Search and Transitioning for Motion Captured Sequences

Suddha Basu* IIT Bombay
Shrinath Shanbhag† IIT Bombay
Sharat Chandran‡ University of Maryland & IIT Bombay

Abstract

Animators today have started using motion captured (mocap) sequences to drive characters. Mocap allows rapid acquisition of highly realistic animation data. Consequently animators have at their disposal an enormous amount of mocap sequences which ironically has created a new retrieval problem. Thus, while working with mocap databases, an animator often needs to work with a subset of "useful" clips. Once the animator selects a candidate working set of motion clips, she then needs to identify appropriate transition points amongst these clips for maximal reuse.

In this paper, we describe methods for querying mocap databases and identifying transitions for a given set of clips. We preprocess clips (and clip subsequences), and precompute frame locations to allow interactive stitching. In contrast with existing methods that view each individual clips as nodes, for optimal reuse, we reduce the granularity.

1 Introduction

Virtual humanoid characters are often driven by hierarchical skeletons consisting of 30 or more degrees of freedom. Animating such characters manually is a daunting task at best. Instead, in recent times, the data for such characters can be acquired directly from a live performer using motion capture devices. Indeed, this method has gained wide acceptance as motion capture is the fastest way to generate rich, realistic animation data.

While working on a new animation sequence, animators may either choose to capture the data afresh or they may choose to synthesize the sequence using existing precaptured clips. The decision is influenced by several factors. The number of existing clips available for reuse, suitability of existing clips for creating the new sequence, suitability of current state of the art techniques for reusing existing clips, availability of the performer for capturing a new sequence, time and cost required for capturing the new sequence are all relevant. Nevertheless, with time, the number of clips already captured increases relentlessly.

Recent ([Bruderlin and Williams 1995], [Unuma et al. 1995], [Gleicher 1997][Gleicher 1998], [Lee and Shin 1999]) motion editing methods allow animators to synthesize new animation in a number of interesting ways. While attempting such synthesis the animator needs to perform two fundamental tasks – search the collection of existing clips for clips matching certain criteria, and compute transition points within a working set of such clips. The complexity of each of these tasks increases exponentially with number of clips. Hence schemes to assist the animator are in order.

Prior schemes such as [Kovar et al. 2002] are one way to compute transition information between clips. However, our observation is that such techniques, when operated at a clip level, do not possess the desired granularity to allow transitioning out at arbitrary frames. This is because they view entire motion clips as a node and only record a single transition between any two given clips. Other schemes such as [Arikan and Forsyth 2002] do not allow the animator to explicitly choose motion clips except for allowing selection of frames at sparsely defined constraint points. While such schemes are able to approximate specified motion requirements, animators often demand more control over the process of motion assembly.

In summary, then, there are two seemingly contrary issues plaguing reuse of motion capture data. The first is the problem of selecting clips that the animator deems relevant. The second is the process of taking parts of these clips for synthesis in a animator controlled way. Too small a set in the first step enables manual control at the expense of variety.

1.1 Our contributions

To address the second issue, we automatically chop individual clips and collect similar sub-clip sequences. Such a subsequence could be even half a dozen frame long. As a result, reuse is possible at a much smaller granularity. Seemingly *unrelated clips* are brought together and thus potentially reused. All such sequences are collected into nodes in a *cluster graph* that works much the same way as motion graphs. Further, we show how subsequences within a node in a cluster graph can be aligned to enable smooth stitching.

To address the first issue, our method allows the animator to *query by example* the motion database. The number of resulting frames is large but the size of the selected clips is a small fraction of the original motion capture database. As a result we avoid creating too large a cluster graph (technically a hypergraph) from the chopped clips.

The rest of the paper is organized as follows. In the next section, we discuss our work in the context of related work. Section 3 has the three main parts of our work. We first discuss how matching clips can be retrieved when presented with a query. Next, we discuss how these clips can be placed into a cluster graph. Finally, we show the process of stitching subsequences by entering a cluster graph node, visiting one of the clip subsequences therein, transiting to another subsequence within the node, and then exiting to another clip. In the last section, we make concluding remarks.

