An Efficient Algorithm for Keyframe-based Motion Retrieval in the Presence of Temporal Deformations

Andreas Baak, Meinard Müller, Hans-Peter Seidel
Max Planck Institut für Informatik and Saarland University
Campus E1 4, 66123 Saarbrücken, Germany
{abaak,mmueller,hpseidel}@mpi-inf.mpg.de

ABSTRACT

In the last years, various algorithms have been proposed for automatic classification and retrieval of motion capture data. Here, one main difficulty is due to the fact that similar types of motions may exhibit significant spatial as well as temporal variations. To cope with such variations, previous algorithms often rely on warping and alignment techniques that are computationally time and cost intensive. In this paper, we present a novel keyframe-based algorithm that significantly speeds up the retrieval process and drastically reduces memory requirements. In contrast to previous index-based strategies, our recursive algorithm can cope with temporal variations. In particular, the degree of admissible deformation tolerance between the queried keyframes can be controlled by an explicit stiffness parameter. While our algorithm works for general multimedia data, we concentrate on demonstrating the practicability of our concept by means of the motion retrieval scenario. Our experiments show that one can typically cut down the search space from several hours to a couple of minutes of motion capture data within a fraction of a second.

Categories and Subject Descriptors

H.3.3 [**Information Storage and Retrieval**]: Information Search and Retrieval; I.3.7 [**Three-Dimensional Graphics and Realism**]: Animation

General Terms

Algorithms, Experimentation, Performance

1. INTRODUCTION

Modern motion capture or mocap technology is capable of accurately tracking and recording human motions at high spatial and temporal resolutions. The resulting 3D mocap data is used in a variety of applications ranging from motion synthesis in data-driven computer animation to motion analysis in fields such as sports sciences, biomechanics, and computer vision [5, 6, 11]. Even though there is a rapidly growing corpus of freely available mocap data, e.g. [2, 10], there is still a lack of efficient motion retrieval systems that work in a purely content-based fashion without relying on manually generated annotations. Here, the main difficulty is due to the fact that similar types of motions may exhibit significant spatial as well as temporal variations [5, 6]. For example, the two kick sequences shown in Fig. 1 are logically related even though they differ considerably with respect to motion speed as well as the direction, the height, and the style of the kick.

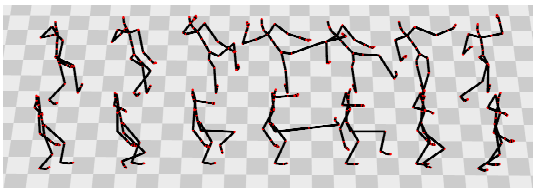

Figure 1: Seven poses of a side kick sequence (top) and a front kick sequence (bottom). Even though the two kicking motions are similar in some logical sense, they exhibit significant spatial and temporal differences.

Most of the previous approaches to motion comparison are based on features that are semantically close to the raw data, using 3D positions, 3D point clouds, joint angle representations, or PCA-reduced versions thereof, see, e. g., [3, 4, 5, 13]. One problem of such features is their sensitivity towards pose deformations which may occur in logically related motions. Furthermore, computational expensive techniques such as dynamic time warping (DTW) are necessary to establish temporal correspondence between related frames [5]. To cope with spatial variations, Müller et al. [9] introduce the concept of *relational features*, which is based on the following observation. Unlike other data types such as 3D shape, image, or video, 3D motion capture data is explicitly based on a kinematic chain that models the human skeleton. This underlying model can be exploited by looking for semantically meaningful boolean relations between specified points of the body. For example, even though there may be large variations between different kicking motions as illustrated by Fig. 1, all such motions share some common characteristics: first the right knee is stretched, then bent, and finally stretched again, while the right foot is raised during this process. Afterwards, the right knee is once again bent and then stretched, while the right foot drops back to

MIR'08, October 30–31, 2008, Vancouver, British Columbia, Canada.

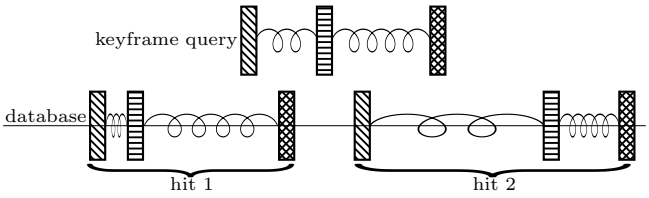

Figure 2: Keyframe-based retrieval in the presence of temporal deformations. A database hit has to contain the queried keyframes in the same order and within specified time bounds controlled by a stiffness parameter. Note that the query and the two hits exhibit different temporal deformations.

the floor. In other words, by simply checking some characteristic poses in the temporal context, one can exclude all motions in the database that do not share the characteristic progression of relations.

As the main contribution of this paper, we introduce a novel keyframe-based search algorithm. A *keyframe query* consists of a sequence of keyframes, where each keyframe is specified by a boolean feature vector that describes characteristic relations of a specific pose. Then, the general strategy is to extract all parts from the unknown motion database that exhibit feature vectors matching the keyframe feature vectors in the correct order within suitable time bounds. One important property of our algorithm is that it allows for explicitly controlling the degree of temporal deformations in the retrieval process. Intuitively spoken, the neighboring query keyframes are connected with elastic springs which can be expanded and compressed by a certain factor specified by what we refer to as *stiffness parameter*, see Fig. 2. Even though our algorithm can handle temporal variations, it works with a standard inverted file index as used in text retrieval [12]. Significantly speeding up and drastically reducing memory requirements, our strategy is ideally suited to cut down the search space in a preprocessing step before applying more refined analysis methods to rank and further process the reduced data set. We will demonstrate such a two-stage retrieval procedure by combining our keyframe-based search with the DTW-based retrieval strategy using motion templates as described in [8]. A similar approach, proposed by Wu et al. [13], proceeds in two stages: First start and end frames of possible candidate clips are identified utilizing a pose-based index and then the actual distance from the query is computed via DTW. However, having no explicit control over temporal deformation of the keyframes, a stong limitation on the preprocessing is imposed.

The remainder of the paper is organized as follows. In Sect. 2, we summarize the concept of relational features as introduced in [9] while fixing the notation. In Sect. 3, as the main contribution of our paper, we describe in detail an efficient keyframe-based search strategy. To prove the practicability of our algorithm, we describe various experiments and show how our algorithm can be applied to speed up previous motion retrieval strategies, see Sect. 4. We conclude in Sect. 5 with prospects of future work. Further references will be given in the respective sections.

