )
            + node->pParent->cost;

        float min = acrossDiagonal;
        node->north = false;
        node->west = true;
        if( diagonalAcross < min )
        {
            node->north = true;
            node->west = true;
            min = diagonalAcross;
        }
        if( diagonalDiagonal < min )
        {
            node->north = true;
            node->west = true;
            min = diagonalDiagonal;
        }
        if( diagonalUp < min )
        {
            node->north = true;
            node->west = true;
            min = diagonalUp;
        }
        return min;
    }

    // if the point is (X>2, m_NoDownPoints) calculate back and diagonal
    if( ( node->coordinates.x > 2) && ( node->coordinates.y == m_NoDownPoints ) )
    {
        PATH * bendingPoint;
        PATH * diagonalPoint;

        float   diagonalUp;
        float   diagonalDiagonal;
        float   diagonalAcross;
        float   acrossDiagonal;
        float   straight;
```

lyit | Institiúid Teicneolaíochta Leitir Ceanainn
Letterkenny Institute of Technology

```
        bendingPoint = findNode((int)node->coordinates.x - 1,(int)m_NoDownPoints - 2);
        diagonalPoint = findNode((int)node->coordinates.x - 1, (int)m_NoDownPoints - 1);

        diagonalUp = ( stretchingWork(node->I, node->J, diagonalPoint->I, diagonalPoint->J)  * stretchWeight ) +
            ( bendingWork2(node->I, diagonalPoint->I, bendingPoint->I, node->J, diagonalPoint->J, bendingPoint->J) * bendWeight )
            + diagonalPoint->cost;
        diagonalDiagonal = ( stretchingWork(node->I, node->J, diagonalPoint->I, diagonalPoint->J)  * stretchWeight ) +
            ( bendingWork2(node->I, diagonalPoint->I, bendingPoint->pParent->I, node->J, diagonalPoint->J, bendingPoint->pParent->J)↙
 * bendWeight )
            + diagonalPoint->cost;
        diagonalAcross = ( stretchingWork(node->I, node->J, diagonalPoint->I, diagonalPoint->J)  * stretchWeight ) +
            ( bendingWork2(node->I, diagonalPoint->I, diagonalPoint->pParent->I, node->J, diagonalPoint->J, diagonalPoint->pParent↙
->J) * bendWeight )
            + diagonalPoint->cost;
        acrossDiagonal = ( stretchingWork(node->I, node->J, node->pParent->I, node->pParent->J)  * stretchWeight ) +
            ( bendingWork2(node->I, node->pParent->I, diagonalPoint->pParent->I, node->J, node->pParent->J, diagonalPoint->pParent->↙
J) * bendWeight )
            + node->pParent->cost;
        straight = ( stretchingWork(node->I, node->J, node->pParent->I, node->pParent->J)  * stretchWeight ) +
            ( bendingWork2(node->I, node->pParent->I, node->pParent->pParent->I, node->J, node->pParent->J, node->pParent->pParent->J)↙
 * bendWeight )
            + node->pParent->cost;

        float min = acrossDiagonal;
        node->north = false;
        node->west = true;
        if( diagonalAcross < min )
        {
            node->north = true;
            node->west = true;
            min = diagonalAcross;
        }
        if( diagonalDiagonal < min )
        {
            node->north = true;
            node->west = true;
            min = diagonalDiagonal;
        }
        if( diagonalUp < min )
        {
            node->north = true;
            node->west = true;
            min = diagonalUp;
        }
        if(straight < min )
        {
            node->north = false;
            node->west = true;
```

lyit | Institiúid Teicneolaíochta Leitir Ceanainn Letterkenny Institute of Technology

```
            min = straight;
        }
        return min;
    }

    // if the point is (m_NoAcrossPoints, 2)...
    if( ( node->coordinates.x == m_NoAcrossPoints) && ( node->coordinates.y == 2 ) )
    {
        PATH * up;
        PATH * bendingPoint;

        float upDiagonal;
        float diagionalUp;
        float diagionalDiagional;
        float diagionalAcross;

        up = findNode(m_NoAcrossPoints, 1);
        bendingPoint = (m_NoAcrossPoints - 1, 0 );

        upDiagonal = ( stretchingWork(node->I, node->J, up->I, up->J)  * stretchWeight ) +
            ( bendingWork2(node->I, up->I, bendingPoint->I, node->J, up->J, bendingPoint->J) * bendWeight )
            + up->cost;
        diagionalUp = ( stretchingWork(node->I, node->J, up->pParent->I, up->pParent->J)  * stretchWeight ) +
            ( bendingWork2(node->I, up->pParent->I, bendingPoint->I, node->J, up->pParent->J, bendingPoint->J) * bendWeight )
            + up->pParent->cost;
        diagionalDiagional = ( stretchingWork(node->I, node->J, up->pParent->I, up->pParent->J)  * stretchWeight ) +
            ( bendingWork2(node->I, up->pParent->I, bendingPoint->pParent->I, node->J, up->pParent->J, bendingPoint->pParent->J) *
    bendWeight )
            + up->pParent->cost;
        diagionalAcross = ( stretchingWork(node->I, node->J, up->pParent->I, up->pParent->J)  * stretchWeight ) +
            ( bendingWork2(node->I, up->pParent->I, up->pParent->pParent->I, node->J, up->pParent->J, up->pParent->pParent->J) *
    bendWeight )
            + up->pParent->cost;

        float min;
        min = diagionalAcross;
        node->north = true;
        node->west = true;
        if( diagionalDiagional < min )
        {
            node->north = true;
            node->west = true;
            min = diagionalDiagional;
        }
        if( diagionalUp < min )
        {
            node->north = true;
            node->west = true;
            min = diagionalUp;
```

lyit | Institiúid Teicneolaíochta Leitir Ceanainn Letterkenny Institute of Technology

```
        }
        if( upDiagonal < min )
        {
            node->north = true;
            node->west = false;
            min = upDiagonal;
        }
        return min;
    }

    // if point is (m_NoAcrossPoints, y > 2)...
    if( node->coordinates.x == m_NoAcrossPoints )
    {
        PATH * up;
        PATH * bendingPoint;

        float upUp;
        float upDiagonal;
        float diagonalUp;
        float diagonalDiagonal;
        float diagonalAcross;

        up = findNode(m_NoAcrossPoints, 1);
        bendingPoint = (m_NoAcrossPoints - 1, 0 );

        upUp = ( stretchingWork(node->I, node->J, up->I, up->J)  * stretchWeight ) +
            ( bendingWork2(node->I, up->I, bendingPoint->pNextNode->I, node->J, up->J, bendingPoint->pNextNode->J) * bendWeight )
            + up->cost;
        upDiagonal = ( stretchingWork(node->I, node->J, up->I, up->J)  * stretchWeight ) +
            ( bendingWork2(node->I, up->I, bendingPoint->I, node->J, up->J, bendingPoint->J) * bendWeight )
            + up->cost;
        diagonalUp = ( stretchingWork(node->I, node->J, up->pParent->I, up->pParent->J)  * stretchWeight ) +
            ( bendingWork2(node->I, up->pParent->I, bendingPoint->I, node->J, up->pParent->J, bendingPoint->J) * bendWeight )
            + up->pParent->cost;
        diagonalDiagonal = ( stretchingWork(node->I, node->J, up->pParent->I, up->pParent->J)  * stretchWeight ) +
            ( bendingWork2(node->I, up->pParent->I, bendingPoint->pParent->I, node->J, up->pParent->J, bendingPoint->pParent->J) *
    bendWeight )
            + up->pParent->cost;
        diagonalAcross = ( stretchingWork(node->I, node->J, up->pParent->I, up->pParent->J)  * stretchWeight ) +
            ( bendingWork2(node->I, up->pParent->I, up->pParent->pParent->I, node->J, up->pParent->J, up->pParent->pParent->J) *
    bendWeight )
            + up->pParent->cost;

        float min;
        min = upUp;
        node->north = true;
        node->west = false;
        if( diagonalAcross < min )
        {
```

Institiúid Teicneolaíochta Leitir Ceanainn
Letterkenny Institute of Technology
lyit

```
            node->north = true;
            node->west = true;
            min = diagionalAcross;
        }
        if( diagionalDiagional < min )
        {
            node->north = true;
            node->west = true;
            min = diagionalDiagional;
        }
        if( diagionalUp < min )
        {
            node->north = true;
            node->west = true;
            min = diagionalUp;
        }
        if( upDiagional < min )
        {
            node->north = true;
            node->west = false;
            min = upDiagional;
        }
        return min;
    }

    // if the point is on the second row (x, 2) there is no vertical stretching
    if( node->coordinates.y == 2 )
    {
        PATH * up;
        PATH * bendingPoint;

        float upDiagional;
        float diagionalUp;
        float diagionalDiagional;
        float diagionalAcross;
        float acrossAcross;

        up = findNode((int)node->coordinates.x, 1);
        bendingPoint = findNode((int)node->coordinates.x - 1, 0 );

        upDiagional = ( stretchingWork(node->I, node->J, up->I, up->J)  * stretchWeight ) +
            ( bendingWork2(node->I, up->I, bendingPoint->I, node->J, up->J, bendingPoint->J) * bendWeight )
            + up->cost;
        diagionalUp = ( stretchingWork(node->I, node->J, up->pParent->I, up->pParent->J)  * stretchWeight ) +
            ( bendingWork2(node->I, up->pParent->I, bendingPoint->I, node->J, up->pParent->J, bendingPoint->J) * bendWeight )
            +up->pParent->cost;
        diagionalDiagional = ( stretchingWork(node->I, node->J, up->pParent->I, up->pParent->J)  * stretchWeight ) +
            ( bendingWork2(node->I, up->pParent->I, bendingPoint->pParent->I, node->J, up->pParent->J, bendingPoint->pParent->J) *
    bendWeight )
```

lyit | Institiúid Teicneolaíochta Leitir Ceanainn
Letterkenny Institute of Technology

