Innovation Project Report 2008

An Investigation into Animation Blending

By Michael Cole
BACVA3

Abstract

The art of animation blending has been used in games for many years, from the original ‘Quake to modern day PS3 games. It is the art of seamlessly merging from one animation to another to imitate movements in the real world. Unlike traditional animation, animation in games is not a linear process, moving from frame to frame, but is composed of lots, some times as many as 150, separate animations. The main objective of this project is to investigate existing methods of animation blending in real time and to use these techniques to solve the problem of creating my own blending software. The secondary objectives are to research methods of exporting animation files from Autodesk Maya 8.5 into a format that can be loaded into a C++ program. The format will have to deal with Joint Hierarchies, Geometry, and Animation for use in animation blending.

Contents

1. Introduction

The first section will investigate how animation blending has been achieved in modern computer games; explaining in some detail the differences in techniques and analysing each methods advantages and disadvantages.

The second section will go into more detail of how exactly animation blending is achieved using a weighted joint hierarchy system and inverse kinematics, as well as discussing some mathematical techniques used in animation. I will then go into which techniques I have chosen for my prototype and why, and discuss its initial design.

The third section critically analyses my prototype, with particular emphasis on discussing problems in production.

The final section deals with my conclusion, discussing what worked well and what would be improved in future production.

1.1 Animation Blending Techniques

There are two main animation techniques that can be used to define animation in computer graphics. One technique is simply describing 3D points for each frame. The other uses some form of rotation and translation matrix based around a joint skeleton that would in turn move these 3D points.

1.1.1 Point to Point Blending

The original ‘Quake’ for example used the first technique.
The Quake Engine was released in 1996 by id Software and featured this first example of 3D real time rendering, and boasted 3D character animation and blending. This method was little different to a flip book animation. Each 3D position was predefined for each point on the character mesh for each frame in the way of successive object models. Quick succession of these models gave the illusion of movement. Each model in effect would be like a panel in a hand drawn animation.
(Quake details from http://en.wikipedia.org/wiki/Quake_engine)

Blending animations together relied on interpolation, which is discussed later. Over time one set of animation models would use interpolation to morph into another by changing the interpolation value from the start animation to the end.

There are two main benefits of this technique. Firstly it is easy to implement, as only a simple loading sequence is needed to play an animation and a reasonably straightforward interpolation to blend.

Secondly the animator has complete control over each individual point on the character mesh, meaning that while an animation is playing there will be no strange skin deformation.

However there are major drawbacks. The blending would not look very realistic. Linear interpolation of points would deform the mesh into strange shapes before finishing at the end position. Other forms of interpolation that may solve this are difficult if not impossible with a point to point system. You could of course simply cut to each animation. This would stop any undesired morphing but would result in a noticeable sudden jump from one position to another. Also, as only one animation can be played at a time a fully animated character would need a lot of animation files, for example you would need separate animations for running, running with a weapon, running up a hill, down a hill and so on.

When processor power was a big limiting factor in computer graphics point to point blending was a viable option. This technique however is now widely considered obsolete.

1.1.2 Joint/Skeleton Hierarchy controlled Geometry

More modern games use the second technique mentioned earlier. Rotation Matrices or Quaternions are used to control a Joint Hierarchy structure that in turn controls the 3D points of a geometry mesh. The Joint movements are controlled by an animation blending system. This system can call on a variety of stored animation sequences to create the final output animation.

A Joint Hierarchy or Skeleton Hierarchy is supposed to represent a simplistic skeleton, mostly for a humanoid character. Each Joint can have multiple child Joints, each of which in turn can have further children. Child Joints inherit the translations, rotations, and in some cases scaling, values of there parents (see figure 1.1).

