EFFICIENT INDEXING AND RETRIEVAL OF MOTION CAPTURE DATA BASED ON ADAPTIVE SEGMENTATION

Meinard Müller, Tido Röder, and Michael Clausen

University of Bonn
Department of Computer Science
{meinard,roedert,clausen}@cs.uni-bonn.de

ABSTRACT

In this paper we propose a unified approach to efficient indexing and content-based retrieval of human motion capture data as used in data-driven computer animation or computer vision. Opposed to other data formats such as music or video, the kinematic chain (a kind of human skeleton) as underlying model of motion capture data allows to introduce qualitative boolean features describing geometric relations of specified points of the body. In combination with such geometric features, we introduce the concept of adaptive temporal segmentation of motion data streams, which accounts for the spatio-temporal invariance needed to identify logically related motions. This allows us to adopt efficient indexing and fault-tolerant retrieval methods such as fuzzy search. Here, the crucial point is that our adaptive segmentation not only adjusts to the granularity of the feature function but also to the fuzziness of the query. We present experimental results on a test data set of more than one million frames corresponding to 180 minutes of motion capture data. The linearity of our indexing algorithms guarantees the scalability of our results to much larger data sets.

1. INTRODUCTION

In the past two decades, motion capturing has become an important tool for synthesizing realistic motion sequences for computer-animated movies or video games. The generation of motion capture data still constitutes a time-consuming and expensive process due to the high cost of recording equipment and the required manual post-processing. Therefore, numerous suggestions have been made how to reuse motion capture data by modifying and adapting existing motion clips via editing and morphing techniques, see, e.g., [1, 5, 7, 9]. Such techniques depend on motion capture databases covering a broad spectrum of motions in various characteristics. Only recently, larger collections of motion material such as [4] have become publicly available.

In order to fully exploit motion databases, one needs efficient algorithms for content-based and fault-tolerant search allowing to automatically identify and extract user-specified motions. Here, the motion clips to be retrieved may be specified in various ways at different semantic levels. One possible specification could be a rough textual description such as "a kick of the right foot followed by a punch". Another query mode would involve a short query motion clip, the task being to automatically locate all database motions that are in some sense similar to the query. This kind of problem, commonly referred to as *content-based retrieval*, is at the heart of this paper.

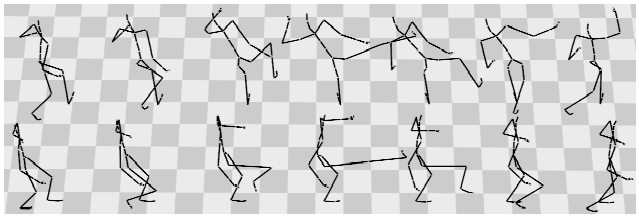

Fig. 1. Upper: seven poses from a side kick sequence. Lower: corresponding poses for a front kick.

The crucial point in content-based motion retrieval is the notion of "similarity" used to compare different motions. Intuitively, two motions may be regarded as similar if they represent variations of the same action or sequence of actions, see [7]. Here the variations may concern the spatial as well as the temporal domain. For example, the kicks shown in Fig. 1 describe the same kind of motion even though they differ considerably with respect to direction and height of the kick. In other words, *logically similar* motions need not be *numerically similar*, as is also pointed out in [7].

To handle queries in form of an example motion clip, we suggest new types of *qualitative, geometric features* opposed to *quantitative, numerical features* used in previous approaches, see Sect. 1.1. These features, as illustrated by Fig. 3, exploit the existence of a skeletal model underlying the motion capture data format, which allows each frame of the motion data stream to be interpreted as a skeletal pose. We show how pose-based geometric features, accounting for spatial invariance, induce an *adaptive temporal segmen-*

tation of the motion data streams, accounting for temporal invariance. Based on these two concepts, the motion capture data streams are transformed into coarse *sequences of geometric configurations*, see Sect. 2; two motion clips are considered as similar if they coincide (more or less) at this sequence level.

The trick is that by incorporating robustness against spatio-temporal variations (as typical for logically related motions) into the geometric features and adaptive segments, we are able to employ standard information retrieval techniques using an index of inverted lists [12]. Here, the feature vectors serve as index words, and indexing is carried out at the segment level rather than at the frame level. In particular, the time and space required to build and store our index structure is *linear*, opposed to the *quadratic* complexity of strategies based on dynamic time warping (DTW), cf. Sect. 1.1.

To account for uncertainties in the query, we adopt the concept of *fuzzy search*, see [3]. Here, our contribution is to extend the concept of adaptive temporal segmentation, adjusting segment lengths within a match not only to the granularity of the feature function but also to the fuzziness of the query. The crucial point is that the resulting *adaptive fuzzy hits* can be computed very efficiently using the same index structure as for the case of exact hits.

The remainder of this paper is organized as follows. We close this section with a discussion of related work and some basic facts about motion capture data, while fixing the notation for the subsequent sections. In Sect. 2 we give an overview of the two concepts of geometric features and adaptive temporal segmentation, details having been described in the work [8]. Sect. 3 constitutes the core of this paper. Based on the query-by-example paradigm, we show how logically related motions can be efficiently retrieved from a motion database using a linear-space index structure. We then introduce the concept of fuzzy queries and adaptive fuzzy hits and give a detailed account on how to retrieve such hits. We present experimental results in Sect. 4 and conclude with possible directions for future work in Sect. 5.