2 Related Work

[Kovar et al. 2002], [Lee et al. 2002] describe techniques to create new motion sequences from a corpus of motion data. Each technique essentially clusters similar motion into nodes. Then it builds a graph of nodes, where each edge represents a transition between nodes. A walk through the cluster node graph results in synthesis of new motion sequences. The techniques differ in metrics used for clustering, pruning schemes and control criteria for node walk. [Arikan and Forsyth 2002] describe a technique using novel search method based around dynamic programming to interactively synthesize motion from annotations. The user paints a timeline with annotations and the system assembles the motion sequence by extracting suitable frames from a motion database. A scheme for synthesizing missing degree of freedoms and adding details to specified degrees of freedom for a roughly specified motion sequence is described in [Pullen and Bregler 2000], [Pullen and Bregler 2002]. Their method uses correlation between the various degrees of freedom within each motion sequence. Research in indexing mocap data for quick search is in its nascent stage. [Keogh et al. 2004] have proposed a novel technique to speed up similarity search under uniform scaling, based on bounding envelopes for indexing human

*e-mail: skbasu@cse.iitb.ac.in
†e-mail: svs@it.iitb.ac.in
‡e-mail: sharat@acm.org

motion. [Liu et al. 2005] demonstrate a data-driven approach for representing, compressing and indexing human-motion based on piecewise-linear components obtained using K-means clustering.

3 Our Method

In this section we describe our *query by example* and *cluster graph* scheme to enable search and transitions for mocap data. We view motion data consisting of a bundle of signals. Each signal represents a sequence of sampled values for one degree of freedom (DOF) of an articulated skeletal model. These signals are sampled at discrete instances of time with a uniform sampling interval to yield a motion clip. In each frame, the sampled values of the different DOFs at each joint determine the configuration of an articulated figure for that frame. Often the root of the skeletal hierarchy contains six DOFs (three for translation and three for rotation about x, y and z axis), whereas the rest of the nodes contain three DOFs (only rotation).

3.1 Query By Example

Our first step is to search for the 'matching clips in the mocap database, corresponding to an animator sketched sequence. For the sake of exposition, we use only the signal corresponding to the most significant DOF. Later, we relax this and consider a set of signals. Thus we have the problem

"Given a set S of N signals, and a query signal, find the best P matches for this query from S."

The first step in retrieving queries is to identify the continuous monotonic portions (termed **fragments**) of the signals. The signals are clipped at points where the first derivative changes sign. This is a reasonable choice for a motion sequence as at these points the joint angles just start changing from one direction to another.

Fragments are important in our method. For example, matches are allowed to be disjoint. That is, matches found may not necessarily have the i^{th} and $(i+1)^{th}$ fragments occurring consecutively in the original signal. Fragments enable "mining" of possibly unusually related clips. Fragments also enable robust scaling. For instance, if the query signal is sampled at half or double the rate, the method returns similar matches. For brevity, we skip details of scaling in this version of the paper.

3.1.1 Matching

Matching between two fragments of equal length is easy and related to the error $\text{err}_{i,j}$ between fragment i of the query signal and candidate fragment j of one of the target signals. For each i, we find the K minimum values of $\text{err}_{i,j}\ \forall j$ and store the corresponding values of j. These values are the indices of the best matching fragments for fragment i and enable us to determine useful clips.

For the sake of determining the quality of the match, we not only retrieved clips, but also stitched portions of them together. We constructed (as in [Pullen and Bregler 2002]) our matching windows in a way so as to maximize the number of consecutive fragments in the path.

3.1.2 Multiple DOF Consideration

Here the query consists of a set of s signals, each representing some different degree of freedom for some joint angle but occurring at the same time. Similarly, the search space now consists of N sets of signals. Each set has s signals, and the q^{th} signal represents the same joint angle as the q^{th} signal in the query set.

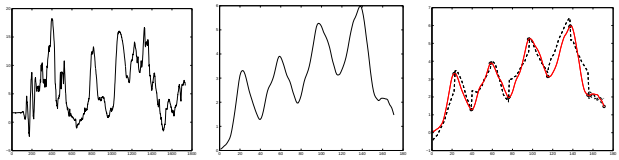
Two significant changes are required to apply the single DOF algorithm to this multi-signal case. First the error calculation is over the s signals. Additionally we choose a *driving signal* which drives the fragmentation of the signals. Fragmentation of each individual signal at points where the first derivative changes sign would be an incorrect approach simply because these break points would not coincide in time. Thus we first choose an index q_d and fragment the q_d^{th} signal in each set using the method discussed earlier. The remaining signals in each set are fragmented at same points of time as the q_d^{th} signals are broken.