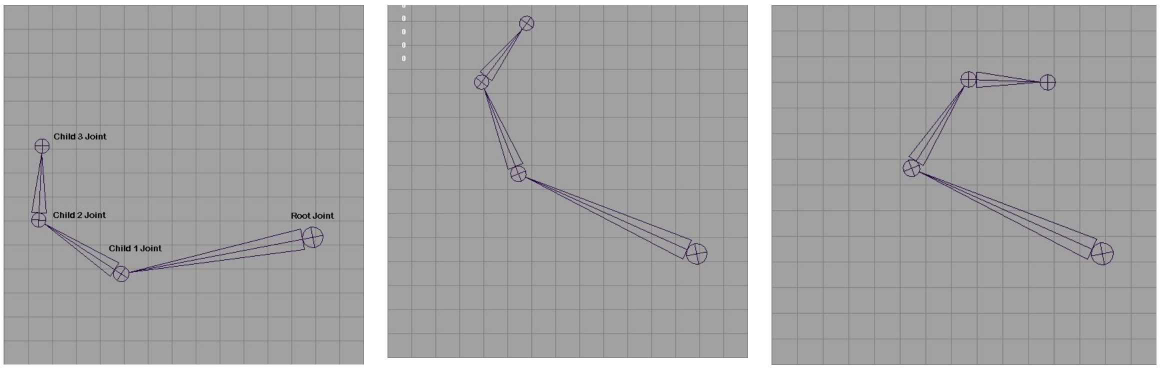

figure 1.1 Example of Joint Chain inheritance of Rotations. The first image shows a four joint chain. The second shows how all the joints inherit the rotation from the highest joint in the chain. The third again shows how every joint lower in a chain inherits from the parent.

The Joints form a Hierarchy Tree with each node inheriting all the values from the nodes above it. All the nodes in the tree eventually link back to the Root Node which is the centre of the character

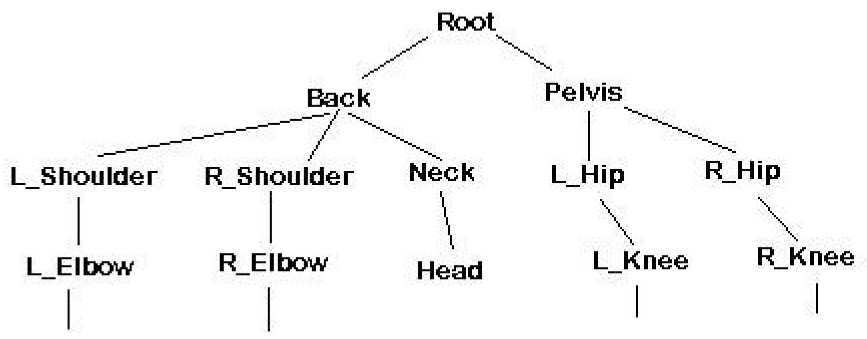

figure 1.2 Example of Hierarchy Tree starting from a Root Node with two child nodes, the Back and the Pelvis etc.

Each vertex on a geometry mesh has a weighting to one or more joints. This means that when a joint rotates the vertex points with a weighting to that joint will move with it.

