Université de Montréal (Département d'informatique et de recherche opérationnelle)
Published May 24th 2007

Technical Report 1296
Automatic Construction of Compact Motion Graphs

Philippe Beaudoin[1] Michiel van de Panne[2] Pierre Poulin[1]
[1]Université de Montréal [2]University of British Columbia

Abstract

Motion capture data often requires substantial processing before it becomes useful. We propose a technique that automatically distills a compact motion graph from an arbitrary collection of motion capture data. At its heart, the process identifies clusters of similar motions which we call "motion bundles". Motion bundles and their encompassing motion graph provide a readily understandable structuring of the motion data. They can serve as a shared tool in support of common types of motion processing, including motion segmentation, motion compression, the creation of blend spaces, and the identification of connectivity to support motion resequencing. We use a novel string-based representation of motions to help find motion bundles. Users can specify a preference for long-duration bundles or bundles containing many motions. We demonstrate results using data for boxing, walking and various exercise motions, and we show that meaningful partitions are retained in the face of noise.

Categories and Subject Descriptors (according to ACM CCS): I.3.7 [Three-Dimensional Graphics and Realism]: Animation

1. Introduction

Data driven character animation requires first collecting a sufficiently large set of motion-captured movements and then processing them to give the required structure for a given application. Processing motions to induce a useful structure has been the focus of much animation research over the past ten years and continues to be an important topic. Seen in the broadest terms, the goal is that of using *analysis* in order to extract models that can then be useful in the *synthesis* of new motions.

Various types of analysis have been developed in support of specific uses, including motion segmentation, motion resequencing, motion blending, query-based motion retrieval, motion compression, and motion annotation. However, the motion analysis tools for these applications have often been developed in isolation of one another. For example, basic motion graphs store the connectivity information required for motion resequencing, but do not detect similar or repeated motions. Similarly, motion compression techniques do not typically pay heed to global motion connectivity.

In counterpoint to specialized approaches, more general approaches have also been proposed for extracting structure from motion corpora. These can be seen as simultaneously achieving motion compression, modeling the connectivity for resequencing, and modeling motion blending or generalization. A first category of approaches uses dynamic statistical models from the machine learning literature, and includes hidden Markov models (HMMs) and switched linear dynamical systems (SLDSs). These stochastic models have achieved success for motion segmentation and categorization, but have, with a few exceptions, met with less success for motion synthesis. A second, more recent approach has been the development of *parametric* [HG07] or *fat* [SO06] motion graphs. In comparison to statistical models, these models are significantly more transparent to end users because they are more consistent with current methodologies for developing character motion models for games and interactive simulations. Their resequence-and-blend based approach for generating motions is also well known to produce high quality results.

The work we present in this paper follows the fat or parametric motion graph approach. Motions in a motion corpus are segmented and clustered into self-similar *motion bundles*. Importantly, the motion-bundle extraction algorithm treats segmentation and clustering together in a single, coupled process. The motion bundles are augmented with connectivity information among motion bundles in order to create a compact motion graph. Our particular contributions are two-fold. First, in contrast to previous work, we develop a fully automated approach to identifying and constructing good clusters of similar motions. Second, we introduce the use of a string-based representation of motions and cast motion clustering as a string clustering problem. The method

supports user preferences in achieving the desired compromise between producing long-duration bundles or highly parameterized bundles containing many motions. The motion bundles can potentially be used for a combination of purposes, including motion compression, motion blending, motion resequencing, and motion retrieval.

The paper is organized as follows. Section 2 discusses related work. Section 3 details the motion-bundle creation algorithm. Section 4 shows how the motion bundles can be used to produce a graph structure. Next, Section 5 presents and analyzes the experimental results for the automatic extraction of motion bundles as well as for motion-bundle graph creation and traversal. Finally, Section 6 concludes by discussing the advantages and limitations of the proposed techniques.

2. Related Work

The past decade has spawned a growing body of work that examines the analysis, synthesis, compression, and retrieval of human motion data. We structure our survey of related work according to the primary goal of the proposed techniques.

Automatic **segmentation** of motions into distinctive smaller fragments has been investigated in support of a number of applications, including motion compression [LM06], motion classification [BSP*04, SB05, FMJ02], and motion retrieval [LZWM05, SB05]. Approaches used for segmentation include angular-velocity zero crossing detection [FMJ02], entropy metrics [SB05], and temporally-local models based on probabilistic principal component analysis (PCA) [BSP*04, LM06].

Motion compression techniques extract spatial or temporal structure from motions in order to achieve compact representations. Arikan [Ari06] introduces a compression scheme that breaks the motion into fixed-duration segments and then builds a compressed representation for these. In [LZWM05, LM06], poses are assigned to a finite set of linear subspaces. The linear subspaces are allocated using a recursive division approach [LZWM05] or the subspaces formed by temporally contiguous frames of the motion. The use of the subspaces to support blending or resequencing is not explored.

Motion query techniques extract structure in order to be able to efficiently identify motions that are similar to a query motion. This has recently been an active area of research. One common approach is to develop and apply a segmentation rule and then cluster the resulting fragments [BSC05, LZWM05]. Motion queries are then performed by looking for a given cluster transition signature. An alternate model is to look for patterns in binary-valued relational features [MRC05, MR06] or extracted keyframes [SB05], and to construct efficient searches based on these. Search strategies can also be informed by user-weighted notions of important features [FF05] and can be made to efficiently support time warps [KPZ*04, SB06]. Another approach builds a precomputed 'match web' from a pairwise comparison of all frames in the motion corpus, which can then be used to quickly retrieve motions that are similar to query motions also selected from the corpus [KG04].

Statistical models of motion are intended to be general and can in theory be used for both analysis and synthesis. In parametric statistical models, the original motion data is discarded and thus original motion retrieval is not an option. Hidden Markov models or variations thereof were first suggested in [BH00, TH00]. Switched linear dynamical systems are proposed in [PRM00, LWS02]. Several stochastic models are applied towards the construction of natural-motion classification oracles in [RPE*05].

