Andrew Mitchell – Innovations report

1 Abstract

This paper will present the ideas necessary to implement a system which manages independent breakdancing animations and blends between them, which will lead to the creation of seamless animation sequences consisting of a variety of cycles. The sequences will be blended in real time.

More specifically; this will involve using motion capture to capture the breakdancing animations, transferring the animation on to a generic skeleton, converting the animations into looping cycles, creating a suitable method for blending between them and finally creating a system which can manage these transitions and create indefinite combinations of moves

The system will be developed under Maya using mel, however it will designed in such a way to make it portable to other applications / environments.

2 Introduction

The purpose of this project is to develop a means of taking separated animations cycles and combining/blending them to form a seamless animation.

To do this I would need some animation cycles, I decided to use breakdancing animations because the motion is very complex so it would test the limits of the system which I will produce. Rather than key frame the animations, I decided that motion capture would be the best way to obtain the animation, it could then be cleaned up and looped.

Once the animation cycles have been looped, I will develop a way to blend between them and then create a system to manage this blending so that the individual animations can be executed in a sequence.

The initial phase of this project was to investigate how blending animations was approached by games designers as this would provide knowledge that would lead to the development of the final implementation.

The terms ‘Finite State machine’ will be used ‘FSM’ interchangeably in the rest of this paper

3 Research

3.1 Case Study: “Animation Blending: Achieving Inverse Kinematics and More” - Jerry Edsall, 2003

(Courtesy of gamastura.com)

The basis of this paper is such that, although traditional IK can be used to solve physical movements such as walking with accurate positional results, it does not give a sense of realism / believability to the motion. It is very hard for a mathematical algorithm to calculate distribution of weight in a given movement, and also sometimes what is technically right, does not necessarily look right. The paper presents a method which only utilises artist created animations which can be blended together to create IK.

This diagram represents the blending engine used in Mechwarrior:

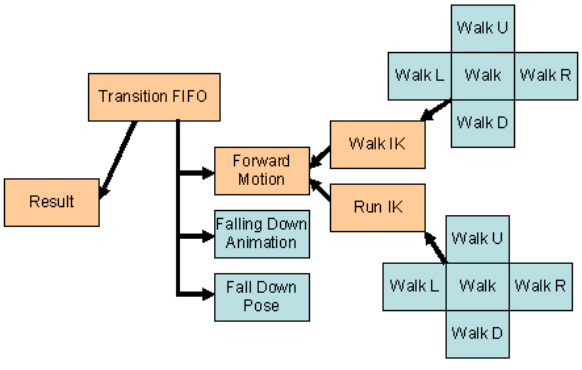


Diagram 1

Blend management Diagram, (Edsall, 2003)

“The blend manager handles the hierarchical blending using blender nodes which each represent a hierarchical layer of the blend tree.” (Edsall, 2003)

Each node in this tree has a weight associated with it, this weight is the amount that that particular node affects the animation. Each frame, the blend tree is evaluated with all it’s nodes and respective weights to calculate current animation of the character. Each independent animation also has a velocity associated with it, the blending of these velocities is also handled by the blending manager. To give an example, if the character is blending between a walk and a run, the resulting jog animation will be created.

Designing the node system in a hierarchy, gives the animator a little more freedom in how them allocate the weights. Calculating weights for a large (eg more that 4), linear system becomes very complex. This way localised decisions can be made, and then the weights normalised for final interpolation.

To summarise:

“The process of blending includes four steps. The first step is to build the tree, which the simulation does based on the movement states of the character. The next step is to advance the time on each animation in the tree. Third, the list is flattened and all the weights are unified. The last step is to interpolate the current position of each source animation and then blend that value into the final result buffer” (Edsall, 2003)

The paper also discusses various types of blend:

Cross Fade: blend one animation into another (a->b), by decreasing the weight of a and simultaneously increasing the weight of b.

Continuous Blend: Add one animation to another by increasing it’s weight from 0. The result will be a combination of the two animations.

Feather Blend: Based on the idea that not all the joints in the blend have a unified weight. I.e. A character running and a stationary character firing a gun. With feather blending these can be combined to form a running character firing.

Single-axis continuous blending:

Single axis blending is a method which can combine a number of animation cycles at any one point, this is probably best described in a diagram:

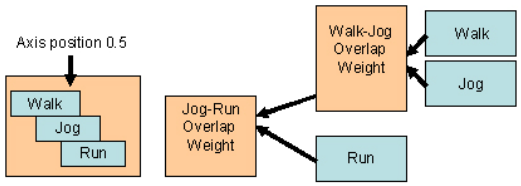


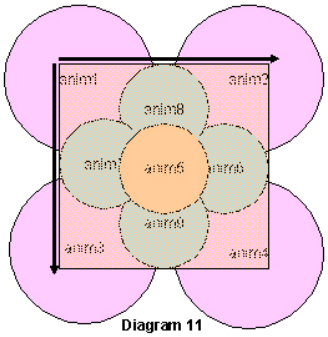
Diagram 9

Single axis continuous blending Diagram, (Edsall, 2003)

As the axis moves position in the above diagram, the motion changes from a walk to a run.

Multi-axis continuous blending:

The multi axis blend allows blending between multiple axes. In this implementation each animation is represented by a circle. The centre point of each circle represents the animation full weight it’s perimeter represents the animation at zero weight. The blended node then steps through the nodes and calculates their distance from the input to determine the overall weighting. Here is a diagram to further clarify this method:



Multi-axis Continuous blending diagram (Edsall, 2003)

Priority blend trees Priority trees are additional to the blending methods, their task is to make sure that certain animations are played regardless of there weighting.

