An Eigen-based motion retrieval method for real-time animation

Pengjie Wang\textsuperscript{a,b,}\textsuperscript{*}, Rynson W.H. Lau\textsuperscript{c}, Zhigeng Pan\textsuperscript{d}, Jiang Wang\textsuperscript{c}, Haiyu Song\textsuperscript{a}

\textsuperscript{a} College of Computer Science \& Engineering, Dalian Nationalities University, Dalian 116600, China
\textsuperscript{b} State Key Laboratory of CAD \& CG, Zhejiang University, Hangzhou 310027, China
\textsuperscript{c} Department of Computer Science, City University of Hong Kong, Hong Kong, China
\textsuperscript{d} Digital Media and HCI Research Center, Hangzhou Normal University, Hangzhou 310023, China

\textbf{Article history:}
Received 5 August 2013
Received in revised form 7 November 2013
Accepted 7 November 2013
Available online 18 November 2013

\textbf{Keywords:}
Eigenspace retrieval
Motion retrieval
Real-time animation
Real-time motion retrieval

\textbf{Abstract}
Research on real-time 3D animation is attracting a lot of attention in recent years due to the popularity of emerging applications such as distributed virtual environments and computer games. One of the important issues in real-time animation is that the existing motion retrieval techniques generally have a high matching time because they are typically based on matching time-series, making them less suitable for use with large motion databases. In this paper, we propose a different approach to motion retrieval, called Eigen-based Motion Retrieval (or EigenMR), to address this limitation of the existing methods by performing motion retrieval in the transform domain instead of the time domain. To differentiate the motion of different body parts, we propose to perform the matching on individual body parts as well as on the whole body. Our approach has the important advantage that each body part can be represented by an index of fixed size, consisting of a number of eigenvectors and the corresponding eigenvalues. As a result, our approach has constant time complexity based on the number of motion files in the database instead of the size of the database. The experimental results show that our approach is both efficient and accurate compared with some of the latest methods. When applied to a motion database of 4 GB in size, our method requires approximately 20% of the standard time, making it more suitable for real-time animation.

\textbf{1. Introduction}

Motion capture data have been widely used in 3D animation. Due to the popularity of real-time 3D applications in recent years, such as second life and computer games, research on real-time data-driven animation is attracting a lot of attention. Data-driven animation focuses on how to use pre-recorded motions in the database to do example-based motion synthesis. For example, in an online game, we may need to search for similar motions from a database, taking the current action as the input motion. Based on the retrieved motions, we may construct a continuous, parameterized space of motion to provide accurate and efficient control [19,28]. Another example is generating reactive motion where we need to stitch two motions together into a continuous motion with a smooth transition [18]. A solution to this latter example is to retrieve motions from the database that are similar to both motions to be stitched. The best matched motion is then edited and used to provide a transition between the two motions. Hence, the performance of a motion retrieval algorithm, in terms of both computation time and matching accuracy, is important for real-time 3D applications.

Conversely, as users of 3D applications expect increasing motion realism, the motion database needs to be increased to accommodate more diverse types of motion. However, with the increase in the size of the motion database, a more efficient motion retrieval algorithm is needed.

Existing motion retrieval methods can be roughly classified into DTW (Dynamic Time Warping)-based methods [8,14,26,36], the uniform-scaling-based method [17], and geometry-based string matching methods [12,29,30]. Because these methods are based on time-series matching, the memory cost in storing the motion indices and the computational cost in matching the motions are proportional to the size of the motion file.

In this paper, we present a different approach to motion retrieval. Instead of performing the matching process in the time domain, we perform the matching process in eigenspaces that are transformed from the dimension-reduced feature matrices. Our idea is based on the observation that similar motions have nearly linear correlated eigenvectors. Fig. 2 shows the primary eigenvectors of four types of motion: running (blue), walking (red), jumping (green) and kicking (yellow). We can see that for the same type of motion, the primary eigenvectors are very similar. In [2], it is observed that different actions within a single motion have different eigenspaces that can be used to assist with motion segmentation. This observation agrees with our finding in this paper that different types of motion have different eigen-patterns. Hence, we may take advantage of these eigen-patterns to
discriminate different types of motion. In addition, we have also introduced some measures to ensure that similar motions obtain high matching scores, including six sets of features in two hierarchical levels, a length normalization process, and a different approach to computing the matching scores.

Because our matching process is based on comparing a fixed number of eigenvectors and eigenvalues, it is much more efficient in terms of both matching speed and index memory cost. In addition, as the transform-domain eigenvectors are able to capture the significant features of a complete motion, our approach can obtain logically similar matching results independent of the size of the motion files. Note that in this work, we refer to “logically similar motions” as the motions that share the same action goal while having some variations. Here, variations may be in the spatial as well as the temporal domains [19,29,12]. For example, a basketball dribbling motion may have variants such as forward dribbling and sideways dribbling, and a raising arm motion may have variants such as raising the two arms alternatively or raising them together. They are all referred to as logically similar motions.