1.1. Related work

Only recently, motion capture data has become publicly available on a larger scale as provided by [4], reinforcing the demand for efficient indexing and retrieval methods. Such methods are necessary to efficiently retrieve logically related motions, which can then be processed via editing and morphing techniques, see, e.g., [1, 5, 7, 9] and the references therein. So far, only little work has been published on motion capture indexing and retrieval based on the query-by-example paradigm. We give a short overview of the relevant literature. To identify motions similar to a given query motion, the authors of [13] proceed in two stages: they first identify start and end frames of possible candidate clips utilizing a pose-based index and then compute the actual distance from the query via DTW. In [7], numerically similar motions are identified by means of a DTW-based index structure termed *match web*. A multi-step search spawning new queries from previously retrieved motions allows for the identification of logically similar motions using numerically similar motions as intermediaries. However, major drawbacks to DTW are its quadratic running time and storage requirements, making DTW infeasible for large data sets. To speed up similarity search, the authors of [6] use an index structure based on bounding envelopes allowing to identify similar motion fragments that differ by some uniform scaling factor with respect to the time axis. In comparing individual frames of the data streams, all of these approaches rely on *numerical* cost measures.

The idea of considering *geometric* (combinatorial, relational, qualitative) features instead of numerical (metrical, quantitative) features is not new and has already been applied in other domains such as visual object recognition in 2D and 3D or action recognition and tracking, see, e.g., [2, 10] and references therein. The following observations are of fundamental importance, see [2]: firstly, relational structures are not only interesting for general recognition problems (due to their invariance properties) but also ideally suited for indexing (due to their discrete nature). Secondly, *relational* similarity of shapes correlates quite well with *perceptual (logical)* similarity. These principles motivate the usage of sequences of geometric configurations in order to identify logically similar movements.

1.2. Motion capture data

The most common recording technology for motion capture data uses an array of digital cameras to three-dimensionally track reflective markers attached to a live actor's body, see, e.g. [11]. The tracking data can then be post-processed to obtain a multi-stream of 3D trajectories corresponding to the joints of a fixed skeletal *kinematic chain* as indicated by Fig. 2. A full set of 3D coordinates describing the joint positions of a kinematic chain for a fixed point in time is also referred to as a *pose*. In this paper, a motion capture data stream is thought of as a sequence of poses, typically sampled at 30–600 Hz. Mathematically, a pose can be regarded as a matrix $P \in \mathbb{R}^{3\times J}$, each column of P corresponding to the 3D coordinates of a joint. Here, J denotes the number of joints, e.g., $J = 24$ in the skeleton shown in Fig. 2. Then, a motion capture data stream D can be viewed as a function $D : [1 : T] \to \mathcal{P} \subset \mathbb{R}^{3\times J}$, where T denotes the number of poses, $[1 : T] := \{1, 2, \ldots, T\}$ corresponds to the time axis (for a fixed sampling rate), and $\mathcal{P}$ denotes the set of poses. The 3D curve described by a single joint is termed *trajectory*, so motion capture data can also be viewed as a multi-stream of time-synchronized trajectories, see Fig. 5. In a sense, motion capture data has a much richer seman-

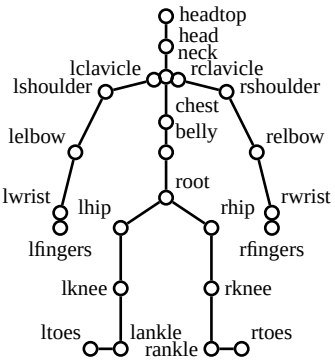

Fig. 2. Skeletal kinematic chain model consisting of rigid *body segments* flexibly connected by *joints*, which are highlighted by circular markers and labeled with joint names.

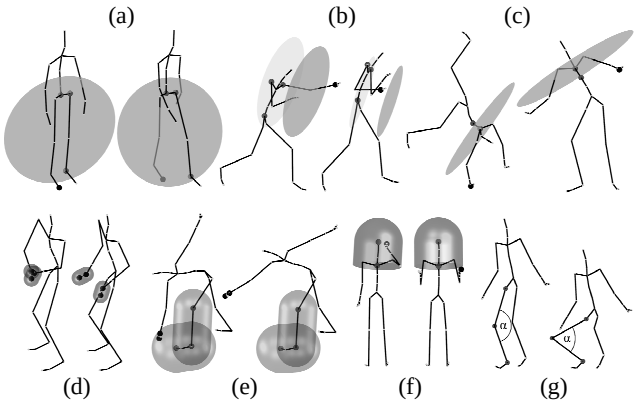

Fig. 3. Various geometric features describing geometric relations between the joints of a pose.

tic content than, for example, pure video data of a motion, since the position and the meaning of all joints is known for every pose. In recent approaches such as [6, 7] poses are compared based on a numerical distance measure defined in a joint-wise manner. Our approach differs fundamentally by exploiting the geometric relations between the joints, which capture salient characteristics of logically related motions.

2. GEOMETRIC FEATURES AND INDUCED TEMPORAL SEGMENTATION

We now define various kinds of *geometric features* describing geometric relations between specified points of the kinematic chain for a fixed, isolated pose. To this end, we need the notion of a *boolean feature*, which we describe mathematically as a boolean function $F : \mathcal{P} \to \{0,1\}$. Obviously, any boolean expression of boolean functions (evaluated pose-wise) is a boolean function itself, the most prominent examples being the conjunction $F_1 \wedge F_2$ and the disjunction $F_1 \vee F_2$ of boolean functions F_1 and F_2. Forming a vector of f boolean functions for some $f \geq 1$, one obtains a combined function $F : \mathcal{P} \to \{0,1\}^f$. Subsequently, F will be referred to as a *feature function* and the vector $F(P)$ as a *feature vector* or simply a *feature* of the pose $P \in \mathcal{P}$. Any feature function can be applied to a motion capture data stream $D : [1 : T] \to \mathcal{P}$ in a pose-wise fashion, which is expressed by the composition $F \circ D$.

