A Database Architecture For Real-Time Motion Retrieval

Charly Awad, Nicolas Courty, Sylvie Gibet

A Database Architecture For Real-Time Motion Retrieval

Charly Awad, Nicolas Courty and Sylvie Gibet
VALORIA, University of Bretagne-Sud
Vannes, FRANCE
{charly.awad nicolas.courty sylvie.gibet}@univ-ubs.fr

Abstract

Over the past decade, many research fields have realized the benefits of motion capture data, leading to an exponential growth of the size of motion databases. Consequently indexing, querying, and retrieving motion capture data have become important considerations in the usability of such databases. Our aim is to efficiently retrieve motion from such databases in order to produce real-time animation. For that purpose, we propose a new database architecture which structures both the semantic and raw data contained in motion data. The performance of the overall architecture is evaluated by measuring the efficiency of the motion retrieval process, in terms of the mean time access to the data.

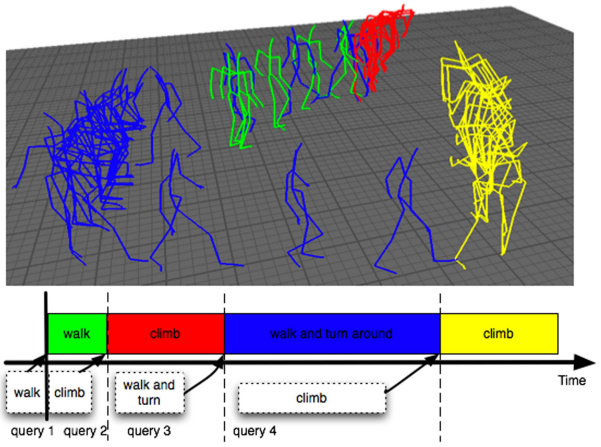

Figure 1. Creating animation from motion chunks extracted from the database.

1 Introduction

With the development of new technologies for motion capture, databases of human motions have become more accessible and yielded new methods for motion indexing, and retrieval, and computer-generated animation. As these databases become larger and more difficult to maintain, there is a need to develop more efficient and accurate ways to store, access, and process the data. The challenge has become even more important as the number of potential applications has increased, including the video game and animated films industries, gesture analysis of sport performances, and action recognition in computer vision based frameworks. The problem addressed here is the real-time access and retrieval of motion stored in large motion capture databases. Our goal is to use motion chunks extracted from the database in a motion synthesis loop that produces new animations from previously captured motions, as depicted in Figure 1. Such an animation system requires the ability to query the database at interactive framerates, constituting a particularly appealing challenge, since motion databases can contain several hours of captured motions to be assimilated in multidimensional time series.

A key aspect of motion data is that it contains multiple levels of information, from the semantic level to the time series level. Semantic labels (i.e., metadata) can be used depending on the type of movement: for example we may find labels such as gait, jump, or others that correspond to everyday motion or sport actions; for expressive motion such as dance labels can define key postures, or identify meaningful units of conversational gestures. Motion retrieval techniques may choose to exploit either the whole movement sequence, or some motion segments after temporal segmentation, the segments having been identified by their labels or time stamps. In all cases, and whatever the type of motion, we may consider that they contain both raw information (motion files) and semantic information (segmented and annotated motion files). The current objective is to propose a new database architecture which exploits both information, and to measure the performance of different motion queries. The final aim is to search within the database for specific movement sequences that answer the conditions expressed as semantic values in a query, and to select the motion chunk

that can best adapt to the animation context. The overview of the different processes is depicted in Figure 2.

To achieve these goals, different factors that influence access to the database are studied. Among these factors, we consider the database architecture and implementation, the different access modes to the raw data, and the structure of the semantic data.

The outline of this paper is as follows: Section 2 presents related work in the area of motion representation and retrieval techniques. Section 3 details the database architecture, separating the raw database, the semantic database, and the way both databases can be simultaneously used for animation. Section 4 is dedicated to results obtained using our architecture. Finally Section 5 is devoted to a conclusion and to possible future research directions.