Fig. 1 shows a summary of our approach. EigenMR is composed of a pre-processing stage and a retrieval stage. The pre-processing stage computes the motion index of each motion file in the database. It has three steps. First, we define six sets of features. One set is for the features of the whole body, and five sets are for the local features of five body parts. This step converts the original motion into six sets of feature matrices, each with columns representing different positions and rows representing different frames. Second, we construct a covariance matrix from each feature matrix. Third, we obtain the eigenvectors and corresponding eigenvalues from the covariance matrix. Here, we only take the most significant ones to form the motion index for matching. Note that each set of features forms a single index called the subindex, and the six subindices together form an index for the motion file.

The runtime motion retrieval stage has two passes. The first pass computes the matching score of the subindices for the whole body between the query motion and the database motion. The second pass then computes a single match score for the six pairs of subindices between the query motion and the database motion.
Here, we use the correlation coefficients of the eigenvectors weighted by the corresponding eigenvalues to evaluate the matching score of a pair of subindices.

The main contributions of our work include the following:

- We propose a method to compare the similarity of two motions in the transform domain (eigenspaces), instead of the time domain. The resulting motion index, while being efficient in matching speed and memory cost, captures the significant and holistic features of the motion data well.
- We propose an efficient method to compute the matching score of two motions based on eigenvector correlation.

The rest of this paper is organized as follows: Section 2 summarizes related work on motion retrieval; Section 3 presents our method for pre-processing the database motions; and Section 4 presents our method for runtime retrieval of database motions. Section 5 extensively evaluates the performance of the proposed motion retrieval method through a number of experiments. Finally, Section 6 briefly concludes this work.

2. Related work

A number of projects have been conducted to reuse human motion data in various ways [32]. They include constructing and traversing motion graphs [1,5,22], interpolating between motion clips that are visually similar and correspond to the same logical action [15,19], projecting motion into a low dimensional space for editing and compression [39], learning statistical dynamic models from human motion [6,24], modeling interactive 3D facial expressions [21] and simulating biped behaviors [23].

Automatically searching for similar motions is of great importance in motion synthesis and reuse. Methods for efficiently and accurately selecting and retrieving similar motions from a large motion database are a challenging problem. In this section, we briefly summarize existing motion retrieval methods that are based on both matching time-series (Section 2.1) and matching in a reduced space (Section 2.2).

2.1. Motion retrieval based on matching time-series

Because motion data are multi-attribute time-series, it is straightforward to apply dynamic time warping (DTW) [8,14,26,31,36], uniform scaling [17] or the Smith–Waterman algorithm [41] for motion retrieval. However, these methods are based on numerical comparison and thus may not be time-efficient. In addition, as logically similar motions may not necessarily be numerically similar [19], methods based on DTW or uniform scaling may not always have high retrieval accuracy because their similarity measures are based on numerical comparison in the spatial domain.

Chiu et al. [8] apply the self-organizing map (SOM) to cluster the rotation angles of each joint separately. The start and end points of the query motion are searched first. The candidate segments are then compared with the query motion using DTW to obtain the final result. Sakamoto et al. [35] propose an image-based user interface for retrieving motion data using a SOM for indexing. After training the SOM with motion data, motion clips are indexed using the SOM nodes. Retrieval can then be performed by searching a well-defined hash table using node-based comparison. Wu et al. [41] divide the human character model into three partitions to reduce the spatial complexity and measure the temporal similarity of each partition by a SOM and the Smith–Waterman algorithm. The overall similarity between two motion clips can be achieved by integrating the similarities of individual body partitions. However, all of these SOM-based methods suffer from high computational cost because of the expensive SOM construction process. Conversely, our method has a low pre-processing time, i.e., one that is linearly proportional to the number of motion files in the database.

Kovar and Gleicher [19] propose a logical retrieval method based on numerical matching by pre-computing the “match webs” to describe potential matches between motion subsequences. An iterative search that spawns a new query for each match in previous queries helps identify logically similar motions using numerically similar motions as intermediaries. However, this method has a pre-processing time that is quadratic to the size of the database. In addition, if a motion query is not already in the database, this method needs to compute the match webs between the input query and all of the motions in the database, which can be very time consuming for large databases. On the contrary, our method only needs to compute an index for each input query for similarity comparison.

Deng et al. [12] present an example-based logical retrieval method by decomposing motions into body parts and extracting common motion patterns. A motion clip can be represented as a string of pattern indices, and motion retrieval becomes simple string matching. This method achieves accurate and perceptual-consistent retrieval. However, the data structures use amount to nearly 5% of the size of the motion database and may be expensive for large databases. In addition, this method requires executing a k-means process to cluster motions for retrieval, which is computationally expensive. On the contrary, our method is extremely efficient and does not have this pre-processing requirement.

Muller et al. [29] propose an efficient semantics-based motion retrieval method, where the user inputs a query motion as a set of time-varying geometric feature relationships. It essentially transforms the motion retrieval problem into a binary matching problem in a geometric relationship space. However, specifying well-defined geometric (semantic) relationships for highly dynamic motions (e.g., boxing) is non-trivial, especially for novice users. Our method does not require the user to specify semantic information. We automatically extract the significant holistic features from the input motions to form the motion indices.