There are numerous classes of semantically meaningful geometric features. We explain the essential ideas by means of the examples shown in Fig. 3 and refer to [8] for a more detailed account. As a first example, consider the oriented plane determined by the left ankle, the left hip, and the root joint (Fig. 3 (a)). When the right foot lies in front of that plane, the geometric feature F^r is defined to assume the value one, otherwise zero. Obviously, the feature $F^r(P)$ is 1 for a pose P corresponding to a person standing upright. It assumes the value 0 when the right foot moves to the back or the left foot to the front, which is typical walking or running motions (Fig. 5). Interchanging corresponding left and right joints in the definition of F^r and flipping the orientation of the resulting plane, we obtain another feature function denoted by F^l. Let us have a closer look at the feature function $F := F^r \wedge F^l$, which is 1 if and only if both, the right and the left toes, are in front of the respective planes. It turns out that F is very well suited to characterize any kind of walking or running motions. If a data stream $D : [1 : T] \to \mathcal{P}$ describes such a locomotion, then $F \circ D$ exhibits exactly two peaks for any locomotion cycle, from which the speed of the motion can easily be deduced (see Fig. 4). Furthermore, the feature F is invariant under global orientation and position, the size of the skeleton, and various local spatial deviations such as sideways and vertical movements of the legs. Of course, F also leaves the upper body unconsidered.

A similar feature is defined by the oriented plane fixed at the left and right shoulders and the root, shifted one humerus length to the front: checking whether the left hand is in front of or behind this plane, one obtains a feature suitable to identify left punches, see Fig. 3 (b). Similarly, considering the plane that is normal to the spine and fixed at the neck, one can check whether a hand is below or above the neck (Fig. 3 (c)). Other types of geometric features express whether specified points of the body are close together, yielding, for example, "touch" detectors for the two hands, a hand and a leg, or a hand and the head (Fig. 3 (d)–(f)). We also use geometric features to check if certain parts of the body such as the arms or the legs are bent or stretched (Fig. 3 (g)). Note that geometric features are invariant under global transformations and local spatial deformations.

Induced temporal segmentation

Let $F : \mathcal{P} \to \{0,1\}^f$ be a fixed feature function. Then an *F-segment* of a data stream D is defined to be a substream

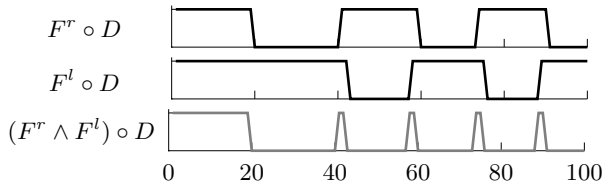

Fig. 4. Boolean features F^r, F^l, and $F^r \wedge F^l$ applied to the 100-frame walking motion $D = D_{\text{walk}}$ of Fig. 5.

of D of maximal length consisting of consecutive frames that exhibit the same feature value. For example, applying the feature function $F^2 := (F^r, F^l) : \mathcal{P} \to \{0,1\}^2$ to the walking motion $D = D_{\text{walk}}$ results in a temporal segmentation of D into 11 segments, as shown in Fig. 5. Note that no pose with feature vector $(0,0)$ appears in this example.

It is this feature-dependent segmentation that accounts for the postulated temporal invariance, the main idea being that motion capture data streams can now be compared at the segment level rather than at the frame level. To be more precise, let us start with the sequence of F-segments of a motion capture data stream D. Since each segment corresponds to a unique feature vector, the segments induce a sequence of feature vectors (or *sequences of geometric configurations*, as denoted in the introduction), which we also refer to as *F-feature sequence* of D and denote by $F[D]$. If M is the number of F-segments of D and if $D(t_m)$ for $t_m \in [1:T]$, $1 \leq m \leq M$, is any pose of the m-th segment, then $F[D] = (F(D(t_0)), F(D(t_2)), \ldots, F(D(t_M)))$. For example, the data stream $D = D_{\text{walk}}$ and the feature function F^2 from Fig. 5 yields

$$F^2[D] = \big(\tbinom{1}{1},\tbinom{0}{1},\tbinom{1}{1},\tbinom{1}{0},\tbinom{1}{1},\tbinom{0}{1},\tbinom{1}{1},\tbinom{1}{0},\tbinom{1}{1},\tbinom{0}{1},\tbinom{1}{1}\big). \quad (1)$$

Obviously, any two adjacent vectors of the sequence $F[D]$ are distinct. The crucial point is that time invariance is incorporated into the F-segments: two motions that differ by some deformation of the time axis will yield the same F-feature sequences. In general, fine features, i.e., feature functions with many components, induce segmentations with many short segments, whereas coarse features lead to a smaller number of long segments.

The main idea is that two motion capture data streams D_1 and D_2 can now be compared via their F-feature sequences $F[D_1]$ and $F[D_2]$ instead of comparing the data streams on a frame-to-frame basis. This has several advantages:

1. One can decide which aspects of the motions to focus on by picking a suitable feature function F.
2. Since spatial and temporal invariance are already incorporated in the features and segments, one can use efficient methods from (fault-tolerant) string matching to compare the data streams instead of applying cost-intensive techniques such as DTW at the frame level.