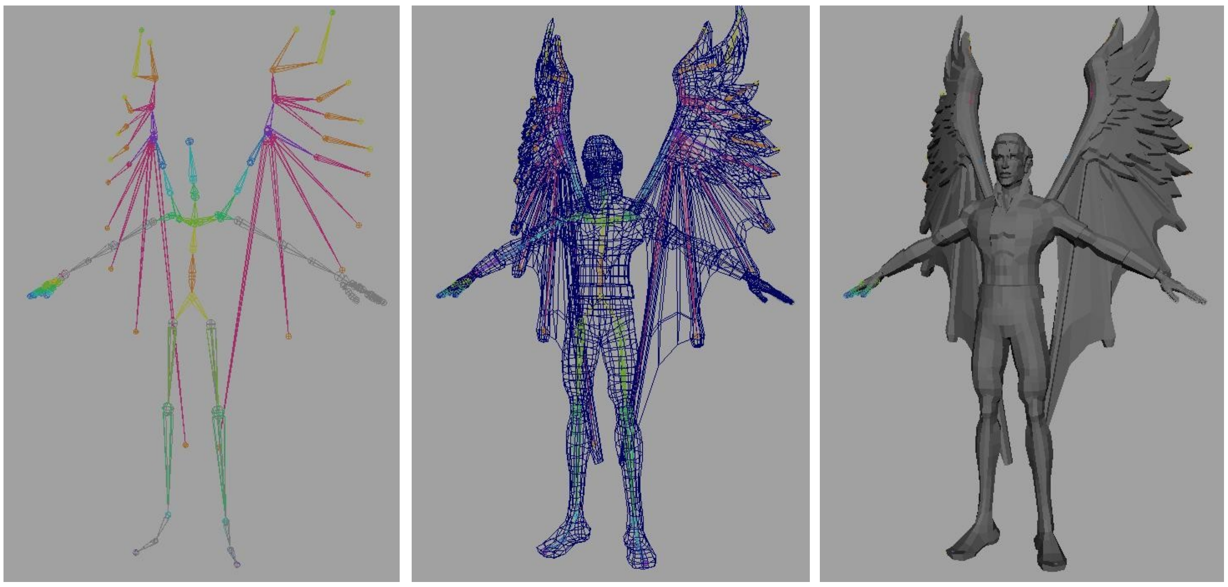

figure 1.3 Skeleton Hierarchy with Geometry mesh Skin. Character taken from Master Class Project, Screen shots from Maya. Joint Hierarchy used in prototype.

Each Joint can have animation applied to it in the form of a rotation or a translation, although generally only the root joint would have a translation. As the animations are not directly connected to the geometry it is possible using this technique to produce real animation blending. Multiple animations can be blended into a single rotation matrix that can then be applied to the joint hierarchy and, in turn, to the geometry.

Of the two techniques the Joint Hierarchy approach allows for much smoother animation blending and requires much less memory to store the information, i.e. one rotation value per frame per Joint rather than new coordinates for every vertex point per frame. It has one key aspect that the point to point system does not. When blending, the joint hierarchy system has an actual point in space to rotate around, whereas a point to point system does not. This point in space means that rotation values from two separate animations can be interpolated and applied to these joints rather than taking two points in space and simple interpolating between them.

1.2 Mathematical Techniques used in Animation Blending

A Joint Hierarchy System requires 3D rotation and translation to perform an animation. More specifically it requires transformation around arbitrary axes as each Joint would have its own orientation. There exist several different techniques for describing rotations and transformations in 3D space.

1.2.1 3D Euler Rotations

In 2D rotation a vertex is rotated about a point, be it the origin or some other arbitrary position. In 3D rotations it is rotated around an axis. Again this could be one of the orthogonal axes (x, y and z) or some arbitrary axis. Rotating around the z axis, or rotating on a plane parallel to the XY plane could be described algebraically as:

$x' = x \cos(ß) – y \sin(ß)$
$y' = x \sin(ß) + y \cos(ß)$
$z' = z$

Altering these equations can provide methods of rotating around the x and the y axis.

To rotate around an arbitrary axis as is required in animation it takes the combination of 5 matrices to calculate a new position. Firstly the arbitrary axis must be rotated until it is in parallel to one of the orthogonal axis. The axis must then be moved back to the origin with a negative translation matrix. The desired rotation would then be applied.

To complete the transformation the point must be moved back to its original orientation. This is done by using the reverse of the previous two matrices; translating the axis back to its parallel position and the inverse rotation matrix to rotate back to the original orientation of the joint.

[-translate] [-jointOrient] [rotate] [+jointOrient] [+translate]

Translations are very simple to calculate.

$x' = x + t_x$
$y' = y + t_y$
$z' = z + t_z$

1.2.2 Quaternions

A quaternion works differently to Euler Matrices. It is associated with vectors rather than individual points as Euler Rotations are. A Quaternion is based on a quadruple of real numbers and is defined as

$Q = [s, v]$

Where s is a scalar and v is a 3D vector. With v in its component form we have

$Q = [s + x\mathbf{i} + y\mathbf{j} + z\mathbf{k}]$

i j k being vector complex numbers.

Quaternions can be used to rotate points about an arbitrary axis in a similar way to Euler rotations in that it takes one transforming quaternion and then the negative of that that transform to complete the rotation.

Any point such as P(x, y, z) can be defined in quaternion form as

$P = [0 + x\mathbf{i} + y\mathbf{j} + z\mathbf{k}]$
and any axis can be defined as a unit vector

$\mathbf{u} = [x_u\mathbf{i} + y_u\mathbf{j} + z_u\mathbf{k}]$.