Tang et al. [37] propose a motion similarity measure to emulate the human perception by considering the joint relative distance as features to be matched. Tang and Leung [38] follow up on this work and propose to select different features for different input queries in an adaptive manner and update the weighting of the features during the relevance feedback. However, as these methods do not consider subspaces, the feature size can be very large.

Chao et al. [7] present a motion retrieval method based on hand-drawn sketches. They introduce a novel hierarchical encoding scheme using a set of orthonormal spherical harmonic basis functions on the trajectories of the important joints. The matching process is a multi-pass process and can retrieve logically similar motions. However, this method requires the user to be familiar with the motions in the database, which may be difficult in practice.

Recently, Kapadia et al. [16] propose a motion retrieval method for large databases. They define a set of motion features that represent a wide array of structural, geometric and dynamic features of human motions and a key to describe each feature. The motions are then encoded as keys. A trie-based data structure is organized to provide efficient mapping from user input key sequences to motions. Based on a specially designed key trie structure, the method can achieve fast matching independent of the database size. Ho et al. [13] explore retrieving motions of characters in close contact. They define rational tangles to distinguish the relationship of two characters. Matching two motions thus becomes matching the tangles of the two motions.
2.2. Motion retrieval in a reduced space

A popular way to reduce the motion retrieval time is to transform the original high-dimensional motion data to a lower dimension representation and conduct matching in the reduced space. Keogh et al. [17] introduce the concept of bounding envelopes to accelerate similarity search under uniform scaling. Muller et al. [29] convert motion sequences into binary geometry feature strings for matching. Deng et al. [Deng05] first identify the motion patterns of each motion clip and then convert the clip into a string of pattern indices. Zhu et al. [42] propose a quaternion space sparse decomposition model that decomposes each motion into a dictionary and some weights. Motion retrieval is accomplished by computing the distance between the dictionaries of the query motion and of the motion in the database. In [8,35,41], a SOM is used to convert the nonlinear high-dimensional human motion data into a linear low-dimensional space. Krug et al. [20] propose a compact feature set to represent the original motion. It is composed of the positions of four end-effectors and the head of the human body. Based on this feature, local neighborhood search and global similarity search can be efficiently achieved using a k-nearest neighbor search. Liu et al. [27] propose index motion sequences by clustering every pose in the database into groups that are represented as simple linear models. Each motion sequence can then be represented as the trajectory of the pose clusters through which this motion sequence passes. The output feature set has a much lower dimension. Liu et al. [26] adopt a key-frame extraction algorithm to reduce the number of motion frames. Motion matching is then performed on the reduced postures using DTW. Lin [25] proposes a geometric feature-based indexing method to pre-process the database by partitioning motions into segments and clustering similar motions. Motion retrieval is then performed by comparing the peaks of the motion curves. Choi et al. [9] propose a motion retrieval and visualization method for motion data based on user-drawn stick figures. To match motions, they convert each motion into a sequence of stick figures, and the matching process is performed on the reduced 2D relative joint positions. However, they need to build stick figure sequences for each motion segment, which makes it difficult to apply their method to large motion databases.

In addition to the above methods, several studies have achieved motion matching or retrieval in the reduced eigenspace. Forbes and Fiume [14] project motion sequences into a weighted PCA space where the weights are either user-defined or based on the significances of different body parts. They then apply DTW to measure the similarity between two motion sequences. Choi et al. [10] reduce the number of dimensions of motion data by PCA and use the projected data to do motion clustering. They then assign each frame to an individual cluster. By constructing a trained dictionary from the indexed motion data, they employ SVM to identify the input query motion. Pradhan and Prabhakaran [33] use SVD (Singular Value Decomposition) to map the multi-dimensional motion data onto a reduced feature space that is defined on the computed feature components for the corresponding 3D joint matrices. They then construct a hierarchical indexing structure on body segments. A query becomes a traversal of the indexing structure for best matches. This method supports queries on both sub-body and whole-body motions. In [34], PCA is proposed for motion trajectory retrieval from video sequences. The method segments each motion trajectory into sub-trajectories and then uses PCA to transform these sub-trajectories into a reduced dimension subspace spanned by the data-dependent orthogonal bases. Matching is performed in this low-dimensional subspace.

Matching or recognition of other geometry data in a lower dimensional space is also popular. For example, Turk and Pentland [40] propose to recognize faces in the eigenspace by projecting them onto a dimension-reduced space for recognition. Readers may refer to [34] for a comprehensive study on this.

In summary, the motion retrieval methods discussed above perform motion matching in a dimension-reduced space. Hence, they still belong to the category of time-series matching methods. While [12,27,25] try to cluster the motions and convert them into cluster indices, the SOM-based methods [8,35,41] project the nonlinear motions into a dimension-reduced linear space. [25,26] improve the efficiency of motion matching by reducing the number of postures that need to be matched. The eigenspace-based methods [14,10,33,4,40] project the motions into a PCA space before motion matching in the dimension-reduced eigenspace.

Our method is fundamentally different from all of the above methods in that our method conducts the comparison directly on the coordinate axes of the eigenspace instead of the projected data. Therefore, our method is a transform domain method instead of a time domain method. Directly comparing the coordinate axes may have the problem that different types of motion may produce the same or similar coordinate axes, degrading the retrieval accuracy. Therefore, we also propose to compare the eigenspace of each of the five body parts individually to minimize mismatches. The overall matching score is a weighted combination of the six matching scores derived from the six feature sets (the whole body and the five body parts). Our experimental results show that the retrieval accuracy of our method is generally better than that of some popular methods and that our retrieval time is significantly lower.

