Proceedings of the 7th International Conference in Central Europe on Computer Graphics, Visualization and Interactive Digital Media '99 (WSCG '99), University of West Bohemia, Plzen, Czech Republic, 8-12 February, 1999. Editor: Vaclac Skala. ISBN: 80-7082-490-5 Pages: 225-232.

MOTION BLENDING USING A CLASSIFIER SYSTEM

Tony Polichroniadis and Neil Dodgson

Computer Lab
New Museum Site
University of Cambridge
Pembroke Street
Cambridge
England

tpp21@cam.ac.uk http://www.cl.cam.ac.uk/users/tpp21
nad@cam.ac.uk http://www.cl.cam.ac.uk/users/nad

ABSTRACT

Motion blending is commonly thought of as creating the transition of an animated figure to the first frame of a piece of motion. We describe a new way of thinking about motion blending by removing the assumption that each piece of motion must start at the first frame. We describe a classifier system that finds the nearest match in a set of animation frames to a given state of an animated figure and show how this classifier can be used to create better motion blends. We also describe how the parameters for this system were optimised using a genetic algorithm.

The classifier given was found to be efficient enough to work in real time with many articulated figures.

CR Categories: I.3.7 [Computer Graphics]: Animation

Keywords: computer animation, motion control, human figure animation, motion-blending.

INTRODUCTION

With the current trend of building libraries of human motion segments it is becoming increasingly important to find robust ways of creating the transitions between arbitrary pieces of captured motion in order to create long, visually correct animations.

We describe a new way of thinking about motion blending by removing the assumption that each piece of motion must start at the first frame. We describe a classifier system that finds the nearest match in a set of animation frames to a given state of an animated figure and show how this classifier can be used to create better motion blends.

The following is a formal description of what we mean by finding a motion blend. The human figure's state is given as some point in joint angle space ($\rho = \Re^{dof}$) and a position and orientation of the root (an arbitrary point on the figure, we choose it to be the base of the spine). Joint angle space is the orientation of each of the joints in an animated figure and *dof* is the number of degrees of freedom of the figure. We are given the following:

- An initial position of the figure $\{I \mid I \in \rho\}$.
- An initial rotational velocity of the figure's d.o.f.s: $\left\{S \mid S \in \frac{d\rho}{dt}\right\}$
- A segment of motion, time from 0 to t_1: $\{M(t) \mid t \in \Re, t \in [0,t_1], M(t) \in \rho\}$, i.e. how the joints vary over time.
- A set of times, indexed from 0 to i_1: $\{ Key_i \mid i \in \aleph, i \in [0,i_1], Key_i \in \Re,$ $Key_i \in [0,t_1] \}$, which are called keyframes.

A good choice of keyframes is important to our motion-blending algorithm. They must represent important points in the motion such as turning points.

Now we must define a new sequence of motion: $B(t)$, that blends from I into M. We make no assumptions about how long B is, or the position of the root at the end of B. This is very important to our design philosophy. For example, in the case of walking we believe that, if an autonomous actor's body is put into some initial state and then told to walk, the computer should decide where its first footstep lies.

The remainder of this paper is split into 6 sections. Section 1 describes previous work on this problem to show where we fit into other research. Sections 2 and 3 describe our algorithm for doing motion blending. Section 4 describes how we optimised the parameters to this algorithm. Section 5 describes our implementation of this system, and finally, Section 6 rounds off the paper and describes possible further work.

BACKGROUND

The majority of solutions to this approach have relied on having previous knowledge of the set of basis motions. They will either:

1. Make sure that the end of one piece of motion perfectly fits into the start of the next piece.

or

2. Have a previously created library of motions that blend from the end of any piece of motion to the start of any other [Moraw90a].

The first approach requires laborious hand crafting, in all but the simplest situations. The second approach requires a number of blends that increases as the square of the number of pieces of motion.