```
            + up->pParent->cost;
        diagionalAcross = ( stretchingWork(node->I, node->J, up->pParent->I, up->pParent->J)  * stretchWeight ) +
            ( bendingWork2(node->I, up->pParent->I, up->pParent->pParent->I, node->J, up->pParent->J, up->pParent->pParent->J) *
    bendWeight )
            + up->pParent->cost;
        acrossAcross = ( stretchingWork(node->I, node->J, node->pParent->I, node->pParent->J)  * stretchWeight ) +
            ( bendingWork2(node->I, node->pParent->I, node->pParent->pParent->I, node->J, node->pParent->J, node->pParent->pParent->J)
     * bendWeight )
            + node->pParent->cost;

        float min;
        min = diagionalAcross;
        node->north = true;
        node->west = true;
        if( diagionalDiagional < min )
        {
            node->north = true;
            node->west = true;
            min = diagionalDiagional;
        }
        if( diagionalUp < min )
        {
            node->north = true;
            node->west = true;
            min = diagionalUp;
        }
        if( upDiagional < min )
        {
            node->north = true;
            node->west = false;
            min = upDiagional;
        }
        if( acrossAcross < min )
        {
            node->north = false;
            node->west = false;
            min = acrossAcross;
        }
        return min;
    }

    // if the point is on the second column (2,y) there is no horizontal stretching.
    if( node->coordinates.x == 2 )
    {
        PATH * up;
        PATH * bendingPoint;

        float upUp;
        float upDiagional;
```

lyit | Institiúid Teicneolaíochta Leitir Ceanainn Letterkenny Institute of Technology

```
        float diagonalUp;
        float diagonalDiagonal;
        float diagonalAcross;
        float acrossDiagonal;

        up = findNode((int)node->coordinates.x, (int)node->coordinates.y - 1 );
        bendingPoint = findNode((int)node->coordinates.x - 1, (int)node->coordinates.y - 2 );

        upUp = ( stretchingWork(node->I, node->J, up->I, up->J)  * stretchWeight ) +
            ( bendingWork2(node->I, up->I, bendingPoint->pNextNode->I, node->J, up->J, bendingPoint->pNextNode->J) * bendWeight )
            + up->cost;
        upDiagonal = ( stretchingWork(node->I, node->J, up->I, up->J)  * stretchWeight ) +
            ( bendingWork2(node->I, up->I, bendingPoint->I, node->J, up->J, bendingPoint->J) * bendWeight )
            + up->cost;
        diagonalUp = ( stretchingWork(node->I, node->J, up->pParent->I, up->pParent->J)  * stretchWeight ) +
            ( bendingWork2(node->I, up->pParent->I, bendingPoint->I, node->J, up->pParent->J, bendingPoint->J) * bendWeight )
            + up->pParent->cost;
        diagonalDiagonal = ( stretchingWork(node->I, node->J, up->pParent->I, up->pParent->J)  * stretchWeight ) +
            ( bendingWork2(node->I, up->pParent->I, bendingPoint->pParent->I, node->J, up->pParent->J, bendingPoint->pParent->J) *
    bendWeight )
            + up->pParent->cost;
        diagonalAcross = ( stretchingWork(node->I, node->J, up->pParent->I, up->pParent->J)  * stretchWeight ) +
            ( bendingWork2(node->I, up->pParent->I, up->pParent->pParent->I, node->J, up->pParent->J, up->pParent->pParent->J) *
    bendWeight )
            + up->pParent->cost;
        acrossDiagonal = ( stretchingWork(node->I, node->J, node->pParent->I, node->pParent->J)  * stretchWeight ) +
            ( bendingWork2(node->I, node->pParent->I, up->pParent->pParent->I, node->J, node->pParent->J, up->pParent->pParent->J) *
    bendWeight )
            + node->pParent->cost;

        float min;
        min = upUp;
        node->north = true;
        node->west = false;
        if( diagonalAcross < min )
        {
            node->north = true;
            node->west = true;
            min = diagonalAcross;
        }
        if( diagonalDiagonal < min )
        {
            node->north = true;
            node->west = true;
            min = diagonalDiagonal;
        }
        if( diagonalUp < min )
        {
            node->north = true;
```

Institiúid Teicneolaíochta Leitir Ceanainn | Letterkenny Institute of Technology lyit

```
            node->west = true;
            min = diagonalUp;
        }
        if( upDiagonal < min )
        {
            node->north = true;
            node->west = false;
            min = upDiagonal;
        }
        if( acrossDiagonal < min )
        {
            node->north = false;
            node->west = true;
            min = acrossDiagonal;
        }
        return min;
    }

        PATH * bendingPoint;
        PATH * diagonalPoint;

        float upUp;
        float upDiagonal;
        float diagonalUp;
        float diagonalDiagonal;
        float diagonalAcross;
        float acrossDiagonal;
        float acrossAcross;

        bendingPoint = findNode((int)node->coordinates.x - 1, (int)node->coordinates.y - 2);
        diagonalPoint = findNode((int)node->coordinates.x - 1, (int)node->coordinates.y - 1);

        upUp = ( stretchingWork(node->I, node->J, diagonalPoint->pNextNode->I, diagonalPoint->pNextNode->J)  * stretchWeight ) +
            ( bendingWork2(node->I, diagonalPoint->pNextNode->I, bendingPoint->pNextNode->I, node->J, diagonalPoint->pNextNode->J, ↙
    bendingPoint->pNextNode->J) * bendWeight )
            + diagonalPoint->pNextNode->cost;
        upDiagonal = ( stretchingWork(node->I, node->J, diagonalPoint->pNextNode->I, diagonalPoint->pNextNode->J)  * stretchWeight ↙
     ) +
            ( bendingWork2(node->I, diagonalPoint->pNextNode->I, bendingPoint->I, node->J, diagonalPoint->pNextNode->J, bendingPoint↙
    ->J) * bendWeight )
            + diagonalPoint->pNextNode->cost;
        diagonalUp = ( stretchingWork(node->I, node->J, diagonalPoint->I, diagonalPoint->J)  * stretchWeight ) +
            ( bendingWork2(node->I, diagonalPoint->I, bendingPoint->I, node->J, diagonalPoint->J, bendingPoint->J) * bendWeight )
            + diagonalPoint->cost;
        diagonalDiagonal = ( stretchingWork(node->I, node->J, diagonalPoint->I, diagonalPoint->J)  * stretchWeight ) +
            ( bendingWork2(node->I, diagonalPoint->I, bendingPoint->pParent->I, node->J, diagonalPoint->J, bendingPoint->pParent->J)↙
     * bendWeight )
            + diagonalPoint->cost;
        diagonalAcross = ( stretchingWork(node->I, node->J, diagonalPoint->I, diagonalPoint->J)  * stretchWeight ) +
```

Institiúid Teicneolaíochta Leitir Ceanainn
Letterkenny Institute of Technology

```
            ( bendingWork2(node->I, diagonalPoint->I, diagonalPoint->pParent->I, node->J, diagonalPoint->J, diagonalPoint->pParent↙
    ->J) * bendWeight )
            + diagonalPoint->cost;
        acrossDiagonal = ( stretchingWork(node->I, node->J, node->pParent->I, node->pParent->J)  * stretchWeight ) +
            ( bendingWork2(node->I, node->pParent->I, diagonalPoint->pParent->I, node->J, node->pParent->J, diagonalPoint->pParent->↙
    J) * bendWeight )
            + node->pParent->cost;
        acrossAcross = ( stretchingWork(node->I, node->J, node->pParent->I, node->pParent->J)  * stretchWeight ) +
            ( bendingWork2(node->I, node->pParent->I, node->pParent->pParent->I, node->J, node->pParent->J, node->pParent->pParent->J)↙
     * bendWeight )
            + node->pParent->cost;

        float min = upUp;
        node->north = true;
        node->west = false;
        if( upDiagonal < min )
        {
            node->north = true;
            node->west = false;
            min = upDiagonal;
        }
        if( diagonalUp < min )
        {
            node->north = true;
            node->west = true;
            min = diagonalUp;
        }
        if( diagonalDiagonal < min )
        {
            node->north = true;
            node->west = true;
            min = diagonalDiagonal;
        }
        if( diagonalAcross < min )
        {
            node->north = true;
            node->west = true;
            min = diagonalAcross;
        }
        if( acrossDiagonal < min )
        {
            node->north = false;
            node->west = true;
            min = acrossDiagonal;
        }
        if( acrossAcross < min )
        {
            node->north = false;
            node->west = true;
```

Institiúid Teicneolaíochta Leitir Ceanainn | Letterkenny Institute of Technology lyit

```
            min = acrossAcross;
        }
        return min;
}


//////////////////////////////////////////////////////////////////////////////////////////
//
// CSederberg::stretchingWork
//      Takes in 4 points and returns the work required to stretch one line to the other
//
//////////////////////////////////////////////////////////////////////////////////////////

float CSederberg::stretchingWork(MYPOINT a, MYPOINT b, MYPOINT c, MYPOINT d)
{
    const float Cs = 0.5f;
    // This is a constant and is used so the bending work doesn't have
    // too great a say in the algorithm.
    const float Ks = .1f;

    // the exponential to infer a degree of elasticity into the stretch
    const float Es = 1;

    float work; // the value to be returned.
    float L0;
    float L1;

    // need to multiply up the y values by about 10,000 otherwise they don't
    // have an impact on the length of a segment and the work value comes out
    // at almost 0, as both lines will be pretty much the same length.
    float average = ( a.y + b.y + c.y + d.y ) / 4;
    int count = 0;
    int scale = 1;
    int intAverage = (int)average;

    if( average != 0 )
    {
        while ( intAverage == 0  )
        {
            average = average * 10;
            intAverage = (int)average;
            count++;
        }
    }
    if ( count != 0 )
    {
        scale = pow(10, count);
    }
```

Institiúid Teicneolaíochta Leitir Ceanainn
Letterkenny Institute of Technology
lyit