3. EigenMR pre-processing

In this section, we present our method for pre-processing the motion data for retrieval in EigenMR. We first define six feature sets and then discuss the motion index extraction process.

3.1. Feature definition

As shown in Fig. 3, the six sets of features are organized as a two-layer hierarchy, where the first layer is the feature set for the whole body and the second layer contains five feature sets representing the five body parts (four limbs and the head-torso). Because the original joint angular representation of motion data is in a high dimensional space, we reduce the dimension of the whole body features. Here, we define our geometry features for the whole body based on [20], where they define the normalized positions of the four limbs and the head as the motion features, because the end-effectors of a motion reflect more significant aspects of the overall motion of the character. In addition to the feature set for the whole body, we define five feature sets for the five body parts in the second layer. These features are also defined in the normalized position representation, where four feature sets correspond to the joints of four limbs and one feature set corresponds to the head and torso. The detailed joint segmentation is shown in Fig. 4.
3.2. Motion index extraction

In EigenMR, we first compute six sets of eigenvectors and eigenvalues for each motion in the motion database in a preprocessing step. These six sets of eigenvectors and eigenvalues are then stored as an index to the corresponding motion file.

We define the feature vector as

\[ F = (f_1, f_2, f_3, ..., f_d) \]

where \( f_1, f_2, ..., f_d \) are the feature vectors of one of the six feature sets defined in Section 3.1 and \( d \) is the total number of feature vectors. Each feature \( i \) can be written as \( f_i = (f_{i1}, f_{i2}, ..., f_{in})^T \), with \( n \) being the number of frames in the motion.

Then, we define a mean vector:

\[ \bar{F} = (\bar{f}_1, \bar{f}_2, \bar{f}_3, ..., \bar{f}_d) \]

where \( \bar{f}_i = (1/n) \sum_{j=1}^{n} f_{ij} \) and a difference vector:

\[ \delta F = F - \bar{F} = (\delta f_1, \delta f_2, \delta f_3, ..., \delta f_d) \]

We compute the covariance matrix of \( \delta F \) as follows:

\[ R = (r_{jk})_{d \times d} \]

where

\[ r_{jk} = \frac{1}{n} \sum_{i=1}^{n} \delta f_{ij} \delta f_{ik} \]

We then obtain \( s \) eigenvalues of \( R \), with \( \lambda_1 \geq \lambda_2 \geq ... \geq \lambda_s \geq 0 \), and the corresponding eigenvectors, \( v_1, v_2, ..., v_s \), where \( s \) is the rank of \( R \). The weight of an eigenvalue is defined as

\[ \alpha_i = \frac{\lambda_i}{\sum_{k=1}^{s} \lambda_k} \]

We maintain the leading \( m (m \leq s) \) largest eigenvalues until the accumulated weight is higher than a threshold \( AccThr \), where \( AccThr \leq 1 \). We refer to \( m \) as the length of the motion index.

Then, the final eigenvector matrix becomes

\[ V = (v_1, v_2, ..., v_m) \]

There are six sets of \( V \) and the corresponding eigenvalues. We store these sets and values as the index of this motion.

4. Runtime motion retrieval

During runtime, we compare the matching score between the query motion and each motion in the database. In this section, we present our runtime matching algorithm given an input motion and a motion in the database. Both motions are pre-processed based on the method discussed in Section 3.2.

4.1. Basic correlation coefficients

To compare an input query motion, \( V_{Q} \), with a motion in the database, \( V_{D} \), we compute the correlation coefficient of each of the six sets of “patterns” (eigenvectors and eigenvalues) of \( V_{Q} \) and \( V_{D} \). We refer to each set of patterns as a subindex. Hence, there are a total of six subindices for each motion file. Referring to Eq. (7), each subindex in \( V_{Q} \) or \( V_{D} \) can be defined as follows:

\[ V_{Q} = (v_{q1}, v_{q2}, ..., v_{qm}) \]

\[ V_{D} = (v_{d1}, v_{d2}, ..., v_{dm}) \]

The covariance between vectors \( v_{qi} \) and \( v_{dj} \) is

\[ \text{cov}(v_{qi}, v_{dj}) = \text{E}(v_{qi})\text{E}(v_{dj}) - \text{E}(v_{qi})\text{E}(v_{dj}) \]

where \( \text{E}(X) \) is the expectation of vector \( X \).

We then define the correlation coefficient as

\[ \rho_{ij} = \frac{\text{cov}(v_{qi}, v_{dj})}{\sqrt{\text{D}(v_{qi})\text{D}(v_{dj})}} \]

where \( \text{D}(X) \) is the variance of vector \( X \), \( \text{D}(v_{qi}) \neq 0 \), and \( \text{D}(v_{dj}) \neq 0 \).