They both have the disadvantage of requiring all the pieces of motion to be enumerated, precluding the possibility of hand animating any parts of the animation or having animation automatically generated in real time. To put it simply, the animator is stuck with the library of animations for which blends have been computed. They both also face the problem that the basis motions must fully complete before changing to a different piece of movement.

Several other groups have attempted to solve the general motion-blending problem by designing an algorithm that creates an interpolation between pieces of motion. This has been approached by couching it as an optimisation problem [Rose96a] or as a signal-processing problem [Brude95a, Witki95a, Unuma95a].

In contrast to the other methods, we think of the motion blending as having a second part *in addition* to the interpolating system. Before the interpolator is called, a classifier system finds the closest match in *M* to *I*. The match can then be used by the interpolator to create a better motion blend by missing out superfluous frames from *M*.

For instance, consider a throwing motion: the hand is pulled back, then moved forwards fast. If in *I* the arm is already pulled back part of the way, then the blend should be from the partially pulled back position to fully pulled back, and then the throw. The classifier removes the need to move forwards in between, something which all previous methods would have insisted upon.

This process works best for looped motion (such as walking), because the blend does not need to reach a specific position exactly at the end, as long at it tends towards the correct motion. For motion that isn't looped, for instance a triple jump, it still works well, as the matches found are often very good. However, a constraint must be imposed that forces the matching procedure to give results that lie within certain bounds within *M*. For instance, in the case of the triple jump, a match in mid-skip would be no good.

THE CLASSIFICATION ALGORITHM

This section describes the algorithm used to find the closest matching set of joint angles in a short piece of motion. We use joint angles as the input, as this gives the classification algorithm more information than end effector position alone. After trying many standard classifier systems, such as neural nets, we settled on a system that works on three levels. A rough guess is made for each limb. Then the whole body is examined to see if it is possible to classify it under one value of *t*. Finally, a refinement to the answer is made by each joint so that they all have individual guesses of *t*.

It is important to point out that while this classifier is not provably correct, it works very well in practice.

Macro frames

The first step in the algorithm is performed at the limb level. It tries to find out what stage in the animation each limb is nearest, to the resolution of the nearest key frame, (we call this guess t_l). This is done by scoring each keyframe in the animation by summing a weighted squared difference between that keyframe's joint angles, and the actual joint angles of *I* and a squared difference between the keyframe's joint velocities and the actual joint velocities:

$$Score_P(Key_i) =$$

$$\sum_{\forall d \in dof} weight_P[d] * (I[d] - M(Key_i)[d])^2$$

$$Score_V(Key_i) =$$

$$\sum_{\forall d \in dof} weight_v[d] * (S[d] - Diff_d(Key_i))^2$$

Where:

$$Diff_d(Key_i) =$$

$$(M(Key_i)[d] - M(Key_{i-1})[d]) / (Key_i - Key_{i-1})$$

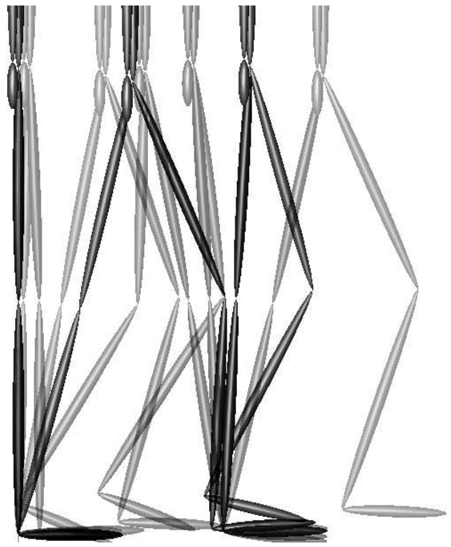
For more discussion of the weights, see the Parameter Optimisation section[1].

[1] In this paper, reference is often made to a specific limit or special scaling factor. These values are all discussed in the Parameter Optimisation section.