2. RELATIONAL MOTION FEATURES

In the following, a motion capture data stream is modeled as a sequence $D = (P_1, P_2, \ldots, P_N)$ of poses $P_n \in \mathcal{P}$

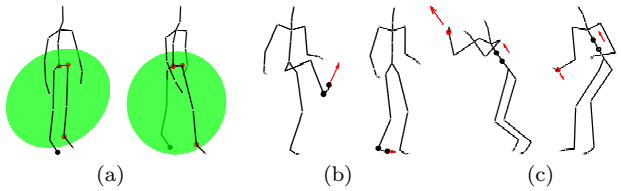

Figure 3: Various boolean relational features that encode spatial, velocity-based, as well as directional information between the joints of a pose.

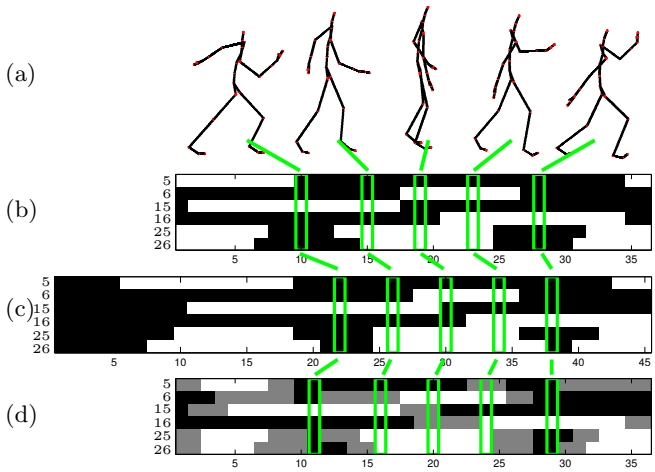

Figure 4: Skiing exercise motion. (a): Poses of a motion at frame positions 10, 15, 19, 23, and 28 (30 Hz). (b): Feature matrix of the skiing motion used in (a). The label numbers of the features correspond to the features used in [8]. Black encodes the feature value one, white encodes the value 0. (c): Feature matrix of another execution of the skiing exercise. (d): Motion template of the skiing motion class trained with 15 training motions from the HDM05 [10] motion database.

for $n \in [1 : N] := \{1, 2, \ldots, N\}$ (w.r.t. a fixed sampling rate), where $\mathcal{P}$ denotes the set of all poses. Here, each pose consists of a full set of 3D coordinates describing the joint positions of a skeletal kinematic chain for a fixed point in time. The idea of *relational features* as introduced by Müller et al. [9] is to describe semantically interpretable, boolean aspects of a pose or a short sequence of poses expressing actions or interactions of certain body parts. Mathematically, a relational feature is a boolean function $F : \mathcal{P} \to \{0, 1\}$ that only assumes the values zero and one. Forming a vector of f boolean features for some $f \geq 1$, one obtains a combined feature $F : \mathcal{P} \to \{0, 1\}^f$ referred to as a *feature function*. A feature function F is applied to a mocap data stream D in a pose-wise fashion: $F(D) := (F(P_1), F(P_2), \ldots, F(P_N))$.

As an example of a relational feature, consider the oriented plane determined by the center of the hip (the root), the left hip joint, and the left foot (Fig. 3 (a)). When the right foot lies in front of that plane, the relational feature F^{15} is defined to assume the value zero, otherwise one. Interchanging corresponding left and right joints in the definition of F^{15} and flipping the orientation of the resulting plane, we obtain another feature function denoted by F^{16}. The combination (F^{15}, F^{16}) of these features is useful to characterize the motion of the lower part of the body, see Fig. 4 (b),

rows 15 and 16. Here, the feature values for one execution of the motion have been visualized. Relational features may also encode velocity-based information. For example, one may check if the absolute velocity of the right foot exceeds a certain velocity threshold, see Fig. 3 (b). The feature F^{25} encodes this relation for the right foot. F^{26} does the same job for the left foot. By checking whether the velocity of the right hand in the direction which is determined by the belly and chest, one obtains a feature that tests whether the right hand is moving upwards or not, see Fig. 3 (c). The values of the features F^5 (right hand to top) and F^6 (left hand to top) can be seen in Fig. 4 (b). For further details including the specification of various generic features and a discussion of threshold selection we refer to [6, 8, 9]. In this paper, we use the feature set described in [8], which comprises $f = 39$ relational features and has been specifically designed to focus on full-body motions. The main point is that even though relational features discard a lot of detail contained in the raw motion data, important information regarding the overall configuration of a pose is retained. Moreover, relational motion features are invariant under global orientation and position, the size of the skeleton, and local spatial deformations of a pose.

Applying a feature function F with f components to a motion data stream D of length N in a pose-wise fashion yields a *feature matrix* $X \in \{0,1\}^{f \times N}$, see Fig. 4 (b) and (c). The n^{th} column of X then contains the feature values of frame n and will be denoted by $X(n) := F(P_n)$, $n \in [1:N]$. Given a set of training motions representing a given motion class, one can learn a *motion template* (MT) that explicitly encodes the consistent and the variable aspects of the motion class, see [8]. Here, a motion template can be thought of as a generalized feature matrix which is obtained by suitably averaging the feature matrices of the training motions, see Fig. 4 (d). Müller et al. [8] suggest MT-based motion retrieval using a variant of dynamic time warping (DTW) to locally compare a motion template with the feature matrices of the unknown motion data. In this paper, as will be explained in the next sections, we speed up MT-based retrieval by prepending a keyframe-based retrieval component to reduce the search space and then apply MT-based retrieval on the reduced data set. Intuitively, the keyframes can be thought of as a small number of characteristic columns of a generalized feature matrix.

3. KEYFRAME-BASED SEARCH

We now describe our keyframe-based search algorithm. To account for temporal deformations within related motion segments, our algorithm allows for controlling the admissible distances between neighboring keyframes by means of an explicit *stiffness parameter*. Despite possible temporal deformations, our algorithm is very efficient and works with a standard inverted file index. In Sect. 3.1 and Sect. 3.2 we describe the index structure and introduce the query, hit, and match concept, respectively. The details of the main algorithm and a discussion of its running time behavior are presented in Sect. 3.3, where we also illustrate the operation mode of our recursive algorithm by means of an explicit example.

3.1 Indexing

Let $\mathcal{D} = (D_1, D_2, \ldots, D_I)$ denote a database consisting of mocap data streams or documents D_i, $i \in [1:I]$. For simplicity, we may assume that the database $\mathcal{D}$ consists of one large document $D = (P_1, \ldots, P_N)$. (This can be achieved by concatenating the documents $D_1, \ldots, D_I$ while keeping track of document boundaries in a supplemental data structure). Note that due to their boolean nature, relational features are ideally suited for indexing. Let F be a fixed feature function having f relational features as its components. Then for each feature vector $v \in \{0,1\}^f$ one stores the *inverted list* $L(v)$ consisting of the indices $n \in [1:N]$ with $v = F(P_n)$. In other words, $L(v)$ shows which of the poses of D exhibit the feature vector v. In a preprocessing step, we construct a query-independent index structure $I_F^{\mathcal{D}}$ consisting of the 2^f inverted lists $L(v)$, $v \in \{0,1\}^f$. Note that one only has to store the non-empty lists. Furthermore, to control the number of index words, one can also split up the feature function into several feature functions and then work with the resulting smaller indices in parallel, see [9]. The elements of the inverted lists are stored in ascending order, accounting for efficient union and intersection operations in the subsequent query stage. To further reduce the size of the index, the elements of each list $L(v)$ are run-length encoded.

