A Real-time Database Architecture for Motion Capture Data

Pengjie Wang[1,2] Rynson W.H. Lau[3] Mingmin Zhang[1] Jiang Wang[3] Haiyu Song[2] Zhigeng Pan[1,4]
[1]State Key Laboratory of CAD & CG, Zhejiang University, Hangzhou, 310027, China
[2]College of Computer Science & Engineering, Dalian Nationalities University, Dalian, 116600, China
[3]Department of Computer Science, City University of Hong Kong, Hong Kong
[4]Digital Media and HCI Research Center, Hangzhou Normal University, Hangzhou, 310023, China
wpj@dlnu.edu.cn, rynson@cs.cityu.edu.hk, zmm@cad.zju.edu.cn, {wj11hk, songhaiyu, zhigengpan}@gmail.com

ABSTRACT
Due to the popularity of motion capture data in many applications, such as games, movies and virtual environments, huge collections of motion capture data are now available. It is becoming important to store these data in compressed form while being able to retrieve them without much overhead. However, there is little work that addresses both issues together. In this paper, we address these two issues by proposing a novel database architecture. First, we propose a lossless compression algorithm to compress the motion clips, which is based on a novel Alpha Parallelogram Predictor (APP) to estimate the degree of freedom (DOF) of each child joint from its immediate neighbors and parents that have already been processed. Second, we propose to store selected eigenvalues and eigenvectors of each motion clip, which only require a very small amount of memory overheads, for faster filtering of irrelevant motions. Based on this architecture, real-time queries become a three-step process. In the first two steps, we perform a quick filtering to identify relevant motion clips in the database through a two-level indexing structure. In the third step, only a small number of candidate clips are uncompressed and accurately matched with a Dynamic Time Warping algorithm. Our results show that users can efficiently search clips from this losslessly compressed motion database.

Categories and Subject Descriptors
H.3.3 [**Information Storage and Retrieval**]: Information Search and Retrieval – *retrieval models*

General Terms
Algorithms, management, performance, design, experimentation.

Keywords
Motion capture data, motion compression, motion indexing.

1. INTRODUCTION
Motion capture (mocap) data are now widely used in many applications, including movies, games, animation and virtual environments. As huge collections of mocap data are being produced, there is a need to design effective motion compression and retrieval methods for the storage and retrieval of these data. However, there is little work that considers both the indexing and compression of mocap data.

*Area Chair: Wei Tsang Ooi

MM'11, November 28–December 1, 2011, Scottsdale, Arizona, USA.

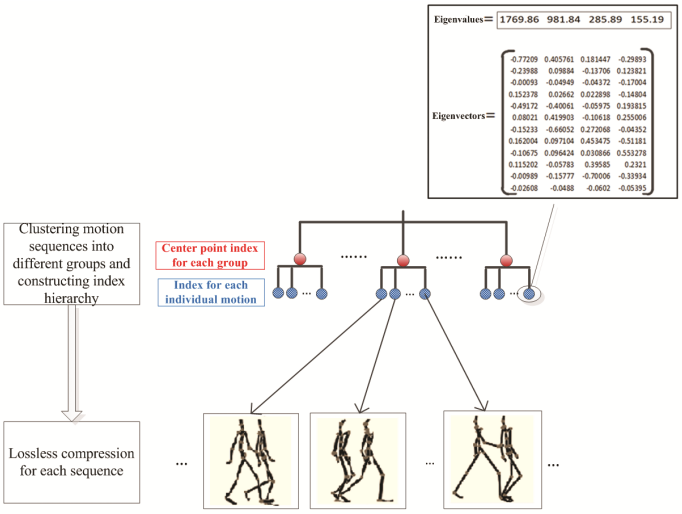

Figure 1. System Overview.