4.2. Length normalization

In Section 4.1, we assume that the corresponding subindices in \( V_{D} \) and \( V_{Q} \) have the same length. However, because we take the leading largest eigenvalues one by one from a subindex until the accumulated weight is higher than \( AccThr \) (Eq. (6)), the resulting subindex lengths may be different for different motions. Hence, we need to normalize their lengths. We let \( \text{Length}_{Q} \) and \( \text{Length}_{D} \) be the subindex lengths of the query and database motions, respectively. We perform the following operations:

- Compute: \( \text{diff} = \text{Length}_{Q} - \text{Length}_{D} \).
- If \( \text{diff} \) is equal to 0, no further operations are needed.
- If \( \text{diff} \) is larger than a threshold, \( \text{Length}_{Thr} \), we assume that the two motions are different and no further operations are needed.
- Equalize \( \text{Length}_{Q} \) and \( \text{Length}_{D} \) by adding more eigenvectors to \( V_{Q} \) or \( V_{D} \).

4.3. Matching score computation

Let \( \alpha_1, ..., \alpha_m \) be the weights of the leading eigenvalues of a subindex, as defined in Eq. (6), where \( m \) is the subindex length after length normalization. We perform the following steps to obtain the weighted correlation coefficient:

- For each \( v_{di} \) in \( V_{D} \), we find the best match \( v_{qi} \) from \( V_{Q} \). Here, the best match refers to the \( v_{qi} \) that produces the highest correlation coefficient with \( v_{di} \) according to Eq. (11).
- Remove \( v_{qi} \) from \( V_{Q} \).
- Set \( \text{MatchQ}[i] = s \), where \( \text{MatchQ}[i] \) is an array indicating the best matches of \( V_{Q} \) in \( V_{D} \).
- Repeat the above steps until all vectors in \( V_{D} \) and \( V_{Q} \) are matched.
Now, the weighted correlation coefficient can be computed as

\[
\rho = \frac{\sum_{i=1}^{m} \alpha_i \rho_i \cdot \text{MatchQ}_i}{\sum_{i=1}^{m} \alpha_i}
\]  

(12)

Then, we obtain the final \(\rho_Q\) as follows (note that we set \(\rho\) to zero to punish it if it is smaller than the threshold, \(\text{PunishThr}\)):

\[
\rho_Q = \begin{cases} \rho, & \text{when } \rho \geq \text{PunishThr} \\ 0, & \text{when } \rho < \text{PunishThr} \end{cases}
\]  

(13)

We then swap \(V_D\) and \(V_Q\) and repeat the whole process. This should give us another \(\rho\), referred to as \(\rho_D\). The final matching score for this subindex is the maximum of \(\rho_Q\) and \(\rho_D\).

4.4. The final matching process

From Section 4.3, we have computed six matching scores for the six subindices according to Eq. (12). Based on these scores, the final matching process is shown in Fig. 5.

The runtime retrieval operation is a two-pass comparison. During the first pass, the matching score for the whole body, referred to as \(\rho_{\text{pass1}}\), is computed according to Eq. (12). We set a threshold, \(\rho_{\text{pass1}}\), to filter out all motions in the database that are unlikely to be similar. In the second pass, we compute another matching score for each of the remaining motions as follows:

\[
\rho_{\text{pass2}} = w_{\text{whole}}\rho_{\text{whole}} + w_{\text{RightArm}}\rho_{\text{RightArm}} + w_{\text{LeftArm}}\rho_{\text{LeftArm}} + w_{\text{RightLeg}}\rho_{\text{RightLeg}} + w_{\text{LeftLeg}}\rho_{\text{LeftLeg}}
\]  

(14)

Here, we obtain each \(\rho\) in the RHS of Eq. (14) from Section 4.3. \(\rho_{\text{pass2}}\) is the weighted sum of all \(\rho\) values. Because the matching score for the whole body captures the global motion similarity, it should weigh more in the final matching score. Conversely, because the matching score for a body part describes the similarity of that part, we may set different weights for different body parts when retrieving some specific types of motion. For example, if our focus is a kicking motion, we should give a higher weight to the legs. Here, in our experiments, we only consider general queries and set all body parts to have the same weight. Note that the sum of the six weights should be 1.

In the second pass, we set another threshold, \(\rho_{\text{pass2Thr}}\), to further filter the result set derived from the first pass. The final result set includes those motions with matching scores that are higher than the two pre-defined thresholds.

5. Experimental results

We have implemented the proposed algorithm in Visual C++ 8.0. To evaluate the performance of the proposed method, we have performed a number of experiments on a PC with an Intel Duo Core 2 GHz CPU and 2 GB RAM.

5.1. Computational and memory Costs

To examine the computation times and storage cost of EigenMR, we have tested it on the five database sizes shown in Table 1. The first and second databases are from CMU [11]. The other three databases are created by randomly duplicating some of the motion files in the first and second databases for the purpose of comparing the processing time.