Motion graphs aim to address the motion resequencing problem by automatically identifying points where motions are sufficiently similar that they support transitions between motions, and hence allow resequencing. They have been introduced in various forms in recent years [TH00, AF02, KGP02, LCR*02, LWS02] and resemble the *move trees* [Men00] that have long been a staple for game-based character motion.

***Fat* or *parametric* motion graphs** [SO06, HG07] support both motion resequencing and motion blending. This is achieved by building a motion graph from sets of parameterized motions, where motions within a set can be blended. Sets of parameterized motions can be constructed manually [PSKS04] or with the help of a motion query algorithm that can retrieve similar motions given an example query motion [KG04, KS05]. The parametric motions in [HG07] are constructed using the techniques developed in [KG04] to efficiently find and resample sets of motions that are similar to a given query motion. A user-specified distance threshold value determines how large the returned set of similar motions will be. Where the similar motions are considered to start and end is implicit in the start and end of the query motion chosen by the user. As a result, the nature and duration of each parameterized motion set is in effect user-specified. The parametric motions in [KS05] are identified with the help of string matching, where the alphabet denotes the various support phases (right leg, left leg, double support, flight) for walking and running.

Our work is closest in spirit to that of the fat and parametric motion graphs described above. However, our goals are complementary. Whereas the focus in existing work has been on user-driven graph construction, the details of establishing a specific parameterization, and the specific details of graph traversal, our focus is on the *automated construction* of such motion graphs. We aim to find natural partitions of the motion data into self-similar motion sequences which we call *motion bundles*. The size and duration of motion bundles are determined in an unsupervised fashion from

the data, with the help of a single parameter that embodies user preferences for their shape (duration vs size).

Another unique aspect of our work is the use of strings to perform motion comparisons and motion bundle extraction. A common approach to comparing two motion sequences involves computing pose distances between all pairs of frames using a distance matrix and placing thresholds on the maximum allowable pose distances. We replace the role of such distance matrices with a simple lookup scheme. Poses are classified as belonging to a particular precomputed pose cluster. Distances between pose cluster centers are then precomputed and a binary matrix is used to record all cluster pairs that are closer than a given threshold distance to each other. This greatly reduces the cost of pairwise frame-to-frame comparisons. The string representation further supports the efficient search for large motion bundles.

We note that the work of [KS05] models motions as strings, but in a very limited fashion. The motion graph approach of [LCR*02] makes use of pose clusters much like ours, which can be labelled to achieve a string-based representation like ours (see Figure 4 in [LCR*02]). However, their use of cluster path signatures is to build *cluster trees* which are used for a different purpose than our motion strings. We are unaware of other work that uses string-based motion representations to perform tasks such as efficient sequence clustering applied to motion capture data.

3. Motion-Bundle Algorithm

3.1. Overview

We begin our discussion by describing the application of the motion-bundle extraction algorithm to a toy example. Figure 1(a–g) shows how motion bundles are extracted for a set of motions in a simple two dimensional space. The frames for this toy example are marked in red in (a). The motion sequences from which they come are the curves traced in (f) and (g). The goal of the algorithm is to produce motion-bundle partitions. As shown in (f) and (g), there exists more than one way to naturally partition the motions into motion bundles. A parameter ρ controls the preferred type of partitioning; $\rho > 1$ gives preference to longer motion bundles, at the possible expense of having fewer motions in any given bundle.

The first step in the process is to reduce the dimensionality of all poses through the application of PCA This is followed by pose clustering, as shown in Figure 1(a), for all poses. Clusters are modeled as isotropic Gaussians, *i.e.*, using a mean value and a single variance parameter. Individual poses are assigned labels, shown as letters, according to their most-likely cluster. The motions can now be converted into motion strings using the letters associated with each pose. Sequential repetitions of letters are removed. All the motion clips in the motion corpus can be represented as strings and then further concatenated into a single long string representing all motions. Figure 1(b) shows a portion of this long string for the toy example. The collection of motions that runs horizontally is represented by the substring GOACEN, and thus multiple occurrences of this substring can be seen. An additional *partitioned flag*, $p(j)$, is used to flag points where adjacent motion clips have been concatenated. Later, $p(j)$ will also be used to flag subsequences that have already been incorporated into motion bundles and that are therefore ineligible for use in new motion bundles.

The construction of motion bundles requires knowing which letters are 'nearby' other letters. This can be captured using a binary *accessibility matrix* **A**, as illustrated in Figure 1(c). This will be used to help identify non-identical substrings that nevertheless represent similar motions. For example, the motions represented by GOACEN and GOACHCEN are very similar and should be eligible to be clustered together.

The clustering algorithm works by choosing a seed point to build a motion bundle and then constructing the largest possible motion bundle from that seed point. An instance of the most frequent letter (among currently unpartitioned letters in the string) is chosen as the seed point. A particular instance of the letter C is chosen for our example, as shown in Figure 1(d). This seed point will then spawn a number of *seed substrings* in order to search for the largest motion bundle that contains the given seed point. The complete details of this process will be described shortly. Once the largest motion bundle is found, all the substrings that it encompasses are marked as 'partitioned', as shown in Figure 1(e), and the process is then repeated with a new seed point.

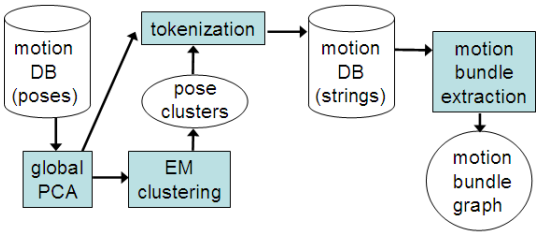

Figure 2: *System Overview.*

Figure 2 gives a block-diagram overview of the the system.