In this paper, we try to address these two problems by proposing a novel database architecture for motion capture data. Our architecture is composed of two levels, the index level and motion clips level, as shown in Figure 1. First, motion clips are clustered into many groups using the proposed novel distance measure, EigenDis, which is defined in the eigenspace of motion clips. Second, the two-level indexing structure is defined. The high-level of the indexing structure is represented by the center point index of each cluster, while each node in low level of the indexing structure stores the leading eigenvectors and the eigenvalues of a motion clip and a pointer to the corresponding motion clip. Each motion clip in the database is compressed losslessly by using our novel Alpha Parallelogram Predictor (APP), which estimates the degree of freedom (DOF) of each child joint from its immediate neighbors and parents that have been processed. The prediction parameter, alpha (α), is adaptively chosen from a carefully designed lookup table. Our results show that the user can efficient search the motion data from the losslessly compressed motion database, as our two-level indexing structure can filter irrelevant motion clips effectively and reduce the candidate set, and only the final set of clips need to be decompressed.

2. RELATED WORKS
The need for compact motion representation has resulted in a number of compression techniques for mocap data. Arikan [1] uses Bezier curves to represent motion clips, and then applies clustered PCA to reduce dimensionality. Beaudoin et al. [3] adapt standard wavelet compression on joint angles to compress mocap data. Tournier et al. [12] use principal geodesics analysis to build

a descriptive model of the pose data of a motion, keeping only the leading principal geodesics. This model is then used in an inverse kinematics system to synthesize poses that not only match the end-joint constraints but also with the interior joint positions. Given this pose model, only the compressed end-joint trajectories and the root joint's position and orientation need to be stored in order to recover the motion. All these methods are lossy method. Therefore, they throw away some information from the data in order to achieve high compression ratios.

On the other hand, automatically searching for similar motions is of great importance in motion synthesis and reusing. Since motion data are multi-attribute time-series, it is straightforward to apply dynamic time warping (DTW) [4, 6] or uniform scaling [8] for motion retrieval. However, these methods are based on numerical comparison and are typically very slow. Kovar and Gleicher [9] propose pre-computing "match webs" to describe potential matches among motion subsequences, in order to improve run-time retrieval efficiency. However, this method has a very high pre-processing time. In addition, if a query is not already in the database, this method will need to compute the match webs between the query and all the motions in the database. Deng et al. [5] propose to decompose motions into body parts and extract common motion patterns for each part. A motion clip can be represented as a string of pattern indices, and so motion retrieval becomes simple string matching. However, the data structures used amount to nearly 5% of the size of the motion database, which can be expensive for large databases. Muller et al. [11] 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, such as boxing, is non-trivial, especially for novice users.

In summary, current state-of-art motion retrieval algorithms do not work with current compression algorithms, since compression algorithms introduce an overhead to real-time applications – the motion files need to be decompressed before use. In addition, all existing compression methods for motion capture data are lossy. However, if we want to edit a motion file compressed with a lossy compression method, we will need to uncompress it before editing and then compress it again afterwards. Each time we compress the motion file, the lossy compression method will introduce additional error to the file. Hence, a motion file needed to go through a number of such editing operations at different times will result in a compound error. Thus, a lossless compression method is needed. In this paper, we address these two problems by proposing a database architecture that integrates our proposed lossless compression method with our two-level indexing structure and a tree-step motion retrieval method.

3. DATABASE CONSTRUCTION

The raw motion data are high-dimensional time-series, which need dimension reduction. In this paper, we accomplish dimension reduction by adopting the feature set defined by [10], which uses positions of four end effectors and head as the features of the motion. After this process, the curve simplification algorithm is applied to reduce the number of frames to 20% of the original one. Finally, an adaptive *k*-means clustering algorithm is used to group similar motion clips together. To cluster motions, we need an effective and efficient distance measure. We propose to compute a matching score of two clips in eigenspace as our distance measure, called *EigenDis*.