The transform quaternion is defined as

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

where θ represents the desired rotation.

$\mathbf{q}^{-1}$ is the inverse of **q.** The formula would therefore be $P' = \mathbf{q}\, P\, \mathbf{q}^{-1}$. This would rotate any point around an arbitrary axis defined by **u** by the angle θ.

Quaternions have several advantages over Euler Rotations. Firstly to perform the same calculation Euler Rotations require 5 matrices while Quaternions require only require 3. This is more efficient and therefore quicker to calculate.

Secondly, interpolation, which will be covered in a later section, is far easier with Quaternions than with Euler Angles. Interpolation of rotations is the key to realistic animation blending.

1.2.3 Interpolation

Linear Interpolation

This is a simple mathematical technique that takes an initial value, an end value and position between the two.

The general formula is: $n = n_1 + t(n_2 - n_1)$

'n_1' is the initial value, and 'n_2' is the end value. The value 't' controls the interpolated value, and is some where between 0 and 1, i.e. when $t = 0$, $n = n_1$, and when $t = 1$, $n = n_2$. All values of t between 0 and 1 will give an interpolated value.

It is this technique that is used in Point to Point blending. It works by creating a linear line in 3D space between two positions and interpolates along this line. As discussed before Point to Point blending is obsolete, however it can be useful to Joint Hierarchy blending in a number of ways. For example, Maya exports 24/25 frames per second. A programmes frame rate would rarely run at exactly the correct speed, so linear interpolation can be used with a millisecond counter to calculate a value between frames.

Interpolation of Quaternions

Quaternions are based on vectors. It is not possible to simple use the same interpolation technique as discussed previously on each component. It would still take into account the orientation of the joint but would ignore the change in magnitude. To successfully interpolate between two vectors while maintaining both magnitude and orientation we can use the formula

$$\mathbf{V} = \frac{\sin((1-t)\theta)}{\sin(\theta)}\mathbf{V}_1 + \frac{\sin(t\theta)}{\sin(\theta)}\mathbf{V}_2$$

where θ is the angle between the two vectors.

The same formula can be used for quaternions where θ is the angle between the two. If θ is not known is can be calculated using the 4D dot product.

This would produce a vertex that would interpolate magnitude as well as direction.

(Mathematical Equations and details taken from Essential Mathematics for Computer Graphics Fast by John Vince)

2. Detailed Outline of How Animation Blending is Achieved

This section will go into more detail on how animation blending is achieved, introducing the idea of joint weights and different types of blending. It will then go into some detail on choices for the prototype.

2.1. Weighting and Blending

A blending of animations can be achieved by simply interpolating quaternion rotations from two or more animations. However the blender is not able to know how important certain animations are and such would blend each animation equally. If you wished to combine a walk cycle with an animation of some one holding a gun it would result in some partly standing still and partly walking animation which is not desired. To combat this, the idea of Joint weightings is introduced.

Taking the previous example, in a walk cycle the important aspects of the walk are the legs, while the arm movement is only secondary and less important. In the 'holding a gun' animation however the important part of the animation is the arms and the legs are secondary. Weights are signs of importance to the blending system within an animation. If you introduce the idea of weighting in your interpolation between the previous two animations you would have the legs walking while the arms carry a gun.

The weighting system is not confined to separating the arms and legs. It can be used to prioritise any part of the skeleton, such as an arm waving or a kicking action. These weights are similar to that used in Maya when skinning a mesh to a joint hierarchy. Some joints have a large influence on the movement of the geometry while others have no influence at all.

To combine quaternion interpolation with weights we can use the following formula.

$Q = (Q_1, Q_2, I(W_1(t), W_2(t))$

where Q1 and Q2 and the two Quaternions to interpolate and W1 and W2 are the weighting on each Quaternion.

With this calculation it now takes into account how important each joint is to the animation. For example if a leg joint has a weighting of 1 in a run animation and 0 in a waving animation then blender will output the rotation fully from the running joint and not at all from the waving. If however the leg joints had a weighting of 0.5 in both animations then the blender would output a rotation interpolated exactly between the two.