```
    a.y = a.y * scale;
    b.y = b.y * scale;
    c.y = c.y * scale;
    d.y = d.y * scale;

    L0 = sqrt( pow( a.x - c.x, 2 ) + pow( a.y - c.y, 2 ) );
    L1 = sqrt( pow( b.x - d.x, 2 ) + pow( b.y - d.y, 2 ) );
    // work function

    work = fabs(L0 - L1)/5;

    if( work < 0 )
    {
        work = work * - 1;
    }
    return work;
}


/////////////////////////////////////////////////////////////////////////////////////////
//
//  CSederberg::plotPath
//      Keeps making new nodes and calculating their cost.
//
/////////////////////////////////////////////////////////////////////////////////////////

void CSederberg::plotPath()
{
    m_pNewPathNode = new PATH[m_NoDownPoints * m_NoAcrossPoints];
    for(int j = 0; j < m_NoDownPoints; j++)
    {
        for(int i = 0; i < m_NoAcrossPoints; i++)
        {
            m_pNewPathNode[ ( j * m_NoAcrossPoints ) + i ].I.x = m_pAcross[i].x;
            m_pNewPathNode[ ( j * m_NoAcrossPoints ) + i ].I.y = m_pAcross[i].y;
            m_pNewPathNode[ ( j * m_NoAcrossPoints ) + i ].J.x = m_pDown[j].x;
            m_pNewPathNode[ ( j * m_NoAcrossPoints ) + i ].J.y = m_pDown[j].y;

            m_pNewPathNode[ ( j * m_NoAcrossPoints ) + i ].coordinates.x = (float)i;
            m_pNewPathNode[ ( j * m_NoAcrossPoints ) + i ].coordinates.y = (float)j;

            // now insert the new node before the tail
            if( ( ( j * m_NoAcrossPoints ) + i ) == 0 )
            {
                m_pGridPath->pNextNode = m_pNewPathNode;
                m_pNewPathNode[0].pParent = m_pGridPath;
                m_pNewPathNode[0].pNextNode = m_pGridEnd;
                m_pGridEnd->pParent = &m_pNewPathNode[0];
            }
```

Institiúid Teicneolaíochta Leitir Ceanainn
Letterkenny Institute of Technology

```
            else
            {
                if( ( ( j * m_NoAcrossPoints ) + i ) == ( m_NoDownPoints * m_NoAcrossPoints ) )
                {
                    m_pGridEnd->pParent = &m_pNewPathNode[m_NoDownPoints * m_NoAcrossPoints];
                    m_pNewPathNode[ ( j * m_NoAcrossPoints ) + i ].pParent = &m_pNewPathNode[ ( j * m_NoAcrossPoints ) + i - 1 ];
                    m_pNewPathNode[ ( j * m_NoAcrossPoints ) + i - 1 ].pNextNode = &m_pNewPathNode[ ( j * m_NoAcrossPoints ) + i ];
                    m_pNewPathNode[ ( j * m_NoAcrossPoints ) + i ].pNextNode = m_pGridEnd;
                }
                else
                {
                    m_pNewPathNode[ ( j * m_NoAcrossPoints ) + i ].pParent = &m_pNewPathNode[ ( j * m_NoAcrossPoints - 1 ) + i ];
                    m_pNewPathNode[ ( j * m_NoAcrossPoints - 1 ) + i ].pNextNode = &m_pNewPathNode[ ( j * m_NoAcrossPoints ) + i ];
                    m_pNewPathNode[ ( j * m_NoAcrossPoints ) + i ].pNextNode = m_pGridEnd;
                    m_pGridEnd->pParent = &m_pNewPathNode[ ( j * m_NoAcrossPoints ) + i ];
                }
            }
            m_pGridEnd->pParent->cost = ( calculateWork(m_pGridEnd->pParent) );
        }
    }

    PATH * currentNode = m_pGridPath->pNextNode;
    // if the pNextNode is 0, we have reached the tail node.
    while( currentNode->pNextNode != 0 )
    {
        currentNode->cost = currentNode->cost;
        gridInfo<<currentNode->cost<<",";
        if( currentNode->coordinates.y != currentNode->pNextNode->coordinates.y )
        {
            gridInfo<<"\n";
        }
        currentNode = currentNode->pNextNode;
    }
    gridInfo<<"\n";

    // reuse current node.
    float cost = 0.0f;
    PATH * costingNode;

    currentNode = m_pGridEnd->pParent;
    do
    {
        // no need to check work values - must go in a certain direction
        if( ( currentNode->coordinates.x == 0 ) && ( currentNode->coordinates.y == 0 ) )
        {
            // at the (0,0) position - do nothing;
        }
        if( currentNode->coordinates.x == 0 )
        {
```

Institiúid Teicneolaíochta Leitir Ceanainn | Letterkenny Institute of Technology lyit

```
            // at the side of the grid - force up
            currentNode->north = true;
            currentNode->west = false;
            m_vOptimalPath.push_back(currentNode->coordinates);
        }
        if( currentNode->coordinates.y == 0 )
        {
            // at the top of the grid - force across
            currentNode->north = false;
            currentNode->west = true;
            m_vOptimalPath.push_back(currentNode->coordinates);
        }
        if( ( currentNode->coordinates.x == 1 ) && ( currentNode->coordinates.y == 1 ) )
        {
            // at position (1,1) - force diagional
            currentNode->north = true;
            currentNode->west = true;
            m_vOptimalPath.push_back(currentNode->coordinates);
        }
        // must check work values - direction is unset.
        if( ( currentNode->coordinates.x == 1 ) && ( currentNode->coordinates.y == 0 ) )
        {
            // need to check diagional and up
            // diagional
            costingNode = findNode(currentNode->coordinates.x - 1, currentNode->coordinates.y - 1);
            cost = costingNode->cost;
            currentNode->north = true;
            currentNode->west = true;
            costingNode = findNode(currentNode->coordinates.x, currentNode->coordinates.y - 1);
            if( costingNode->cost < cost )
            {
                currentNode->west = false;
            }
            m_vOptimalPath.push_back(currentNode->coordinates);
        }

        if( ( currentNode->coordinates.x == 0 ) && ( currentNode->coordinates.y == 1 ) )
        {
            // need to check diagional and side
            // diagional
            costingNode = findNode(currentNode->coordinates.x - 1, currentNode->coordinates.y - 1);
            cost = costingNode->cost;
            currentNode->north = true;
            currentNode->west = true;
            costingNode = findNode(currentNode->coordinates.x - 1, currentNode->coordinates.y);
            if( costingNode->cost < cost )
            {
                //cost = costingNode->cost;
                currentNode->north = false;
```

Institiúid Teicneolaíochta Leitir Ceanainn | Letterkenny Institute of Technology lyit

```
        }
        m_vOptimalPath.push_back(currentNode->coordinates);
    }
    MYPOINT previousPoint;
    bool across = true;
    bool up = true;
    // we need to check if the choice of the next node is restriced by an up or across movement.
    // however, if we are just starting, this won't be an issue. The start is detected by looking
    // at the next node variable of the current node - this will point to the end node on startup.
    if(currentNode->pNextNode == m_pGridEnd)
    {
        // need to check diagonal, side and up
        // diagonal
        costingNode = findNode(currentNode->coordinates.x - 1, currentNode->coordinates.y - 1);
        cost = costingNode->cost;
        currentNode->north = true;
        currentNode->west = true;
        //up
        costingNode = findNode(currentNode->coordinates.x, currentNode->coordinates.y - 1);
        if( costingNode->cost < cost )
        {
            cost = costingNode->cost;
            currentNode->west = false;
        }
        // side
        costingNode = findNode(currentNode->coordinates.x - 1, currentNode->coordinates.y);
        if( costingNode->cost < cost )
        {
            currentNode->north = false;
            currentNode->west = true;
        }
        m_vOptimalPath.push_back(currentNode->coordinates);
    }
    else
    {
        if( ( currentNode->coordinates.x > 1 ) && ( currentNode->coordinates.y > 1 ) )
        {
            // before doing anything - check what the previous point was - from this decide if either
            // up or across moves are illegal.
            previousPoint = m_vOptimalPath[m_vOptimalPath.size() - 1];
            // Diagonal
            if( ( ( previousPoint.x - 1 ) == currentNode->coordinates.x ) && ( ( previousPoint.y - 1 ) == currentNode->coordinates↙
.y ) )
            {
                across = true;
                up = true;
            }
            if( ( ( previousPoint.x ) == currentNode->coordinates.x ) && ( ( previousPoint.y - 1 ) == currentNode->coordinates.y )↙
```

Institiúid Teicneolaíochta Leitir Ceanainn Letterkenny Institute of Technology lyit

```
            {
                across = false;
                up = true;
            }
            if( ( ( previousPoint.x - 1 ) == currentNode->coordinates.x ) && ( ( previousPoint.y ) == currentNode->coordinates.y )
    )
            {
                across = true;
                up = false;
            }


            // need to check diagonal, side and up
            // diagonal
            costingNode = findNode(currentNode->coordinates.x - 1, currentNode->coordinates.y - 1);
            cost = costingNode->cost;
            currentNode->north = true;
            currentNode->west = true;
            //up
            if( up == true )
            {
                costingNode = findNode(currentNode->coordinates.x, currentNode->coordinates.y - 1);
                if( costingNode->cost < cost )
                {
                    cost = costingNode->cost;
                    currentNode->west = false;
                }
            }
            // side
            if( across == true )
            {
                costingNode = findNode(currentNode->coordinates.x - 1, currentNode->coordinates.y);
                if( costingNode->cost < cost )
                {
                    currentNode->north = false;
                    currentNode->west = true;
                }
            }
            m_vOptimalPath.push_back(currentNode->coordinates);
        }
        // if the point has x < 1 or y < 1 it needs to be forced along to the origin.
        else
        {
            if( ( currentNode->coordinates.x == 1 ) && ( currentNode->coordinates.y > 1 ) )
            {
                currentNode->north = true;
                currentNode->west = false;
                m_vOptimalPath.push_back(currentNode->coordinates);
            }
```

Institiúid Teicneolaíochta Leitir Ceanainn
Letterkenny Institute of Technology
lyit