4. Motion Index and EigenDis

In [2], it is observed that different actions within a single motion have different eigenspaces, and motion segmentation can be achieved by taking advantage of this. Inspired by this observation that each type of motion has its unique eigenvector "pattern", we propose to extract these "patterns" as motion index, and define EigenDis as the pattern matching score of two clips. In run-time, the matching scores between the input query clip and the clips in the database are used to pre-filter irrelevant clips.

First, we represent the original motion clips as a low-dimensional matrix composed of *d* vectors:

$$\mathbf{F} = (f_1, f_2, f_3, ..., f_d) \quad (1)$$

where $f_1, f_2, \ldots, f_d$ are the position vectors, *d* indicates the total number of angles, each defined as $f_i = (f_{1,i}, f_{2,i}, ..., f_{n,i})^{\mathrm{T}}$, and *n* is the number of frames in the motion. We define a mean vector:

$$\overline{\mathbf{F}} = (\overline{f}_1, \overline{f}_2, \overline{f}_3, \ ..., \overline{f}_d)$$

where $\overline{f}_i = \frac{1}{n}\sum_{j=1}^{n} f_{j,i}$, and a difference vector:

$$\hat{\mathbf{F}} = \mathbf{F} - \overline{\mathbf{F}} = (\hat{f}_1, \hat{f}_2, \hat{f}_3, ..., \hat{f}_d)$$

We compute the covariance matrix of $\hat{\mathbf{F}}$ as:

$$\mathbf{R} = (r_{j,k})_{d \times d} \quad (2)$$

where $r_{j,k} = \frac{1}{n}\sum_{i=1}^{n} \hat{f}_{i,j} \cdot \hat{f}_{i,k}$. We then obtain the *s* eigenvalues of **R**, $\lambda_1 \geq \lambda_2 \geq ... \geq \lambda_s \geq 0$, and the corresponding eigenvectors $v_1, v_2, \ldots, v_s$, 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 keep the leading *m* ($m \leq s$) largest eigenvalues until the accumulated weight is higher than a threshold *AccThr*. (In our implementation, we set it to 0.9.) We refer to *m* as the *length of the motion index*. Then, the final eigenvector matrix becomes:

$$\mathbf{V} = (v_1, v_2, \ldots, v_m) \quad (3)$$

We store **V** and the corresponding eigenvalues as the motion index of this motion, as shown in Figure 1. Assuming that the query motion is **VQ** and a motion in the database is **VD**, we compute the matching score of these two indexes as,

$$\rho = \sum_{i=1}^{m} w_i \rho_{i,i} \quad (4)$$

where w_i is defined as the i^{th} eigenvalue over the total eigenvalues of the query index, and $\rho_{i,i}$ is the correlation coefficient between the i^{th} eigenvector of **VD** and the i^{th} eigenvector of **VQ**.

4.1 Two-level Index Construction

After the clustering process, a two-level indexing structure is constructed. The low level is the index for a specific motion clip. The content of an index node includes eigenvectors (Eq. (3)) and the corresponding eigenvalues. At the high level, each cluster stores a center point index. This center point is computed from the *k*-means clustering process. Each motion clip, as pointed to by a low level index node, is compressed losslessly using our Alpha Parallelogram Predictor (APP) method.

4.2 Lossless Compression of Motion Data

We adapt and extend the traditional parallelogram predictor [7] of mesh compression to compress the motion capture data in a lossless way, by introducing a prediction parameter, alpha (α), and its lookup table.