3.2 Query, Hit, and Match Concept

As mentioned in the introduction, certain types of motions typically exhibit characteristic relations that already discriminate these motions from most other types of motions. For example, a cartwheel motion can be distinguished from most other motions simply by checking if the body is upside down in the execution of the motion. Or, a skiing exercise motion is characterized by a diametrical backward and forward swinging of arms and legs coupled with a joint air and landing phase of the two feet, see Fig. 4. The idea is to express the characteristic relations of a keyframe pose by a suitable feature vector $v \in \{0,1\}^f$ with respect to a fixed feature function F. Since using all components of F is often too restrictive, we allow an entire set $V \subseteq \{0,1\}^f$ of alternative feature vectors to describe the characteristic relations. In the following, such a set V is simply referred to as *keyframe*.

A *keyframe query* of length K is a tuple $(\mathbf{V}, \mathbf{d})$ consisting of a sequence $\mathbf{V} = (V_1, \ldots, V_K)$ of keyframes $V_k \subseteq \{0,1\}^f$, $k \in [1:K]$, and a sequence $\mathbf{d} = (d_1, \ldots, d_{K-1})$ of keyframe distances $d_k \in \mathbb{N}_0$, $k \in [1:K-1]$. Here, d_k specifies the distance (in frames) of the neighboring keyframes V_k and V_{k+1}. To account for temporal deformations, we introduce a *stiffness parameter* $\boldsymbol{\sigma} = (\sigma_1, \ldots, \sigma_{K-1})$, $\sigma_k \in [0,1]$, which controls the degree of expansion and compression allowed in the matching process. A *hit* in the database document $D = (P_1, \ldots, P_N)$ with respect to the query $(\mathbf{V}, \mathbf{d})$ is a sequence $(n_1, \ldots, n_K)$ of increasing indices $1 \leq n_1 \leq \ldots \leq n_K \leq N$ so that the following two conditions are fulfilled:

$$\forall k \in [1:K]: \qquad F(P_{n_k}) \in V_k \tag{1}$$

$$\forall k \in [1:K-1]: \qquad \sigma_k \cdot d_k \leq n_{k+1} - n_k \leq \frac{1}{\sigma_k} \cdot d_k \tag{2}$$

Here, condition (1) implies the occurrences of the characteristic keyframe poses and condition (2) ensures that the distances of two consecutive keyframes are within the tolerated time bounds specified by $\boldsymbol{\sigma}$ and the query distances. Note that choosing $\sigma_k = 1$ implies that the keyframe distance of keyframes V_k and V_{k+1} within a hit have to coincide with the distances within the query (i. e., there is no deformation tolerance). On the other hand, for $\sigma_k = 0$ (we

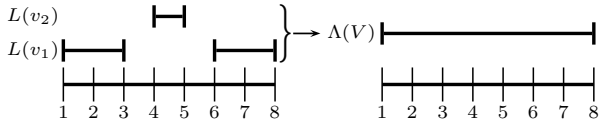

Figure 5: To compute the inverted keyframe list of a keyframe $V = \{v_1, v_2\}$, corresponding inverted lists are combined. The run-lengh-encoding step can reduce the total number of segments.

then set $\frac{1}{\sigma_k} = \infty$) there are no deformation bounds—the keyframes within a hit simply have to appear in the order as specified by the query.

The number of different hits may explode with decreasing stiffness. For example, a small deviation in one of the keyframe positions already defines, in mathematical terms, a different hit. In applications, one is typically not interested in all hits but only in a set of representative hits. We therefore soften the frame-based notion of a hit and assume a segment-focused view. For each query keyframe V_k, $k \in [1:K]$, we define an *inverted keyframe list*

$$\Lambda_k := \Lambda(V_k) := \bigcup_{v \in V_k} L(v). \tag{3}$$

Again we sort the list in ascending order and look for maximal runs of consecutive indices (similar to run-length encoding), see Fig. 5. Each such run is defined by a *segment* $[s:t]$ with integers $s \leq t$, where s denotes the start frame and t denotes the end frame of the segment. Then, one can encode the inverted keyframe list Λ_k by a sequence

$$\Lambda_k = ([s_{k,1}:t_{k,1}], \ldots, [s_{k,\ell_k}:t_{k,\ell_k}]), \tag{4}$$

of segments, where ℓ_k denotes the number of segments. Note that, because of the maximality of the runs, one has $t_{k,i}+1 < s_{k,i+1}$ for $i \in [1:\ell_k - 1]$. Now, a sequence $M = (p_1, \ldots, p_K)$ with $p_k \in [1:\ell_k]$, $k \in [1:K]$, is called a *match* in D with respect to the query $(\mathbf{V}, \mathbf{d})$, if there exists a hit $H = (n_1, \ldots, n_K)$ with

$$\forall k \in [1:K] : s_{k,p_k} \leq n_k \leq t_{k,p_k}. \tag{5}$$

In this case, we also say that the match M *contains* the hit H. In other words, a match specifies a sequence of segments (rather than a sequence of frames) containing at least one hit. In the following, we think of p_k being a pointer to the segment $[s_{k,p_k} : t_{k,p_k}]$, see Fig. 6. The motivation of this notion becomes clear in Sect. 3.3 when describing the main algorithm.

Of course, a match may contain several hits. For a given match M let $R(M) = [s:t]$ be the segment (given by start frame s and end frame t) of minimal length that comprises all hits contained in M. $R(M)$ will also be referred to as *hit relevant range* of M. For example, assume that the match $M = (p_1, p_2) = (1, 3)$ of Fig. 6 contains exactly the three hits $H_1 = (3, 7)$, $H_2 = (4, 7)$, and $H_3 = (4, 8)$, then $R(M) = [s:t] = [3:8]$.