The systems are not without their drawbacks, memory usage to store all animations and computation power for blending but providing the system is fully optimised, it is a far superior alternative to ik solutions.

Conservation of momentum:

"The interactions of each animation must be carefully analysed for continuity since animations that are not designed to blend together often provide undesirable effects." (Edsall, 2003)

When blending between a walk and a run, one needs to consider that the character would speed up / slow down gradually as opposed having a sudden burst of speed. For example the run must be slowed down to the speed of the walk at the beginning of the blend, and then the walk sped up to the speed of the run at the end of the blend. This will ensure cadence is preserved, and the blend looks natural.

When 2 animations are significantly different ie, walking and climbing, a straight blend between them would look very unnatural indeed. This would require at least one point in the animation cycles at which the character is in exactly the same position, if this is not the case then some transition animation would be needed.

Case study 1: summary

This article provided a very detailed account of how an animation system was implemented in the Mechwarrior game and provided a lot of inspiration. I think the idea of feather blending would be particularly useful for breakdancing.

3.2 Case Study: "Non-Linear Animation" - George Maestri, 2001

This paper describes the process of NLA (non linear animation), The following is a summary of some of the key features.

"NLAs allow the animator to think and create above the level of the keyframe" (Maestri, 2001)

The principals of NLA are to have a library of basic animations e.g walk, creep, run, and also some bridging animations. Once you have these fundamental parts, the idea is that you can apply them to a number of standardised characters and order the animations in whichever way you choose. NLA requires animation skeletons to be standardised, i,e with similar joint structures and naming conventions so that animation can easily be transferred from one character to another.

With the base animations complete, it is also possible to layer animations to create more characterisation. For example: Starting with a basic walk, one could add a hip twist, or even a limp. These layers could be specific to a certain character but the underlying motion would be the same. This would be a very efficient way of creating a number of individual characters.

Some of the other advantages would be, speed changes, the ability to change the length of the animation block/s on the fly. Also offsetting, allowing the animation to be shifted around in a global scene.

Transitions between cycles

In order to create a convincing blend between animation a number of factors must be considered.

It is important when changing between cycles that the transition is easy to calculate as possible. In games the blending algorithms are generally quite simple, just finding the shortest distance between two points. Make sure that the run and the walk both start from the same foot, otherwise this would involve the character stepping back half a step in the blend. Special moves should be able to blend with other special moves and also standard motions.

This paper also presented some interesting ideas regarding motion blending, a lot of the ideas presented are similar to that of the Mechwarrior example explained slightly differently, but the principle remained the same.

Whilst reading the above articles and a few other less in-depth features, I came across the term “finite state machine” on more than one occasion. As I was unsure as to what it meant, I decided to investigate:

3.3 Finite State Machines

“Finite State Automata, also known as finite state machines or FSMs, are a thereotical device used to describe the evolution of an object based on its current state and outside influences. The present state of the object, its history, and the forces acting upon it can be analysed to determine the future state of the object. Usually, finite state machines are represented in terms of state transition tables. While theoretically interesting, in general there do not seem to be very many real world applications that take advantage of the properties of finite state automata.”
(Gruber, 1995)

This definition of finite state machines was a little to general for my taste, I was still unsure as to how I could use one with regards to animation system or how one could implemented in general. I decided to consult some games programming books in order to get a more relevant definition.

In ‘Game programming gems 3’, Dante Treglia presents an example of an FSM (finite state machine) when used to manage the AI of a computer controlled character. The character had a series of emotions (uncaring, mad, rage etc..) these can be referred to as the *states*, and a series of inputs (character attacking, character walking away etc..). These inputs were combined into a ‘state transition matrix’, in the form:

Old state -> input -> new state

Here is a simple example

Uncaring -> character attacking -> rage (Treglia, 2001)

Although this example illustrates an unrelated use of a FSM, it successfully illustrated how useful it would be for designing a system to manage my animation cycles.

Now i would like to discuss the theory behind my own blending engine.

4 Implementation

4.1 Theory

Let the animator do all the work – By this I mean, don’t leave any of the animation up to mathematical equations (ik/interpolation), as this results in unrealistic, weightless animation. Instead provide a means of blending / overlapping two animations, alternatively creating transition animations which can preserve cadence. In the case of the breakdancing moves I think it would be necessary to identify blend points that is, the position in the motion that is closest to the next motion. This would reduce the number of transition animations I will need

Use a hierarchical state machine to manage the system – an FSM has been present in most of the examples I have researched, I think it would be highly useful for managing an animation system. This would include holding a list of animations waiting to be executed, and controlling which blending animations are used.

Complex Nodes / states which hold user specifc data – Rather than each node / state just contain the animation cycle, it should hold information regarding the speed of that animation, appropriate blending points specific to the motion into which it will be blended etc.

I considered using the trax editor, Maya’s primary NLA tool for job of managing the animation cycles This would allow the animations to be edited in a similar way to music tracks. However, although the system was going to based primarily in Maya, I wanted to create a more generic design that could be used in other applications or even in a games environment so the fsm approach seemed the most appropriate.

As far creating sequences of animations, a keyable state node within maya would be sufficient.

Information a node / state would contain:

Animation Data – This would be the keyframe info etc used to animate the character

Animation speed – Depending on how experienced a breakdancer is will depend on how fast he can execute moves. So the speed of a move / animation should be variable, this could also give rise to a loss of balance if the move is executed too fast resulting in falling over.

Transition Data – Depending on the next move in sequence a certain transition animation will be required and that transition will also need to take place a certain point in the current animation cycle. The relevant transition animation will be pointed to by this node, and also the transition point/frame in the current cycle.