```
                else
                {
                    if ( ( currentNode->coordinates.x > 1 ) && ( currentNode->coordinates.y == 1 ) )
                    {
                        currentNode->north = false;
                        currentNode->west = true;
                        m_vOptimalPath.push_back(currentNode->coordinates);
                    }
                }
            }
        }
        // move the currentNode on to the new cheapest node
        if( ( currentNode->north == true ) && ( currentNode->west == true ) )
        {
            currentNode = findNode(currentNode->coordinates.x - 1, currentNode->coordinates.y - 1);
        }
        else
        {
            if ( ( currentNode->north == true ) && ( currentNode->west == false ) )
            {
                currentNode = findNode(currentNode->coordinates.x, currentNode->coordinates.y - 1);
            }
            else
            {
                if ( ( currentNode->north == false ) && ( currentNode->west == true ) )
                {
                    currentNode = findNode(currentNode->coordinates.x - 1, currentNode->coordinates.y );
                }
            }
        }
    }
    while ( ( currentNode->coordinates.x != 0 ) && ( currentNode->coordinates.y != 0) );
    if( ( currentNode->coordinates.x == 0 ) && ( currentNode->coordinates.y == 0 ) )
    {
        // at the (0,0) position - push this point onto the queue
        m_vOptimalPath.push_back(currentNode->coordinates);
    }
}


////////////////////////////////////////////////////////////////////////////////////////////
//
//  CSederberg::insertPathNode
//      Handles next node pointers and parent pointers when a new node is instered.
//
////////////////////////////////////////////////////////////////////////////////////////////

void CSederberg::insertPathNode(PATH *pPreviousNode, PATH * pNewNode)
{
```

lyit | Institiúid Teicneolaíochta Leitir Ceanainn Letterkenny Institute of Technology

```
    PATH * pNextTemp;
    PATH * pParentTemp;

    pNextTemp = pPreviousNode->pNextNode;
    pParentTemp = pPreviousNode;
    pPreviousNode->pNextNode = pNewNode;
    pNewNode->pNextNode = pNextTemp;
    pNewNode->pParent = pParentTemp;
    pNextTemp->pParent = pNewNode;
}


/////////////////////////////////////////////////////////////////////////////////////////
//
//  CSederberg::findNode
//      Returns a pointer to the node at (x,y)
/////////////////////////////////////////////////////////////////////////////////////////

PATH * CSederberg::findNode(int x, int y)
{
    return &m_pNewPathNode[ ( m_NoAcrossPoints * y ) + x ];
}


/////////////////////////////////////////////////////////////////////////////////////////
//
// CSederberg::findOptimalPath
//  Fills a vector array with the optimal path from (0,0) to (m_NoAcrossPoints, m_NoDownPoints)
//
/////////////////////////////////////////////////////////////////////////////////////////

void CSederberg::findOptimalPath()
{
    if(m_vOptimalPath.empty() != true )
    {
        m_vOptimalPath.clear();
    }
    if(m_vPathResult.empty() != true )
    {
        m_vPathResult.clear();
    }
    DIRECTION immediate = diagional;
    DIRECTION further = diagional;

    // this puts all the points in and gets the best path through them.
    plotPath();

    // empty the function vector
    m_vFunctionVector.clear();
```

Institiúid Teicneolaíochta Leitir Ceanainn
lyit | Letterkenny Institute of Technology

```
    int size = (int)m_vOptimalPath.size();

    MYPOINT result;

    int outputPosition = 0;
    int pathPosition = 1;

    // left moves recordes how many times we move left on the grid, so we know which across point
    // to map to a down point in the case of a substution.
    int leftMoves = 0;
    // similaryly for moving down
    int downMoves = 0;

    bool exit = false;

    do
    {
        // clear the function vector
        m_vFunctionVector.clear();
        // diagional
        MYPOINT diagPoint1;
        MYPOINT diagPoint2;
        diagPoint1 = m_vOptimalPath[size - 1 - pathPosition];
        diagPoint2 = m_vOptimalPath[size - pathPosition];
        if( ( diagPoint1.x == diagPoint2.x + 1 )
            && ( diagPoint1.y == diagPoint2.y + 1 ) )
        {

            // This is where merging the pose and the animation signals occurs
            // Currently, it is set to shift the animation about the pose
            // A 50/50 blend should be coded here.
            // shifting
            // get the average value of the down signal
            float average = 0.0f;
            for( int i = 0; i < m_NoDownPoints; i++ )
            {
                average = average + m_pDown[i].y;
            }
            average = average/m_NoDownPoints;
            float shift = m_pAcross[0].y - average;

            // next point is on a diagional - straight swap
            MYPOINT pointToPush;
            pointToPush.x = m_pAcross[(int)m_vOptimalPath[size - pathPosition].x].x;
            //Put in the shift line when mixing a pose with an animation
            pointToPush.y = m_pDown[(int)m_vOptimalPath[size - pathPosition].x].y + shift;
            m_vPathResult.push_back(pointToPush);
```

Institiúid Teicneolaíochta Leitir Ceanainn | Letterkenny Institute of Technology lyit

```
            pathPosition++;
        }

        // across
        else
        {
            MYPOINT acrossPoint1;
            MYPOINT acrossPoint2;
            MYPOINT pushingPoint;
            acrossPoint1 = m_vOptimalPath[size - 1 - pathPosition];
            acrossPoint2 = m_vOptimalPath[size - pathPosition];
            if( ( acrossPoint1.x == acrossPoint2.x + 1 )
                && ( acrossPoint1.y == acrossPoint2.y ) )
            {
                // push the point to the averaging vector
                pushingPoint = m_pAcross[(int)acrossPoint2.x/*leftMoves*/];
                m_vFunctionVector.push_back(pushingPoint);

                // while the next point is across as well, push it to the fuction vector
                do
                {
                    // push point to function vector
                    pushingPoint = m_pAcross[(int)m_vOptimalPath[size - 1 - pathPosition].x];
                    m_vFunctionVector.push_back(pushingPoint);
                    pathPosition++;
                    if( ( size - pathPosition ) == 0 )
                    {
                        exit = true;
                    }
                    else
                    {
                        if ( ( m_vOptimalPath[size - 1 - pathPosition].x == m_vOptimalPath[size - pathPosition].x + 1 )
                           && ( m_vOptimalPath[size - 1 - pathPosition].y == m_vOptimalPath[size - pathPosition].y ) )
                        {
                            exit = false;
                        }
                        else
                        {
                            exit = true;
                        }
                    }
                } while( exit == false );
                exit = false;
                result = average(m_vFunctionVector);
                m_vPathResult.push_back(result);
                m_vFunctionVector.clear();
                pathPosition++;
            }
            // down
```

lyit Institiúid Teicneolaíochta Leitir Ceanainn | Letterkenny Institute of Technology

```
        else
        {
            bool twoOr4 = false;
            m_vFunctionVector.clear();
            MYPOINT downPoint1 = m_vOptimalPath[size - 1 - pathPosition];
            MYPOINT downPoint2 = m_vOptimalPath[size - pathPosition];
                if( ( downPoint1.x == downPoint2.x )
                 && ( downPoint1.y == downPoint2.y + 1 ) )
                {
                    m_vFunctionVector.push_back(m_pAcross[(int)downPoint2.x - 1 ]);

                    if ( (int)downPoint2.x < m_NoAcrossPoints )
                    {
                        m_vFunctionVector.push_back(m_pAcross[(int)downPoint2.x]);
                    }
                    else
                    {
                        m_vFunctionVector.push_back(m_pAcross[(int)downPoint2.x - 1]);
                    }
                    // check that the value is in range

                    twoOr4 = true;
                    if( (int)downPoint2.x + 1 < m_NoAcrossPoints )
                    {
                        m_vFunctionVector.push_back(m_pAcross[(int)downPoint2.x + 1 ]);
                    }
                    else
                    {
                        m_vFunctionVector.push_back(m_pAcross[(int)downPoint2.x - 1]);
                    }

                    // need to scale up the y values so they aren't considers a straight line
                    // first, get the average of the points
                    int count = 0;
                    int scale = 1;
                    float average = 0.0f;
                    bool scaledEnough = false;
                    bool abort = false;
                    int r;
                    for(r = 0; r < m_vFunctionVector.size(); ++r)
                    {
                        float number = m_vFunctionVector[r].y;
                        if( ( number < 0.0000001) && ( number > -0.0000001 ) )
                        {
                            abort = true;
                        }
                    }
                    if(abort == false)
                    {
```

```
        do
        {
            for ( int q = 0; q < m_vFunctionVector.size(); q++ )
            {
                average = m_vFunctionVector[q].y * pow(10, count);
                if ( average > 1.000000f )
                {
                    scaledEnough = true;
                }
            }
            if( scaledEnough == false )
            {
                count++;
            }
        }
        while( scaledEnough == false );
    }
    // need to reduce the value of count by 1, as it has been increased once after the
    // counting was meant to finish.
    if(abort == true)
    {
        scale = 0;
    }
    else
    {
        scale = count;
    }
    scale = pow(10.0f,scale);
    for( int s = 0; s < m_vFunctionVector.size(); s++ )
    {
        m_vFunctionVector[s].y = m_vFunctionVector[s].y * scale;
    }