In the case that hit relevant ranges of several matches overlap, we consider, as a further reduction, the union of these ranges instead of the individual ranges. This is motivated by our strategy that in a first stage of a multistage retrieval procedure (as a kind of preselection) it suffices to locate coarse candidate excerpts of the database that contain the keyframe query at least once. As an example, consider the four ranges shown in Fig. 7. The ranges $R(M_1) = [1:7]$, $R(M_2) = [2:11]$, and $R(M_3) = [4:10]$ overlap, whereas $R(M_4) = [13:15]$ is disjoint to the other ranges. The union of the first three ranges defines the segment $[1:11]$. Note that the union does not change, when considering only the first two ranges leaving out the range $R(M_3)$. We then say that M_3 is an *irrelevant match*. Our keyframe-based search algorithm to be presented next, may actually leave out some matches, but for these one can prove that they are irrelevant matches.

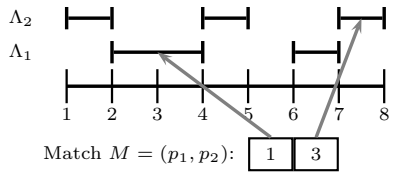

Figure 6: Two inverted keyframe lists Λ_1 and Λ_2. Here, p_1 points to the segment $\Lambda_1(p_1) = [2:4]$ and p_2 to the segment $\Lambda_2(p_2) = [7:8]$.

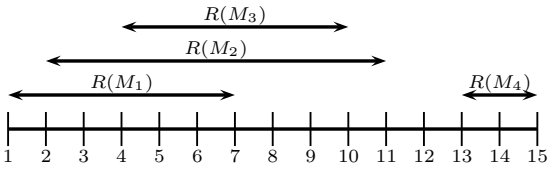

Figure 7: Ranges $R(M_i)$ (indicated by arrows) of four different matches M_i, $i \in [1:4]$. The ranges $R(M_1) = [1:7]$, $R(M_2) = [2:11]$ and $R(M_3) = [4:10]$ overlap, whereas $R(M_4) = [13:15]$ is disjoint to the other ranges.

3.3 Main Algorithm

Before describing our main algorithm, we need some further notation. Generalizing the above notion, a *segment* is an element of the set

$$\mathcal{S} := \{[s:t] : s \in \mathbb{Z} \cup \{-\infty\}, t \in \mathbb{Z} \cup \{\infty\}, s \leq t\} \cup \{\emptyset\}. \tag{6}$$

The intersection of two segments is defined as $\emptyset$ in case one of the segments is empty. Otherwise, for segments $[s_1:t_1] \in \mathcal{S}$ and $[s_2:t_2] \in \mathcal{S}$, we define

$$[s_1:t_1] \cap [s_2:t_2] := \begin{cases} \emptyset, \text{ if } \quad t_1 < s_2 \text{ or } t_2 < s_1 \\ [\max(s_1, s_2):\min(t_1, t_2)], \text{ else.} \end{cases} \tag{7}$$

Our keyframe-based search algorithm consists of a main procedure called KeyframeBasedSearch, and a recursive procedure called RecursiveSearch. The input consists of the inverted file index $I_F^{\mathcal{D}}$, a keyframe query $(\mathbf{V}, \mathbf{d})$ with $\mathbf{V} = (V_1, \ldots, V_K)$ and $\mathbf{d} = (d_1, \ldots d_{K-1})$, as well as a stiffness parameter $\boldsymbol{\sigma} = (\sigma_1 \ldots, \sigma_{K-1})$. Recall from Sect. 3.1 that the index $I_F^{\mathcal{D}}$ does not depend on the query. The algorithm outputs unions of hit relevant ranges, which comprise all matches except for possibly some irrelevant matches. These hit ranges as well as the pointers $(p_1, \ldots, p_K)$ are given by global variables and are consistent in both procedures. In the following, we illustrate the functioning of our algorithm by means of an explicit example with three keyframes $\mathbf{V} = (V_1, V_2, V_3)$ and frame distances $\mathbf{d} = (3, 5)$. As stiffness parameter we use $\boldsymbol{\sigma} = (0.5, 0.6)$, see also Fig. 8 (a).

The procedure KeyframeBasedSearch takes care of the initialization and the first step. In Line 3, the inverted keyframe lists $\Lambda_1, \ldots, \Lambda_K$ are computed, see (3). Recall that

Algorithm Keyframe-based Search

Input: keyframe query $(\mathbf{V}, \mathbf{d})$, comprising K keyframes
stiffness parameter $\boldsymbol{\sigma}$
inverted file index $I_F^{\mathcal{D}}$
Output: hitRanges, the union of all hit relevant ranges
Global: $(p_1, \ldots, p_K)$, hitRanges

1: **procedure** KeyframeBasedSearch($\mathbf{V}$, $\mathbf{d}$, $\boldsymbol{\sigma}$, $I_F^{\mathcal{D}}$)
2: **for** $k \leftarrow 1$ **to** K **do**
3: $\Lambda_k \leftarrow \bigcup_{v \in V_k} L(v)$
4: $p_k \leftarrow 1$
5: **for** $p_1 \leftarrow 1$ **to** ℓ_1 **do**
6: admissibleRange $\leftarrow \mu_1(\Lambda_1(p_1))$
7: RecursiveSearch(2, admissibleRange)

8: **procedure** RecursiveSearch(k, $[s:t]$)
9: **while** $p_k \leq \ell_k \wedge t_{k,p_k} < s$ **do** $p_k \leftarrow p_k + 1$
10: pointerIncremented $\leftarrow$ FALSE
11: intersection $\leftarrow \Lambda_k(p_k) \cap [s:t]$
12: **while** intersection $\neq \emptyset$ **do**
13: **if** $k = K$ **then**
14: hitRanges $\leftarrow$ hitRanges $\cup R((p_1, \ldots, p_K))$
15: **else**
16: admissibleRange $\leftarrow \mu_k$(intersection)
17: RecursiveSearch($k+1$, admissibleRange)
18: $p_k \leftarrow p_k + 1$
19: pointerIncremented $\leftarrow$ TRUE
20: **if** $p_k > \ell_k$ **then** intersection $\leftarrow \emptyset$
21: **else** intersection $\leftarrow \Lambda_k(p_k) \cap [s:t]$
22: **if** pointerIncremented = TRUE **then** $p_k \leftarrow p_k - 1$

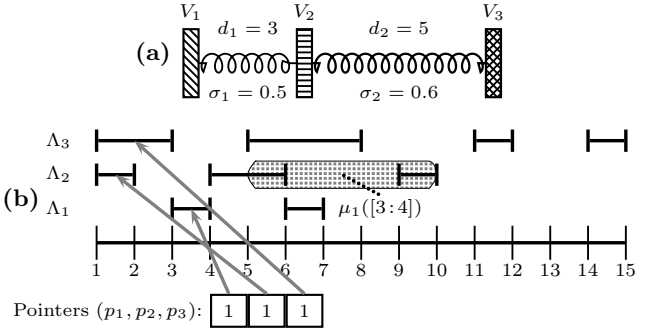

Figure 8: (a): Keyframe query of our running example. (b): Resulting keyframe lists Λ_1, Λ_2, and Λ_3. The pointers (p_1, p_2, p_3) are initialized to point to the first segments of the respective list. The range $\mu_1([3:4])$ is indicated by the gray area.