3.1.3 Experimental Results and Observations

The aim of experimentation was to see how the above methods work on real data sets. For our experiments, we have used various *.bvh* files [bvh 2004]. In all the examples cited here, we have taken walk sequences as queries.

Single Sequence Queries

As shown in Fig. 1(c), we find the results quite satisfactory. Note that the resulting output from the dance clip consists of non-contiguous fragments. We also observe that there is a significant resemblance of the query subsequence from walk clip with the dance sequence for the joint angle and degree of freedom considered.



(a) Input: A dance clip appearing in the database

(b) Input: A query sequence extracted from a walk clip

(c) Output: Stitched sequence from the dance clip overlaid on the query.

Figure 1: Sample result of querying by example.

We carried out a series of experiments to examine the qualitative performance of the algorithm. Different sub-sequences, as in 1(b), from the walk clip were queried against the dance clip in Figure 1(a). The performance was judged using the total error in the best match found and the number of sudden jumps in the synthesized paths as parameters. Table 1 shows some results of the experiments.

Observations Query-by-example using a single degree of freedom allows us to mine mocap database, retrieve interesting fragments, and stitch them together. Two drawbacks stymie this approach, however.

First, the time complexity for finding the min-cost path is exponential w.r.t. the number of fragments in the query ($O(K^{N_{frag}})$ where N_{frag} is the number of fragments of the query signal. For K matches, the total number of paths is $K^{N_{frag}}$. Here a path is just an ordered tuple of indices $\{j_1, j_2, j_3, \ldots, j_{N_{frag}}\}$, where each j_i is among the best K matches for fragment i. Although, K is chosen to be 3, and N_{frag} varies from 2 to 7 in most cases, the optimization of this step is very essential. In our implementation, we discarded paths which do not have any '0-cost' transition (paths in which all joining fragments are non-consecutive). This reduces the storage space to vary linearly with the total number of '0-cost' transitions.

Nevertheless, when N_{frag} is large, we do not have the luxury of choosing K too generously.

Second, and perhaps as significant, when we ran the method using multiple degree of freedoms, we found the quality of the match to be not acceptable.

3.2 Cluster Graph

Our experiments in the previous section forced a different approach. Instead of trying to provide a fully automated way of searching and synthesizing a motion clip, we use only the first step to retrieve candidate clips. Subsequences of these candidate clips are organized into a cluster graph. The cluster graph is then used for synthesizing new clips.

The cluster graph structure consists of nodes and edges. Nodes contain frames from one or more clips. Frames within a node are "similar;" that is, the error between any two frames is below a threshold. Edges are obtained from the natural sequential ordering of clips within nodes. Figure 2 shows an example.

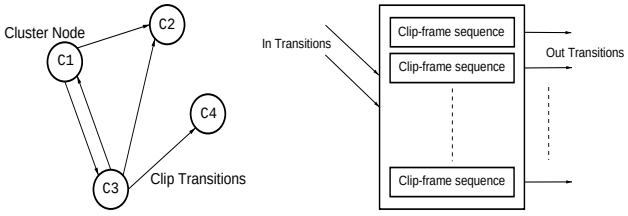


Figure 2: Cluster graphs contain clip subsequences organized efficiently.

Within a node, the frames are sorted by clips and time. Contiguous sequences of frames are collected together into a structure called *clip-frame sequence*. We maintain out-transitions for each clip-frame sequence for each cluster node. Maintaining one transition per clip-frame sequence automatically prunes away transitions from contiguous frames.

3.2.1 Cluster graph advantages

- Cluster graphs provide a level of granularity smaller than those of motion graphs. Traditional motion graphs record only one transition point between each pair of clips. As a result this data structure has been used to find transitions between two clips without enforcing a hard time constraint. For the realtime version, time is an important factor. We may need to transition between two clips at precise, or more controlled instants in time. Therefore it is useful to maintain as many distinct transition points as possible. Note further that transition points occur in bunches. Cluster graphs prune multiple transition points lying very close to each other temporally.

- Cluster graphs enable looping; that is it enables an animator to create a walk sequence that can go on forever, instead of being limited to the 10 second footage. A node containing multiple clip-frame sequences from the same clip indicates the possibility of looping.