3.2. Pose Preprocessing

In this section, we give a more detailed description of the pose preprocessing and introduce the notation that is used when describing subsequent steps of the algorithm. Each pose in the database is expressed using a vector containing the joint angles, the root height, the root pitch and roll angles, the root horizontal speed and the root angular speed

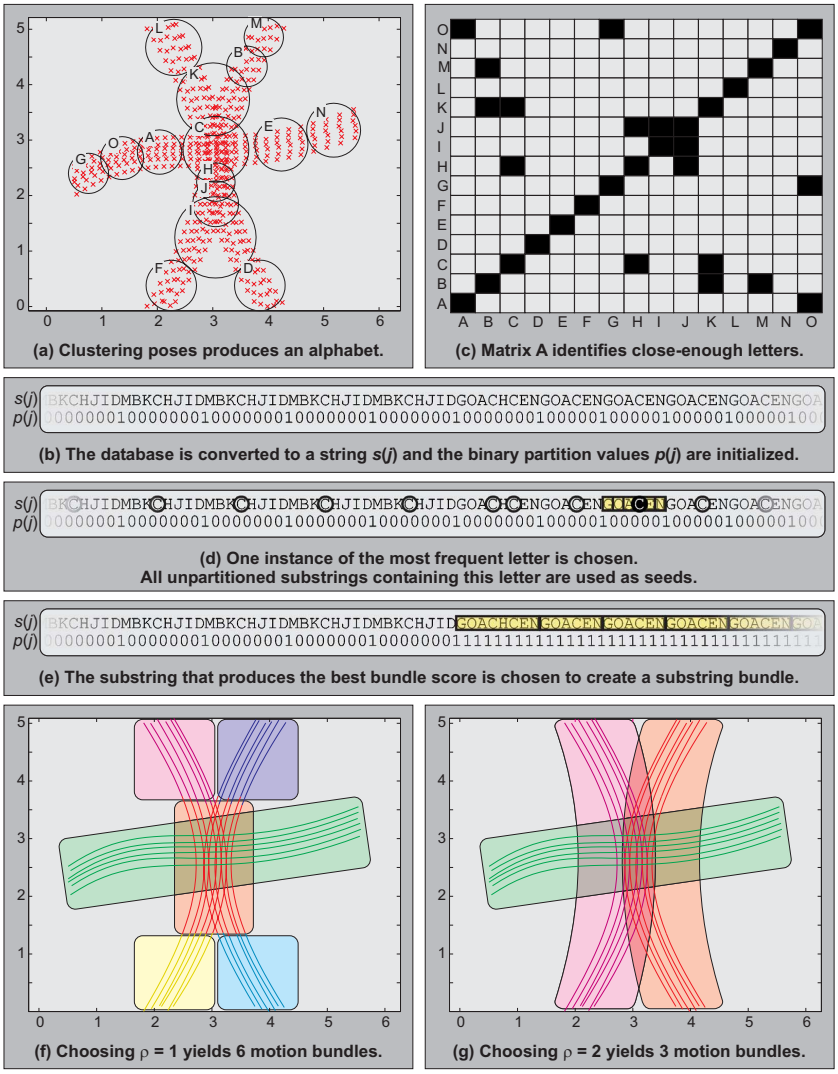

Figure 1: *The various steps of the motion-bundle algorithm on a 2D toy example.*

with respect to the vertical Y axis. This makes the pose representation independent of the facing direction and the location on the plane, as is commonly desired. The database can be expressed as an ordered list of n vectors $\Theta(i)$, $1 \leq i \leq n$. A submotion $\Theta(i), \Theta(i+1), ..., \Theta(i')$ is denoted by a pair of indices (i, i').

All test sequences we use to build the databases shared the same bone hierarchy, leading to a 62-dimensional pose space. We perform a global PCA where all the $\Theta(i)$ are projected into a subspace of fewer dimensions to yield $\hat{\Theta}(i)$. We perform projections that keep 70% of the total variance, corresponding to keeping 7–12 principal components, depending on the dataset. Global PCA helps speed up the pose clustering process.

Pose clusters are created using a mixture of multidimensional isotropic Gaussian distributions. The parameters of this model are estimated using a standard Expectation Maximization (EM) algorithm. The total number κ of clusters needs to be specified by the user. We experimented with a number of other cluster models and parameter estimation techniques. The best results were obtained with models that could accommodate many clusters, each having only a few parameters.

Each cluster can be assigned a letter from a κ-letter alphabet. All motions can therefore be represented as a n-letter string by associating each pose $\hat{\Theta}(i)$ with the cluster that maximizes its likelihood. This string can be further simplified by removing consecutive repetitions of the same letter,

leading to a m-letter string $s(j)$ with $1 \leq j \leq m$. Moreover, for each letter $s(j)$ we can store the set of consecutive frames $\mathcal{F}(j)$ that map to this letter. The average of this set is noted $\bar{\mathcal{F}}(j)$. A substring $s(j), s(j+1), ..., s(j')$ is denoted by a pair (j, j').

Cluster-to-cluster distances are important for determining when two letters are close enough to be considered equal. Given μ_a and $\sigma_a \mathbf{I}$ as the mean and the covariance matrix of the Gaussian corresponding to the cluster labeled a, we define the distance between two clusters as:

$$d(a,b) = \left| \frac{\mu_b - \mu_a}{\min(\sigma_a, \sigma_b)} \right|. \quad (1)$$

This represents the maximum of the mutual Mahalanobis distances of the cluster centers. We further precompute a binary *accessibility matrix* $\mathbf{A}$ that has entries of 1 for "close enough" clusters, defined as

$$\mathbf{A}_{[a,b]} = \begin{cases} 1 & \text{if } d(a,b) \leq \tau \\ 0 & \text{otherwise} \end{cases} \quad (2)$$

where τ is the user-defined maximum distance between two clusters if they are to be considered equal.

3.3. Motion-Bundle Creation

The motion-bundle creation process works by first building bundles from the string-based representation of motions, followed by a second step that maps the result back to the original motions. We postpone the discussion of this second step until Section 3.5. Substring-bundle creation works by iteratively partitioning the motion string. To track the algorithm as it progresses, a binary partition value is associated with each transition between two consecutive letters of the string. This value is noted $p(j)$ with $1 \leq j \leq m-1$ so that $p(j)$ corresponds to the interval between $s(j)$ and $s(j+1)$. By default, all values of $p(j)$ are initialized to 0, indicating that the whole string is unpartitioned. It is possible to initialize some values of $p(j)$ to 1 to indicate that two consecutive letters belong to different motion clips. This way, the algorithm will never include that interval into a substring bundle.