each list Λ_k, $k \in [1:K]$, consists of a sequence of ℓ_k segments, see (4). In Line 4, the pointers $(p_1, \ldots, p_K)$ are all initialized to the value one, thus pointing to the first segments of the respective lists. For our running example, this state is also illustrated by Fig. 8 (b). The three keyframe lists are

$$\begin{aligned} \Lambda_1 &= ([3:4], [6:7]), \\ \Lambda_2 &= ([1:2], [4:6], [9:10]), \\ \Lambda_3 &= ([1:3], [5:8], [11:12], [14:15]). \end{aligned} \tag{8}$$

Now, the **for**-loop in Line 5 runs over all segments $\Lambda_1(p_1)$, $p_1 \in [1:\ell_1]$. Note that these segments exactly contain the database frames that match to the first keyframe V_1. In other words, for each hit $H = (n_1, \ldots, n_K)$ one has $n_1 \in \Lambda_1(p_1)$ for some $p_1 \in [1:\ell_1]$. Line 6 specifies an admissible search range $\mu_1(\Lambda_1(p_1))$ for the second keyframe V_2. More generally, given a segment of candidate frames for the k^{th} keyframe, μ_k computes the admissible range, which is specified by d_k and σ_k, for the next keyframe. Here, the function $\mu_k := \mu_{\sigma_k, d_k} : \mathcal{S} \to \mathcal{S}$, $k \in [1:K-1]$, is defined as

$$[s:t] \mapsto \begin{cases} [s + \lceil \sigma_k \cdot d_k \rceil : t + \left\lfloor \frac{1}{\sigma_k} \cdot d_k \right\rfloor], & \text{if } \sigma_k > 0 \\ [s:\infty], & \text{if } \sigma_k = 0. \end{cases} \tag{9}$$
$$\emptyset \mapsto \emptyset$$

To illustrate this definition, we consider our running example for the case $p_1 = 1$. Then, $\Lambda_1(p_1) = [3:4]$ and $\mu_1([3:4]) = \mu_{\sigma_1, d_1}([3:4]) = \mu_{0.5,3}([3:4]) = [3 + \lceil 0.5 \cdot 3 \rceil : 4 + \lfloor \frac{1}{0.5} \cdot 3 \rfloor] = [3 + 2 : 4 + 6] = [5:10]$. In other words, if the first keyframe lies within the segment $[3:4]$, then the second keyframe must lie within the segment $[5:10]$ to fulfill condition (2), see Fig. 8 (b). Finally, Line 7 triggers the recursion starting with the second keyframe and the admissable range $\mu_1(\Lambda_1(p_1))$.

The procedure RecursiveSearch starts with fast forwarding the current pointer p_k (Line 9) until the list end is reached or until the current segment $\Lambda_k(p_k) = [s_{k,p_k} : t_{k,p_k}]$ does not lie entirely to the left of the admissible range $[s:t]$. In our example, this is the case for $p_2 = 2$, where $\Lambda_2(p_2) = [4:6]$. Line 11 calculates the intersection of the current segment and the admissible range. The intersection defines a segment of candidate frames that match keyframe V_k and fulfill the distance condition (2) for at least one frame of the previous segment $\Lambda_{k-1}(p_{k-1})$. In our example, the intersection is $[4:6] \cap [5:10] = [5:6]$.

In the **while**-loop, starting with Line 12, all segments in Λ_k that non-empty intersect with the admissible range $[s:t]$ are considered. Here, the increment of the pointer p_k and computation of the intersections is handeled between Line 18 and Line 21. In the case $k = K$, each such intersection contributes to a hit (this directly follows from what was said above). Therefore, in Line 14, the hit relevant range of a resulting match is computed and the union is formed with previously computed hit relevant ranges. An example for this step will be discussed later. In the case $k < K$, a new admissible range is computed (Line 16), and a recursion is triggered with the $(k+1)^{\text{th}}$ keyframe (Line 17).

We continue our example with $p_2 = 2$ and the non-empty intersection $[5:6]$. In Line 16, the admissible range $[8:14] = \mu_2([5:6])$ is computed. Line 17 triggers another call of RecursiveSearch for the third keyframe. At this recursion level, p_3 is incremented to $p_3 = 2$ (Line 9) and the intersection of the current segment $\Lambda_3(p_3)$ and the admissible range is $[5:8] \cap [8:14] = [8:8]$. Now, the condition $k = K$ is fulfilled. The pointers $(p_1, p_2, p_3) = (1, 2, 2)$ define a match and Line 14 extends the union of the hit relevant ranges by $R((1,2,2)) = [3:8]$. At this point we note that the hit relevant range $[s:t]$ of a given match can be efficiently computed. Here, the end frame of the intersection, calculated in Line 11, yields t. To calculate s, one has to only consider the K intersections that yield the found match during the recursion. Then, Line 18 sets $p_3 = 3$ and the resulting intersection is $[11:12] \cap [8:14] = [11:12]$ (Line 21). The pointers $(p_1, p_2, p_3) = (1, 2, 3)$ define another match with $R((1,2,3)) = [3:12]$, which is merged with the previously found hit relevant range by means of the union operator in Line 14. After incrementing to $p_3 = 4$, Line 20 sets the intersection to $[14:14]$ and $R((1,2,4)) = [3:14]$ is processed in line Line 14. Now, incrementing the pointer p_3 in Line 18 exceeds the list boundary, so the empty intersection, set in

M	Hits	$R(M)$
(1,2,2)	(3,5,8)	[3:8]
(1,2,3)	(3,5,11), (3,6,11), (3,6,12), (4,6,11), (4,6,12)	[3:12]
(1,2,4)	(3,6,14), (4,6,14)	[3:14]
(1,3,4)	(3,9,14), (3,9,15), (4,9,14), (4,9,15), (4,10,14), (4,10,15)	[3:15]
(2,3,4)	(6,9,14), (6,9,15), (6,10,14), (6,10,15) (7,9,14), (7,9,15), (7,10,14), (7,10,15)	[6:15]

Table 1: Matches found by the proposed algorithm for our running example, hits that are contained in these matches, and their hit relevant ranges $R(M)$.