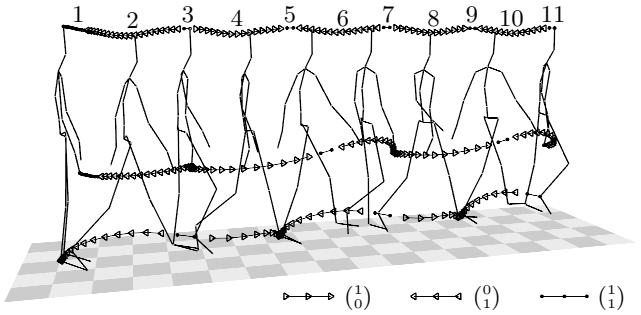

Fig. 5. F^2-segmentation of $D = D_{\text{walk}}$, where F^2-equivalent poses are indicated by identically marked trajectory segments. The trajectories of the joints 'head-top','rankle', and 'rfingers' are shown.

3. In general, the number M of segments is much smaller than the number T of frames, which accounts for very efficient computations.

In the next section, we will see how our concept leads to an efficient method of indexing motion capture data in a semantically meaningful way.

3. INDEXING AND FAULT-TOLERANT RETRIEVAL

In this section, we use the following notations. The database consists of a collection $\mathcal{D} = (D_1, D_2, \ldots, D_I)$ of motion capture data streams or documents D_i, $i \in [1:I]$. To simplify things, we may assume that $\mathcal{D}$ consists of one large document D by concatenating the documents $D_1, \ldots, D_I$, keeping track of document boundaries in a supplemental data structure. The query Q consists of an example motion clip. We fix a feature function $F : \mathcal{P} \to \{0,1\}^f$ and use the notation $F[D] = \vec{w} = (w_0, w_1, \ldots, w_M)$ and $F[Q] = \vec{v} = (v_0, v_1, \ldots, v_N)$ to denote the resulting F-feature sequences of D and Q, respectively. The document $D = D_{\text{walk}}$ with $F = F^2$, as shown in Fig. 5, will serve as a running example.

3.1. Index

To index our database $\mathcal{D}$ with respect to F, we use standard techniques based on inverted lists, see, e.g., [12]. For each feature vector $v \in \{0,1\}^f$ we store the *inverted list* $L(v)$ consisting of the indices $m \in [1:M]$ of the sequence $\vec{w} = (w_0, w_1, \ldots, w_M)$ with $v = w_m$. $L(v)$ tells us which of the F-segments of D exhibit the feature vector v. For our example $D = D_{\text{walk}}$ and $F = F^2$ we obtain from (1) the inverted lists $L(\binom{1}{1}) = \{1,3,5,7,9,11\}$, $L(\binom{0}{1}) = \{2,6,10\}$, $L(\binom{1}{0}) = \{4,8\}$, and $L(\binom{0}{0}) = \emptyset$. Inverted lists are sets represented as sorted, repetition-free sequences. Depending

on the context, our notation for inverted lists and derived objects will switch between sequence and set notation.

In a preprocessing step, we construct an index $I_F^{\mathcal{D}}$ consisting of the 2^f inverted lists $L(v)$, $v \in \{0,1\}^f$. Since we store segment positions of the F-segmentation rather than frame positions in the inverted lists, and since each segment position appears in exactly one inverted list, the index size is proportional to the number M of segments of D. Additionally, we store the segment lengths, so that the frame positions can be recovered. Furthermore, we keep the entries k within each list $L(v)$ sorted. This accounts for fast computations of unions and intersections of the inverted lists using merge operations or binary search, see also [3].

In Sect. 4.1 we will report on the running time behavior and memory requirements of our actual implementation. We then also discuss how to handle the case for feature functions with a large number of components by splitting up an index with a large number of lists into several indices consisting of a smaller number of lists.

3.2. Hits

Recall that two motion clips are considered as similar (with respect to the selected feature function) if they exhibit the same feature sequence. Adapting concepts from [3], we introduce the following notions. Let $\vec{w} = (w_0, w_1, \ldots, w_M)$ and $\vec{v} = (v_0, v_1, \ldots, v_N)$ be the feature sequences of the database and the query, respectively. Then an *exact hit* is an element $k \in [0:M]$ such that $\vec{v}$ is a subsequence of consecutive feature vectors in $\vec{w}$ starting from index k. In other words, writing in this case $\vec{v} \sqsubset_k \vec{w}$, one obtains

$$\vec{v} \sqsubset_k \vec{w} \;\; :\Leftrightarrow \;\; \forall i \in [0:N] : \; v_i = w_{k+i}. \tag{2}$$

The set of all exact hits in the database $\mathcal{D}$ is then given by

$$H_{\mathcal{D}}(\vec{v}) := \{k \in [0:M] \mid \vec{v} \sqsubset_k \vec{w}\}. \tag{3}$$

Obviously, $H_{\mathcal{D}}(\vec{v})$ can be evaluated very efficiently by intersecting suitably shifted inverted lists:

$$H_{\mathcal{D}}(\vec{v}) = \bigcap_{n \in [0:N]} (L(v_n) - n), \tag{4}$$

where the addition and substraction of a list and a number is understood component-wise for every element in the list. Instead of additively adjusting the inverted lists (which are in general long) one can also adjust the lists appearing as intermediate results (which are in general much shorter). From (4) it is easy to see that the following iterative algorithm, which will be extended in Sect. 3.3, computes the set of exact hits.