If two animations had a weighting of 1 on the legs the blender would output a value midway between the two. Both input rotations would have an equal effect on the output. However, an animation with a weighting of 1 on the legs means that the leg animation should not be altered. Therefore if you combine two animations of weighting 1 you may get a strange result, such as a man running and kicking at the same time. To prevent this from occurring it would be down to the blender to not attempt to blend two such animations together.

Animations weighting, while not available in Maya, is available in other 3D animation packages such as 'Granny' which is designed for interactive 3D applications. It has a built in run time animator among other things, which is equipped to handle animation playback, modification, blending and inverse kinematics. This software could be used to reline animations and add joint weightings.

(Granny Website http://www.radgametools.com/granny.html)

2.1.1 Animation Types

There are several different types of animation blending that can be used. The two main types are transition blending and animation blending.

Transition blending
This is the type of blending that would be used to move from a walk to a run. The start animation would be the walk cycle. Then over a period of a few frames the interpolation between the walk rotations and the run rotations would slowly shift the animation from one to the other resulting in a continuous loop of the run animation.

Animation blending
This merges two or more animations together completely. This is where weightings come in useful. It is possible to merge a run with a wave and play both animations at the same time. Because of the weighting system the waving animation does not affect the legs and vice versa. It also does not matter if the waving animation is longer that the run animation as the animation is being calculated on a frame by frame basis. Once the end of the running animation is reached it will loop back round to the beginning.

The final technique used in blending is a direct Inverse Kinematic control of a joint. This means that a joint is controlled directly, and not by an input from an animation. An example of how this could be used would be a character 'look at'. The head of the character would rotate separately from the animation that is playing. If this procedural rotation is fed through the animation blender it would result in a smooth unnoticeable blend.

(details of animation blending found at http://www.gamasutra.com/features/20030704/edsall_02.shtml)

2.1.2 Systems Using Joint Hierarchy Blending

As stated before most modern day computer games use some form of skeleton hierarchy blending system. The newer the game the more complex the system. The MechWarrior games were a series in the mid ninety's that used a skeleton hierarchy. It uses a mixture of transition blending, animation blending and Inverse Kinematics to control the movements of the character.

Over View of the MechWarrior System

It is composed of two main systems, the animation playback system and the blending system. The animation system handles the actual displaying to screen of the character while the the blending system deals with outputting of blended animation based on a database of animations. Once calculated the blender would output display instructions to the animation playback system.

Quaternions are used to calculate rotations of joints and in the interpolation of animations. The blender is comprised of a tree of blender nodes linking it to a variety of functions such as a transition node, which stores a start and and end animation. The blender calculates the weightings for all animations involved and flattens and normalizes them ready for use in the animation playback.

Image taken from the original MechWarrior.

This system uses most of what has been discussed. The blender has been designed to portray the weight of the characters which are supposed to be 10 metres tall, and does so successfully. The Animation is clunky a heavy. It also uses procedural Inverse kinematics successfully as 'look ats' and gun sights. However, the system uses more than 150 animations per character. While calculations for one character are not too tasking the game often involves many MechWarrior characters at the same time. This can result in quickly escalating processor demands which can slow game play down on slower machines. A simpler blending system would allow for more overhead.

(MechWarrior system spec. from
http://www.gamasutra.com/features/20030704/edsall_01.shtml)

2.2 Choices for the Prototype

2.2.1 Exporting from Maya

Before any animation blending can take place it is necessary to export the animation from Maya into a format that can be loaded into the prototype software. This file format will be required to contain information on Joint position and Orientation, Geometry and Animation values.

OBJ. exporter
This file format exports geometry from Maya into a format that can be easily loaded into a C++ program. However it does not export any kind of Joint or Animation data. I briefly mention this file format because it could be used in conjunction with another method to export the required data.

My own file format
It is possible to write a script within Maya that would take values from the file and write them to a document. This format would have been useful as it would allow you to specify exactly which data to include in the file, such as rotations, skin weights etc, and would allow the creation of layout that would optimize loading sequence into a C++ program.