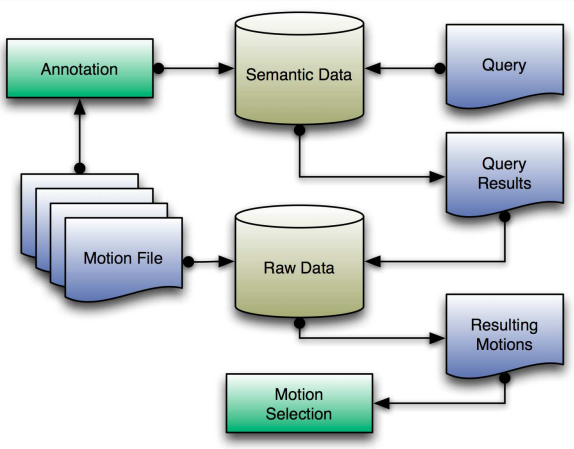

Figure 2. Overview of the architecture

2 Related Work

The brute-force search in a large motion database (which consists of comparing paired-wise each frame in the database) makes the process of retrieving motion captured data highly costly in computation time. Thus, several motion representation methods have been proposed in an attempt to solve this problem. One of these methods is clustering, used by Basu et al. in [2], to construct cluster graphs in which they grouped similar frames from one or more motion clips in nodes, and to sort frames within every node by clips and time. Liu et al. used in [12] a method that clusters poses from a motion database into groups. Geometric features are used in [11], where Lin cuts motions into segments and group motions based on a class of geometric features. Also in [14], Müller et al. propose to pre-process the motion database by dividing motions using qualitative geometric features. Dimensionality reduction techniques are another motion representation method used to reduce computational time; in [5], Forbes and Fiume project the data into a weighted Principal Component Analysis (wPCA) space. Another example of dimensionality reduction is the Isometric feature mapping used by Xiang et al. in [15]. Annotation is another way of presenting motion data, consisting of cutting a motion sequence into smaller motion chunks and labeling them with semantic information. Kendon proposes in [7] a manual segmentation method based on identifying movement phases in a video sequence. In [9], Kita et al. extend the work from [7] by applying the idea of movement phase identification to sign language video sequences. In [1], Arikan et al. use a semi-automatic annotation algorithm divided into two parts: the first part consists of a manual annotation, and the second uses the Support Vector Machine (SVM). In [3], Chao et al. present a different annotation approach for Tai Chi Chuan movements, which consists of building a Motion Index Table composed of Motion Clips and defining a Basic Motion Text as a set of sentences. Another method that Morales uses in [13], consists of storing motion captured data by converting it to an XML format. Chung et al. apply the same concept in [4] using a standard mark-up language MCML (Motion Capture Mark-up Language).

Our method stands out from the methods mentioned above, as it permits us to retrieve motions in real-time by simultaneously querying both a semantic and raw database.

3 Database Architecture

The proposed architecture, shown in Figure 2, is divided into two parts: data representation, and data retrieval and selection. The original methodology used for the data representation makes the motion retrieval faster, and presents the data in two formats: raw data and semantic data.

3.1 Data Representation

In this section, we detail motion data in the databases. The use of two different databases (a raw database and a semantic database) is justified by the fact that the two types of data (non-segmented raw data and segmented semantic data) are handled differently.

3.1.1 Raw Database

Motions are traditionally stored on the hard drive with various formats (.bvh, .fbx, .asf/amc, etc.). Interpreting those files amounts to building an internal representation of the motion in CPU memory. In our system, this internal representation is a collection of poses, each pose being composed of a root position and an ordered vector of quaternions (joint rotations). This representation is globally con-