(1) $L^0 := L(v_0) + 1$
(2) $L^{n+1} := (L^n \cap L(v_{n+1})) + 1$ for $n = 0, \ldots, N-1$
(3) $H_{\mathcal{D}}(\vec{v}) = L^N - (N+1)$

As an illustration, we apply this algorithm to our running example $D = D_{\text{walk}}$ and $F = F^2$ and the query sequence $\vec{v} = (\binom{1}{0},\binom{1}{1},\binom{0}{1})$. Recall that in this case $L(\binom{1}{1}) = \{1,3,5,7,9,11\}$, $L(\binom{0}{1}) = \{2,6,10\}$, and $L(\binom{1}{0}) = \{4,8\}$.

(1) $L^0 := L(\binom{1}{0}) + 1 = \{5,9\}$
(2) $L^1 := (\{5,9\} \cap \{1,3,5,7,9,11\}) + 1 = \{6,10\}$
$L^2 := (\{6,10\} \cap \{2,6,10\}) + 1 = \{7,11\}$
(3) $H_{\mathcal{D}}(\vec{v}) = L^2 - 3 = \{4,8\}$

In other words, there are two exact hits for $\vec{v}$ starting with the fourth and eighth element of $\vec{w}$, respectively.

3.3. Fuzzy hits

In many situations, the user may be unsure about certain parts of the query and wants to leave certain parts of the query unspecified. Or, the user might want to mask out some of the f components of the feature function F to obtain a less restrictive search leading to more hits. To handle such situations, we introduce the concept of *fuzzy search*.

The idea is to allow at each position in the query sequence a whole set of possible, alternative feature vectors instead of a single one. This is modeled as follows. A *fuzzy set* is a subset $V \subset \{0,1\}^f$ with corresponding list $L(V) := \bigcup_{v \in V} L(v)$. Then a *fuzzy query* is defined to be a sequence $\vec{V} = (V_0, V_1, \ldots, V_N)$ of fuzzy sets such that $V_n \cap V_{n+1} = \emptyset$ for $0 \leq n < N$. (In case the latter intersection condition is not fulfilled, we iteratively conjoin adjacent sets with nonempty intersection until we end up with a sequence having the desired property. Note that this procedure corresponds to coarsening the segmentation of the query.) Extending the definition in (2), a *fuzzy hit* is an element $k \in [0:M]$ such that $\vec{V} \sqsubset_k \vec{w}$, where

$$\vec{V} \sqsubset_k \vec{w} \;\; :\Leftrightarrow \;\; \forall i \in [0:N] : \; V_i \ni w_{k+i}. \tag{5}$$

Obviously, the case $V_n = \{v_n\}$ for $0 \leq n \leq N$ reduces to the case of an exact hit. Similar to (3), the set of all fuzzy hits is defined to be

$$H_{\mathcal{D}}(\vec{V}) := \{k \in [0:M] \mid \vec{V} \sqsubset_k \vec{w}\}. \tag{6}$$

In analogy to (4), the set $H_{\mathcal{D}}(\vec{V})$ can be computed via

$$H_{\mathcal{D}}(\vec{V}) = \bigcap_{n \in [0:N]} (L(V_n) - n). \tag{7}$$

This formula shows that the complexity of computing $H_{\mathcal{D}}(\vec{V})$ is proportional to $\sum_{n=0}^{N} |V_n|$ and not to $\prod_{n=0}^{N} |V_n|$, as might be suspected in view of (5), see also [3].

3.4. Adaptive fuzzy hits

The concept of fuzzy search as introduced above is not yet exactly what we want: so far, the fuzziness only refers to the

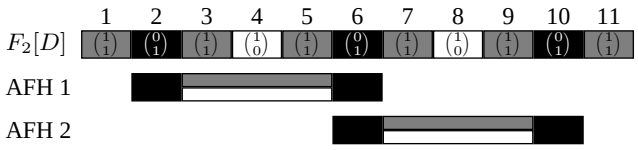

Fig. 6. Upper row: feature sequence $\vec{w} = F^2[D]$. Below: two adaptive fuzzy hits (AFH).

spatial domain (allowing alternative choices for the pose-based features) but ignores the temporal domain. More precisely, the segmentation of the document D is only determined by the feature function, disregarding the fuzziness of the fuzzy query $\vec{V}$. For example, considering the fuzzy query $\vec{V} = (V_0, V_1, V_2)$ with $V_0 = V_2 = \{\binom{0}{1}\}$ and $V_1 = \{\binom{1}{0},\binom{1}{1}\}$, one easily checks that $H_{\mathcal{D}}(\vec{V}) = \emptyset$.