The motion data is first split into some equal-sized clusters before prediction, with each cluster having t poses. Figure 2 shows a human skeleton model and an example of APP. The right hand side of Figure 2 only shows two columns and some rows of cluster i. Assuming that c is a DOF index of child joint C, p is the DOF index of the parent of joint C. The numbers of DOFs of joint 18 and joint 16 are indexed by c and p, respectively. For cluster i, we can predict $DOF_{b+j,c}$ using the following formula:

$$P_{b+j,c} = DOF_{b+j-1,c} + \alpha\,(DOF_{b+j,p} - DOF_{b+j-1,p}) \quad (5)$$
$$b = t(i-1)$$

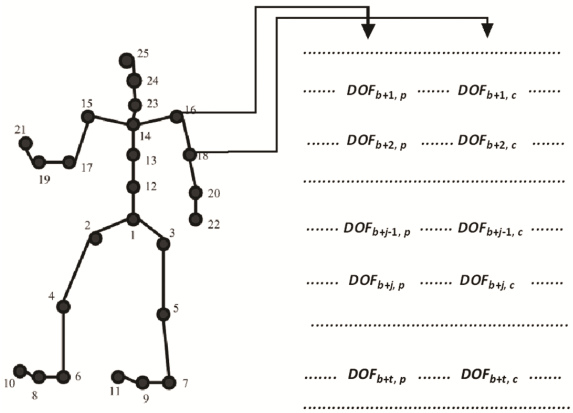

Figure 2. The virtual human skeleton and an example of APP.

When using Eq. (5) for prediction, the prediction error of DOF index c of cluster i is:

$$\varepsilon_{i,c} = \sum_{j=1}^{t}\left(DOF_{b+j,c} - P_{b+j,c}\right)^2$$
$$= \sum_{j=1}^{t}\left((DOF_{b+j,c} - DOF_{b+j-1,c}) + \alpha(DOF_{b+j-1,p} - DOF_{b+j,p})\right)^2 \quad (6)$$

where $DOF_{b+j,c}$ is the actual value corresponding to the prediction value $P_{b+j,c}$. We obtain the minimum of $\varepsilon_{i,c}$, when:

$$\alpha = \frac{\sum_{j=1}^{t}(DOF_{b+j,p} - DOF_{b+j-1,p})(DOF_{b+j-1,c} - DOF_{b+j,c})}{\sum_{j=1}^{t}\left(DOF_{b+j,p} - DOF_{b+j-1,p}\right)^2} \quad (7)$$

The α value computed from Eq. (7) is referred to as the ideal α value, since we can obtain a minimum error if we use it for prediction. Unfortunately, each ideal α value will occupy 32 bits to store. With a lot of α values to store, it will offset the compression ratio significantly. To address this problem, we first construct a α lookup table. Given an ideal α value, we simply look up the nearest α value from this lookup table and use it for prediction as described in Eq. (5). In this way, we only need to store the indices to the lookup table.

5. A TREE-STEP RETRIEVAL METHOD FOR COMPRESSED MOTIONS

Based on our database architecture, we can elaborate our run-time three-step motion retrieval algorithm as follows:

1. ***Group-filtering*** – This step is to branch the query motion to candidate groups by calculating matching score (EigenDis) between the query index and the center point index of each group. We set a threshold *FilterThrLel1* to remove all groups which scores are below this threshold. Others are returned as the resulting candidate groups.
2. ***Clip-filtering*** – This step is to identify the candidate clips from the candidate groups obtained from the first step. For each clip in the candidate groups, we compute one final matching score, which is the EigenDis between the query index and the index of the candidate clip. The matching is between the eigenvalues and eigenvectors of the two indices as in Eq. (4). We set a threshold *FilterThrLel2* to filter all clips which scores are below this threshold. Others are returned as the resulting candidate set.
3. ***Motion decompression and comparison*** – This step performs a detailed matching. We first uncompress each candidate motion clip and then use a time-series matching algorithm to compare between the query motion and this candidate motion. In our implementation, we use the DTW algorithm to perform the time-series matching.

With the above three-step retrieval algorithm, we can quickly remove a large number of irrelevant motion clips without involving decompression. We only need to decompress and execute the expensive DTW algorithm for a small number of motion clips to identify the matched motion clips.