The values of $p(j)$ separate the string into partitioned letters and unpartitioned letters. We say that letter $s(j)$ is *partitioned* if and only if $p(j-1) = p(j) = 1$, and that it is otherwise *unpartitioned*. In the same way, substring (j, j') is partitioned if and only if $p(j) = p(j+1) = ... = p(j'-1) = 1$, and unpartitioned if and only if $p(j) = p(j+1) = ... = p(j'-1) = 0$. It is possible for a substring to be neither partitioned nor unpartitioned.

The creation of every motion bundle begins with the identification of a seed point. This is taken as the most frequently occurring unpartitioned letter. Intuitively, this corresponds to a pose cluster that still has many motions passing through it, but that does not yet belong to a motion bundle. One unpartitioned instance of this letter (say at index c of the string) is then randomly selected; $s(c)$ therefore equals this most frequent letter. This index is the starting point for the creation of the current substring bundle.

All unpartitioned substrings containing the letter at index c, up to a maximal user-defined substring length λ, are then used as *seed substrings*. For each of these seed substrings, we look through the full motion string for all similar and unpartitioned substrings. Identifying good motion bundles requires a definition of 'similar' that is broader than simply identifying copies of the substring in question. Details of this process are given in Section 3.4. The end product of the overall search is a distinct set of matching substrings for each possible choice of seed substring.

Each of these sets of substrings can be regarded as defining a different potential motion bundle, and we thus need to choose which one will give the largest motion bundle. We define a *bundle volume* V related to the bundle dimensions as follows:

$$V = (\mathcal{B}_h - 1)(\mathcal{B}_w)^{\rho} \quad (3)$$

where $\mathcal{B}_h$ is the bundle height (the number of substrings in the bundle) and $\mathcal{B}_w$ is the bundle width (the average number of frames in the motions corresponding to the included substrings, see Section 3.4). ρ is a bias parameter that lets the algorithm favor bundles with more or less frames; its effect is discussed in Section 3.6.

The substring bundle having the largest volume V is identified and kept. If all potential bundles contain a single motion, then the one with the fewest frames is kept. Following that, the values of $p(j)$ are updated so that all substrings in the chosen bundle are now marked as being partitioned. The algorithm then repeats using a newly chosen seed point. Motion bundles that contain a single submotion are said to be degenerate. As a final clean-up step, it is possible to merge together degenerate motion bundles that are temporally adjacent in the motion database.

3.4. Identifying Similar Substrings

The process of identifying substrings similar to a seed substring is inspired by the search algorithm proposed by [KG04], although we compare pose clusters rather than poses. For a given subsequence, we need to identify all other subsequences with which we can build a cluster-to-cluster registration curve that satisfies some constraints. We begin by defining the binary relation $M(ss_1, ss_2)$ that is true if and only if substring ss_1 matches substring ss_2. We will construct this relation so that it is reflexive and symmetric.

Suppose we have two substrings ss_1 and ss_2 identified respectively by (j_1, j_1') and (j_2, j_2'). We can build a substring-to-substring binary accessibility matrix **SSA** as

$$\mathbf{SSA}_{[a-j_1, b-j_2]} = \mathbf{A}_{[s(a), s(b)]} \quad (4)$$

with $j_1 \leq a \leq j_1'$ and $j_2 \leq b \leq j_2'$. For $M(ss_1, ss_2)$ to

be true there must exist, within **SSA**, a valid path of 1 starting from cell $\mathbf{SSA}_{[0,0]}$ and ending at $\mathbf{SSA}_{[j'_1-j_1,j'_2-j_2]}$. Suppose we have such a candidate path going through cells $[a_1,b_1],[a_2,b_2],...,[a_l,b_l]$, with $a_1=b_1=0, a_l=j'_1-j_1, b_l=j'_2-j_2$, and where $\mathbf{SSA}_{[a_i,b_i]}=1$ for all i. This path is valid if and only if $a_i \leq a_{i+1} \leq a_i+1$ and $b_i \leq b_{i+1} \leq b_i+1$ for $0 \leq i < l$. The reflexivity of $M(\cdot,\cdot)$ comes from the fact that all the elements on the diagonal of **A** are 1. The symmetry of the relation is straightforward from the symmetry of the definition.

Provided we have a seed substring ss_0, we can build the set of all substrings for which $M(ss_0,ss_a)$ holds true. This is not an equivalence set, however, since the transitivity of the relation is not guaranteed. Making this an equivalence set would be an advantage in the context of our greedy algorithm since the same substring bundle would be constructed regardless of the substring used as a seed.

To turn this into an equivalence set, we simply apply the transitive closure of the $M(\cdot,\cdot)$ relation. The set $\mathcal{M}$ of the substrings similar to the seed substring ss_0 is therefore defined as

$$\mathcal{M} = \left\{ss_0, ss_i | ss_j \in \mathcal{M} \text{ and } M(ss_i,ss_j) \Rightarrow ss_i \in \mathcal{M}\right\}. \quad (5)$$

In practice, it can sometime happen that some substrings in $\mathcal{M}$ correspond to motions whose length differ significantly from others in the set. In this case, we relax the criterion that $\mathcal{M}$ be an equivalence set and remove the problematic substrings. The length of the motion corresponding to substring (j,j') can be estimated as $\bar{\mathcal{F}}(j')-\bar{\mathcal{F}}(j)$. A motion is deemed too different if the ratio of its length to the average motion length is less than a user-defined factor γ or more than $1/\gamma$.

3.5. Fine-grained Substring to Submotion Mapping

Motion bundles, represented in terms of substrings, need to be mapped back to particular frame sequences of the original motions. While this step is largely trivial, there remains a small-but-non-negligible issue. Multiple successive frames of a motion sequence will often map to the same letter in a substring. Arbitrarily choosing one of these frames for the first or last letter of a substring may lead to motions that are not as well aligned as they could be. It is thus advantageous to further optimize, at the frame level, where a given motion enters and exits a motion bundle.

To do so, we first identify the indices of all letters of the database appearing at the beginning or the end of a substring. The set containing these indices is noted $\mathcal{J}$. Mapping back from substring to submotion is then simply a matter of finding a frame $f(j)$ for each $j \in \mathcal{J}$.