    // this should return a bspline
    // figure out how many down moves are carried out
    // insert each down move into the bspline and get a value out
    // this value goes into the result
    bSpline cSpline;
    // this gets the bSpline object
    // list out the values of the m_vFunction vector - for debugging only
    MYPOINT firstSplinePoint = m_vFunctionVector[0];
    MYPOINT secondSplinePoint = m_vFunctionVector[1];
    MYPOINT thirdSplinePoint = m_vFunctionVector[2];
    cSpline = getBSpline(m_vFunctionVector);
    int noUps = 1;
    float increment = 0.0f;
    MYPOINT splinePoint;
    do
    {
```

Institiúid Teicneolaíochta Leitir Ceanainn
Letterkenny Institute of Technology

```
        pathPosition++;
        noUps++;
        if( ( size - pathPosition ) == 0 )
        {
            exit = true;
        }
        else
        {
            if( ( m_vOptimalPath[size - 1 - pathPosition].x == m_vOptimalPath[size - pathPosition].x )
              && ( m_vOptimalPath[size - 1 - pathPosition].y == m_vOptimalPath[size - pathPosition].y + 1 ) )
            {
                exit = false;
            }
            else
            {
                exit = true;
            }
        }

    } while( exit == false );
    exit = false;
    // noUps has the number of up moves.
    // this should be divided into 3
    increment = 3/(float)noUps;

    // previous x/time value - this is needed as the spline function returns values between
    // 0 and 3. This will mess up the time of the samples, so they need to be offset by
    // the time of the sample before the b spline occurs.
    // This time should be set to the value that was last pushed to the result vector.

    int previousTimeSize = m_vPathResult.size();
    float previousTime = m_vPathResult[previousTimeSize - 1].x;
    float functionVectorSize = m_vFunctionVector.size();

    // take values from the spline here
    float UValue = 2.0f;
    // the number of points is one less than the number of up moves because
    // the first point in the up moves is a diagonal move from the previous point
    // and as such, is treated as a diagonal move.
    int numberOfOutputPoints = noUps ;
    if ( twoOr4 == true )
    {
        // subtract 2 from the number of elements in the function vector

        numberOfOutputPoints;
        increment = 1.0f/*functionVectorSize*/ / ( numberOfOutputPoints + 1 );
    }
    else
    {
```

Institiúid Teicneolaíochta Leitir Ceanainn | Letterkenny Institute of Technology

```
                            // subtract 4 from the number of elements in the function vector
                            numberOfOutputPoints;
                            increment = functionVectorSize / ( numberOfOutputPoints + 1 );
                        }
                        twoOr4 = false;
                        int pointsOutput = 0;
                        while ( pointsOutput != numberOfOutputPoints )
                        {
                            splinePoint = cSpline.getUResult(UValue);
                            // undo the scale
                            splinePoint.y = splinePoint.y / scale;
                            // debug variables
                             float endTime = m_vFunctionVector[m_vFunctionVector.size() - 1].x;
                             float startTime = m_vFunctionVector[2].x;
                            // end debug variables

                            m_vPathResult.push_back(splinePoint);
                            pointsOutput++;
                            UValue = UValue + increment;
                        }
                    }
                    pathPosition++;
                }
            }
        }
        while ( pathPosition < size - 1 );

        // push a straight average of the last 2 points.
        MYPOINT lastResultPoint;
        // The time of the last point will be the same as the time of one of the last samples
        // The rotation of the last point will be an average of the 2 last samples.
        lastResultPoint.x =  m_pAcross[m_NoAcrossPoints -1].x;
        lastResultPoint.y = ( m_pAcross[m_NoAcrossPoints - 1].y + m_pDown[m_NoDownPoints - 1].y) / 2;
        // The range of values in m_vPathResult needs to be 0-31, but its 0-30, so taking an average between point 29,
        // and the value destined for 30, and putting that in 30 and the point for 30 in 31.
        int pathsize = m_vPathResult.size();
        float x = m_vPathResult[pathsize-1].x;
        float y = m_vPathResult[pathsize-1].y;
        MYPOINT average;
        // x holds the time of the second last sample.
        average.x = x;
        average.y = ( y + lastResultPoint.y ) / 2;
        m_vPathResult.push_back(average);
        m_vPathResult.push_back(lastResultPoint);
        int PathResultSize = m_vPathResult.size();
        m_vOptimalPath.clear();
}
```

Institiúid Teicneolaíochta Leitir Ceanainn
lyit | Letterkenny Institute of Technology

```
//////////////////////////////////////////////////////////////////////////////////////////
//
// CSederberg::average
//      Takes in a vector of numbers, and returns the average of the numbers
//
//////////////////////////////////////////////////////////////////////////////////////////

MYPOINT CSederberg::average(std::vector<MYPOINT> &value)
{
    int i = (int)value.size();
    float sumX = 0.0f;
    float sumY = 0.0f;
    for(int j = 0; j < i; j++)
    {
        sumX = sumX + value[j].x;
        sumY = sumY + value[j].y;
    }
    MYPOINT mean;
    mean.x = sumX/i;
    mean.y = sumY/i;
    return mean;
}


//////////////////////////////////////////////////////////////////////////////////////////
//
//  CSederberg::bSplineEvaluator
//      Takes in a vector of numbers and forms a bSpline from them.
//      Returns the center of the Spline.
//
//////////////////////////////////////////////////////////////////////////////////////////

MYPOINT CSederberg::bSplineEvaluator(std::vector<MYPOINT> &value)
{
    bSpline cSpline;

    // get the size of the vector
    int i = (int)value.size();
    // iterate through the value vector passing each point into a bspline object.
    for( int j = 0; j < i; j++ )
    {
        cSpline.enterControlPoint(value[j]);
    }
    return cSpline.getUResult((float)(i+1)/2);
}


//////////////////////////////////////////////////////////////////////////////////////////
//
```

lyit Institiúid Teicneolaíochta Leitir Ceanainn Letterkenny Institute of Technology

```
//  CSederberg::getBSpline
//      Takes in a vector of numbers and retuns a bSpline from them.
//
////////////////////////////////////////////////////////////////////////////////////////

bSpline  CSederberg::getBSpline(std::vector<MYPOINT> &value)
{
    bSpline cSpline;

    // get the size of the vector
    int i = (int)value.size();
    // iterate through the value vector passing each point into a bspline object.
    for( int j = 0; j < i; j++ )
    {
        cSpline.enterControlPoint(value[j]);
    }
    return cSpline;
}


////////////////////////////////////////////////////////////////////////////////////////
//
//  CSederberg::bendingWork2
//      Takes in 6 points, to form 2, 2 semgent lines. Gets the difference in angles
//      between the 2 lines.
//
////////////////////////////////////////////////////////////////////////////////////////

float  CSederberg::bendingWork2(MYPOINT a, MYPOINT b, MYPOINT c, MYPOINT d, MYPOINT e, MYPOINT f)
{
    // check if one point is a local minimum, and if the other is a local maximum
    // if this is the case, one of the angles needs to be inverted before getting
    // the cos as per the paper

    bool invert = false;
    bool firstmax = false;
    bool firstmin = false;
    bool secondmax = false;
    bool secondmin = false;
    if( ( b.y > a.y ) && ( b.y > c.y ) )
    {
        firstmax = true;
    }
    else
    {
        if ( ( b.y < a.y ) && ( b.y < c.y ) )
        {
            firstmin = true;
        }
```

lyit | Institiúid Teicneolaíochta Leitir Ceanainn | Letterkenny Institute of Technology

```
    }

    if( ( e.y > d.y ) && ( e.y > f.y ) )
    {
        secondmax = true;
    }
    else
    {
        if ( ( e.y < d.y ) && ( e.y < f.y ) )
        {
            secondmin = true;
        }
    }

    if ( ( firstmax == true ) && ( secondmin == true ) )
    {
        invert = true;
    }
    else
    {
        if ( ( firstmin == true ) && ( secondmax == true ) )
        {
            invert = true;
        }
        else
        {
            invert = false;
        }
    }

    float average = ( a.y + b.y + c.y + d.y + e.y + f.y ) / 6;
    int count = 0;
    int scale = 1;
    int intAverage = (int)average;

    if( average != 0 )
    {
        while ( intAverage == 0  )
        {
            average = average * 10;
            intAverage = (int)average;
            count++;
        }
    }
    if ( count != 0 )
    {
        scale = pow(10, count);
    }
```

Institiúid Teicneolaíochta Leitir Ceanainn
lyit | Letterkenny Institute of Technology

```
// scale up the y values by 100000 as they are too small
// when compared to the time between samples
//int scale = 10000;
a.y = a.y * scale;
b.y = b.y * scale;
c.y = c.y * scale;
d.y = d.y * scale;
e.y = e.y * scale;
f.y = f.y * scale;
// take the first 3 points, centre them at [0,0]
a.x = a.x - b.x;
a.y = ( a.y - b.y );

c.x = c.x - b.x;
c.y = ( c.y - b.y );

b.x = 0.0f;
b.y = 0.0f;

float hypOne = sqrt( ( a.x * a.x ) + ( a.y * a.y ) );
float cosOne;
float sinOne;
float firstAngle;
MYPOINT anglePoint;
// if the hyp is 0, a is at the origin. Assume its on the positive x axis giving
// cos = 1 and sin = 0.
if( FloatAlmostEquals(hypOne, 0.0f) )
{
    cosOne = 1.0f;
    sinOne = 0.0f;
    firstAngle = 3.1415920f;
}
else
{
    cosOne = a.x/hypOne;
    sinOne = a.y/hypOne;


    // anglePoint holds c after it has been rotated to refelect a being lined up with the x axis.

    anglePoint.x = ( cosOne * c.x ) + ( sinOne * c.y );
    anglePoint.y = ( -sinOne * c.x ) + ( cosOne * c.y );