To adjust the temporal segmentation to the fuzziness of the query, we proceed as follows: let $\vec{w} = (w_0, w_1, \ldots, w_M)$ be the feature sequence of D. Supposing $w_k \in V_0$ with $w_{k-1} \notin V_0$ for some index $k_0 := k \in [0 : M]$, we determine the maximal index $k_1 \geq k_0$ with $w_m \in V_0$ for all $m = k_0, k_0 + 1, \ldots, k_1 - 1$ and concatenate all segments corresponding to these w_m into one large segment. By construction, $w_{k_1} \notin V_0$. Only if $w_{k_1} \in V_1$, we proceed in the same way, determining some maximal index $k_2 > k_1$ with $w_m \in V_1$ for all $m = k_1, k_1 + 1, \ldots, k_2 - 1$, and so on. In case we find a sequence of indices $k_0 < k_1 < \ldots < k_N$ constructed iteratively in this fashion we say that $k \in [0 : M]$ is an *adaptive fuzzy hit* and write $\vec{V} \sqsubset_k^{\text{ad}} \vec{w}$. The set of all adaptive fuzzy hits is given by

$$H_{\mathcal{D}}^{\text{ad}}(\vec{V}) := \big\{k \in [0 : M] \mid \vec{V} \sqsubset_k^{\text{ad}} \vec{w}\big\}. \tag{8}$$

We continue our running example $D = D_{\text{walk}}$ and $F = F^2$ from Fig. 5. Let's once more consider the fuzzy query $\vec{V} = (V_0, V_1, V_2)$ with $V_0 = V_2 = \{\binom{0}{1}\}$ and $V_1 = \{\binom{1}{0},\binom{1}{1}\}$, then $L(V_0) = L(V_2) = (2, 6, 10)$ and $L(V_1) = (1, 3, 4, 5, 7, 8, 9, 11)$. Looking at the feature sequence $\vec{w}$ illustrated by Fig. 6, one easily derives that $H_{\mathcal{D}}^{\text{ad}}(\vec{V}) = \{2, 6\}$. That is, there are exactly two adaptive fuzzy hits in D, denoted by AFH 1 and AFH 2 in Fig. 6 (opposed to $H_{\mathcal{D}}(\vec{V}) = \emptyset$). We have $k_0 = 2$, $k_1 = 3$, $k_3 = 6$ for the first hit and $k_0 = 6$, $k_1 = 7$, $k_2 = 10$ for the second hit. In the case of the first hit, for example, this means that V_0 corresponds to segment 2 of $\vec{w}$, V_1 to segments 3–5, and V_2 to segment 6, amounting to a coarsened segmentation of D.

Now, the important point is that $H_{\mathcal{D}}^{\text{ad}}(\vec{V})$ can also be computed efficiently using the same index $I_F^{\mathcal{D}}$ as in the case of an exact hit. Before describing the algorithm, we need to introduce some more notations. Note that the list $L(V)$ for a fuzzy set V may contain consecutive segment indices (opposed to an inverted list $L(v)$). We consider sequences of consecutive segment indices in $L(V)$ having maximal length. Suppose $L(V)$ consists of $K + 1$ such sequences with starting segments $r_0 < r_1 < \ldots < r_K$ and lengths $t_0, t_1, \ldots, t_K$; then we define $R(V) := (r_0, r_1, \ldots, r_K)$ and $T(V) := (t_0, t_1, \ldots, t_K)$. For example, if $L(V) = (2, 4, 5, 6, 9, 10)$ then $R(V) = (2, 4, 9)$ and $T(V) = (1, 3, 2)$. Obviously, one can reconstruct $L(V)$ from $R(V)$ and $T(V)$. Note that by the maximality condition one has $r_k + t_k < r_{k+1}$ for $0 \leq k < K$. As usual, let $\vec{w} = (w_0, w_1, \ldots, w_M)$ correspond to the database and $\vec{v} = (v_0, v_1, \ldots, v_N)$ to the query. Then, extending the algorithm in Sect. 3.2, we can compute the set $H_{\mathcal{D}}^{\text{ad}}(\vec{V})$ as follows:

(1) $R^0 := R(V_0) + T(V_0)$, $T^0 := T(V_0)$
(2) Suppose R^n and T^n are known, $n \in [0 : N - 1]$.
Assume $R^n = (p_0, \ldots, p_I)$, $T^n = (q_0, \ldots, q_I)$,
$R(V_{n+1}) = (r_0, \ldots, r_J)$, $T(V_{n+1}) = (t_0, \ldots, t_J)$,
$R^n \cap R(V_{n+1}) = (p_{i_0}, \ldots, p_{i_K}) = (r_{j_0}, \ldots, r_{j_K})$
for suitable indices $0 \leq i_0 < \ldots < i_K \leq I$ and
$0 \leq j_0 < \ldots < j_K \leq J$. Then set
$R^{n+1} := (R^n \cap R(V_{n+1})) + (t_{j_0}, \ldots, t_{j_K})$
$T^{n+1} := (q_{i_0}, \ldots, q_{i_K}) + (t_{j_0}, \ldots, t_{j_K})$
(3) $H_{\mathcal{D}}^{\text{ad}}(\vec{V}) = R^N - T^N$

To illustrate the algorithm, we continue our running example. From the lists $L(V_0) = L(V_2) = (2, 6, 10)$ and $L(V_1) = (1, 3, 4, 5, 7, 8, 9, 11)$, we obtain $R(V_0) = R(V_2) = (2, 6, 10)$, $T(V_0) = T(V_2) = (1, 1, 1)$ and $R(V_1) = (1, 3, 7, 11)$, $T(V_1) = (1, 3, 3, 1)$. Then

(1) $R^0 := (2, 6, 10) + (1, 1, 1) = (3, 7, 11)$
$T^0 = (1, 1, 1)$
(2) $R^1 := ((3, 7, 11) \cap (1, 3, 7, 11)) + (3, 3, 1)$
$= (6, 10, 12)$
$T^1 := (1, 1, 1) + (3, 3, 1) = (4, 4, 1)$
$R^2 := ((6, 10, 12) \cap (2, 6, 10)) + (1, 1) = (7, 11)$
$T^2 := (4, 4) + (1, 1) = (5, 5)$
(3) $H_{\mathcal{D}}^{\text{ad}}(\vec{v}) = R^2 - T^2 = (2, 6)$.