3.2.2 Constructing cluster graphs

Once a similarity metric has been selected between two frames, an algorithm reminiscent of Kruskal's minimum spanning tree algorithm is used to create clip-frame sequences. We first preprocess all n frames so that we have the distance set D for all $\binom{n}{2}$ frames. Next,

1. Sort D into $\pi = (o_1, o_2, \ldots, o_k)$ by non-decreasing values.
2. Start with $F^0 = \{\}$.
3. Repeat Step 4 for $o_q = o_1, \ldots, o_k, k = \|D\|$.
4. Construct F^q given F^{q-1} as follows. Let o_q correspond to frame pair (i, j). If o_q is small compared to a threshold, then
   (a) Add frames (i, j) to F^{q-1} and refer to this pair as a single frame called I_j.
   (b) Remove from D all references to either i or j.
   (c) Set distance I_j for all frame pairs (I_j, p), $p \neq i, k \neq j$ to be the minimum of the distances (i,k) and (j,k). Insert these distances maintaining the sorted order.

The algorithm can be efficiently implemented. Specifically each insert and removal in Step 4 is $O(\log n)$.

3.3 Clip-frame Sequence Transitioning

Once clip-frame sequences are clustered in a graph, each cluster node contains similar frames. We now find the best frames in each motion clip within a node for smooth intra-cluster transitions.

First, we find the best point of transition between two motion clip-frame sequences. We choose the frame pair for which the sum of distances between them, and the derivatives at the respective points, are minimum. Using derivatives ensures continuity of the stitched motion curves. Next, we iteratively determine a set of ordered m-tuple where m is the number of clip-frame sequences. Each element of the set contains m frame indices such that these frames in the corresponding clip-frame sequences are the closest matches for each other. Hence, we transition between two clip-frame sequences in the cluster node, at these frames, to get a smooth motion sequence. Figure 3 shows plots for three degrees of freedom using this implementation for transitioning between two clips. The transition between the two clip-frame sequences is smooth and continuous.

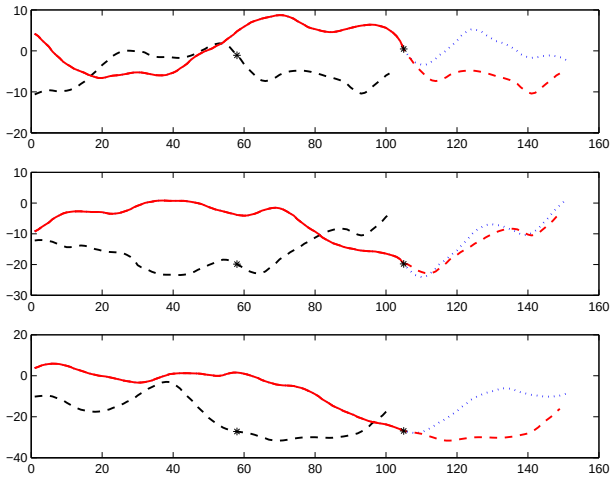


Figure 3: Two input clips are stitched. The first one is 150 frames long, and is shown in two parts: solid and dotted. The second is 100 frames long and is shown as dashed. The stitching is overlaid on the input to show the smoothness across three different degrees of freedom. In any of these three, the output clip starts from the first clip, and meets the second clip at the 103rd frame. The portion starting from the 58th frame of the second clip is appended at this transition point. Notice that the stitched sequence abandons the dotted portion of the first clip.

No.	N_{frag} of query	error in best match			no. of sudden jumps		
		$K=2$	$K=3$	$K=4$	$K=2$	$K=3$	$K=4$
1	7	0.538	0.540	1.005	6	5	3
2	6	1.024	1.024	1.024	4	4	4
3	5	1.412	1.412	1.737	4	4	3
4	8	1.525	1.525	1.525	5	5	5
5	7	0.555	0.726	0.726	5	3	3
6	5	2.330	2.330	2.618	4	4	3
7	5	2.105	2.105	2.105	4	4	4
8	6	0.965	1.886	1.886	4	3	3
9	7	0.538	0.540	1.005	6	5	3
10	6	1.024	1.024	1.024	4	4	4
15	5	2.105	2.105	2.105	4	4	4
25	13	7.169	7.169	7.169	6	6	6

Table 1: Sample results from a series of experiments with the dance sequence (Figure 1(a)) as target. The number of fragments in the target was approximately 319, and walk sequences similar to Figure 1(b) were the queries.

```
S = L_{1,2}
for i = 2 to m − 1 do
  T = {}
  for each element (a,...,p) in S do
    if (p,q) ∈ L_{i,i+1}, then
      Add (a,...,p,q) to T.
    end if
  end for
  S = T
end for
```

Figure 4: The alignment algorithm.