Line 20, causes the **while**-loop to stop. In Line 22, the pointer p_3 is decremented to the last value, where the intersection was non-empty. In our example, we then have $p_3 = 4$. Note that the decrementation is necessary to find all hit relevant ranges. Although the matches already comprise the last segment of the third inverted keyframe list, the hit relevant ranges of the matches found so far do not include frame 15. Decrementing p_3 and continuing the algorithm ensures finding the correct hit relevant ranges. The recursion returns to the point where RecursiveSearch(3, (8,14)) was called (Line 17). The pointer p_2 is incremented to $p_2 = 3$ (Line 18) and intersection $= [9:10] \cap [5:10] = [9:10]$ is calculated (Line 21). The **while**-loop is repeated and in Line 16 the admissible range is set to $\mu_2([9:10]) = [12:18]$. The subsequent recursive call (Line 17) leads to the match $(1, 3, 4)$. Finally, the pointer p_1 is increased leading to another match $(2, 3, 4)$.

Table 1 shows all matches M found by our algorithm along with all hits contained in the respective match and the resulting hit relevant ranges $R(M)$. Actually, there are two matches, $(1, 3, 3)$ and $(2, 3, 3)$, which are not found by the algorithm. These matches, however, are irrelevant since $R((1, 3, 3)) = [3:12]$ and $R((2, 3, 3)) = [6:12]$ are contained in unions of hit relevant ranges of the other matches. Also recall that the actual output of the algorithm consists of the union of all $R(M)$, thus avoiding an explosion of the output size. In our example, this results in a single segment [3:15].

4. EXPERIMENTS

Our keyframe-based search algorithm works for general time-dependent multimedia data and is designed for efficiently handling temporal deformation between the query and database keyframes. We will demonstrate the practicability of our concept by means of the motion retrieval scenario, where one typically encounters such deformations between semantically related motion sequences. In Sect. 4.2, we describe some experiments showing that our algorithm is often able to limit the search space from several hours to a couple of minutes of motion capture data within a few milliseconds (ms). The so reduced data set can then be ranked and analyzed by more refined alignment techniques. We will demonstrate such a two-stage retrieval strategy by using the motion templates [8] for postprocessing the keyframe-based matches, see Sect. 4.3. Here, we also report on some experiments to demonstrate the effect of the stiffness parameter on the final retrieval result. Information on the experimental data set and the used keyframes can be found in Sect. 4.1.

4.1 Experimental Data Set and Keyframes

For our experiments, we used the HDM05 database, which consists of 210 minutes of motion data contained in 324 files. Making up an average length of 39 seconds, each file consists of a sequence of different actions. A detailed description of the motion files[1] can be found in [10]. From the HDM05 database we cut out 1327 short motion clips, which were organized into 57 motion classes, each containing 10 to 50 realizations executed by various actors. These motion classes were used to generate keyframes in a semi-automatic process. Here, using one half of the motion clips of each class as training data, we computed the quantized motion templates based on the 39 relational features as described in [8]. These features were divided up into three feature sets: One for the upper body, one lower body, and one mixed set. We then selected 3 to 9 representative columns along with their distances of each class motion template as keyframes. For details of a similar procedure we refer to [8]. To avoid false negatives (at the expense of having more false positives), we manually post-processed the keyframes by adding or removing suitable feature vectors from the keyframes, see Sect. 3.2.

At this point, we note that the focus of this paper is not on the fully automated generation of keyframes, but on the efficient and deformation-tolerant retrieval based on a given set of suitable keyframes. In order to fully automate the keyframe generation, one may use genetic algorithms to learn keyframes from positive as well as negative training motions, see [7]. Further keyframe selection methods based on the analysis of joint attribute sequences are described by Assa et al. [1]. Once suitable keyframes are generated for a specific motion class, they can be used as queries to arbitrary databases. Furthermore, we concatenated the motion class keyframes to generate longer and more complex queries describing sequences of different actions, see Sect. 4.3.

To prove the applicability of our algorithm, we conducted several experiments. The presented algorithm has been implemented in a mixture of C and Matlab®. All experiments were executed on an AMD Athlon™ 64 X2 5000+ (using only 1 core) with 3.5GB of RAM.

4.2 Data Reduction

As shown in Table 2, we used the generated keyframes as an input for our algorithm to reduce the search space for 57 query motion classes. For each query, K denotes the number of keyframes. $\sum l_v$ amounts to the number of segments in the inverted lists that have to be processed in the query. To demonstrate the efficiency of our algorithm, $\mathbf{t^K}$ shows the keyframe search time in milliseconds, and $\%(\mathbf{D})$ shows the size of the reduced search space in percent with regard to the database size. The time taken to reduce the search space using our algorithm comprises on average only 12.5ms to reduce the search space to less than 5% of the entire database. As the table shows, the search time depends on the size of the processed inverted lists. Small searching times, like 0.3ms for query ID1, are due to queries that contain keyframes describing infrequent poses in the database. For query ID1, characteristic poses for a cartwheel, which occur only in few other motions in the database, were used as keyframes. In contrast to this, using the 39 full body motion features from [8], the class ID49 (turnRight) can not be distinguished from the standing pose. Here, using a total of 9 keyframes, more than 60000 segments have to be processed. Because of the unspecific keyframes, all standing

[1]The files have been made available at `http://www.mpi-inf.mpg.de/resources/HDM05/`.