We wish to find the keyframe that would best *follow I.* The lowest $weight_p$ gives the *nearest* keyframe to *I*. As the velocity term describes transitions between keyframes, the smaller of $weight_v$ and $weight_{v+1}$ will be used to decide if the correct keyframe[2] is Key_i or Key_{i+1}, i.e. velocity is used to determine if the figure is approaching or leaving the nearest keyframe.

Co-ordination

After a rough guess has been made for each of the limbs, control is passed to the top level. At this point limbs can decide to change their guess of t_l to one guessed at by another limb. This is done by finding the most confident limb (the one with the lowest $score_p$), and suggesting that the other limbs change to that limb's keyframe. Each will do so if its time differs by more than a fixed constant from the most confident limb, and the drop in confidence is lower than another fixed constant.



This is important in the case of creating a blend from standing still into walking. In the standing position both legs are in the support position, which means one leg is half way through the animation compared to the other. By making the decision to make both legs choose the same time we don't get awkward postures while one leg 'catches up' with the other. The picture above shows how a leg can smoothly transition into the middle of the walking motion after co-ordination has chosen a single point in time for the whole figure.

[2] It is assumed that all arithmetic with frame numbers in looped motion is to be done modulo the number of frames, so it should read $Key_{(i+1) \bmod i_1}$

Refinement

Now, each d.o.f. has the opportunity to give a better estimate for its own individual guess at the time (denoted by t_d). As the guess made by each limb is the immediately following keyframe, it is a matter of calculating how long it should take for each d.o.f to reach that keyframe. The value can be subtracted from the initial guess to give a better estimate.

The amount of time it takes for a joint to reach its keyframed value depends on two factors:

- The time it takes to reach the required joint position, which we approximate by:

$$t_d = Key_i - Abs(I[d] - M(Key_i)[d]) / Abs(Diff(Key_i))$$

- The amount of time it takes to match velocity with the keyframe's required velocity, which we approximate by:

$$t_d -= Abs(S[d] - \tfrac{\partial}{\partial t} M(t_d)[d]) / Abs(Diff(Key_i))$$

This gives a time estimate for individual joints that is higher granularity than the initial keyframe estimate.

CALCULATING THE MOTION BLEND

As can be seen from the algorithm described above, each d.o.f. calculates its own value of t_d. The question this section resolves is how we use these values to compute a motion blend.

Before doing so, we must introduce the concepts of Front and Furthest. Front gives the time in M that I is closest to, and Furthest is the time in M that is furthest away from Front but is still a possible estimate.

Computing Front and Furthest is trivial in a non-looped animation. They are simply the maximum and minimum values of t_d, (Shown in Figure 1).

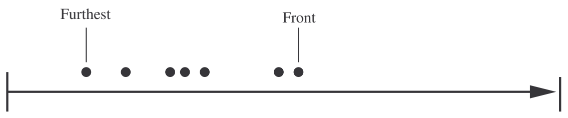


Figure 1.

For a looped animation, one must find the smallest range of time in looped space that encloses all t_d. Furthest and Front are then the d.o.f.s at the start and end of the set, (Shown in figure 2). In a tie (two d.o.f.s with the same t_d), the decision can be made arbitrarily.

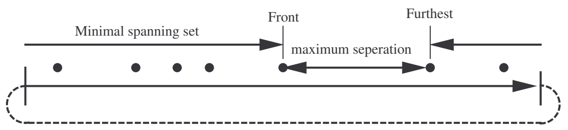


Figure2

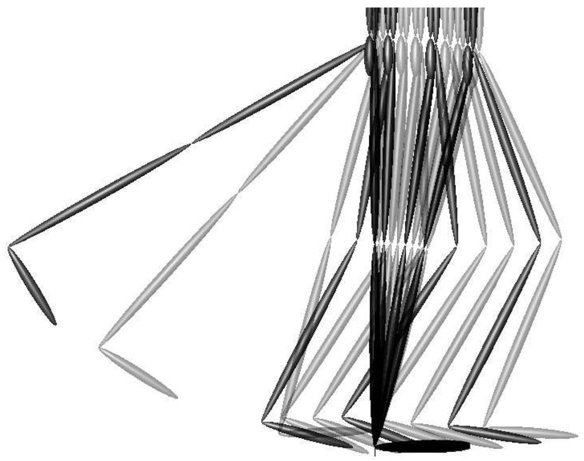
Any of the methods described in Section 1 could be combined with the classifier. All of the methods described naively assume $t = 0$ for all d.o.f.s in the figure. To take advantage of the classifier, one must blend into the position at t = Front. To show why, we consider looped and non-looped motion separately.