Given such a mapping, we can evaluate the quality of the motion alignment within a bundle. To do so we define the alignment score for motion bundle $\mathcal{B}$ as

$$\left|\text{cov}\left\{\Theta(i)\right\}\right| + \left|\text{cov}\left\{\Theta(i')\right\}\right| + \text{var}\left\{i'-i\right\} \quad (6)$$

where i and i' respectively indicate all the starting frames and all the ending frames of motions in $\mathcal{B}$. The first two terms measure the discrepancy of the start and end poses of the bundle, respectively. The last term measures the discrepancy in motion durations. The total alignment score for a given assignment of the $f(j)$ is the sum of the alignment scores over all the motion bundles.

We look for an assignment that minimizes the total alignment score. To do so, we rely on the following stochastic search algorithm. First, we use the initial guess $f(j)=\bar{\mathcal{F}}(j)$ for all $j \in \mathcal{J}$. Then an index $j' \in \mathcal{J}$ is randomly selected and a direct search is performed within $\mathcal{F}(j')$ to find the assignment $f(j')$ that minimizes the total alignment score. The algorithm is repeated with a newly selected random index until the total alignment score cannot be improved or for a predefined number of iterations. There is no guarantee that this algorithm converges, although we have always obtained good results in practice.

3.6. Parameter Selection

In this section, we cover the various user-defined parameters that drive the above algorithm. The user needs to define κ, the number of pose clusters. This number determines the precision of the algorithm. Too few clusters could mean that distant poses will be considered similar. On the other hand, too many clusters can slow down the algorithm by increasing the size and density of the accessibility matrix **A**. Fortunately, the algorithm is not overly sensitive to parameter κ. For example, in the case of the boxing sequence, we obtained very good results with values of κ between 100 and 250. For values of κ under 100, most of the extracted motion bundles were still meaningful although the algorithm started to bundle together different motions like right jabs and left jabs. An alternative would be to use the Bayesian Information Criterion of a mixture model.

Parameter τ dictates the maximum distance between two clusters that are to be considered equal by the matching algorithm. It is hard to find a good fixed value for this parameter. We therefore automatically estimate it by analyzing the distance between adjacent letters in the database string. All the results presented in this paper were obtained using $\tau = \text{median}\{d(s(j),s(j-1))|2 \leq j \leq m\}$.

The maximal seed substring length, λ, is not necessarily intuitive and it is simpler to specify λ', the maximal submotion length in frames. To convert from λ' to λ we can use the average size of the clusters by taking

$$\lambda = \lambda' \frac{m-1}{\sum_{j=2}^{m} \bar{\mathcal{F}}(j) - \bar{\mathcal{F}}(j-1)}. \quad (7)$$

In practice, we used $\lambda' = 200$ frames.

Parameter ρ lets the user drive the relative importance of bundle width versus bundle height. This parameter mostly

affects the nature of the obtained motion bundles. Example of the impact of these parameters are given in Section 5. We usually choose $\rho = 1$ to favor a balance between bundle width and height.

Parameter γ dictates the minimum acceptable ratio between a submotion length and the average length of the motions within a bundle. In practice, this parameter only marginally affects quality and reasonable results can be obtained even with $\gamma = 0$, *i.e.*, no motions are ever rejected. We noticed that a choice of $\gamma = 0.8$ usually yields a moderate improvement in the quality of the results.

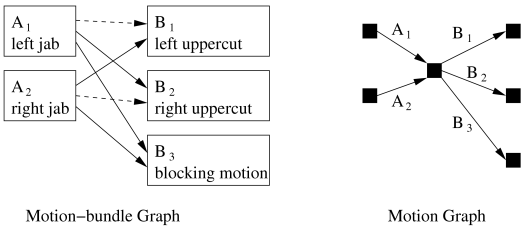

Figure 3: *Graphs using nodes-as-motions (left) vs nodes-as-transitions (right).*

4. Motion-Bundle Graphs

Motion bundles can be seen as a way of folding the motion database so that different submotions overlap. This naturally creates a directed graph structure where each node is a motion bundle and each edge is a valid transition from one motion bundle to the another one. For convenience, we call this a motion-bundle graph.

Parameterized motion graphs [HG07] use a similar graph structure, namely having parameterized motions as nodes and transitions as edges. As noted in [HG07], this structure has advantages over the motions-as-edges, nodes-as-transitions structure used in traditional motion graphs [KGP02] and fat graphs [SO06]. Specifically, in a nodes-as-transitions motion graph, any motion coming into a node must be able to transition into any motion leaving from that node. This makes it difficult to create good hub nodes with rich connections and also necessitates short-duration blends from incoming onto outgoing motions. In contrast, the nodes-as-motions model supports both long blend intervals during transitions, as well as rich parameterizations of motions using multi-way blends [HG07].

The example shown in Figure 3 illustrates the structural benefits of the nodes-as-motions structure, shown on the left. Solid edges indicate feasible transitions between motions. In the nodes-as-transitions representation this same set of constraints cannot be directly matched. In Figure 3(right), motions A_1 and A_2 are represented as edges and both can transition to B_3. This implies that they reach a unique transition pose. Since we also have transitions $A_1 \rightarrow B_2$ and $A_2 \rightarrow B_1$, we are forced to add the new transitions $A_1 \rightarrow B_1$ and $A_2 \rightarrow B_2$, as illustrated by the dashed edges in Figure 3(left), even though such transitions may not be desirable.

Building a motion-bundle graph from the motion-bundle structure is straightforward. First, a node B is created for each bundle $\mathcal{B}$. Then, an edge $B_1 \rightarrow B_2$ is added if there exist poses i_1, i_2, i_3 such that the corresponding sequences $(i_1, i_2) \in \mathcal{B}_1$ and $(i_2, i_3) \in \mathcal{B}_2$.