6. RESULTS AND DISCUSSIONS

To evaluate the performance of the proposed method, we have implemented it in C++ and performed our experiments on a PC with an Intel Core Duo 2.26GHz CPU and 2GB RAM. Figure 3 shows our system interface. The red motion is the query. The blue motions are the best three motions and the remaining matched motions are in green. In this section, we first study the performance on the database construction process. We then study the run-time query performance of our method.

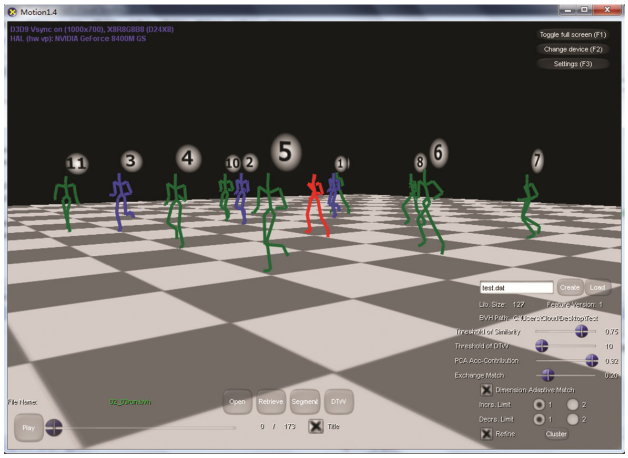

Figure 3. The system interface.

6.1 Performance of Database Construction

To examine the storage requirement and computation times of the proposed method, we have tested it on three databases of different sizes. In Table 1, column 1 shows the size of the input databases and the number of motion files. Columns 2, 3 and 4 show the index size, α look up table size and the compressed motion data size, respectively. Column 5 shows the total database size. Columns 6, 7 and 8 show the index extraction time, clustering time and compression time for constructing the database, respectively. Column 9 shows the total database construction time,

which is the sum of columns 6, 7 and 8. During the clustering process, the EigenDis threshold (***ClsThr***) that we use to determine if a clip should belong to a group is 0.75. There are also two compression parameters that we need to determine: the number of groups, t, and the α lookup table. Here, we set t=10 and the α lookup table is {-1, -0.5, -0.25, 0, 0.25, 0.5, 1, 2}. From Table 1, we can see that by using our lossless compression method, we can effectively reduce the sizes of the databases to roughly a quarter of their original sizes.

Table 1. Database size and construction time, with *ClstThr* = 0.75, *t*=10 and α lookup table={-1, -0.5, -0.25, 0, 0.25, 0.5, 1, 2}.

Orig. DB Size (MB) / No. of Files	Size (MB)				Time (s)			
	Index size	Alpha size	Comp. size	Total size	Indexing time	Clustering time	Comp. time	Total time
39 / 134	0.23	0.25	9.76	10.24	2.23	0.23	14.79	17.25
172 / 121	0.21	0.23	45.25	45.69	9.98	0.17	63.67	73.82
290 / 233	0.40	0.52	72.78	73.70	15.90	0.45	107.49	123.84

6.2 Run-time Retrieval Performance

In this experiment, we apply some random queries from four types of motion on the first database mentioned in Table 1. This database includes 39 walking, 33 running, 14 jumping, 10 basketball-dribbling, 5 kicking ball, 3 drinking, 4 climbing, 8 cartwheeling, 5 picking up, 10 hoping and 3 dancing motions. Figure 4 compares the *precision* and *recall* of the proposed method with the non-filtering method (full search with the DTW matching process), and the non-DTW method (simply filtering the motion clips with the proposed indexing structure). We can see that with the proposed indexing structure, the recall values of our method (as well as the non-DTW method) are similar to those of the non-filtering method. When applying the non-DTW method, the precision values are not very good as some irrelevant motions still exist. By using our proposed method, the precision values are almost the same as the non-filtering method.