In the non-looped case, it is obvious to see that any choice other than Front would result in the Front d.o.f. going backwards to catch up with the choice, then forwards again when the rest of the animation is played.

In the looped case, we are trying to minimise the worst difference between any individual t_d and the time chosen for the whole figure. This is an equivalent problem to choosing Front.

The number of frames allocated for the blend will depend on the time between Furthest and Front. In other words, if they are far apart, then it will take longer for Furthest to catch up with Front.

The calculated t_d can be used further to compute an even better blend. If there are keyframes between any t_d and Front, then the blend can go via those keyframes, instead of just moving straight into



In the above example, the figure starts off in the awkward position of having the left leg thrown back, and is made to walk. The algorithm decides the right leg is in the support phase, and the left leg is nearest the back swing. In order to blend into walking, the figure must swing the left leg forward. By using the in between keyframes as hints, it knows to bend at the knee when it comes forward, instead of just throwing its leg forward, straight.

By using the algorithm shown we found that no further optimisation was needed to create realistic looking blends. Simply using Hermite curves to create smooth transitions is enough. However, we are only interested in looking at motions in which dynamics takes a limited participation. For blending movement in sports, for instance, it is more likely that the figure will end up in dynamically unfeasible positions, or unbalanced. More research would be needed to give this algorithm the concept of dynamics.

PARAMETER OPTIMISATION

As we mentioned at the start of the article, there are many parameters to this algorithm. The parameters include the weights for the d.o.f.s. and constants for the co-ordination algorithm. A more in depth discussion of the specific parameters is given in Appendix A.

To determine good values for these parameters, we formulate the selection of these parameters as an optimisation problem. As all the parameters are contributing to the algorithm in a non-trivial manner, standard optimisation approaches are not applicable. We have decided to use a genetic algorithm to optimise the parameters [Back96a] [Goldb89a]. In order to do this, we must decide on a measure of fitness for a set of parameters. It was decided that one definition of a good motion blend was one that managed to get the figure into a useful state as soon as possible.

Please note that this is subtly different from getting the figure to move as quickly as possible. The algorithm has no ability to change how fast the joints transition, just what they transition to. This is because the angular velocity is decided by the refinement algorithm, that chooses how far away each joint is from its required position, and hence the minimum amount of time for the transition. In other words, we are optimising the controller to make decisions that do not result in the figure going into pointless positions while blending.

An alternative to optimising for an efficient controller would be to optimise towards a minimal torque solution, i.e. one that favoured blends that required the smallest amount of energy to be used.

To characterise our requirement, we set the figure into a number of random start states, then blended it into a walk motion. The fitter sets of parameters would get the figure walking sooner than unfit sets, so the further the figure walks, the better the set of parameters. Fitness was hence defined as $1.0/(1.0+\max(dist)-dist_i)$ because this gives small improvements in distance a large enough genetic advantage.

After running the genetic algorithm, the new set of parameters found resulted in the character moving 1.5 times further than our initial choice. Visually, the motion was more convincing.

IMPLEMENTATION

The described algorithm was implemented in C++ on top of our agent based animation environment, *Jake*. The parameters for the genetic algorithm are shown in Table 1.

Parameter	Value
Population Size	50
Number of random positions	100
Number of generations	50
Mutation probability	1%

Table 1.

The genetic algorithm took roughly 3 hours to complete on 195MHz R10000. The classifier takes around 200 milliseconds, on an eight keyframe animation, on the same computer.