It is possible to produce an animation that follows any valid path through this graph. To do so, we exploit the similarity of motions within a bundle and smoothly blend from the entry motion to the exit motion. More precisely, assume a desired path $B_1 \rightarrow B_2 \rightarrow B_3$. Then $B_1 \rightarrow B_2$ guarantees that we can find i_1, i_2, i_3 such that $(i_1, i_2) \in \mathcal{B}_1$ and $(i_2, i_3) \in \mathcal{B}_2$. Similarly, $B_2 \rightarrow B_3$ guarantees that we can find i'_2, i'_3, i'_4 such that $(i'_2, i'_3) \in \mathcal{B}_2$ and $(i'_3, i'_4) \in \mathcal{B}_3$. To play the motion corresponding to B_2, we simply blend smoothly from submotion (i_2, i_3) to submotion (i'_2, i'_3).

Even though the blended animations are similar and well-aligned at the beginning and at the end, it is possible that their intermediate poses are not well aligned. Moreover it is possible that their contact constraints differ. Various strategies could be used to solve these problems, including time warping, registration curves, and inverse-kinematics-based corrections. Because these techniques are already well studied, we do not cover them in this paper. The animations presented in the accompanying video do not use any of these strategies, and display the raw output of the motion bundles. While this gives a clear picture of the quality that can be achieved without extra cleanup, moderate foot skate and other minor motion artifacts can sometimes be observed.

The basic motion-bundle graph has one node for every motion bundle. However, degenerate bundles do not add anything to the graph given that they have exactly one incoming and one outgoing edge. In our graphical representation, we therefore replace any degenerate motion bundle with a single edge representing a transitional animation. We merge together multiple edges that would be created between two motion bundles. Nodes are represented has boxes with an height proportional to $\sqrt{\mathcal{B}_h}$ and a width proportional to $\mathcal{B}_w$.

5. Results

5.1. Extracting Motion Bundles

We tested the motion-bundle algorithm on various collections of motions, ranging in size up to a 7300 frame sequence sampled at 120 Hz for about 1 minute of motion. The entire set of test sequences amounts to approximately 10 minutes of motion. Some of the collections we used for testing contain a single activity: an actor shadow boxing, performing various exercises, or walking around in an erratic fashion. Other collections contained clips representing

different kinds of motions. All the animations were taken directly from the CMU Motion Capture Database [Uni] and use a sample rate of 120 Hz. Some motions contain small pose errors or pose noise that is typical of uncleaned motion-capture data. All tests were executed on an Athlon64 3500+ processor with 2 GB of memory.

The most costly step of the motion-bundle extraction algorithm is pose clustering. This process takes 460 seconds for the 7300 frame database when using 250 clusters. The motion-bundle extraction then took 76 seconds for this example. Motion-bundle extraction for a 1900 frame walking dataset took 4 seconds. The final motion-bundle structures need very little space, requiring only pointers to the starting and ending frames for each motion subsequence in a motion bundle. The total size of the motion bundles for all the databases use about 100 KB of disk space in an uncompressed text format.

The content of a motion bundle is difficult to illustrate in printed form, and so we encourage the reader to view the video associated with this paper. However, Figure 4 visualizes two of the motion bundles extracted for a boxing sequence by showing the middle frame for all the motions contained in the bundles. The original 5400 frame boxing sequence contains an assortment of punch, dodge, and stepping motions. The results were obtained with $\rho = 1$. The first bundle contains 20 right punches (36 frames) and the second contains 7 left punches (32 frames). For each bundle we give the average number of frames per motion. During the same run, the algorithm also produced non-degenerate bundles of various sizes for stepping and idle motions.

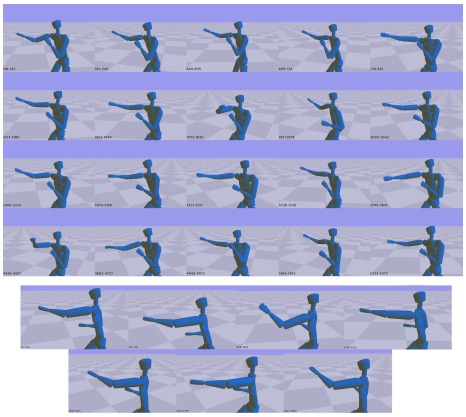

Figure 4: *Content of two motion bundles. The middle frame from 20 right punch motions (top) and 7 left punch motions (bottom).*

Manual annotation of the original boxing motion capture sequence results in the identification of 5 right uppercuts, 22 right punches and 11 left punches. The style of the right and left punches vary, although most of them are quick jabs of various strength. Since there are less repetitions of the other styles, they are less likely to be favored by the greedy step of the algorithm. For example, the algorithm will occasionally segment out 2 "hammer-style" right punches in a distinct motion bundle.

The chosen value of ρ affects the shape of the extracted bundles. For example, $\rho = 2$ favors longer bundles. In the boxing sequence, this results in motion bundles containing punch combinations and longer stylized punches. One bundle contains 6 left-right jab combinations (69 frames), while others contain 3 right uppercuts (89 frames), 2 "hammer-style" right punches (51 frames), 4 strong right jabs (42 frames), and 5 weak right jabs and 1 weak left jab which constitutes a spurious match (23 frames).

In order to test the sensitivity of the extracted bundles with respect to noise, we ran the algorithm with $\rho = 1$ on a noisy version of the boxing animation. We added Gaussian noise of amplitude 0.1σ to each degree of freedom. The extracted motion-bundle structure is similar to the one obtained without noise, although some differences appear. For example, the right-punch motion bundle now contains 19 right punches (41 frames average length) and the left-punch bundle containts 7 motions (30 frames).

The content of all these motion bundles can be seen in the accompanying video.

5.2. Motion-Bundle Graphs

Producing the motion-bundle graphs from the motion-bundle information takes less than 15 milliseconds for all our examples. These graphs embed all the motions from the source motion capture dataset. Much of this motion is included as motion-bundle nodes, while the remainder is contained in the edges.

An important question to ask is "does the algorithm extract a reasonable graph structure from the unstructured datasets?". We present a number of results to help answer this question. Figure 5 shows the motion-bundle graph for the walking dataset (1900 frames at 120 Hz), which was computed using 90 pose clusters. We manually layout and annotate the resulting graph. Five principal motion bundles are found, each of which contains highly similar motions.