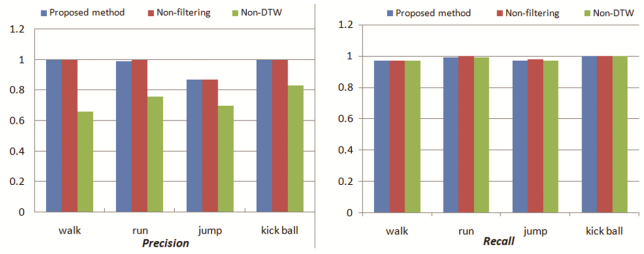

Figure 4. The *precision* and *recall* comparison between the proposed method (*FilterThrLel1*=0.2 and *FilterThrLel2*=0.45), the non-filtering method and the non-DTW method.

Table 2 shows the *response time per query* comparison of the proposed method, the non-filtering method (full search with the DTW matching process on the compressed motions) and non-compression method (our method but without compressing the motions). We use the same three databases as shown in Table 1 for this experiment. We can see that the proposed method can significantly reduce the *response time per query* through the filtering process with the indexing structure, as compared with the non-filtering method. The reduction is even more significant when the size of the database is large.

Table 2. The *response time per query* comparison between the proposed method, the non-filtering method and the non-compression method with number of candidate clips = 5.

DB Size(MB)	Proposed method (s)	Non-filtering (s)	Non-compression (s)
39	0.28	3.06	0.1
172	0.76	69.81	0.24
290	0.81	142.54	0.28

7. CONCLUSIONS

In this paper, we have looked at both compression and retrieval of motion capture data and proposed a database architecture to address both issues at the same time. We have proposed a lossless compression method for motions with a compression ratio of 25% of the original database size. We have also introduced a three-step motion retrieval method to effectively accelerate the retrieval process by filtering irrelevant motions with the proposed indexing structure. We have demonstrated the performance of the architecture by a number of experiments.

8. ACKNOWLEDGMENTS

The correspondence author of this paper is Zhang Mingmin. This paper was partially supported by NSFC (61170318, 61173124), two Fundamental Research Funds for the Central Universities (DC10040113, DC10040111), and two SRG grants from City University of Hong Kong (7002576 and 7002664).

REFERENCES

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

[2] J. Barbic, A. Safonova, J. Pan, C. Faloutsos, J. Hodgins, N. Pollard. Segmenting motion capture data into distinct behaviors. *Proc. Graphics Interface*, pp.184-193, 2004.

[3] P. Beaudoin, P. Poulin, M. van de, Panne. Adapting wavelet compression to human motion capture clips. *Proc. Graphics Interface*, pp. 313–318, 2007.

[4] C. Chiu, S. Chao, M. Wu, S. Yang, H. Lin. Content-based retrieval for human motion data. *Visual Communication and Image Representation*, 15(3):446-466, 2004.

[5] Z. Deng, Q. Gu, Q. Li. Perceptually consistent example-based human motion retrieval. *Proc. ACM I3D,* pp.191-198*,* 2009.

[6] K. Forbes, E. Fiume. An efficient search algorithm for motion data using weighted PCA. *Proc. ACM SCA*, pp.67-76, 2005.

[7] M. Isenburg, P. Alliez. Compressing polygon mesh geometry with parallelogram prediction. *Proc. IEEE Visualization*, pp. 141-146, 2002.

[8] E. Keogh, T. Palpanas, V. Zordan, D. Gunopulos, M. Cardle, Indexing Large Human-Motion Databases, *Proc. VLDB*, 2004.

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

[10] B. Krüger, J. Tautges, A. Weber, A. Zinke. Fast local and global similarity searches in large motion capture databases, *Proc. ACM SCA*, pp. 1-10, 2010.

[11] M. Muller, T. Roder, and M. Clausen. Efficient content-based retrieval of motion capture data. *ACM Trans. on Graphics*, 24(3):677–685, 2005.

[12] M. Tournier, X. Wu, C. Nicolas, A. Élise, R. Lionel. Motion Compression using Principal Geodesic Analysis. *Proc. Eurographics*, pp. 355-364, 2009.