There are many draw backs to this however. Mainly it would take a long time to design a file format that could deal with all the different possibilities that Maya files could contain and present them in a way that could be easily loaded into C++. An Obj file could be used to cut down on some of the work, although this would creating problems when linking the skinning values to the mesh. Even with a successful format it could only be used by a C++ program with a loader written especially for it. Meaning the animation files and blending program could never be used in industry or with any existing software.

There is an existing file format that was designed by Sony to be a universal file exporter that could be used to transfer files from different animation packages such as Maya and 3D Studio, as well as to be used for independent software development.

Collada

This is an XML (eXtensible Mark Up language) file format that describes the structure of data. It does not have a fixed set of elements like HTML, but rather, it is a metalanguage, or a language for describing other languages, in this case Maya. It can potentially describe everything from Geometry to Particles and Orient Constraints. This format has many benefits over the other options. As it already exists no time need be wasted designing your own format, and whatever files are produced could be loaded into any software with a Collada Importer.

Collada DOM (Document Object Model)
This is a loader design specifically for use in software development. It provides a structure for loading in a Collada document and accessing elements within to be stored in user defined class structures.

2.2.2 The Prototype

The aim of this project was to develop a C++ program that could take animations from Maya 8.5 and blend them together. My research has determined that to create realistic animation blending the Joint Hierarchy system produces far smoother results than a point to point blending system. While it comes at a cost of more complicated implementation the results justify the means.

However, Maya does not include an easy way of applying weightings to joints as it designed primarily for special effects and not for designing animation for games. Therefore to add weightings to an existing animation it would have to be conducted outside of Maya either within the Collada document or once the Collada document is loaded into the C++ programme.

Adding code to a Collada file is risky as it is designed to create a structure that can be recognised by a variety of Collada importers. Changing code without knowing detailed information on its structure could result in errors. Adding Code to a C++ programme to add weights is also not a viable option for the prototype. Each character that is loaded into the programme is more than likely to have different Joint structures and different names for those joints., making an automatic weighting system extremely complicated. Also animations can vary, so adding default weights, if it were even possible, to a walk animation for example would not necessarily give a decent result.

In short the only way to add such weights to a skeleton would be to write a separate programme entirely that could load in various animation files and allows the user to add specific weights to selected joints. It would then have to export the animation to a file format that an animation blending engine would recognise.

Also quaternions, which would be preferable in a more complicated system, will not be used as its implementation would be time consuming, and OpenGL has pre built Matrices functions that work with Euler Rotations. Blending using this technique however may be harder.

The prototype is designed to load in multiple animation files and transition blend between them. The user is able to choose an animation which would then either commence or blend into the current animation.

The prototype uses four animations, a Stand, a Walk, a Run and a Jump.

To move from a walk to a run is a good example of a straight forward transition blend. From this the prototype will show how well it achieves the goal of a smooth blend, even without joint weightings. If the animations are in sync with each other, meaning the left foot is moving forward together in both animations, then the results could be quite smooth. However if the change occurs when the animations are out of sync then the blend would not be as realistic.

The Standing animation is included to provide a test on how well a walk or a run blends back to a standing stationary position and vice versa. Again joint weighting should not play to great a part in successful blending as the weighting would be uniform across the entire skeleton.

The Jump again has its own purpose. Unlike the other three animations once it has finished it would have to return to the previous animation, for example, running taking a jump and then continuing to run. This will show how smoothly it blends from one type of movement to another.

3. Construction of the prototype

Collada is by far the most satisfactory way of exporting files for software development. Collada Maya exporters are freely available on the internet and are easily connected to the Maya software. The importer for software development, Collada DOM, is harder to set up as it contains many libraries and headers that must be connected to the project correctly before development can occur. You can either compile the libraries yourselves under whichever compiler you desire or you have to figure out the setup that was used to compile the libraries to begin with and match them for your software set-up. Needless to say a lot of time was wasted figuring out a way to connect it.

A Loader has to be written to transfer the data stored in the animation file into a data structure that can be accessed by C++. The uses the Collada DOM data structure and function to access elements and their data within a file.

The data structure for the joint hierarchy uses a modified binary tree file structure, each node pointing to its children. Only a pointer to the root node is required to read the entire binary tree. Reading can be performed swiftly with a recursive function.