Consider 3 animation cycles: A, B, C

These animations will give rise to 6 transition animations:

A – B
A – C
B – A
B - C
C – A
C – B

The first thing to note is that the transition animations are going to increase by a power of 2 each time a core animation is added. This would require a large amount of memory as the animation library increased. In order to reduce this overhead it would be better to split these transition animations up smaller animations so that they could be pieced together to form many different transitions. This is something I will need to investigate further when I look at other animations.

Next lets consider the job of the FSM itself; the FSM will initially need to be set with a state (animation) e.g. A. This animation will continue to play until the next state is input eg B. At that point the FSM will check in the state node to see the next available point in the current cycle that a transition can take place. It will then also take a handle to the transition animation that will need to be played. After the transition animation is played the current state will be set B. As new moves are input from the player they will be added to a que in the FSM. This queue will contain a handle to the next animation, the appropriate transition animation, and the frame at which to perform the transition itself.

This will be the basis for the animation system.

4.1 Motion Capture

At this point I need to put what I have detailed in the above into practice. I went to the motion capture lab with a friend to capture the necessary animation cycles. This is a list of the animations that were captured:

Uprock
Footwork
Swipes
Windmills
Flares
1990’s
Headspins

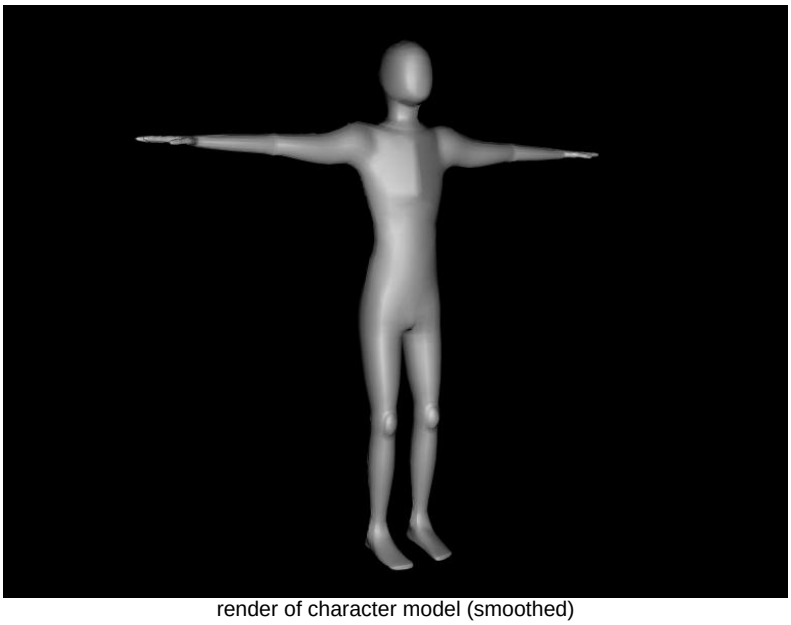
See innovations – making of.avi

I decided to catch a large variety of moves as I wasn’t sure which ones were going to capture accurately and which would blend together well. This was the first time the motion capture producer (Andy Cousins) had worked with breakdancing so we had no idea how well it would turn out, the session lasted approximately 4hrs.

A few weeks after the capture session, Andy had some encouraging results, the majority of the moves had all captured accurately, so after Andy had transferred the animation into motion builder and set up the joint, they were ready to be transferred to a Maya skeleton.

In preparation for this, I had already created a simple character in Maya, to be used for this purpose. I took the exact measurements of the motion capture skeleton and scaled my character appropriately so that the motion would be preserved as best as possible in the transfer.

Original Character model:



render of character model (smoothed)

Nick (motion capture subject) in mocap suit

The original model and skeleton needed to be adjusted to fit nick's specifications, this allowed the transition of animation data from the motion capture software to be as easy as possible.

4.3 Motion Capture clean up

I had never used any mo-cap data until this point so the task of cleaning it up was a very new experience for me.

The first animation I decided to clean was the swipe. I was unsure whether this would be an easy choice but the motion capture had worked pretty well for the move itself so that was a good start.

See Swipes_orig.avi

The above is a playblast of the animation immediately after I had imported it into Maya (other than scaling it down to ¼ size). The skeleton was keyed at every joint on every frame, this obviously needed to be reduced for economical reasons, but I also needed to preserve the quality of the motion. After some investigation, I found the 'simplify curve' function which is designed to reduce dense key frame data. After running this function it drastically reduced the amount of keyframes on the skeleton to about once every 3-4 frames. In order to check that the quality of the motion had not been lost I overlapped the animation with a copy of the original skeleton, the difference was virtually unnoticeable so I decided to keep the changes.

Although, the fundamental motion was captured accurately the finer details of the move needed a lot of cleaning up. The hands for example are a prime example of this, on none of the frames during the motion do they lie on the plane and in the same place. Since the root is moving every frame they needed to be adjusted for every frame that they sit on the ground, so that the viewer is convinced that the momentum is being generated from that point.