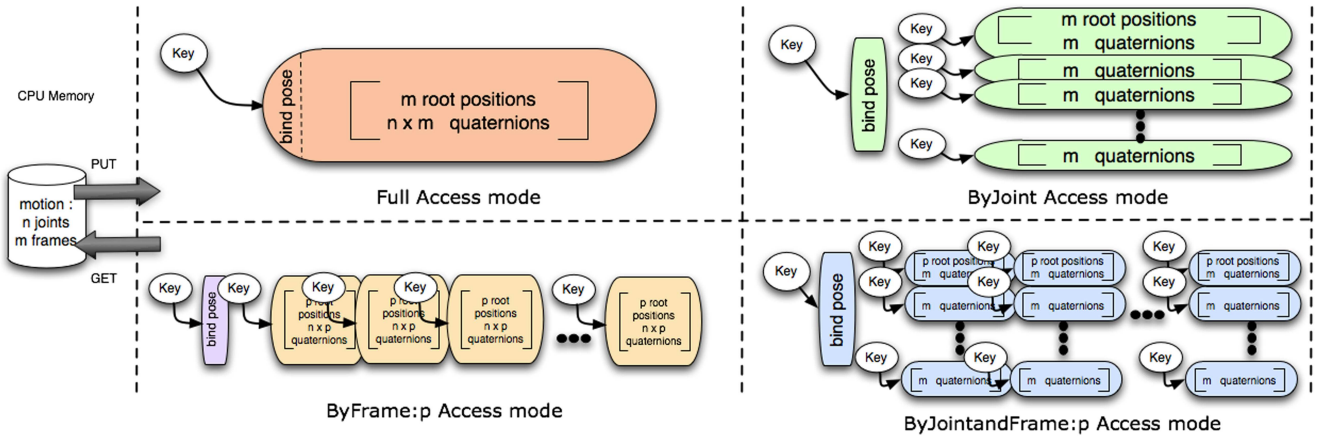

Figure 3. Different access modes to the database

sistent, provided that all the poses share a common hierarchical structure which is commonly named "bind pose". The time needed to read a motion file into this internal representation depends on the complexity of the parser and the amount of geometrical computation (for instance, cumulation of local transforms, switching from Euler angles to quaternions, etc.). This time is usually far from being negligible, and prevents dynamic loads in interactive applications. In our system, such files are loaded and interpreted one time, and stored as a sequence of bits in our database. We wrote our own serialization process for this purpose. Traditional databases function with a set of paired-value data: one key, preferably unique, is associated to the useful data, in our case the motion. The simplest way to proceed is to associate for instance the whole motion file with a unique key which can be chosen as the name of the original data file. The whole sequence is then handled by the database manager, and stored on the hard drive. This approach assumes that when retrieving the motion, all the data will be reconstructed in the CPU memory. In the context of a real-time animation controller, where small pieces of the motion are dynamically combined to achieve a desired goal, this policy is no longer efficient. Our database is designed to handle several access modes to the data. The following modes, depicted in Figure 3, determine particular fragmentation policies that are more or less suited to a particular type of query:

- *FullAccess* - one single motion is associated with one unique key. The first part of the value is the bind pose, followed by a sequence of poses formed by a root position and one quaternion per joint.
- *ByFrame:p* - the whole sequence of frames is divided into packets of p frames; one motion will produce $\left\lceil \frac{n}{p} \right\rceil$ entries in the database if n is the total number of frames in the motion.
- *ByJoint* - the whole motion is divided into m packets, each one corresponding to a given joint of the bind pose. The first joint usually also contains the root position, though this may change if several joints also have some translational degrees of freedom.
- *ByJointandFrame:p* - the whole motion is decomposed into packets that account for the motion of one joint over a sequence of p frames. One motion will produce $m * \left\lceil \frac{n}{p} \right\rceil$ entries in the database if n is the total number of frames and m the number of joints in the motion.

Fragmenting motions in the database is interesting because only a small portion of the motion (containing the query results) is reconstructed in the memory. However, this operation has a cost because it can multiply the number of entries in the database, thus increasing the search time and the index size. In our framework we allow the online modification of how data is stored in the database, effecting the best access mode for a given application. Hence, if the application is likely to access only specific subparts of the skeleton (like the hand for instance) over entire motions, the *ByJoint* access mode is optimal. We propose some quantitative results and compare between those modes in the experimentation section of the paper.

3.1.2 Semantic Database

One way to more easily access significant data in motion data files is to segment and annotate these files. The segmentation process identifies a set of elementary motion units that may be associated with meaningful semantic information. If we only consider a description of the motion, such motion units can either affect the whole body or specific parts of the body. In the latter case, a multi-channel

segmentation can be defined, with each channel being associated to a group of articulators (upper-body, arm, hand, etc.). For example, when describing phonological segments of sign languages, it might be interesting to separate the hand configuration from the arm movement [6]. Moreover, it is possible to distinguish specific phases on each channel, the boundaries of which are characterized by time stamps and are the result of a segmentation process, either manual or automatic. We propose both a spatial and temporal structure of the segmented motion data in our semantic database. The spatial structure refers to a multi-channel decomposition, whereas the temporal structure refers to phase description. We use an XML hierarchical description language which is provided by existing annotation tools such as ANVIL[1] or ELAN[2]. Once the annotation process is finished, the XML annotation documents are stored in the semantic database.