Geometry is only of secondary importance to the prototype as the aim is to achieve animation blending. Skinning therefore is an luxury addition. It is stored using 3D Coordinate classes combined with float arrays and construction data.

Animations are separated into each key able attributes, for example the X axis rotation of the root joint. Each of these are imported through Collada with an 'input', 'output and 'interpolation' node. As Each joint node has a different set of values for each animation loaded. The joint class contains a pointer to these data arrays under the heading of each animation.

The Animation playback system is written in OpenGL code to create a screen that can display the joint hierarchy in a 3D space. It runs through the joint hierarchy drawing the joint to the screen then continuing down the hierarchy. The recursive function is set-up in a way that every child joint inherits any translations or rotations values that have come before it. The actual rotation values are provided by the Blending system.

As this is quite a simplistic blending system it only handles transition blending. The animation that is currently playing, or the 'start' animation, is stored in a node, while the next animation, or the 'end' animation is stored in another. Based on a Lead In value it interpolates between the two animations over time. If a joint has no animation applied to it within an animation the interpolation is calculated using the original orientation and rotation. Once the value has been calculated it is outputted to the animation playback system which displays it to screen.

The User can select between animation using the number keys. When a key is pressed the desired animation is assigned to the next animation node. It is then interpolated with the current animation until it is playing fully.

3.1 How well was it achieved

The collada DOM was unfortunately the most time consuming section of the prototypes development due to a lack of documentation on how to install. Even still it was the right choice of exporters. It created an easy structure to move through and access elements within a file.

The class declarations that were used seemed to store the information worked well and make it easy to access and store desired data.

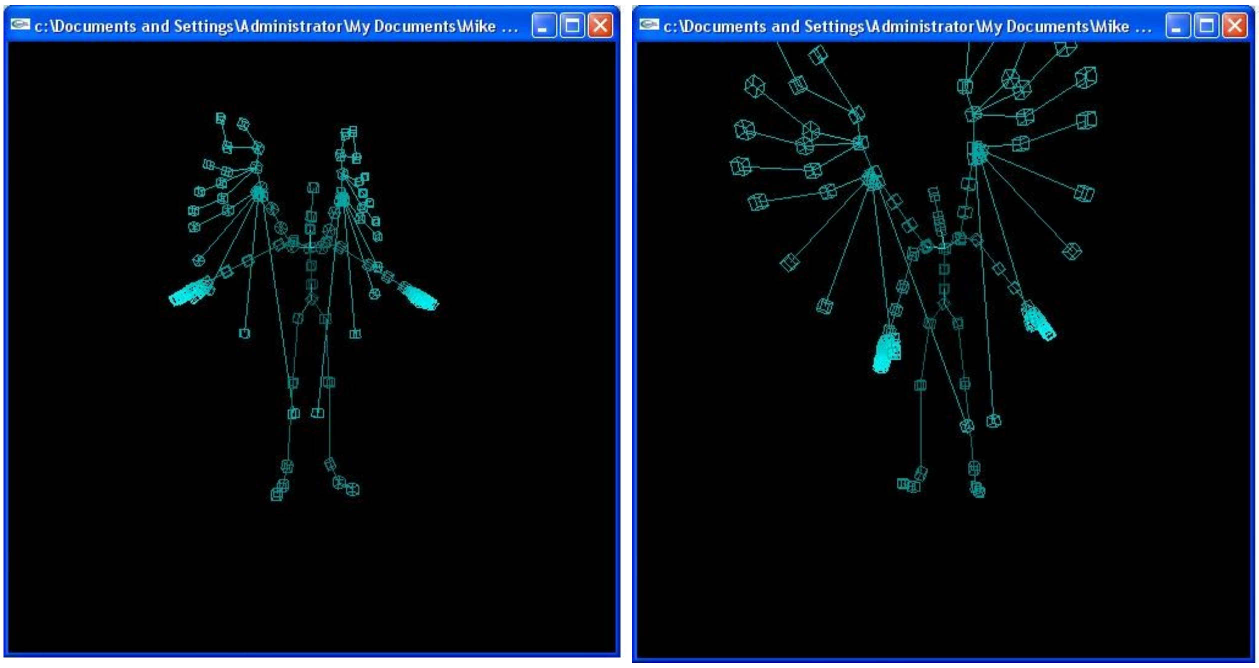
c:\Documents and Settings\Administrator\My Documents\Mike ...
c:\Documents and Settings\Administrator\My Documents\Mike ...

figure 3.1first image shows basic Joint Hierarchy loaded to screen. second show part way through a walk cycle.