Finally, we want to note that fuzzy search can be complemented by the concept of *m-mismatch search*, see [3]. Here, one introduces another degree of inexactness by permitting up to $m < N$ of the fuzzy sets in a query $\vec{V} = (V_0, V_1, \ldots, V_N)$ to completely disagree with the database sequence $\vec{w}$. To maintain a certain degree of control over this mismatch mechanism, it is possible to restrict the mismatchable fuzzy sets within $\vec{V}$.

4. EXPERIMENTS AND RESULTS

We implemented our indexing and retrieval algorithms in Matlab 6 and tested them on two databases $\mathcal{D}^{180}$ and $\mathcal{D}^{60}$ containing roughly 1,300,000/430,000 frames of motion capture data, respectively (180/60 minutes sampled at 120

Hz). The experiments were run on a 3.6 GHz Pentium 4 with 1 GB of main memory.

4.1. Indexing

Recall that our retrieval algorithms are based on merging and intersecting operations of suitable inverted lists. In view of efficiency, it is important to have few lists (leading to few merging and intersecting operations), but at the same time it is important to have short lists (leading to fast merging and intersecting operations), which are mutually exclusive demands. In our experiments, it turned out that a choice of $f \in [8:12]$ results in a good tradeoff between these two requirements. Since it is generally desirable to work with larger numbers of features, we suitably divide the feature set into several groups of 8–12 features and build separate indexes for each of these groups. Queries involving features scattered across several indexes are then processed by querying each respective index to yield intermediary results, which are finally intersected to obtain the desired hits.

In our experiments, we considered 31 boolean features divided into three feature functions F_{lower}, F_{upper}, and F_{mix} consisting of 11, 12, and 8 components, respectively. The feature functions F_{lower} and F_{upper} comprised boolean features expressing properties of the lower/upper part of the body (mainly of the legs/arms, respectively) as described in Sect. 2, whereas F_{mix} consisted of boolean features expressing interactions of the upper and lower part as well as global parameters such as velocities.

The total size of $\mathcal{D}^{180}$ represented in a compact binary double precision format was about 370 MB. Index sizes are linear in the number of segments extracted from the database: for example, the F_{lower}-index for $\mathcal{D}^{60}$ contained roughly 21,000 segments, requiring 0.72 MB of storage, whereas the corresponding index for $\mathcal{D}^{180}$ comprised about twice as many segments (41,000), requiring 1.41 MB. This also shows the drastic amount of data reduction our scheme achieves. The total indexing time is linear in the number of frames, for example, it took 42 seconds to build up the F_{lower}-index for $\mathcal{D}^{60}$, which is roughly one third of the 110 seconds that were needed to build up the index for $\mathcal{D}^{180}$. For both indexes, more than half of the total indexing time was spent on reading in the data. The scalability of our algorithms' running time and memory requirements permits us to use much larger databases than those treated in [7], where the preprocessing step to build up a match web is quadratic in the number of frames (leading, e.g., to a running time of roughly 3,000 seconds for a database containing only 37,000 frames).

4.2. Retrieval

The running time to process a query very much depends on the query length (the number of segments), the respective index, as well as the number of resulting hits. In an experiment, we posed 10,000 random queries (guaranteed to yield at least one hit) for each of six query scenarios to the index I^{180}_{upper}, see Table 1. For example, finding all adaptive fuzzy F_{upper}-hits for a query consisting of $5/10/20$ segments, where each set V_n of alternatives had a size of $|V_n| = 64$, took on average 23–291/28–281/42–294 ms.

Query type, #(segs)	1–9 hits			10–99 hits			$\geq$ 100 hits		
	μ_h	σ_h	μ_t	μ_h	σ_h	μ_t	μ_h	σ_h	μ_t
exact, $\|Q\| = 5$	3.0	2.4	16	44	28	20	649	567	144
exact, $\|Q\| = 10$	1.7	1.6	17	34	22	26	239	147	71
exact, $\|Q\| = 20$	1.1	0.6	19	32	26	36	130	5	52
ad. fuzzy, $\|Q\| = 5$	3.6	2.5	23	44	27	29	1,878	1,101	291
ad. fuzzy, $\|Q\| = 10$	2.4	2.1	28	40	26	35	1,814	1,149	281
ad. fuzzy, $\|Q\| = 20$	2.0	1.9	42	36	24	35	1,908	1,152	294

Tab. 1. Statistics on 10,000 random queries in the F_{upper}-index for $\mathcal{D}^{180}$ for different query modes and lengths, grouped by the hit count, h. μ_h and σ_h are the average/standard deviation of h for the respective group, μ_t is the average query time in milliseconds.

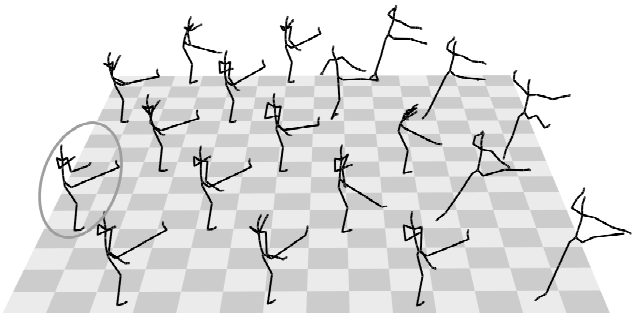

Fig. 7. Selected frames from 19 query-by-example hits for a right foot kick. The query clip is highlighted.