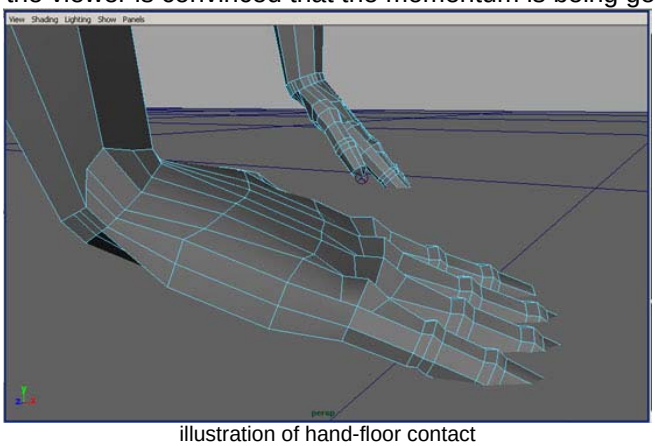


illustration of hand-floor contact

Rather than try and clean up the motion as a whole, I decided to cut out one rotation of the swipe to work on.

This reduced the animation to 22 frames. One method I found effect for maintaining one hand position on the ground was to duplicate the skeleton and apply ik to the duplicate arms, I could then key those arms in place whist the hands were meant to be on the ground I could then use a simple script to copy the rotation values from the ik arms to the fk arms.

4.4 Looping animation

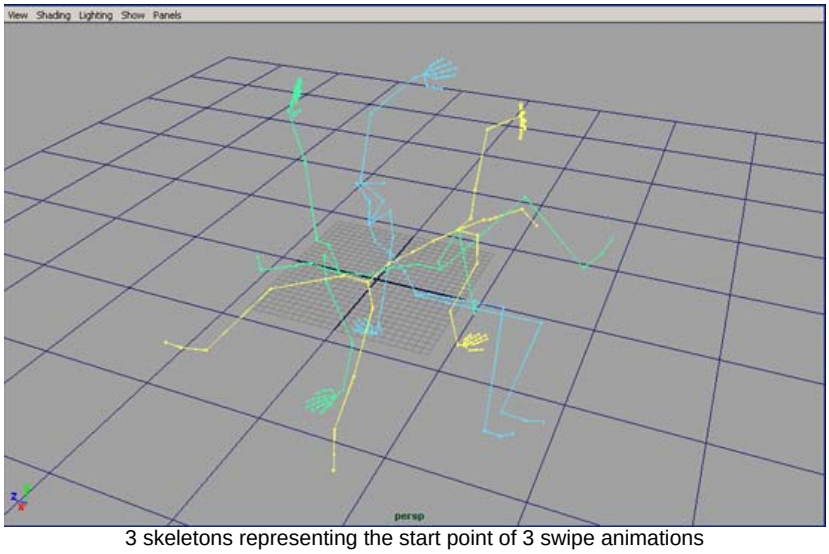
It goes without saying that to loop an animation the start and end pose need to be the same, however, in order for the swipe skeleton to be in the same position requires the swipe to repeat 4 times, and even then the animation needs to be altered slightly for it to be an exact loop. The implications of this are that, now I would need to have 4 blends for every one target animation depending on which of the 4 swipes were currently looping. I could always delay the blend until the appropriate swipe was in motion, but that would cause for a large backlog for multiple animations and mean that the minimum amount of swipes would be 4 which would be very impractical.

Since it was only the root joint that did not loop each swipe, if i could extract the root rotation and translation, i could cycle the other joints and just increment the root each cycle. My first thought towards this dilemma was to parent the skeleton to a group, at the end of each swipe, I could reset the animation and move the group to a new position and start the animation again. If this was all done in the space of one frame it would give the impression of continuity, the swipes could loop indefinitely and I would only need one blending animation since the local rotations did not change.

In order to cycle the animation, I thought that it would be better if the keyframes weren't connected to the time line, instead I created attribute in the parent group of animation and connect all the animation to that. This gave move freedom when moving the animation around.

I blocked out a test of this method by duplicating the one cleaned looped of the swipe animation, moving it and delaying the animation.

See swipes_looped.avi



3 skeletons representing the start point of 3 swipe animations

I tried working with constraints to aim and position the group each cycle in an attempt to automate this process but this lead to some very odd results. Unfortunately because the group was a level above the skeleton, aim, orient, constraints did not give the right results. As far as I can gather when an orient constraint is applied, the local rotate values of the target are applied to the subject, this is okay if the subject has no previous values, but if it does or in this case it is the parent of an object with rotation values it is of no use. The alterative was to create my own aim equation, this could be done by calculating the local axis of the target skeleton and the local axis of the subject skeleton and then using an aiming algorithm align the two using the group, although this seemed feasible it also seemed that it was an over-complex solution to the problem.

I continued to experiment with this problem, and managed to come up with a slightly simpler solution. Due to the nature of the motion ie it rotates around the world y-axis, the y rotation at each cycle was a constant increment (84.1deg) so I could just initiate a relative rotation of this amount each loop. As for the translation, if I retrieved the global coords on the last frame of the cycle, translated the group to that position and reset the animation, the skeleton would remain in the same place.

See appendix 1

Content that I now had the swipes animating indefinitely, I decided to move to the next animation: windmills

The windmills involved much the same clean up process as the swipes, but just with a different motion. One thing I had to focus on was that when the character rolled onto on and off his back that his hands were placed underneath it's body and also that the root moved in a circular motion. I also developed a couple of scripts to remove unwanted keyframes so that the motion remained smooth.

See appendix 2

After cleanup I applied a similar expression to the one used with swipes to get the motion looping, see anim.