ID	query class	K	$\sum l_v$	t^K	%(D)	t^R
1	cartwheelLHandStart1Reps	3	75	**0.3**	**1.2**	234
2	clap1Reps	5	39326	**36.6**	**14.0**	5094
3	clapAboveHead1Reps	5	11959	**11.8**	**1.7**	328
4	depositFloorR	5	18565	**17.5**	**6.0**	1063
5	depositHighR	7	20860	**21.5**	**2.2**	375
6	elbowToKnee1RepsLelbowStart	3	265	**0.5**	**0.6**	78
7	elbowToKnee1RepsRelbowStart	4	1631	**2.0**	**0.6**	63
8	grabFloorR	7	20664	**19.3**	**2.5**	344
9	grabHighR	6	18560	**17.9**	**2.9**	656
10	hopBothLegs1hops	5	23808	**22.0**	**1.0**	125
11	hopLLeg1hops	3	4358	**4.5**	**0.5**	31
12	hopRLeg1hops	4	16965	**15.8**	**1.1**	109
13	jogLeftCircle4StepsRstart	4	2115	**2.8**	**1.4**	203
14	jogOnPlaceStartFloor2StepsRStart	5	28583	**27.6**	**30.4**	8766
15	jogRightCircle4StepsRstart	3	1451	**1.8**	**1.3**	188
16	jumpDown	4	5726	**5.6**	**1.2**	188
17	jumpingJack1Reps	3	780	**1.5**	**0.9**	109
18	kickLFront1Reps	5	30918	**26.9**	**2.0**	250
19	kickLSide1Reps	4	1516	**2.0**	**1.1**	141
20	kickRFront1Reps	5	4302	**5.3**	**1.1**	188
21	kickRSide1Reps	4	11610	**10.5**	**1.2**	172
22	lieDownFloor	3	1784	**2.2**	**2.7**	813
23	punchLFront1Reps	5	11214	**12.5**	**1.4**	234
24	punchLSide1Reps	6	29549	**29.8**	**2.6**	375
25	punchRFront1Reps	6	16880	**19.3**	**2.5**	406
26	punchRSide1Reps	6	45343	**42.8**	**1.7**	266
27	rotateArmsBothBackward1Reps	3	837	**1.0**	**1.5**	172
28	rotateArmsBothForward1Reps	6	6009	**6.5**	**0.7**	78
29	rotateArmsLBackward1Reps	4	6021	**6.0**	**0.6**	78
30	rotateArmsLForward1Reps	3	508	**0.8**	**0.7**	94
31	rotateArmsRBackward1Reps	3	1618	**2.1**	**0.6**	63
32	rotateArmsRForward1Reps	3	298	**0.6**	**0.7**	78
33	runOnPlaceStartFloor2StepsRStart	5	24522	**22.9**	**0.4**	31
34	shuffle2StepsRStart	8	31395	**31.6**	**2.0**	359
35	sitDownChair	6	2793	**3.8**	**2.4**	422
36	sitDownFloor	5	3911	**4.5**	**4.7**	1109
37	sitDownKneelTieShoes	3	372	**0.7**	**2.2**	609
38	sitDownTable	6	18393	**19.9**	**29.3**	16766
39	skier1RepsLstart	5	4157	**5.2**	**0.8**	78
40	sneak2StepsLStart	4	4935	**5.6**	**1.5**	250
41	sneak2StepsRStart	6	18126	**18.3**	**2.0**	328
42	squat1Reps	4	848	**1.2**	**1.0**	141
43	staircaseDown3Rstart	5	9644	**10.5**	**6.5**	1469
44	staircaseUp3Rstart	5	2860	**4.0**	**1.7**	281
45	standUpLieFloor	3	463	**0.7**	**1.7**	375
46	standUpSitFloor	6	7969	**7.9**	**3.6**	672
47	throwBasketball	5	5624	**6.9**	**3.0**	563
48	turnLeft	7	34548	**33.8**	**10.6**	4063
49	turnRight	9	63819	**61.8**	**20.1**	9844
50	walk2StepsLstart	6	12956	**14.8**	**10.5**	4609
51	walk2StepsRstart	5	18547	**18.4**	**13.2**	5875
52	walkBackwards2StepsRstart	5	8641	**9.8**	**1.1**	141
53	walkSidewaysLeft2Steps	3	312	**0.5**	**0.9**	141
54	walkLeftCircle4StepsRstart	6	11447	**13.8**	**13.5**	3953
55	walkOnPlace2StepsLStart	5	14792	**16.0**	**35.9**	17422
56	walkRightCircle4StepsRstart	7	10196	**14.1**	**11.5**	2719
57	walkRightCrossFront2Steps	3	6545	**5.7**	**4.0**	875
∅		4.8	12314	**12.5**	**4.8**	1657
	sequences of actions					
58	ID 6 + ID 42	7	1114	**1.8**	**1.2**	281
59	ID 13 + ID 15	7	3566	**5.1**	**1.3**	438
60	ID 19 + ID 23, ID 26	16	63739	**72.7**	**1.8**	797
61	ID 22 + ID 22	6	3568	**4.2**	**2.1**	1656
62	ID 27 + ID 28	10	12030	**14.4**	**1.0**	219
63	ID 39 + ID 6	8	4423	**5.8**	**0.8**	234
64	ID 44 + ID 43	10	12505	**16.7**	**1.0**	281
65	ID 52 + ID 57	8	15186	**17.5**	**1.7**	594
66	ID 55 + ID 33	10	39314	**42.5**	**1.0**	391
∅		9.1	17272	**20.1**	**1.3**	543

Table 2: Retrieval results on the HDM05 database (3.5 hours of mocap data). K: Number of keyframes used in the query. $\sum l_v$: Number of segments in the processed inverted lists. t^K: Keyframe search time in ms. $\%(D)$: Size of the reduced search space in percent (w.r.t. the size of HDM05). t^R: Time for motion template retrieval on the reduced search space in ms.

poses in the database are contained in the reduced search space. It is important to notice that altough the number of keyframes in this case is rather high, the search time does not explode.

Having a look at the percental size of the reduced search space, for most of the classes, sizes of less than 3% can be achieved. For many classes, like ID6, even better reducing rates are reached, returning less than 1% of the HDM05 database. Unlike these results, some queries do not reduce the search space well. As already mentioned, the keyframes for query ID49 are rather unspecific. As a result, for this query more than 20% of the database is returned. Similarly,

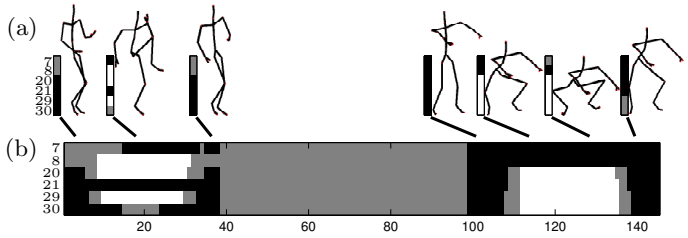

Figure 9: A combination of queries can be used to search for a sequence of motion classes. The feature labels correspond to [8].

for queries ID50 to ID57, some of the reduction rates are not so good due to the large number of walking motions in HDM05 and due to some confusion between various walking styles.

As a further application, the keyframes can be combined to describe a query for an entire sequence of various actions. As an example, the conatenation of the queries elbow-to-knee and squat is shown in Fig. 9 (a). By setting the distance d_3 and the corresponding stiffness parameter σ_3 between the last keyframe of the first motion class and the first keyframe of the second motion class, one can control the time that may elapse between the two actions. A similar approach to scene description has been sketched in [9]. To demonstrate the applicability of this scenario, we have created nine combined queries. The retrieval results are documented in Tab. 2, Lines 58 to 66. Although more keyframes are used, the keyframe search time does not explode. For example, in query 60, comprising 16 keyframes, a total of 63739 segments have been processed in $72.7ms$. In comparison to query ID49, which exhibits a similar number of processed segments, but a smaller number of keyframes, only a small increase in the search time can be observed. Additionaly, for all combined queries the size of the reduced search space is very small. Generally, the more keyframes are used in a query, the less data fits to these keyframes.

4.3 Retrieval Quality

To show the effectiveness of our algorithm in a two-stage retrieval system, we apply motion template retrieval [8] as a ranking of the reduced search space. For queries ID1 to ID57, class motion templates have been used. For queries ID58 to ID66, the class motion templates have been combined as indicated in Fig. 9 (b). In this example, the elbow-to-knee motion template and the squat motion template have been concatenated with a block of 0.5-values, assigning zero cost for this clipping during the ranking process. The output of this postprocessing step is a set of ranked motion clips contained in the reduced search space.