Fig. 8 shows 9 adaptive fuzzy hits for a "squatting" motion (retrieval time: 18 ms), and Fig. 7 shows 19 adaptive fuzzy hits for a "kicking" motion (retrieval time: 5 ms), 13 of which are actual martial arts kicks. The remaining six motions (right hand side) are ballet moves containing a kicking component. A manual inspection of $\mathcal{D}^{180}$ showed that there were no more than the 13 reported kicks in the database. Similar findings hold for other retrieval experiments, including that of Fig. 8, which demonstrates the high recall percentage our technique achieves. As for retrieval precision, we specifically designed our features' offset and threshold values to yield high acceptance rates. The two skeletons to the right of Fig. 8 "sitting down" on a virtual table edge illustrate this fact very well: the relevant feature used in the query for the squatting motion thresholds the knee angle against a relatively high decision value of 120°. Hence, the knees of the sitting skeletons were classified as "bent", leading to the confusion with a squatting motion. A further source of false positive hits is of course the choice of fuzzy alternatives in a query: the ballet jumps in Fig. 7 were found as matches for a kicking motion because only the right leg was constrained by the query, leaving the left

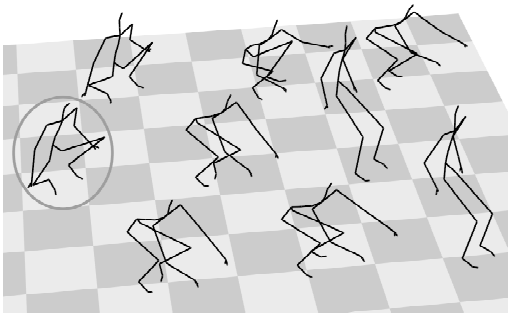

Fig. 8. Selected frames from 9 query-by-example hits for a squatting motion. The query clip is highlighted.

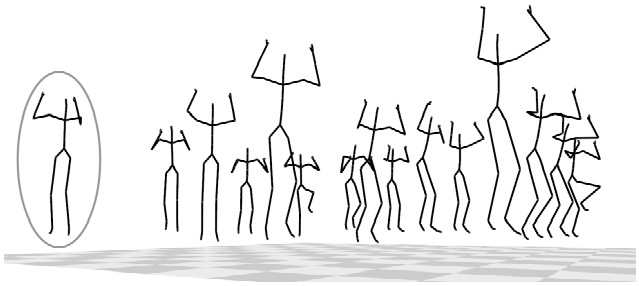

Fig. 9. Selected frames from 15 adaptive fuzzy hits for a jump.

leg free to be stretched behind the body. Fig. 9 shows the top 15 out of 133 hits for a very coarse adaptive fuzzy "jumping" query, which basically required the arms to move up above the neck and back down, while forcing the feet to lift off. The hits were ranked according to a simple strategy based on a comparison of segment lengths. We omitted hits no. 16–30, which also contained some jumps, for space reasons. This example demonstrates how such coarse queries can be applied to efficiently reduce the search space, while retaining a superset of the desired hits.

5. FUTURE WORK

In this paper, we showed that simple geometric relations in combination with adaptive temporal segmentation are a powerful tool for content-based motion retrieval. For the future, we plan to use this method to efficiently provide coarse pre-alignments of logically similar motions, which can then be refined by more involved DTW-based techniques, significantly accelerating the overall alignment procedure. As first experiments showed, geometric features also seem to be a promising tool in handling low-level descriptive queries automatically. Here the idea is that motions can often be easily characterized by a typical progression of geometric constellations corresponding to key poses. For the future, we plan to employ statistical methods to learn such progressions as well as to automatically identify expressive geometric features from typical example motions.

Acknowledgements: Some of the data used in this project was obtained from `mocap.cs.cmu.edu`, which was created with funding from NSF EIA-0196217. Some of the data was obtained from Modern Uprising Studios. Tido Röder is supported by the German National Academic Foundation.

6. REFERENCES

[1] A. Bruderlin and L. Williams. Motion signal processing. *Computer Graphics*, 29:97–104, 1995.

[2] S. Carlsson. Combinatorial geometry for shape representation and indexing. In *Object Representation in Computer Vision*, pages 53–78, 1996.

[3] M. Clausen and F. Kurth. A unified approach to content-based and fault tolerant music recognition. *IEEE Trans. Multimedia*, 6(5):717–731, 2004.

[4] CMU. Carnegie-Mellon MoCap Database. `http://mocap.cs.cmu.edu`.

[5] M. Giese and T. Poggio. Morphable models for the analysis and synthesis of complex motion patterns. *IJCV*, 38(1):59–73, 2000.

[6] 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.

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

[8] M. Müller, T. Röder, and M. Clausen. Efficient content-based retrieval of motion capture data. Submitted for publication, 2005.

[9] K. Pullen and C. Bregler. Motion capture assisted animation: Texturing and synthesis. In *Proc. ACM SIGGRAPH 2002*. ACM Press, 2002.

[10] J. Sullivan and S. Carlsson. Recognizing and tracking human action. In *ECCV '02: Proc. 7th European Conf. on Computer Vision—Part I*, pages 629–644. Springer, 2002.

[11] Vicon. 3D Optical Motion Capture. `http://www.vicon.com`.

[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.*, 2003.