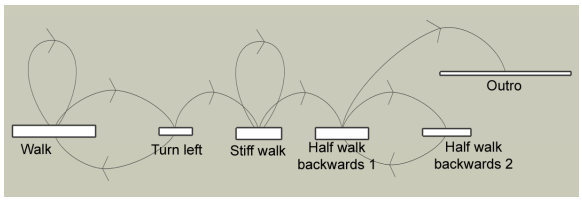

Figure 5: *Motion-bundle graph extracted from the walking sequence.*

Figure 6 shows the resulting graph for the boxing dataset

(5400 frames) using $\rho = 1$. The largest motion bundles correspond to right punches (20 motions, 36 frames), idle behavior (8 motions, 40 frames), left punches (7 motions, 32 frames), and small steps (5 motions, 34 frames). The graph exhibits good connectivity. The effect of favoring long motions ($\rho = 2$) is shown in Figure 7. This produces more discrimination between the types of right punches, and also results in the creation of a right-left combo punch. The graph also readily illustrates gaps in the connectivity. For example, there are no motions to support going directly from a right-left combo to a right uppercut.

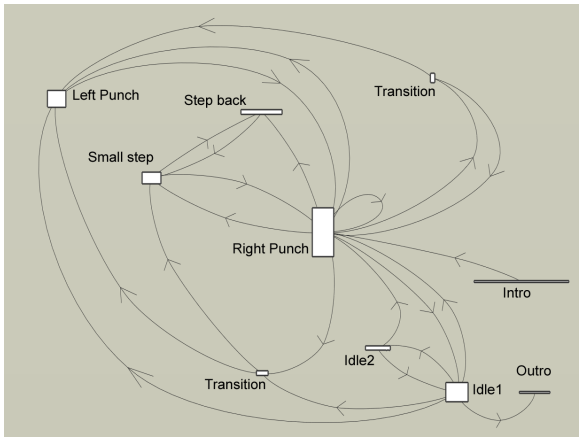

Figure 6: *Motion-bundle graph extracted from the boxing sequence ($\rho = 1$).*

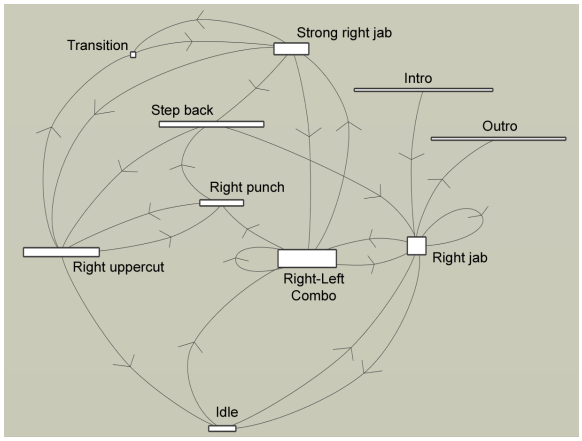

Figure 7: *Motion-bundle graph for the boxing sequences while favoring longer sequences ($\rho = 2$).*

Another question we wish to answer is "does the algorithm extract the same structure for a dataset, independent of another dataset that we might mix in with it?". To test this, we combine the walking and boxing datasets into a single dataset and observe the resulting extracted structure. As can be seen from Figure 8, the result consists of two disjoint graphs, each of which having structure similar, but not identical, to what was obtained when the datasets were considered independantly. 250 pose clusters were used for the combined dataset, which is the same that was used earlier for the boxing dataset alone.

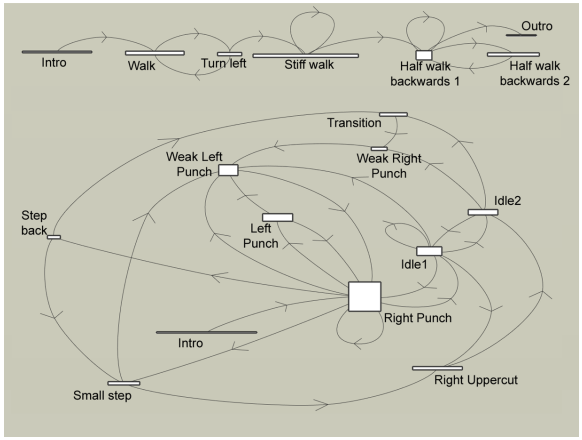

Figure 8: *Motion-bundle graph extracted from the combined walking and boxing sequences.*

We tested the effect of noise on the extracted structure. Figure 9 shows the result of the boxing dataset with noise applied to it. The structure remains similar, although does exhibit differences.

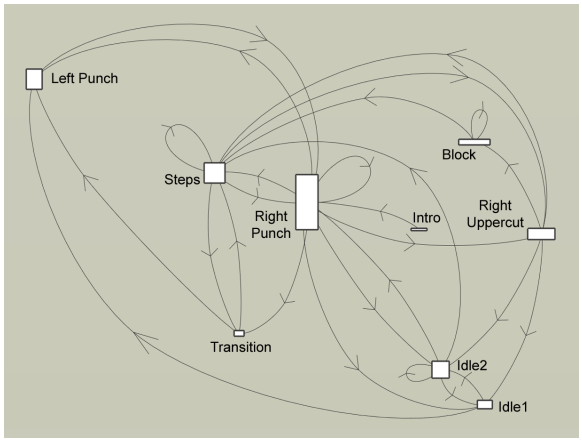

Figure 9: *Motion-bundle graph for the boxing sequences with added noise.*

Lastly, we tested the extracted graphs for resequencing by observing the output for random walks through the graphs. The results are shown in the accompanying video, including a walk through a graph extracted from an exercising animation. Some minor motion artifacts remain because we do not correct for effects such as foot skate during blending.

Published May 24th 2007

6. Conclusion

We have presented a technique that can distill a highly structured representation from unstructured motion data. The resulting motion-bundle graph provides an easily interpretable visualization of the motion data, its repetitive variations, and its connectivity. The user can specify a preference for the desired shape of the extracted motion bundles. The processed result is multi-purpose, potentially usable for motion resequencing, the creation of blend spaces, motion compression, motion segmentation, and motion annotation.