### Table 1

<table>
<thead>
<tr>
<th>Database size (MB)</th>
<th>Index size (MB)</th>
<th>(t_1) (s)</th>
<th>(t_2) (s)</th>
<th>(t_3) (s)</th>
<th>(\Sigma) (s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>(b^1) 56</td>
<td>0.17</td>
<td>8</td>
<td>3</td>
<td>0.01</td>
<td>11</td>
</tr>
<tr>
<td>(b^2) 521</td>
<td>1.55</td>
<td>67</td>
<td>34</td>
<td>0.02</td>
<td>101</td>
</tr>
<tr>
<td>(b^3) 955</td>
<td>2.72</td>
<td>110</td>
<td>65</td>
<td>0.04</td>
<td>175</td>
</tr>
<tr>
<td>(b^4) 2085</td>
<td>6.37</td>
<td>270</td>
<td>197</td>
<td>0.06</td>
<td>468</td>
</tr>
<tr>
<td>(b^5) 4102</td>
<td>11.9</td>
<td>520</td>
<td>409</td>
<td>0.10</td>
<td>930</td>
</tr>
</tbody>
</table>

5.1.1. Pre-processing performance

In Table 1, column 2 shows the size of each database. Column 3 shows the total size of the motion indices. Columns 4 (\(t_1\)), 5 (\(t_2\)) and 6 (\(t_3\)) show the time required for reading in the database, computing the motion indices, and writing out the motion indices, respectively. Column 7 shows the total pre-processing time (the sum of Columns 4, 5, and 6).

We have compared EigenMR with three methods: Muller et al. [29], Deng et al. [12], and Kovar and Gleicher [19]. Figs. 6 and 7 show the pre-processing times and the index sizes of the methods, respectively. Because the index construction time for [19] is approximately 1000 times that of EigenMR (~300 s for a database containing only 37,000 frames), we have not included its results here for the sake of clarity.

We may observe from Fig. 6 that EigenMR has a much lower computational cost in the pre-processing step compared with [29] and [12] (it is only one-fourth of the time of [29]). This lower cost is due mainly to our simple index construction process, which has a complexity that is linear to the number of motion files. In addition, we do not need to run a clustering process on the database motions, whereas [12] needs to run a k-means process and [19] needs to build match webs. Further, because [12] and [19] perform retrieval based on all of the motions in the database, they need to update the whole database as a new motion is added, which is computationally very expensive. This added cost also makes them less suitable for real-time applications, where new
motions may need to be added to the database dynamically. On the contrary, our pre-processing step operates on individual motions. Thus, when a new motion is added to the database, we only need to compute the new index for this new motion. Although the computational cost of [29] is also linear with the size of the database, it requires the user to specify appropriate semantic geometry features. From Fig. 7, we can see that our method has a much lower memory cost for storing the index structure, which occurs because our indices are much simpler than those of the other methods.

5.1.2. Retrieval time

In this experiment, we apply some random queries to the five databases mentioned in Table 1. Fig. 8 compares the retrieval time of EigenMR with [12] and [29]. Because using too many features will significantly slow down [29] and too few features will reduce its retrieval accuracy, we choose to use eight of the 31 features in [29] for a fair comparison. We can see that EigenMR has a much lower retrieval time than the other two methods.

In EigenMR, the retrieval time of each query depends on the size of the database, the length of the index and the allowable difference in index length, LengthThr, defined in Section 4.2. When the difference in index lengths between the query motion and a database motion is larger than LengthThr, we skip the matching process. The computational complexity of the runtime retrieval process of EigenMR is in fact less than \( O(n) \), where \( n \) is the number of motions in the database. There are three other reasons for the efficiency of EigenMR. First, we do not need a complex string-based time-series matching process. Second, we reduce the unnecessary matching time through motion filtering as discussed in Section 4.4. Third, we have the smallest index size, as shown in Fig. 7.

![Fig. 6. Comparison of the pre-processing time.](image)

![Fig. 7. Comparison of the index size.](image)
5.2. Retrieval accuracy

To study the retrieval performance of EigenMR, we collect a set of 132 motions (a total of 56 MB in size and including 77,428 frames) from the CMU motion library [11]. Based on the original semantic labels of the motions, we divide the motions into nine semantic categories: walking, running, jumping, kicking, playing-basketball, climbing-ladder, drinking, picking and cartwheeling, with 31, 33, 24, 6, 10, 12, 3, 5 and 8 motion files, respectively.

Fig. 9 shows four example query motions (in red) and the returned matches (the top three ranked matches in blue and the rest in green). Each query took less than 2 ms to complete. For the jumping motion in Fig. 9(a), the query motion has 369 frames, whereas one of the matched motions has over 2751 frames. A long jumping motion retrieved is composed of ten basic jumping cycles and two turning-around cycles, where each jumping cycle is similar to the input query. For this type of motion, the repeated basic jumping actions share similar eigenspaces and EigenMR handles this type of subsequence retrieval well. For the climbing-ladder motion in Fig. 9(d), the average number of frames in the motions is over 3500. Here, the retrieved motions and the query motion are all similar, but composed of multiple different actions. Refer to the supplementary video for more detail for the results.