    // FloatAlmostEquals(0.f, 0.f);
    // If the point lies on the y axis (x == 0) then the angle is 90 degrees.
    // this would be a divid by 0 - giving infinity, the atan of which is 90.
    if(FloatAlmostEquals(anglePoint.x, 0.0f))
```

Institiúid Teicneolaíochta Leitir Ceanainn
Letterkenny Institute of Technology
lyit

```
        {
            if(FloatAlmostEquals(anglePoint.y, 0.0f))
            {
                firstAngle = 3.1415920f; // 180 degrees
            }
            else
            {
                if(anglePoint.y > 0 )
                {
                    firstAngle = 1.7123889f; // 90 degrees in radians.
                }
                else
                {
                    firstAngle = 4.712388f; // y is negative, 270 degreese.
                }
            }
        }
        // point is not on the x axis, there will be no divide by 0.
        // need to suss out what quadrant the point is in.
        // first quadrant
        else
        {
            if( ( anglePoint.x > 0 ) && ( anglePoint.y >= 0 ) )
            {
                firstAngle = atan( anglePoint.y / anglePoint.x );
            }
            else
            {
                // second quadrant
                if( ( anglePoint.x < 0 ) && ( anglePoint.y >= 0 ) )
                {
                    firstAngle = atan( anglePoint.y / anglePoint.x ) + 3.1415920f;
                }
                else
                {
                    // third quadrant
                    if( ( anglePoint.x < 0 ) && ( anglePoint.y < 0 ) )
                    {
                        firstAngle = atan( anglePoint.y / anglePoint.x ) + 3.1415920f;
                    }
                    // forth quadrant
                    else
                    {
                        firstAngle = 6.2831853f - ( atan( anglePoint.y / anglePoint.x ) );
                    }
                }
            }
        }
    }
```

lyit | Institiúid Teicneolaíochta Leitir Ceanainn
Letterkenny Institute of Technology

```
    d.x = d.x - e.x;
    d.y = ( d.y - e.y );

    f.x = f.x - e.x;
    f.y = ( f.y - e.y );

    e.x = 0.0f;
    e.y = 0.0f;

    float hypTwo = sqrt( ( d.x * d.x ) + ( d.y * d.y ) );
    float cosTwo;
    float sinTwo;

    // if the hyp is 0, a is at the origin. Assume its on the posivite x axis giving
    // cos = 1 and sin = 0.
    float secondAngle = 0.0f;
    if( FloatAlmostEquals(hypTwo, 0.0f) )
    {
        cosTwo = 1.0f;
        sinTwo = 0.0f;
        secondAngle = 3.1415920f;
    }
    else
    {
        cosTwo = d.x/hypTwo;
        sinTwo = d.y/hypTwo;


        // anglePoint holds f after it has been rotated to refelect d being lined up with the x axis.
        anglePoint.x = ( cosTwo * f.x ) + ( sinTwo * f.y );
        anglePoint.y = ( -sinTwo * f.x ) + ( cosTwo * f.y );

        // If the point lies on the y axis (x == 0) then the angle is 90 degrees.
        // this would be a divid by 0 - giving infinity, the atan of which is 90.
        if(FloatAlmostEquals(anglePoint.x, 0.0f) )
        {
            if(FloatAlmostEquals(anglePoint.y, 0.0f) )
            {
                secondAngle = 3.1415920f; // 180 degrees
            }
            else
            {
                if(anglePoint.y > 0 )
                {
                    secondAngle = 1.5707963f; // 90 degrees in radians.
                }
                else
                {
                    secondAngle = 4.7123889f; // y is negative, 270 degreese.
```

lyit Institiúid Teicneolaíochta Leitir Ceanainn | Letterkenny Institute of Technology

```
                }
            }
        }
        // point is not on the x axis, there will be no divide by 0.
        // need to suss out what quadrant the point is in.
        // first quadrant
        else
        {
            if( ( anglePoint.x > 0 ) && ( anglePoint.y >= 0 ) )
            {
                secondAngle = atan( anglePoint.y / anglePoint.x );
            }
            else
            {
                // second quadrant
                if( ( anglePoint.x < 0 ) && ( anglePoint.y >= 0 ) )
                {
                    secondAngle = atan( anglePoint.y / anglePoint.x ) + 3.1415920f;
                }
                else
                {
                    // third quadrant
                    if( ( anglePoint.x < 0 ) && ( anglePoint.y < 0 ) )
                    {
                        secondAngle = atan( anglePoint.y / anglePoint.x ) + 3.1415920f;
                    }
                    // forth quadrant
                    else
                    {
                        secondAngle = 6.2831853f - ( atan( anglePoint.y / anglePoint.x ) );
                    }
                }
            }
        }
    }
    //negatate second angle on invert being true.
    if( invert == true )
    {
        secondAngle = secondAngle * -1;
    }

    float kb = 0.5f;
    float eb = 1.0f;
    float bendingCost = kb * pow( fabs( firstAngle - secondAngle ), eb );
    if( bendingCost < 0 )
    {
        bendingCost = bendingCost * -1;
    }
    return bendingCost;
```

Institiúid Teicneolaíochta Leitir Ceanainn
lyit | Letterkenny Institute of Technology

```
}


/////////////////////////////////////////////////////////////////////////////////////////
//
//  CSederberg::timeReassignment
//      Maps the points in the signal from the time warping to the times created in the
//      timewarping.
//
/////////////////////////////////////////////////////////////////////////////////////////

void CSederberg::timeReassignment(CMorphData &a, CMorphData &result)
{
    for ( int i = 0; i < a.m_noFrequencyBands; i++ )
    {
        for ( int j = 0; j < a.m_noBoneTracks; j++ )
        {
            for ( int k = 0; k < a.m_noSamples; k++ )
            {
                result.m_pBandPassArray[i].SignalBone[j].time[k] = a.m_pBandPassArray[i].SignalBone[j].time[k];
            }
        }
    }
}
} // end namespace IE
```

lyit | Institiúid Teicneolaíochta Leitir Ceanainn
Letterkenny Institute of Technology

```
// Confidential Information of Torc Interactive Limited.  This software contains code, techniques and know-how
// which is confidential and proprietary to Torc.  Not for disclosure or distribution without prior written consent.
// All Rights Reserved. Use of this software is subject to the terms of an end user license agreement.
// Instinct Engine (C) Copyright 2002/2005 Torc Interactive Limited.

#ifndef __CMWAPP_H__
#define __CMWAPP_H__

#include <ieCore/Application.h>
#include <ieCore/System.h>
#include <ieCore/IDiagnostics.h>
#include <ieCore/Utils/CEntityComponentRef.h>
#include <ieGraphics/IGraphics.h>
#include <ieInput/IInputManager.h>
#include <ieSound/IChannelManager.h>
#include <ieConsole/IConsole.h>
#include <ieModels/IBones.h>

#include <ieModels/IAnimation.h>
#include <ieMaths/Quaternion.h>
#include <ieMaths/Vector.h>

#include "CWarping.h"

namespace IE
{

class CMWApp : public Input::IInputEventHandler,
               public Models::IBonesController
{
public:

    CMWApp()
    :   m_pGraphics(0),
        m_pInput(0),
        m_pSoundChannelMgr(0),
        m_pConsole(0),
        m_NumBones(0),
        m_pRotations(0),
        m_pPositions(0),
        m_pBinding(0),
        m_pTimer(0)
    {}

    ieResult    update();

    ieResult    init();

    ieResult    shutdown();
```

lyit | Institiúid Teicneolaíochta Leitir Ceanainn Letterkenny Institute of Technology

```
private:

    ieResult    initEntityManager(IEntityManager * pEntMgr);

    ieResult    updateAnimations(float Time);
    ieResult    writeText();


    ieResult    Example1();

    ieResult    MotionWarpUpdate();

    //Input functions
    ieResult    handleInputEvent(ieConstStr szType,
                                 const IEvent * pEvent);

    bool        getKey(int key);

    //IBonesController
    virtual ieResult registeredController(Models::IBones* pBones);
    virtual ieResult unregisteredController(Models::IBones* pBones);
    virtual ieResult updateController(Models::IBones* pBones);
    virtual ieConstStr getControllerName() {return ieS("CMWApp");}

    //IBonesListener
    virtual ieResult handleBonesStructureChange(Models::IBones* pBones);
    virtual ieResult handleBonesChange(Models::IBones* pBones);
    virtual ieResult handleBonesShutdown(Models::IBones* pBones);

    Graphics::IGraphics *                          m_pGraphics;
    Input::IInputManager *                         m_pInput;
    Sound::IChannelManager *                       m_pSoundChannelMgr;
    Console::IConsole *                            m_pConsole;
    IDiagnostics *                                 m_pDiagnostics;
    CEntityComponentRef<Graphics::ISceneVisibility> m_SceneVis;

    IEntityManager *                               m_pEntityMgr;

    CEntityComponentRef<Models::IAnimation>        m_AnimWalk;
    CEntityComponentRef<Models::IAnimation>        m_AnimPose;
    CEntityComponentRef<Models::IBones>            m_Bones;

    ieUInt16                                       m_NumBones;
    VECTOR*                                        m_pPositions;
    QUATERNION*                                    m_pRotations;

    const ieInt16*                                 m_pBinding;
```

lyit | Institiúid Teicneolaíochta Leitir Ceanainn Letterkenny Institute of Technology

```
    ITimer *                                        m_pTimer;

    CWarping                                        m_Warping;
};

} // end namespace IE

#endif //__CMWAPP_H__
```

lyit | Institiúid Teicneolaíochta Leitir Ceanainn
Letterkenny Institute of Technology

lyit | Institiúid Teicneolaíochta Leitir Ceanainn
Letterkenny Institute of Technology