In Table 2, column t^R, depicts the time in ms used for the ranking step. Using a variant of DTW, the running time depends linearly on the product of the size of the reduced search space and the size of the motion template. In practice, for most of the queries this step takes less than a second. The speed-up with regard to motion template retrieval on the whole database is equal to the reduction rate of the keyframe based search. To demonstrate the quality of the results, precision-recall diagrams for some queries are shown in Fig. 10. For these queries, 15 to 75 relevant documents are contained in the HDM05 database. The recall is very high, which means that the reduced search space,

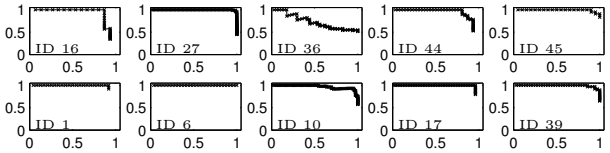

Figure 10: Precision-recall diagrams of hits obtained by queries with the indicated IDs (see Tab. 2).

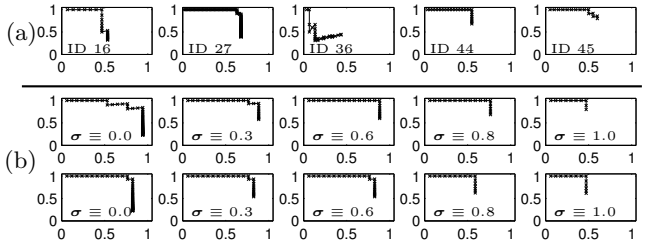

Figure 11: Stiffness experiments. (a): Queries of the first row of Fig. 10 were modified to $\sigma \equiv 1.0$. (b): Precision-recall diagram for query ID59 (top row), and ID60 (bottom row), whereas the manually optimized stiffness values have been modified to the specified values.

obtained by our algorithm with stiffness values around 0.6, still contains most of the relevant documents. Unlike the other queries, on query ID36 ("sit down on floor") false positives occur early in the hit list. Most of the false positives are motions of the class "lie down on floor", which starts in most cases in the HDM05 database with a "sit down on floor"-phase.

To quantify the influence of the stiffness parameter, Fig. 11 shows the results for queries where the stiffness parameter has been set to the value 1.0, thus prohibiting any time deformation between keyframes. In comparison to the corresponding diagrams in the first row of Fig. 10, many relevant documents have been missed due to the denial of temporal deformations.

A further experiment shows the effect of varying the stiffness parameter. Fig. 11 (b), top row, shows precision-recall diagrams for query ID59 with modified stiffness values. Some false positives occur when setting $\boldsymbol{\sigma} \equiv 0$, which are eleminated when using higher stiffness values. For $\boldsymbol{\sigma} \equiv 0.6$, the precision-recall diagram is the same as for a manually optimized stiffness values. A further raise of the stiffness results in a loss of some relevant documents. A similar behavior is demonstrated for query ID60 in Fig. 11 (b), bottom row. Again, no difference in the $\boldsymbol{\sigma} \equiv 0.6$-diagram can be noticed in comparison to the manually optimized query stiffness values. However, the sizes of the reduced search spaces and so the times for the ranking steps are smaller for queries ID59 and ID60 than for the $\boldsymbol{\sigma} \equiv 0.6$-modified queries.

5. CONCLUSIONS

In this paper, we have introduced a novel algorithm for keyframe-based multimedia retrieval, which is able to drastically cut down the search space. In contrast to previous approaches, our index-based algorithm can cope with significant temporal deformations without resorting to computationally expensive techniques such as dynamic time warping. To prove its practicability, we have applied our algorithm within a two-stage motion retrieval scenario. As it turned out, the temporal flexibility introduced by our stiffness concept is necessary to avoid a large number of false negatives in the preselection step. Because of its efficiency, our algorithm may be applied iteratively as a subroutine in classification and learning scenarios. For example, we plan to apply such an approach for automatically learning characteristic keyframes using supervised training algorithms based on positive and negative motion examples.

6. ACKNOWLEDGMENTS

The research was funded by the German Research Foundation (DFG CL 64/5-1) and the Cluster of Excellence on Multimodal Computing and Interaction.

7. REFERENCES

[1] J. Assa, Y. Caspi, and D. Cohen-Or. Action synopsis: Pose selection and illustration. *ACM Trans. Graph.*, 24(3):667–676, 2005.

[2] CMU. Carnegie-Mellon Mocap Database. `http://mocap.cs.cmu.edu`, 2003.

[3] K. Forbes and E. Fiume. An efficient search algorithm for motion data using weighted PCA. In *Proc. 2005 ACM SIGGRAPH/Eurographics Symposium on Computer Animation (SCA 2005)*, pages 67–76. ACM Press, 2005.

[4] E. J. Keogh, T. Palpanas, V. B. Zordan, D. Gunopulos, and M. Cardle. Indexing large human-motion databases. In *Proc. 30th VLDB Conf., Toronto*, pages 780–791, 2004.

[5] L. Kovar and M. Gleicher. Automated extraction and parameterization of motions in large data sets. *ACM Trans. Graph.*, 23(3):559–568, 2004.

[6] M. Müller. *Information Retrieval for Music and Motion*. Springer, 2007.

[7] M. Müller, B. Demuth, and B. Rosenhahn. An evolutionary approach for learning motion class patterns. In *Proc. DAGM*, LNCS 5096. Springer, 2008.

[8] M. Müller and T. Röder. Motion templates for automatic classification and retrieval of motion capture data. In *Proc. of the 2006 ACM SIGGRAPH/Eurographics Symposium on Computer Animation*, pages 137–146. ACM Press, 2006.

[9] M. Müller, T. Röder, and M. Clausen. Efficient content-based retrieval of motion capture data. *ACM Trans. Graph.*, 24(3):677–685, 2005.

[10] M. Müller, T. Röder, M. Clausen, B. Eberhardt, B. Krüger, and A. Weber. Documentation: Mocap Database HDM05. Computer Graphics Technical Report CG-2007-2, Universität Bonn, June 2007. `http://www.mpi-inf.mpg.de/resources/HDM05/`

[11] B. Rosenhahn, R. Klette, and D. Metaxas. *Human Motion Understanding, Modeling, Capture, and Animation*. Springer, 2007.

[12] I. H. Witten, A. Moffat, and T. C. Bell. *Managing Gigabytes*. Morgan Kaufmann Publishers, 1999.

[13] M.-Y. Wu, S. Chao, S. Yang, and H. Lin. Content-based retrieval for human motion data. In *16th IPPR Conf. on Computer Vision, Graphics and Image Processing*, pages 605–612, 2003.