3.3.1 Details

We apply a correlation-based technique on clip-frame sequences M_i and M_{i+1} and get a list $L_{i,i+1}$ of k most suitable pairs of frame indices. $L_{i,i+1}$ contains several pairs of indices (a,b), such that frame f_a^i and f_b^{i+1} resemble each other, and can be obtained in $O(n \log n)$ expected time where n is the size of a typical clip-frame (usually small, between 10-100). Once all $L_{i,i+1}$ are computed ($1 \leq i \leq m$) we look for common indices in adjacent $L_{i,i+1}$ as seen in Figure 4. This algorithm runs in $O(m|\mathcal{S}| * \max|L_{i,i+1}|)$ time. $\mathcal{S}$ and $L_{i,i+1}$ are usually no longer than 20, and m varies from 3-15.

$\mathcal{S}$ typically contains at least one index sequence $j_1, j_2, \ldots, j_m$. Thus if we now want to transition from clip M_4 to M_7, we play M_4 till the j_4^{th} frame and then switch to $(j_7+1)^{th}$ frame in clip M_7. The transition is without any jerks and bumps.

4 Concluding Remarks

In this paper we have provided efficient schemes (with big-Oh expected running times, and supported by experimental results) for the following activities

- Given a (possibly rough) animation sequence, search for clips that contain frame sequences that are similar. Optionally stitch these frame sequences in an optimal way
- Our theoretical and experimental studies indicate that the stitching time is exponential. Sometimes the quality of the automatic stitch may be less than desirable. We therefore create a cluster graph data structure. Cluster graphs preserve the fine granularity of fragments, can be created efficiently and have several advantages (Section 3.2.1).
- Cluster graphs are used interactively by the animator. Although we do not create a fully automated stitched sequence, we efficiently compute the location of possible transition points within a node in the cluster graph, and thus between multiple clips.

References

ARIKAN, O., AND FORSYTH, D. A. 2002. Interactive Motion Generation from Examples. In *Proc. of Siggraph '02*, 483 – 490.

BRUDERLIN, A., AND WILLIAMS, L. 1995. Motion Signal Processing. In *Proc. of Siggraph '95*.

2004. *http://www.bvhfiles.com/bvh_archive/*. Dec.

GLEICHER, M. 1997. Motion Editing with Spacetime Constraints. In *Proc. of the 1997 Symposium on Interactive 3D Graphics*.

GLEICHER, M. 1998. Retargetting Motion to New Characters. In *Proc. of Siggraph '98*.

KEOGH, E. J., PALPANAS, T., ZORDAN, V. B., GUNOPULOS, D., AND CARDLE, M. 2004. Indexing large human-motion databases. In *VLDB*, 780–791.

KOVAR, L., GLEICHER, M., AND PIGHIN, F. 2002. Motion Grahs. In *Proc. of Siggraph '02*.

LEE, J., AND SHIN, S. Y. 1999. A Hierarchial Approach to Interactive Motion Editing for Human like Figures. In *Proc. of Siggraph '99*.

LEE, J., CHAI, J., REITSMA, P. S. A., HODGINS, J. K., AND POLLARD, N. S. 2002. Interactive Control of Avatars Animated with Human Motion Data. In *Proc. of Siggraph '02*.

LIU, G., ZHANG, J., WANG, W., AND MCMILLAN, L. 2005. A system for analyzing and indexing human-motion databases. In *SIGMOD '05: Proceedings of the 2005 ACM SIGMOD international conference on Management of data*, ACM Press, New York, NY, USA, 924–926.

PULLEN, K., AND BREGLER, C. 2000. Animating by Multi-level Sampling. In *Proc. of IEEE Computer Animation 2000*.

PULLEN, K., AND BREGLER, C. 2002. Motion Capture Assisted Animation: Texturing and Synthesis. In *Proc. of Siggraph '02*.

UNUMA, M., ANJYO, K., AND TAKEUCHI, R. 1995. Fourier principles for emotion based human figure animation. In *Proceedings of Siggraph '95*.