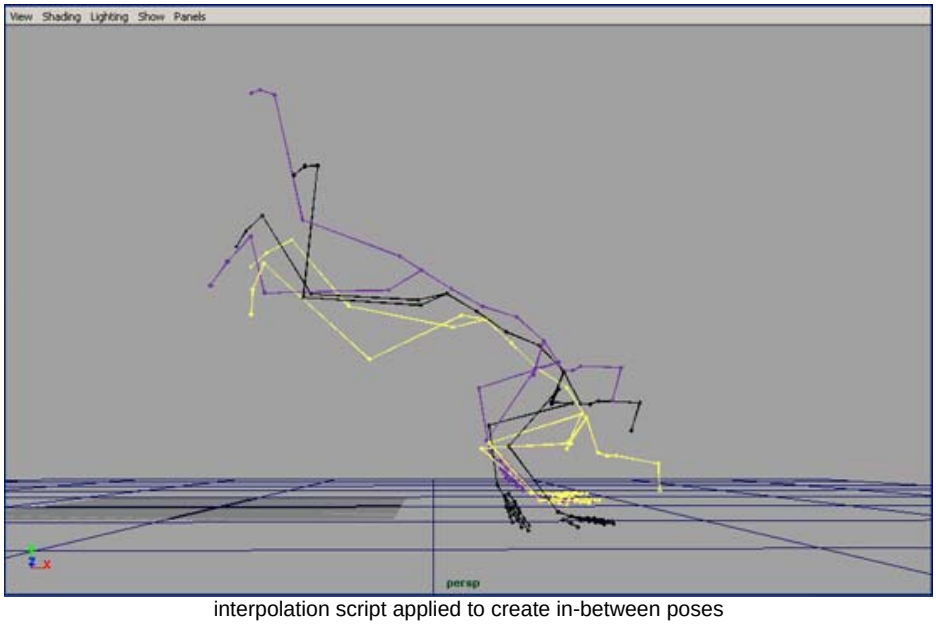
See windmill_looped.avi

4.5 Blending animation cycles

Now I had 2 animations looping the next step was to approach blending them.

This first thing I did was to identify the 'blend-in' pose for the swipe and 'blend-out' pose for the windmill (swipe-windmill transition) and vice versa for the windmill swipe transition I then played the animations sequentially. From this animation I could see that there were only a few frame that would need to be added in between to make the transition smooth.

I was reluctant to use mathematical equations (ie interpolation, quaternion rotation) to move between one motion and the next as most of the blending articles i had read advised against it. However, before disregarding this method totally I decided it would be best to try it in practice. I took the 'blend-in' frame from swipe and the 'blend-out' frame from the windmill copied them to static skeletons, I then wrote a script to generate 3 intermediate poses using interpolation.



interpolation script applied to create in-between poses

The interpolation method did not produce very encouraging results for a blend. Looking at the rotation values from each skeleton it is easy to see why, although the animation poses may look similar, their rotation values are very different. Although methods such as quaternions will take the shortest route between rotation values, this would not keep the hands convincing on the ground as the character changed cycles

Instead I felt another level of abstraction was needed. If I hand created the blends, providing all the joints were in the same position at the blend-out and blend-in poses, the actual rotation values would be of no consequence. With this in mind I hand created 2 blends what perfectly matched the in out poses, each blend had it's own skeleton and parent group with time attribute so that it could be treated in same manner as the same animation cycles.

See innovation – making of.avi

The results appear to be seamless, however the only way to be sure that the blends are convincing i would need to use them repeatedly in a sequence of animations. If I was to do this I would need a way of automating the combinations, this would mean the implementation of a finite state machine.

4.6 Finite state machine

When looping the swipe and windmill cycles, I used a method that involved a constant, relative rotation each cycle. However this approach would not work if I was to combine more than one animation, because the rotation would no longer be constant. So it seemed I did need an approach similar to the mathematical one suggested earlier.

Since I was only dealing with rotation in the y axis, I would not need to worry about 3d rotation matrices, instead the scalar product would provide an angle that would work on the x,z plane, the only problem was which vectors to compare to generate the angle.

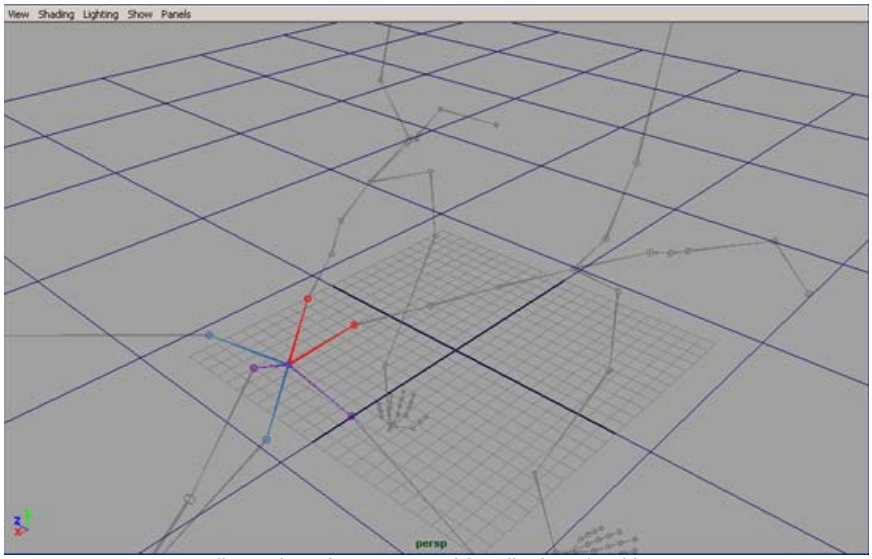


Illustration of vectors used for aligning algorithm

The vectors that I chose to use for the scalar product are highlighted in red in the above picture, that is the first spine joint of both the skeletons. I could calculate this vector by subtracting the world coordinates of the spine joint and the root joint (marked in red above). To determine which way to rotate the skeleton clockwise/anticlockwise (as the scalar product given no indication of direction) I used the cross product, depending on the direction of normal vector produced (+y or –y) by the cross product the algorithm could determine which way to rotate the skeleton.