A key insight of the method is to tackle segmentation and clustering in a *coupled* fashion. The string-based representations of motions we use are highly compact while being sufficient for extracting the inherent structure of motion sequences. Motion bundles can be extracted very efficiently using the string-based representation.

Our work has a number of limitations. We do not currently compute an optimal registration of motions to each other within the motion bundles. There is no absolute guarantee that motion bundles will always be semantically meaningful, nor that they can make the fine distinctions that might be needed to discriminate between two very similar classes of motion. We do not yet build user-designed reparameterizations of motions in a motion bundles, along the lines proposed by [SO06, HG07] for building parameterized edges.

A significant number of directions exist for future work. We wish to investigate the compression of motion-bundle graphs. We want to explore creating separable motion-bundles that can simultaeously capture both global redudancies as well as localized redundancies, such as an arm wave that can be composed with many different full body motions. With some optimizations, an online version of the algorithm could be used to support interactive graph construction, assisting a director in creating a targeted corpus of motion capture data for a given application. Currently, the scalability is limited by the initial clustering phase. We wish to apply fast N-body techniques to this in order to further test scalability.

References

[AF02] ARIKAN O., FORSYTH D.: Interactive motion generation from examples. *ACM Trans. Graphics 21*, 3 (2002), 483–490.

[Ari06] ARIKAN O.: Compression of motion capture databases. *ACM Trans. Graphics 25*, 3 (2006), 890–897.

[BH00] BRAND M., HERTZMANN A.: Style machines. In *Proc. SIGGRAPH '00* (2000), pp. 183–192.

[BSC05] BASU S., SHANBHAG S., CHANDRAN S.: Search and transitioning for motion captured sequences. In *VRST '05: Proc. Symp. Virtual Reality Software and Technology* (2005), pp. 220–223.

[BSP*04] BARBIČ J., SAFONOVA A., PAN J.-Y., FALOUTSOS C., HODGINS J., POLLARD N.: Segmenting Motion Capture Data into Distinct Behaviors. In *Proc. Graphics Interface* (July 2004), pp. 185–194.

[FF05] FORBES K., FIUME E.: An efficient search algorithm for motion data using weighted pca. In *SCA '05: Proc. Symp. Computer Animation* (2005), pp. 67–76.

[FMJ02] FOD A., MATARIC M., JENKINS O.: Automated derivation of primitives for movement classification. *Autonomous Robot 12*, 1 (2002), 39–54.

[HG07] HECK R., GLEICHER M.: Parametric motion graphs. In *Proc. Symp. Interactive 3D Graphics and Games* (2007).

[KG04] KOVAR L., GLEICHER M.: Automated extraction and parameterization of motions in large data sets. *ACM Trans. Graphics 23*, 3 (2004), 559–568.

[KGP02] KOVAR L., GLEICHER M., PIGHIN F.: Motion graphs. *ACM Trans. Graphics 21*, 3 (2002), 473–482.

[KPZ*04] KEOGH E., PALPANAS T., ZORDAN V., GUNOPULOS D., CARDLE M.: Indexing large human-motion databases. In *Proc. VLDB Conf.* (2004), pp. 780–791.

[KS05] KWON T., SHIN S.: Motion modeling for on-line locomotion synthesis. In *SCA '05: Proc. Symp. Computer Animation* (2005), pp. 29–38.

[LCR*02] LEE J., CHAI J., REITSMA P., HODGINS J., POLLARD N.: Interactive control of avatars animated with human motion data. *ACM Trans. Graphics 21*, 3 (2002), 491–500.

[LM06] LIU G., MCMILLAN L.: Segment-based human motion compression. In *SCA '06: Proc. Symp. on Computer Animation* (2006), pp. 127–135.

[LWS02] LI Y., WANG T., SHUM H.-Y.: Motion texture: a two-level statistical model for character motion synthesis. *ACM Trans. Graphics (Proc. SIGGRAPH) 21*, 3 (2002), 465–472.

[LZWM05] LIU G., ZHANG J., WANG W., MCMILLAN L.: A system for analyzing and indexing human-motion databases. In *SIGMOD '05: Proc. ACM SIGMOD Intl. Conf. on Management of Data* (2005), pp. 924–926.

[Men00] MENACHE A.: *Understanding Motion Capture for Computer Animation and Video Games*. Academic Press, 2000.

[MR06] MÜLLER M., RÖDER T.: Motion templates for automatic classification and retrieval of motion capture data. In *SCA '06: Proc. Symp. Computer Animation* (2006), pp. 137–146.

[MRC05] MÜLLER M., RÖDER T., CLAUSEN M.: Efficient content-based retrieval of motion capture data. *ACM Trans. Graphics 24*, 3 (2005), 677–685.

[PRM00] PAVLOVIC V., REHG J., MACCORMICK J.: Learning switching linear models of human motion. In

NIPS '00: Proc. Neural Information Processing Systems (2000), pp. 981–987.

[PSKS04] PARK S., SHIN H., KIM T., SHIN S.: On-line motion blending for real-time locomotion generation. *Comput. Animat. Virtual Worlds 15*, 3-4 (2004), 125–138.

[RPE*05] REN L., PATRICK A., EFROS A., HODGINS J., REHG J.: A data-driven approach to quantifying natural human motion. *ACM Trans. Graphics 24*, 3 (2005), 1090–1097.

[SB05] SO C., BACIU G.: Entropy-based motion extraction for motion capture animation. *Comput. Animat. Virtual Worlds 16*, 3-4 (2005), 225–235.

[SB06] SO C., BACIU G.: Hypercube sweeping algorithm for subsequence motion matching in large motion databases. In *VRCIA '06: Proc. ACM Intl. Conf. Virtual Reality Continuum and its Applications* (2006), pp. 221–228.

[SO06] SHIN H., OH H.: Fat graphs: constructing an interactive character with continuous controls. In *SCA '06: Proc. Symp. Computer Animation* (2006), pp. 291–298.

[TH00] TANCO L., HILTON A.: Realistic synthesis of novel human movements from a database of motion capture examples. In *HUMO '00: Proc. Workshop on Human Motion* (2000), pp. 137–142.

[Uni] UNIVERSITY C. M.: Cmu graphics lab motion capture database.

Published May 24th 2007