The results of the genetic algorithm showed up some interesting features. For the hip values, which were duplicated on the left and right hand side of the body, each time we ran the algorithm they came up with identical values (within statistical error). However, the knee and ankle values varied. This is likely to be due to there being a wide range of values that could be possible for the knee and ankle weights, while the hip values were more constrained as to the values they could take.

FURTHER WORK

The above algorithm works very well in cases where the initial position is fairly close to a position in the animation. However in starting positions that are completely different from any position in the animation the method cannot be expected to perform as well. For instance, in the case where the figure is lying down on the floor, and it is asked to blend into a walk, you could not expect it to formulate the complex piece of motion involved in getting up off the floor. We propose that the concept of a basis motion is redefined as multiple segments of motion that all converge on the last frame of the animation. For instance a walk could be defined as: the looped walk, a blend from a crouch into a walking stance, and lying down into a crouch. In this way we can more fully span the space of figure states. As long as the figure starts off in a state that is 'near' to one of the branches of the tree, a good blend will be ensured.

An alternative approach would be to create libraries of blends for which the algorithm could parasite. Using our nearest match algorithm, the most appropriate blend would be chosen automatically. Work would have to be done in co-ordinating limbs so that they chose frames from the same blend.

APPENDIX A: PARAMETERS TO THE ALGORITHM

This section discusses the parameters to our implementation of the system described in the paper. These are given as example parameters. As the specific algorithms for certain parts of this system are not given, a different implementation may need different parameters.

The proportional weightings for the degrees of freedom (as discussed in Section 2.1) will always be needed. Their values are given in Table 2.

Name	Weight	Value
$Weight_p[hip]$	Hip position	5.5
$Weight_p[knee]$	Knee position	0.2
$Weight_p[ankle]$	Ankle position	0.05
$Weight_v[hip]$	Hip velocity	0.36
$Weight_v[knee]$	Knee velocity	0.12
$Weight_v[ankle]$	Ankle velocity	0.01

Table 2.

In our implementation, the only other part of the algorithm that was parameterised was the co-ordination phase (Section 2.2). The decision criteria was:

```
if( Confidence lost < BribeFactor
AND Difference in t_l >
SuggestFactor )
      ChangeFrameToSuggestedOne()
```

BribeFactor was found to be 1.1 and SuggestFactor was found to be 0.05 seconds (Remember, the smallest, nonzero value for the difference in t_l's, at this stage in the algorithm, is only going to be as small as the smallest difference between any two Key_i s).

ACKNOWLEDGEMENTS

I would like to thank the Rainbow Group for their continued support and good humour with the Jake project. I would also like to thank the EPSRC for their sponsorship during this research.

REFERENCES

[Brude95a] Armin Bruderlin and Lance Williams: Motion Signal Processing in Computer Graphics (SIGGRAPH '95 Proceedings), 1995 pp.97-104

[Witki95a] Andrew Witkin and Zoran Popovic Motion Warping in Computer Graphics (SIGGRAPH '95 Proceedings), 1995, pp.105-108.

[Unuma95a] Munetoshi Unuma, Ken Anjyo and Ryozo Takeuchi Fourier Principles for Emotion-based Human Figure Animation in Computer Graphics (SIGGRAPH '95 Proceedings), 1995 pp.91-95

[Rose96a] Charles Rose et al. Efficient Generation of Motion Transitions using spacetime constraints. in Computer Graphics (SIGGRAPH '96 Proceedings) 1996, pp.155-162

[Moraw90a] Claudia L. Morawetz and Thomas W. Calvert: Goal-Directed Human Animation of Multiple Movements in Graphics Interface'90 pp.60-67

[Back96a] Thomas Back: Evolutionary algorithms in theory and practice: evolution strategies, evolutionary programming, genetic algorithms Oxford: Oxford University Press, 1996

[Goldb90a] David E Goldberg. Genetic algorithms in search, optimization, and machine learning Reading, Mass.; Wokingham: Addison Wesley, 1989.