```
//==============================================================================
// Includes
//==============================================================================
#include <ieCore/IFileManager.h>
#include <ieCore/Time.h>
#include <ieCore/Utils/CPackage.h>
#include <ieGraphics/ComponentIDs.h>
#include <ieGraphics/ISetup.h>
#include <ieGraphics/IDisplay.h>
#include <ieGraphics/IScene.h>
#include <ieGraphics/ILight.h>
#include <ieGraphics/IMaterial.h>
#include <ieGraphics/IDebug.h>
#include <ieGraphics/ITexture.h>
#include <ieGraphics/ScriptValueTypes.h>
#include <ieGraphics_DX9/ISetup_DX9.h>
#include <iePhysics/IEnvironment.h>
#include <ieMaths/MathsUtility.h>
#include <ieCore/Log.h>
#include <ieCore/Utils/CEntityRef.h>
#include <ieCore/Utils/CEntityComponentRef.h>
#include <ieInput/ICommandMapper.h>
#include <ieCore/IUpdateSet.h>
#include <ieCore/IResourceManager.h>

#include "CMWApp.h"
#include "common.h"

namespace IE
{

ieResult CMWApp::init()
{
    IE_TRACE

    ieResult ier;

    if (Failed(ier = GetComponentInstance(Graphics::CID_GRAPHICS, Graphics::IID_GRAPHICS, (void**)&m_pGraphics)))
    {
        return ier;
    }

    // draw loading screen
    Graphics::ISetup_DX9 * p_setup;
    if (Succeeded(m_pGraphics->getSetup()->getInterface(Graphics::IID_SETUP_DX9, (void**)&p_setup)))
    {
        if (IDirect3DDevice9 * p_d3d_device = p_setup->getDirect3DDevice())
        {
            Graphics::ITexture * p_texture;
```

lyit | Institiúid Teicneolaíochta Leitir Ceanainn Letterkenny Institute of Technology

```
        if (Succeeded(m_pGraphics->getTextureManager()->loadTexture(ieS("core/textures/LoadingScreen"), Graphics::
TEXTURE_FLAG_NO_MIPMAP, &p_texture)))
        {
            p_texture->activate(0);

            struct VERTEX
            {
                float   x;
                float   y;
                float   z;
                float   rhw;
                float   u;
                float   v;
            };

            float z = 0.5f;
            float rhw = 1.0f;

            ieInt8 adapter;
            ieInt16 mode;
            if (Failed(ier = m_pGraphics->getSetup()->getActiveAdapterAndMode(&adapter, &mode)))
            {
                return ier;
            }
            IE::Graphics::ADAPTER_MODE info;
            if (Failed(ier = m_pGraphics->getSetup()->getAdapterAndModeInfo(adapter, mode, &info)))
            {
                return ier;
            }
            float w = info.width;
            float h = info.height;

            VERTEX verts[] =
            {
                0, 0, z, rhw, 0, 0,
                    w, 0, z, rhw, 1, 0,
                    w, h, z, rhw, 1, 1,
                    0, h, z, rhw, 0, 1
            };

            WORD indices[] =
            {
                0, 1, 2,
                0, 2, 3
            };

            p_d3d_device->SetVertexShader(NULL);
            p_d3d_device->SetPixelShader(NULL);
            p_d3d_device->SetFVF(D3DFVF_XYZRHW | D3DFVF_TEX1 | D3DFVF_TEXCOORDSIZE2(0));
```

lyit | Institiúid Teicneolaíochta Leitir Ceanainn Letterkenny Institute of Technology

```
                m_pGraphics->getDisplay()->beginScene();
                m_pGraphics->getDisplay()->clear(0, Graphics::RGB_RED, Graphics::CLEAR_TARGET | Graphics::CLEAR_ZBUFFER | Graphics::
CLEAR_STENCIL);
                p_d3d_device->DrawIndexedPrimitiveUP(D3DPT_TRIANGLELIST,
                                                     0,
                                                     4,
                                                     2,
                                                     indices,
                                                     D3DFMT_INDEX16,
                                                     verts,
                                                     sizeof(VERTEX));
                m_pGraphics->getDisplay()->endScene();
                m_pGraphics->getDisplay()->present();
            }
        }
    }

    ieConstStr sz_autoexec = GetSystemConfigVal(ieS("Core.autoexec"));
    if (sz_autoexec)
    {
        ExecuteCommandFile(sz_autoexec);
    }

    if (Failed(ier = GetComponentInstance(ieS("Diagnostics"),
                                          IID_DIAGNOSTICS,
                                          (void**)&m_pDiagnostics)))
    {
        return ier;
    }

    // initialize input
    if (Failed(ier = GetComponentInstance(ieS("Input"),
                                          Input::IID_INPUT_MANAGER,
                                          (void**)&m_pInput)))
    {
        return ier;
    }
    m_pInput->registerEventHandler(static_cast<Input::IInputEventHandler*>(this));

    if (Failed(ier = GetComponentInstance(ieS("Console"),
                                          Console::IID_CONSOLE,
                                          (void**)&m_pConsole)))
    {
        return ier;
    }

    m_pConsole->setActive(false);
```

Institiúid Teicneolaíochta Leitir Ceanainn | Letterkenny Institute of Technology lyit

```
    // Initialize sound
    if (Failed(ier = GetComponentInstance(ieS("SoundChannelManager"),
                                          Sound::IID_CHANNEL_MANAGER,
                                          (void**)&m_pSoundChannelMgr)))
    {
        return ier;
    }

    m_pSoundChannelMgr->reset();

    if (Failed(ier = m_SceneVis.acquire(GetActiveEntityManager(),
                                        ieS("SceneVis"),
                                        Graphics::CID_SCENE_VISIBILITY,
                                        Graphics::IID_SCENE_VISIBILITY)))
    {
        //...has a valid .entities file been loaded? Should we print out a warning / debug message?
        IE_LOG1(LOGTYPE_WARNING, ieS("Unable to acquire SceneVis from EntityManager. Has the correct .entities file been specified?"))
;

        return ier;
    }

    // Test file searching
    IResourceManager * p_resource_mgr;
    if (Succeeded(GetComponentInstance(ieS("ResourceManager"),
                                       IID_RESOURCE_MANAGER,
                                       (void**)&p_resource_mgr)))
    {
        FIND_FILE_HANDLE h_find;

        FILE_INFO * p_info = p_resource_mgr->findFile(true, ieS("*.template"), ieS("templates"), ieS("*.svn"), &h_find);
        if (p_info)
        {
            while (p_info)
            {
                IE_LOG2(LOGTYPE_DEBUG, ieS("Found file: "), p_info->szName);
                p_info = p_resource_mgr->findNextFile(h_find);
            }

            p_resource_mgr->endFindFile(h_find);
        }
    }

    //==============================================================================
    // Create stick entity instance
    //==============================================================================
    GetEntityManager(0, &m_pEntityMgr);

    CEntityRef stick_man;
```

Institiúid Teicneolaíochta Leitir Ceanainn
lyit | Letterkenny Institute of Technology

```
    //Create a stickman from the core/stickman template
    if (Failed(ier = stick_man.create(m_pEntityMgr, ieS("FirstStickMan"), ieS("core/StickMan"))))
    {
        return ier;
    }

    //Grab helper classes
    if (Failed(m_AnimWalk.acquire(m_pEntityMgr, ieS("FirstStickMan.Animation1"), Models::IID_ANIMATION)))
    {
        return IE_F_ERROR;
    }

    if (Failed(m_AnimPose.acquire(m_pEntityMgr, ieS("FirstStickMan.Animation2"), Models::IID_ANIMATION)))
    {
        return IE_F_ERROR;
    }

    //Grab bones data
    if (Failed(m_Bones.acquire(m_pEntityMgr, ieS("FirstStickMan.Bones"), Models::IID_BONES)))
    {
        return IE_F_ERROR;
    }

    if (Failed(m_Bones->registerController(this, 10)))
    {
        return IE_F_ERROR;
    }

    //Init the warping object with the source and target animation
    m_Warping.init(m_AnimWalk, m_AnimPose);

    return ier;
}

//One to one mapping between bones and animation.
//Note, in all cases bones contain one extra bone, the "origin" (used for moving the
//entire skeleton). This is assigned index 0. When applying animations with a one to one
//mapping, it is necessary to "shift" up one index value to accommodate this.
ieResult CMWApp::Example1()
{


    int operation;

    operation = 9;

    ieUInt16 i;
    for (i = 1; i < m_NumBones; ++i)
```

Institiúid Teicneolaíochta Leitir Ceanainn | Letterkenny Institute of Technology lyit

```
{
    switch ( operation )
    {
    case 1:
    // To view the walk animation:
        m_AnimWalk->getPosition(i-1, &m_pPositions[i]);
        m_AnimWalk->getRotation(i-1, &m_pRotations[i]);
        break;

    case 2:
    // To view the sampled animation
        m_AnimWalk->getPosition(i-1, &m_pPositions[i]);
        m_Warping.getRotation(i-1, &m_pRotations[i], m_Warping.getTime());
        break;

    case 3:
    // To view the pass band animation
        m_AnimWalk->getPosition(i-1, &m_pPositions[i]);
        m_Warping.getPassBandRotation(i-1, &m_pRotations[i], m_Warping.getTime());
        break;

    case 4:
    // To view a strand of the low pass
        m_AnimWalk->getPosition(i-1, &m_pPositions[i]);
        m_Warping.getLowPassRotation(i-1, m_pRotations[i], 3/*Low pass band*/, m_Warping.getTime());
        break;

    case 5:
    // To view the pose
        m_AnimPose->getPosition(i-1, &m_pPositions[i]);
        m_AnimPose->getRotation(i-1, &m_pRotations[i]);
        break;

    case 6:
    // To view the time warped animation
        m_AnimWalk->getPosition(i-1, &m_pPositions[i]);
        if( ( m_AnimWalk->getTime() >=2.0f ) && ( m_AnimWalk->getTime() <= 6.0f ) )
        {
            m_Warping.getMorphedRotation(i-1, &m_pRotations[i], m_Warping.getTime());
        }
        else
        {
            m_AnimWalk->getRotation(i-1, &m_pRotations[i]);
        }
        break;

    case 7:
    // to view the warped animation blended with the walk animation
        m_AnimWalk->getPosition(i-1, &m_pPositions[i]);
```

Institiúid Teicneolaíochta Leitir Ceanainn | lyit | Letterkenny Institute of Technology