This algorithm solved the problem of aligning the skeletons so now I just need to create the machine itself.

I decided to create the first version of my fsm in mel. Although I was capable of producing it in the api I felt that I could obtain quicker results using mel for debugging purposes. However this would mean that i wouldn't be able to create in an object orientated manner (c++) which was how i had originally planned to create the fms, but i could still follow the same structure.

See FSM V1.txt

The above fsm executes in maya through the expression editor. In the Maya scene there is locator which has a state attribute, this attribute contains the values “windmill” and “swipe”. Each time 'stateMachine()’ executes, the value of the state attribute is retrieved. If the retrieved state is different to the currently executing state ($STATE), the $isBlend global variable is set.

Depending on the current state of fsm, the appropriate animation is advanced each time it executes. For example, if $STATE = “windmill”, advanceWindmill() will be called. The advance functions themselves increment the time attribute of the relevant animation, they also check to see if the animation needs to be looped (the time attribute is at its maximum) in which case loop() is called, and also whether the animation cycle must be changed ($isblend is set and it is the appropriate frame) in which case blend() is called.

The loop() function is responsible for resetting the current animation cycle and also changing its global coordinates so that the transition is seamless, it does this using the same process as was used in my earlier files.

The blend() function is responsible for seamlessly switching between animation cycles. It does this by calculating the position of the current animation, (the destination), setting the appropriate time value on the new animation, and then moving and aligning the new animation so that it is the correct position for the transition unnoticeable.

This implementation proved successful, by keying the attribute in Maya, you were able to determine which animation is played and when the animation is changed.

Although this version of the fsm was functional, it was only really capable of dealing with 2 main animations and the necessary blending animation. Since I wanted the system to be capable of dealing with a multitude of animations it would need to be modified or redesigned, but this was a good foundation and a necessary step.

If I was going to build a more comprehensive fsm I would need more animations for it to control. I decided to add another loop, I wanted to use something that could be easily blended from the current animation sI had created. My first thought was flares (see appendix) but after a close examination of the Maya file I realised that I didn’t really have a solid loop to work on, my next idea was to use head spin, The head spin had captured particularly well and I had a lot of animation to chose from so this would be my next addition.

Unlike windmills and swipes, head spins have more than one stage in their motion,

See innovation – making of.avi

I would need to use 3 head spin cycles; tick, glide and drill (see appendix) to effectively use them as part of the animation. Although this would mean a lot more work, cleaning 3 cycles as opposed to one, I thought it would provide a interesting addition to the project and so was worth pursuing.

After the cycles were cleaned I needed to consider the blending between them. Unlike the windmill to swipe and swipe to windmill blends the start and end poses were quite similar, so the blends shouldn’t have been as hard to create. However it was important not to lose the sense of momentum as the motions were so fast. I used my interpolation script:

See appendix 2

As a start for the blends which provided a good foundation , I then cleaned up the wayward joints (usually spine and arms) by hand.

Rather than using 6 blends I used the fact that the positions were similar to my advantage, and reduced the blend number to 4:

Tick->glide
Glide->drill
Drill->glide
Glide->tick

There was no need to animate drill->tick and tick->drill as I could simply blend through the glide animation using:

tick->glide, glide, glide->drill, drill.

Which would be just as effective.

In order to improve efficiency within the fsm, it would be nice if I could just consider the above animations as "headspin" and use a lower level program (a simpler fsm) to worry about which exact headspin was currently executing. This would mean that as I added more moves I could abstract the ones with a number of variations (of which there are many in breakdancing) to these lower level fsm which would make the whole program much more manageable.

Another improvement that became apparent was the need for a few physical constraints on the moves themselves. For example, when the drill animation loops it becomes increasing unrealistic, so I could introduce a parameter determined how long the drill/glide had been looping, and if it was past a certain value, blend to tick and then back to the blend/glide. This little automation would be a nice realistic touch without relieving the user of any of their freedom with moves, and would be a very relevant addition in a games environment.

I decided to code an fsm specific to the head spin animations to see how easily it could be implemented.

See fsm V2.txt

This version of the fsm was a lot more comprehensive than the previous version. It was centralised around the idea of an animation queue which contains a list of all the cycles waiting to be executed, the oldest at the front. Each time a new animation is requested, that animation and the necessary blending animations were added to the que. Using this method, the currently executing cycle only had to check against the top member of the stack for blending information. When the animation cycle is replaced, the member of the stack is popped ('popQue()').

The process of adding animations to the stack is an important one, depending on which animation is current or, if there is already a que, mostly recently added animation, the appropriate blending animations must be put on to the stack in the correct order. This gave rise to each headspin having its own blending function:

pushTick(), pushGlide(), pushDrill.

Each 'push…' function would be responsible for ensuring that the correct inbetween animations were inserted before the requested animation. I combined the 'blend' and 'loop' functions into a single function since looping is really just blending animation cycle with itself. The global variable '$blendNum' keeps track of the number of blends in the queue and is also handy for referencing the most recently added animation.

The main fsm function checks the current the current state, '$STATE', and if that state is a head spin passes control to the head spin fsm which can check for the specific head spin to advance.

This version of the fsm was much more open to expansion, with the introduction of an animation que and sub fsm to control groups of similar animations, but to really test it, I needed to introduce the windmill and swipe animation. This, meant more blending animations needed to be created. As I was approaching the end of the project at this point I decided just to animate the windmill->tick and tick->windmill animations and leave the swipe transitions until later if I had time.