OpenGL uses rotations and translations through the matrices. For this reason the blending system was designed around Euler Rotations rather than Quaternions. The use of Quaternions would need an entirely new system to move points around which would be quite complex to design. However, the use of Euler rotation rather than a more complex system of quaternions was not a good choice. Interpolation of Euler rotations is subject to a lot of variations. For example, the resulting blend between the walk and run did not consistently blend together correctly. When the animations were in sync, meaning that same leg was moving forward at the same time, the blend happened quite smoothly. However, if the transition was out of sync it was subject to wide fluctuations caused by the blend software attempting to blend two joints moving away from each other. The blend to stationary was not smooth either. While it started in the correct place and ended in the correct place the transition was very jerky.

As the Aim of the project was animation blending and not animation so due to other priorities it was not possible to include the Jumping animation.

The coding was on the erratic side. This is mainly because the initial, experienced, design failed to take in the entire scope of the task, meaning that additions were constantly needed. While I do have some experience of animation blending my limited knowledge did not realise how much was involved and similarly it was my inexperience that decided on using Euler Rotations rather that quaternions.

While the code was a bit of a unstructured it did however operate correctly. The failure to animate properly was down to the problems existing in interpolation of Euler Rotations.

4. Conclusion

I chose this project because it links with games development I had little knowledge on the topic of animation blending. The main objective of this project was to research methods of animation blending and use this research to develop a prototype that loaded Maya files an blended their animations together. While I was not totally successful in this aim I have gained a great deal of knowledge around the topic, and were I to produce something similar again I would feel more confident it designing the system.

Future versions of the software would need various improvements. The first prototype brought to light certain aspects of the design that had not previously occurred to me. The Joint Hierarchy structure needs to be redesigned to allow easier reading of the entire chain. Blending becomes more difficult if it has to traverse the entire tree to load an animation. More care need be taken in designing this layout.

Quaternions should be used rather than Euler rotations. Whilst Euler rotations may be easier to implement to begin with due to OpenGL all the research points towards Quaternions being superior at control blending. This would mean that a Quaternion system would need to be created that could handle Quaternion maths as well as Quaternion interpolation.

Blender systems have to be designed specifically for specific characters, so more care would need to be taken when designing the input animations. Also more data needs to be stored on each animation, such as Weightings, Lead-In, Lead-Out values and type of animation. This would allow the blender to produce smoothly interpolating animations ready for use in a game.

Bibliography

Literature:

Programming Believable Character For Computer Games (Game Development Series) by Charles River Media.

Essential Mathematics for Computer Graphics Fast by John Vince

Real-time 3D Character Animation with Visual C++ (Book & CD-ROM) (Paperback) by Nik Lever

Sams Teach Yourself C++ in 21 Days (Sams Teach Yourself) by Jesse Liberty

OpenGL Programming Guide: The Official Guide to Learning OpenGL, Version 2.1 (OpenGL) by OpenGL Architecture Review Board, Dave Shreiner, Mason Woo and Jackie Neider

C Pocket Reference by Peter Prinz and Ulla Prinz

Web Sites:

Collada Sites

Introduction to Maya
http://www.gamasutra.com/features/20070329/arnaud_01.shtml

Collada Forums
http://www.collada.org/mediawiki/index.php/Main_Page
(and other related pages on this site)

http://www.feelingsoftware.com/content/view/55/72

http://sourceforge.net/projects/collada-dom

Other Sites

'Granny' HomePage 3D Package http://www.radgametools.com/granny.html

MechWarrior Blending system details
http://www.gamasutra.com/features/20030704/edsall_01.shtml

http://www.robthebloke.org/models/html/Anim_cycles.html