```
                if( ( m_AnimWalk->getLocalTime() >=2.0f ) && ( m_AnimWalk->getLocalTime() <= 6.0f ) )
                {
                    m_Warping.getBlendedWarpRotation(i-1, m_pRotations[i], m_AnimWalk->getLocalTime());
                }
                else
                {
                    m_AnimWalk->getRotation(i-1, &m_pRotations[i]);
                }
                break;

            }
        }

        //If the bones after all operations are in local space, then multiply their matrices
        //into world space.
        m_Bones->multiplyBonesUsingParents();



        return IE_S_OK;
}


//Setting bone position and rotation via a mapping between the bones and animation.
//This is more flexible than one to one mapping, but also slightly more expensive.


ieResult CMWApp::MotionWarpUpdate()
{
    ieUInt16 i;
    for (i = 1; i < m_NumBones; ++i)
    {
        m_Warping.getPosition(i-1, &m_pPositions[i], m_Warping.getTime() );
        m_Warping.getRotation(i-1, &m_pRotations[i], m_Warping.getTime() );
    }

    //If the bones after all operations are in local space, then multiply their matrices
    //into world space.
    m_Bones->multiplyBonesUsingParents();

    return IE_S_OK;
}

//IBonesController
ieResult CMWApp::registeredController(Models::IBones* pBones)
{
    //Same as bones changed
    return handleBonesChange(pBones);
}
```

Institiúid Teicneolaíochta Leitir Ceanainn
Letterkenny Institute of Technology
lyit

```
ieResult CMWApp::unregisteredController(Models::IBones* pBones)
{
    //Same as bones shutdown
    return handleBonesShutdown(pBones);
}

ieResult CMWApp::updateController(Models::IBones* pBones)
{
    //Set absolute times
    m_AnimWalk->setTime(m_pTimer->getTimeSeconds());

    //m_Warping.setTime(m_Warping.getTime() + 0.04f, m_AnimWalk);
    m_Warping.setTime(m_AnimWalk->getLocalTime(), m_AnimWalk);
    Example1();

    return IE_S_OK;
}

ieResult CMWApp::handleBonesStructureChange(Models::IBones* pBones)
{
    //Same as bones change
    return handleBonesChange(pBones);
}

ieResult CMWApp::handleBonesChange(Models::IBones* pBones)
{
    //Get bones memory
    if (Failed(m_Bones->getBonePositions(&m_pPositions)))
    {
        return IE_F_ERROR;
    }
    if (Failed(m_Bones->getBoneRotations(&m_pRotations)))
    {
        return IE_F_ERROR;
    }

    //Store number of bones
    m_NumBones = m_Bones->getNumBones();

    //Set root bone to origin
    m_pPositions[0] = 0.0f;
    m_pRotations[0].identity();

    //Get the model manager
    Models::IModels * p_models;
    if (Failed(GetComponentInstance(Models::CID_MODELS,
                                    Models::IID_MODELS,
                                    (void**)&p_models)))
```

lyit | Institiúid Teicneolaíochta Leitir Ceanainn | Letterkenny Institute of Technology

lyit | Institiúid Teicneolaíochta Leitir Ceanainn
Letterkenny Institute of Technology

```
    {
        return IE_F_ERROR;
    }

    //Grab a binding from the walk animation to the bones
    Models::HSTRING_ARRAY_ID bones_hstring_array = Models::INVALID_HSTRING_ARRAY_ID;
    m_Bones->getStringHandleArray(&bones_hstring_array);

    Models::HSTRING_ARRAY_ID anim_hstring_array = Models::INVALID_HSTRING_ARRAY_ID;
    m_AnimWalk->getStringHandleArray(&anim_hstring_array);

    if (Failed(p_models->getStringHandleArrayBinding(bones_hstring_array,
                                                     anim_hstring_array,
                                                     0,
                                                     &m_pBinding)))
    {
        return IE_F_ERROR;
    }
    return IE_S_OK;
}

ieResult CMWApp::handleBonesShutdown(Models::IBones* pBones)
{
    m_pBinding = 0;
    m_NumBones = 0;
    m_pRotations = 0;
    m_pPositions = 0;
    return IE_S_OK;
}

ieResult CMWApp::initEntityManager(IEntityManager * pEntMgr)
{
    CEntityComponentRef<IUpdateSetController> update_controller;
    if (Failed(update_controller.acquire(pEntMgr, ieS("Update"), ieS("Controller"), IID_UPDATE_SET_CONTROLLER)))
    {
        return IE_F_ERROR;
    }
    CEntityComponentRef<IUpdateSetController> preview_controller;
    if (Failed(preview_controller.acquire(pEntMgr, ieS("Preview"), ieS("Controller"), IID_UPDATE_SET_CONTROLLER)))
    {
        return IE_F_ERROR;
    }

    m_pTimer = update_controller->getTimer();
    ITimer * p_preview_timer = preview_controller->getTimer();

    m_pTimer->pause();
    update_controller->setActive(false);
```

Institiúid Teicneolaíochta Leitir Ceanainn
Letterkenny Institute of Technology
lyit

```
    p_preview_timer->resume();
    preview_controller->setActive(true);

    m_pGraphics->setTime(p_preview_timer->getTimeSeconds(), p_preview_timer->getUpdateDeltaTimeSeconds());
    pEntMgr->update();
    m_pGraphics->update(m_SceneVis.getInterface(), 0);
    NewFrame();
    UpdateTimers(GetLastFrameDelta());

    m_pTimer->resume();
    update_controller->setActive(true);

    p_preview_timer->pause();
    preview_controller->setActive(false);

    return IE_S_OK;
}

bool CMWApp::getKey(int key)
{
    return (HIWORD(GetAsyncKeyState(key)) != 0);
}

ieResult CMWApp::update()
{
    IE_TRACE

    // Check for escape key
    if (getKey(VK_ESCAPE))
    {
        return IE_S_EXIT_SYSTEM;
    }

    // Reset mouse over game window
    SetCursorPos(320, 256);

    // Update frame and timers
    NewFrame();
    if (m_pGraphics->getScripting()->getRecord())
    {
        UpdateTimers(m_pGraphics->getScripting()->getRecordFrameTime());
    }
    else
    {
        UpdateTimers(GetLastFrameDelta());
    }

    // Update console
    if (Failed(m_pConsole->update()))
```

lyit | Institiúid Teicneolaíochta Leitir Ceanainn
Letterkenny Institute of Technology

```
    {
        return IE_S_EXIT_SYSTEM;
    }

    // Update input
    if (Failed(m_pInput->update()))
    {
        return IE_S_EXIT_SYSTEM;
    }

    // Get entity manager
    IEntityManager * p_ent_mgr = GetActiveEntityManager();
    if (!p_ent_mgr)
    {
        return IE_F_NO_ACTIVE_ENTITY_MANAGER;
    }

    // Update entities
    static bool first = true;
    if (first)
    {
        first = false;
        initEntityManager(p_ent_mgr);
    }

    ieResult ier = p_ent_mgr->update();
    if (Failed(ier) ||
        ResultEquals(ier, IE_S_EXIT_SYSTEM))
    {
        return ier;
    }

    // Update diagnostics
    m_pDiagnostics->update();

    // Update graphics
    m_pGraphics->setTime(m_pTimer->getTimeSeconds(), m_pTimer->getUpdateDeltaTimeSeconds());

    if (Failed(m_pGraphics->update(m_SceneVis.getInterface())))
    {
        return IE_S_EXIT_SYSTEM;
    }

    // Update sound
    if (Failed(m_pSoundChannelMgr->update()))
    {
        return IE_S_EXIT_SYSTEM;
    }
```

lyit | Institiúid Teicneolaíochta Leitir Ceanainn Letterkenny Institute of Technology

```
    // Write text
    writeText();

    return IE_S_OK;
}


ieResult CMWApp::shutdown()
{
    m_SceneVis.clear();

    m_Warping.shutdown();

    m_Bones.clear();
    m_AnimWalk.clear();
    m_AnimPose.clear();

    return IE_S_OK;
}

ieResult CMWApp::writeText()
{
    QUATERNION quaternion_rotation;
    m_AnimWalk->getRotation(26, &quaternion_rotation );

    EULER euler_rotation;
    euler_rotation.x = 0.0f;
    euler_rotation.y = 0.0f;
    euler_rotation.z = 0.0f;

    m_pGraphics->getDebug()->drawText(Graphics::RGB_BRIGHT_GREEN,
                                      ieS("X: %f\nY: %f\nZ: %f\n"),
                                      euler_rotation.x*57.2957795,
                                      euler_rotation.y*57.2957795,
                                      euler_rotation.z*57.2957795);
    return IE_S_OK;
}

ieResult CMWApp::handleInputEvent(ieConstStr szType,
                                  const IEvent * pEvent)
{
    // Toggle console on F1 key press
    if (StringCompare(szType, Input::EID_DEVICE) == 0)
    {
        const Input::DEVICE_EVENT * p_device_event = GetEventInterface<const Input::DEVICE_EVENT>(pEvent);

        if (p_device_event->deviceIndex == 1 &&
            p_device_event->controlIndex == 37 &&
            p_device_event->data.switchVal)
```

Institiúid Teicneolaíochta Leitir Ceanainn | Letterkenny Institute of Technology lyit

```
        {
            m_pConsole->setActive(!m_pConsole->getActive());
        }
    }
    return IE_S_OK;
}


}
```

lyit | Institiúid Teicneolaíochta Leitir Ceanainn
Letterkenny Institute of Technology