Supplementary material related to this article can be found online at http://dx.doi.org/10.1016/j.cag.2013.11.008

To study the retrieval accuracy of EigenMR, we have conducted an experiment similar to [19] and [12]. The idea of this experiment is to use the same query motions to search two different databases. One contains a single type of motion, e.g., walking,
and is called a single-type database. The other contains multiple types of motion and is called a mixed-type database. The results from the single-type database are considered to be the ground-truth. Then, the retrieval accuracy can be computed as:

$$\text{Accuracy} = \frac{\text{# of matched retrieved files from mixed} \times \text{type dB}}{\text{# of retrieved files from single} \times \text{type dB}}$$

where the matched retrieved files are the motions retrieved from the mixed-type database that match with motions retrieved from the single-type database [19,12]. This definition is popular in motion retrieval, most likely because it considers both recall and precision at the same time.

In this experiment, we use the database mentioned at the beginning of this section. For each query, we consider only the top 15 retrieved motions. As shown in Fig. 10, EigenMR outperforms [12] and [19] on almost all types of motion. However, [12] slightly outperforms EigenMR on the running motion, mainly because our method does not consider velocity and trajectory.

Note that we have not compared EigenMR with [29] here because [29] requires the user to specify a set of semantic geometry features. Thus, it is difficult to perform a sound comparison between this type of semantic-based retrieval method and the pure example-based retrieval methods of [19] and [12].

5.3. Logically similar retrieval

We argue above that because the transform-domain eigenvectors can capture significant features in a holistic and space-time independent way, EigenMR is capable of performing logically similar retrieval. To demonstrate this ability, Fig. 11 shows the matches of the picking motion. EigenMR can identify all of the picking motions in the database. In addition to the four picking motions with one hand, EigenMR also identifies a picking motion with both hands, although it has the lowest score.

Fig. 12 shows the matches of the forward dribbling motion. In addition to four matches that are forward dribbling motions (matches 1 to 4), EigenMR also identifies two matches of sideways dribbling.

Fig. 10. Comparison of the accuracy of three retrieval methods.

Fig. 11. Five matches on picking motion. The query motion is in red and the top three matches are in blue. (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article).

Fig. 12. Due to the logical similarity of hand-shaking and dribbling, some hand shaking motions is incorrectly retrieved.
dribbling motions (those in red ellipses). However, EigenMR also mistakenly retrieves hand-shaking motions (those in black ellipses) as dribbling motions because their differences are rather subtle. Fortunately, the ranking scores of these mistakenly retrieved motions are the lowest among the matches.

EigenMR can retrieve logically similar motions because the primary eigen-patterns of two logically similar motions should also be similar. Based on this idea, we have implemented some techniques to improve the matching accuracy due to the lack of time-domain information, including the six sets of features in two levels, the length normalization process, and a different way of computing matching scores. Taking the dribbling motion of Fig. 12 as an example, if we use forward dribbling as a query motion, we may also obtain sideways dribbling motions in addition to forward dribbling motions. The proposed method succeeds because forward dribbling motions and sideways dribbling motions share similar movement patterns except that they are different in the facing direction and the leg movements.

6. Conclusion and discussion

In this paper, we have proposed a novel and efficient motion retrieval approach for real-time 3D animation based on matching in the transform domain. Although our motion index is small, it can capture the holistic features of motions well. As a result, our
method can achieve better retrieval performance compared with some of the latest retrieval methods. In addition, our approach is scalable in that it has a linear index construction time based on the size of the database. We have conducted a number of experiments. Our results show that the proposed method is significantly more efficient in matching time compared with some popular retrieval methods. This result is very important in real-time applications. Our method is also generally more accurate than previous methods.

In the following subsections, we discuss our feature definitions, the limitations of our method, and possible future work.

### 6.1. Feature definition

For the features, we have adopted the angular definition shown in Fig. 13 for our experiments. These angles are formed between lines that represent bones or connections of two joints and between the normals of two skeleton plane structures and indicate the relation between different parts of the body. Those in Fig. 13(a) and (b) are designed to measure arm and wrist motions. Those in Fig. 13(c), (d) and (e) are designed to measure leg and ankle movements. Fig. 13(e) also captures the twisting of the legs. The angle in Fig. 13(f) measures the head motion. Those in Fig. 13(g) and (h) measure body leaning and bending. Appendix A shows the definitions in more detail.

| Global features | Angle between CGD plane and MRN plane, where CGD plane is the plane determined by C, G, D, the same with MRN plane. | Angle between CD and Gravity |
| Local features | Angle between CD and TN |
| Head feature | Angle between TN and XY |
| Upper limb features | Angle between FJ and EI |
| Lower limb features | Angle between EI and FD |

### 6.2. Limitations of the proposed method

The main idea of EigenMR is to convert the temporal pattern matching problem of motion retrieval into a simple index comparison, i.e., matching of eigenvectors and eigenvalues in the transform domain at a constant time. The computed motion index captures the motion spans of major features without any dynamic information such as feature motion trajectories and moving speed. As such, the first limitation of our approach is that the matches may sometimes include motions of very different trajectories. For example, if the query motion is forward walking, EigenMR may also return backward walking motions. However, although walking and running are similar and differ mainly on their trajectory patterns, EigenMR can distinguish them well (as shown in Fig. 14), though EigenMR does not perform as well as [12], as discussed in Section 5.2. We believe that the difference between these types of motion, as in many other cases, is not only in their motion trajectory and moving speed. They also have different spans of the hand and leg motions, which affect the computed eigenvectors. This allows EigenMR to differentiate different types of motion.