This led to the third and final version of my fsm.

See fsm V3 final.txt
See fsm – final sequence

All the existing 'push…()' functions needed to be updated to include the windmill and swipe parameters, and new push functions for swipe and windmill were added. I also added the $isHeadspin variable which informed the main fsm to pass control to head spin fsm.

This final implementation demonstrated the majority of the principles that I have discussed regarding the implementation of an animation management system.

See Final Scene.mb
See Final Scene readme.txt

In the above fike you are able to animate the state variable of an im locator in order to create a unique combination of moves.unfortuantely I did not have time to add the physical limitations I discussed earlier, but this was just a minor touch that can be added at a later date.

5 Conclusion

In this project I have implemented a hierarchical finite state machine blending engine in mel which manages a series of cycled /bridging animations in order to create indefinite sequences of animations which can be controlled through a Maya scene.

When I began researching for the project I was under the impression that I would be using a lot of mathematical algorithms in order to blend between the motion cycles (quaternions and such like). But most of the papers I had read advised against using mathematical equations to create animations as it provided no sense of weight / momentum. After trying both the methods myself I decided that keying the blends myself would provide the best results. However towards the later stages of the project when I was working on the head spin blends, it became apparent that perhaps a hybrid approach would probably be best. For example, it is very important that the head stays in contact with the same part of the floor during the rotation of the character as the spine deforms so this must be keyed bay hand, however the movement of the legs between tick and glide good easily be automated using interpolation / quaternions to give the same effect.

If I was to continue working on this project, the next step would be to move all the animation on to one skeleton. In the preliminary stages it was fine to have a few skeletons in the scene one for each motion but as the amount of animations increase the scenes got bigger would eventually become unmanageable. This best approach to solve this would be to store all the rotation values on disk and just load them up as required on to a generic skeleton. This would also be a step towards taking the system out of Maya a using it in a games environment.

This project was my first chance to use motion captured animation which was a great experience, but unfortunately although the fundamental motion was captured well the, clean up of the more detailed elements was a significant task and took and large proportion of the project time. I developed a few scripts to help me with this but ultimately it just involved stepping through each frame checking that all the points that should be in contact with the ground were still in contact. This does raise the question of whether it was worth doing the motion capture at, whether I could have animated the moves from scratch in less time.

The final animation serves well as a demonstration of my animation system. However the addition of the head spin animation fairly late on in the project meant that I was able to animate all the bridging animations (swipe-> headspin and headspin->swipe) and those that I did add those that I did add weren’t entirely to my satisfaction.

I look forward to working on this project in the future to add further animations and create more varied animation sequences.

5 Bibliography

Web:

www.Gamasutra.com (Animation Blending: … - Jerry Edsall) ->1

www.informit.com (Nonlinear Animation - George Maestri)
(Animation For Games - George Maestri)

www.fastgraph.com (Automata Animation - Diana Gruber)

Books:

Essential Mathematics for computer graphics fast – Prof. John Vince

Game Programming gems 1 – Mark Deloura
Game Programming gems 3 – Dante Treglia

Appendix 1 – Looping Expression

```
//$t is the swipe animation time value

if( $t % 22 == 0 )
{
	print "Change Cycle 2";

	$check = 1;

	float $count = $t / 22;

	//start position
	float $s_pos[3] = { 18.766478, 13, -8.184948 };

	//current pos
	float $c_pos[3] = `xform -q -ws -t swipe|spine_1_Base`;

	print $c_pos;

	//new position
	float $n_pos[3];

	$n_pos[0] = $c_pos[0] - $s_pos[0];
	$n_pos[1] = $c_pos[1] - $s_pos[1];
	$n_pos[2] = $c_pos[2] - $s_pos[2];

	print $n_pos;


	//move to new position
	xform -ws -t $n_pos[0] 0 $n_pos[2] swipe;

	float $rot = $count * 84.1;

	//rotate correct amount
	xform -ws -ro 0 $rot 0 swipe;

	//reset animation
	setAttr "swipe.time" 0;

}

else
{
	setAttr "swipe.time" ( $t % 22 );
}
```