Intuitively, our method is most successful with motions that contain only a single action. This is the case for many motion synthesis applications because they only require searching for motions containing similar actions to concatenate the actions or construct a parametric motion space for a single action. Our method can also work well in the scenario where both the query motion and the motions in the database contain multiple actions. An example is the climbing–ladder motion that we mentioned in Section 5.2. This type of motion is composed of multiple actions, but they still show similar eigen-patterns. However, our method cannot handle subsequence retrieval very well theoretically. For example, a query of a walking motion is unlikely to be matched

![Fig. 16. Retrieved motions when using sideways dribbling as a query motion with AccThr set to 0.95 (left) and 0.80 (right).](image)

![Fig. 17. Retrieved motions when using picking as a query motion with LengthThr set to 3 (left) and 6 (right).](image)
with climbing-ladder motions that also contain walking actions. The reason is that the eigen-patterns of the subsequence walking motion is diluted and even completely lost in the extremely long climbing-ladder motion. Conversely, when the input query motion is part of a slightly longer motion in the database, this longer motion may be mistakenly retrieved. This subsequence problem is the second limitation of our approach, although this limitation can be avoided by segmenting each motion into subsequences using a motion segmentation method [2] before matching the query motion with each individual subsequence.

Theoretically, if a very large motion database composed of totally different motions is used, there will be a higher chance of getting similar eigen-patterns from different motions. This is another limitation of our method. One method to address this problem is to perform two-stage retrieval: the proposed method performs a coarse retrieval to filter irrelevant motions, and then a traditional method performs an accurate comparison.

6.3. Future work

For future work, we plan to improve the proposed method in three respects. The first is to incorporate some minimal transition information into the motion index to further improve the retrieval accuracy while keeping the index size small. The second is to incorporate the motion segmentation method to address the subsequence problem. The third is to adopt a two-stage retrieval strategy to improve both accuracy and efficiency.

Acknowledgments

We would like to thank all of the reviewers of this paper for their constructive comments. This work was partially supported by a SRG grant from the City University of Hong Kong (Project number: 7002768), NSFC (Project number: 61300089), NKTSP (Project number: 2013BAD24F00), and two grants from Liaoning Province (Project number: L2013562 and 20131023).

Appendix A. Defining angular Features

Fig. 15 shows a simplified human skeleton model and feature representation. There are a total of 16 vectors, with each vector connecting two joints except the gravity vector that points to the ground. Among these vectors, 10 are bones (in solid lines) and 5 connect non-adjacent joints (in dashed lines).

We divide the geometry features into two parts: global features and local features. Table 2 shows the definitions in detail. These angles represent the topological relationships of different body parts. As mentioned in Section 3.1, the types of motion are mainly specified by the positions of the end-effectors. Hence, we include more angles that are related to the end-effectors, such as ankles, wrist and head, that are as orthogonal to each other as possible. There are a total of 12 angles. This number is enough to minimally cover a wide range of motion types.

Appendix B. Discussion on threshold and parameter

In this section, we discuss the parameter and threshold settings.

• AccThr: This threshold determines how many eigenvectors to maintain. It is defined in Section 3.2. The recommended range of this threshold is [0.90, 0.98]. If this threshold is too small, then some important eigen-pattern information will be lost. If it is too large, then there will be a lot of eigenvectors to compare, increasing the matching time. As an example, in Fig. 16, if we set AccThr to 0.95, then all retrieved motions are dribbling basketball, with sideways dribbling having the highest matching score. If we set AccThr to 0.80, the retrieved motions will include a drinking motion.

• LengthThr and PunishThr: These two thresholds are defined to pre-filter the motions that are unlikely to be similar. LengthThr determines if two motions are definitely different without the need for further comparison. It is typically set in the range of [1.5, 5], depending on the type of motion. Otherwise, a lot of similar motions will be filtered out or dissimilar motions will be included. PunishThr determines how small a matching score should be neglected from consideration. It is typically set in the range of [0.3, 0.6]. As an example, in Fig. 17, if we set LengthThr to 3, then the retrieved motions include four picking motions using one hand and one picking motion using two hands. If LengthThr is set to 6, then the retrieved motions will include a jumping motion that has a higher matching score than the picking motion using two hands.

• 1passThr and 2passThr: These two thresholds are defined in the 2-pass filtering process. 1passThr filters motions with low matching scores on whole body features, and 2passThr filters motions with low matching scores in individual body parts. Normally, the first threshold is smaller than the second one. These two thresholds may be set depending on the preferred number of retrieved motions.

• WWhole, WRightArm, WLeftArm, WTorso, WRightLeg, and WLeftLeg: These thresholds are weights for the six individual matching scores in Eq. (14). We typically set the matching score for the whole body to be higher than those of individual body parts and in the range of [0.4, 0.6]. The weights for individual body parts are typically set in the range of [0.08, 0.12]. However, the sum of these weights should equal to 1. Different weights may also be given when retrieving a specific type of motion. For example, if our focus is on kicking motions, we should give higher weights to the legs.

References


Lin, Y. 2006. Efficient Human Motion Retrieval in Large Databases. In: Proceedings of the GRAPHS.