Appendix 2 – Blending Scripts

```
//array which holds the local names of all the animated joints in the model
string $list_a[22] =
		{
		"spine_1_Base"
		"pelvis_left",
		"knee_left",
		"ankle_left",
		"foot_left",
		"pelvis_Right",
		"knee_right",
		"ankle_right",
		"foot_right",
		"spine_2",
		"spine_5",
		"neck_1_base",
		"clavical_left",
		"clavical_right",
		"shoulder_left",
		"shoulder_right",
		"elbow_left",
		"elbow_right",
		"wrist_left",
		"wrist_right",
		"palm_left",
		"palm_right"
		};


//array which holds the global names of all the animated joints
string $list_b[22] =
		{
		"spine_1_Base"
		"spine_1_Base|pelvis_left",
		"spine_1_Base|pelvis_left|knee_left",
		"spine_1_Base|pelvis_left|knee_left|ankle_left",
		"spine_1_Base|pelvis_left|knee_left|ankle_left|foot_left",
		"spine_1_Base|pelvis_Right",
		"spine_1_Base|pelvis_Right|knee_right",
		"spine_1_Base|pelvis_Right|knee_right|ankle_right",
		"spine_1_Base|pelvis_Right|knee_right|ankle_right|foot_right",
		"spine_1_Base|spine_2",
		"spine_1_Base|spine_2|spine_3|spine_4|spine_5",
		"spine_1_Base|spine_2|spine_3|spine_4|spine_5|neck_1_base",
		"spine_1_Base|spine_2|spine_3|spine_4|spine_5|clavical_left",
		"spine_1_Base|spine_2|spine_3|spine_4|spine_5|clavical_right",
		"spine_1_Base|spine_2|spine_3|spine_4|spine_5|clavical_left|shoulder_left",
		"spine_1_Base|spine_2|spine_3|spine_4|spine_5|clavical_right|shoulder_right",
		"spine_1_Base|spine_2|spine_3|spine_4|spine_5|clavical_left|shoulder_left|elbow_left",
		"spine_1_Base|spine_2|spine_3|spine_4|spine_5|clavical_right|shoulder_right|elbow_right",
		"spine_1_Base|spine_2|spine_3|spine_4|spine_5|clavical_left|shoulder_left|elbow_left|wrist_left",
		"spine_1_Base|spine_2|spine_3|spine_4|spine_5|clavical_right|shoulder_right|elbow_right|wrist_right",
		"spine_1_Base|spine_2|spine_3|spine_4|spine_5|clavical_left|shoulder_left|elbow_left|wrist_left|palm_left",
		"spine_1_Base|spine_2|spine_3|spine_4|spine_5|clavical_right|shoulder_right|elbow_right|wrist_right|palm_right"
		};


//this loop will copy all the keyrframe from one frame in the animation to another frame
//useful for removing frames
for( $c=0; $c<20; $c++ )
		{
		currentTime 0;
		$a = `xform -q -os -ro $list_a[$c]`;
		currentTime 1;
		xform -os -ro $a[0] $a[1] $a[2] $list[$c];
		setKeyframe ( $list_a[$c] + ".rx" );
		setKeyframe ( $list_a[$c] + ".ry" );
		setKeyframe ( $list[_a$c] + ".rz" );
		}
}
```

Appendix 2 – Blending Scripts cont…

```
// This script copies the rotation values on a series of static skeletons and applies them to one model

int $c;
int $x;

for ( $x=0; $x<6; $x++)
{
	currentTime $x;

	for( $c=0; $c<21; $c++ )
	{
		$s = `xform -q -os -ro ( "blend_"+ $x + "|" + $list_b[$c] )`;
		xform -os -ro $s[0] $s[1] $s[2] ( "blend|" + $list_b[$c] );

		setKeyframe ( "blend|" + $list_b[$c] + ".rx" );
		setKeyframe ( "blend|" + $list_b[$c] + ".ry" );
		setKeyframe ( "blend|" + $list_b[$c] + ".rz" );
	}

	$s = `xform -q -os -ro ( "blend_"+ $x + "|spine_1_Base" )`;
	xform -os -ro $s[0] $s[1] $s[2] ( "blend|spine_1_Base" );

	setKeyframe ( "blend|spine_1_Base.rx" );
	setKeyframe ( "blend|spine_1_Base.ry" );
	setKeyframe ( "blend|spine_1_Base.rz" );

	$s = `xform -q -ws -t ( "blend_"+ $x + "|spine_1_Base" )`;
	xform -ws -t $s[0] $s[1] $s[2] ( "blend|spine_1_Base" );

	setKeyframe ( "blend|spine_1_Base.tx" );
	setKeyframe ( "blend|spine_1_Base.ty" );
	setKeyframe ( "blend|spine_1_Base.tz" );

}


//this script connects all the keyed attribute of the skeleton to the blend attribut in the parent group
for( $c=0; $c<20; $c++ )
{
	int $num = ` keyframe -q -kc ( "blend2|" + $list_b[$c] ) `;

	if($num > 0)
	{
		connectAttr -f blend2.time ( $list_a[$c] + "_rotateX.input");
		connectAttr -f blend2.time ( $list_a[$c] + "_rotateY.input");
		connectAttr -f blend2.time ( $list_a[$c] + "_rotateZ.input");
	}
}

	connectAttr -f blend2.time spine_1_Base_rotateX.input;
	connectAttr -f blend2.time spine_1_Base_rotateY.input;
	connectAttr -f blend2.time spine_1_Base_rotateZ.input;

	connectAttr -f blend2.time spine_1_Base_translateX.input;
	connectAttr -f blend2.time spine_1_Base_translateY.input;
	connectAttr -f blend2.time spine_1_Base_translateZ.input;
```

Appendix 2 – Blending Scripts cont…

```
//this script interpolates between two skeleton poses and applies each inbetween pose to a static skeleton

float $t[3];
float $t[3];
float $dif[3];
int $c;


for( $c=0; $c<22; $c++ )
	{

	$w = `xform -q -os -ro ( "windmill|" + $list_b[$c] )`;
	$t = `xform -q -os -ro ( "tick|" + $list_b[$c] )`;

	$dif[0] = $w[0] - $t[0];
	$dif[1] = $w[1] - $t[1];
	$dif[2] = $w[2] - $t[2];

	$dif[0] /= 4;
	$dif[1] /= 4;
	$dif[2] /= 4;

	$t[0] += $dif[0];
	$t[1] += $dif[1];
	$t[2] += $dif[2];

	xform -os -ro $t[0] $t[1] $t[2] ( "blend_1|" + $list_b[$c] );

	$t[0] += $dif[0];
	$t[1] += $dif[1];
	$t[2] += $dif[2];

	xform -os -ro $t[0] $t[1] $t[2] ( "blend_2|" + $list_b[$c] );

	$t[0] += $dif[0];
	$t[1] += $dif[1];
	$t[2] += $dif[2];

	xform -os -ro $t[0] $t[1] $t[2] ( "blend_3|" + $list_b[$c] );

	}
```