3.2 Data Retrieval and Selection

Retrieving data in the database is divided into two parts. The first part of the process consists of querying the semantic database, allowing us to extract information contained in the segmented files. Retrieving data from the semantic database is achieved by specifying one-condition or multiple-condition queries, called *PhaseQuery*. The query results are expressed as sequences of segments, each segment being characterized by time stamps expressing the start and end boundaries and the name of the phase. A simple one condition query, indeed, will probably return a great number of motion candidates, whereas a multiple-condition query will return fewer candidates. In the second part of the process, the query results are interpreted so that each segment triggers access to the raw database, giving the corresponding motion frames. Concerning the time access to the whole database, we expect that there is a compromise between the time-processing of the semantic data and the time-processing of the raw data, the complexity of the request being directly linked to the number of potential results. As the way in which the query is processed may globally affect the query's performance time, we propose two evaluation modes inspired from those available with Oracle Berkeley DB[3], the *Eager* and the *Lazy* modes. In the *Eager* mode, the query results are derived and stored in-memory; as for the *Lazy* mode, minimal processing is performed and the remaining processing is adjourned, thus making it faster in computational time. In our real-time motion synthesis context, choosing among the possible results of the XML query amounts to find a motion which is a good continuation of the previous motion. While "a good continuation" is somehow difficult to define and may result on different acceptations, we used a commonly accepted notion in the computer animation field which use an elastic distance to compare short sequences of motion. This elastic distance, obtained through dynamic time warping, is also well known in the context of time series indexation as well as human motion indexation [8]. In our implementation, we use a simple similarity metric between motion frames defines as the sum of geodesic distances between each pairs of corresponding rotations [10] to compute the similarity matrix. This operation is conducted on the final frames of the reference motion and the equivalent number of first frames of the candidate motion. This operation allows to select a candidate motion which will be easy to concatenate with the current motion with traditional animation techniques.

Algorithm 1 Selection Algorithm

$M\_Frames \Leftarrow$ last 10 frames of the $CurrentMotion$
repeat
  Candidate $C \Leftarrow$ Process $Query$
  $C\_Frames \Leftarrow$ 10 first frames of C
  $DTW\_distance(M\_Frames, C\_Frames)$
until $DTW\_distance$ is smaller than threshold
return C

In the selection algorithm the query is executed and the first result, a potential candidate, is processed. The DTW is applied to calculate the distance between the first 10 frames of the potential candidate, frames retrieved from the raw database, and the 10 ending frames of the current motion. If the distance is larger than a threshold (which is experimentally fixed), then the second result of the query becomes the potential candidate and is then processed. This operation is repeated until the distance is less than the threshold.

4 Experiments

We measured the computational time needed to retrieve motions in order to analyze the efficiency of our proposed architecture; specifically, the time to retrieve motions from the raw database, the time to retrieve motions from the semantic database, and the overall time. We tested our database system over the whole Carnegie Mellon University's (CMU) Graphic Lab motion capture database[4], which includes 2548 bvh files with a total size of 3 GB. The following experiments use annotations of basketball motion capture files; these files correspond to subject "06" in the CMU's database which were stored in our semantic database with a size of 600 KB. The experiments have run on a Macbook Pro dual core 2.4 GHz running with Mac OSX 10.5.5 equipped with 4 GB of memory. As for the

[1] http://www.anvil-software.de
[2] http://www.lat-mpi.eu/tools/elan
[3] http://www.oracle.com/database/berkeley-db
[4] http://mocap.cs.cmu.edu

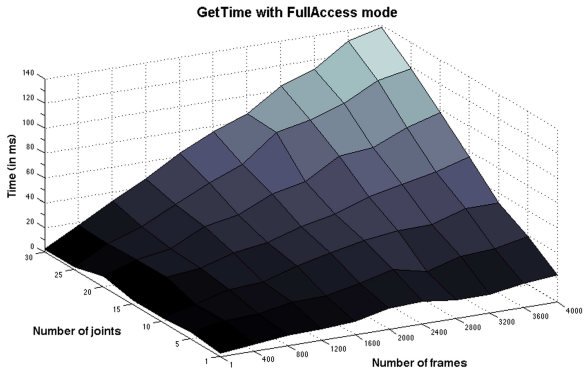

Figure 4. Get time for random access in the database with a full access mode.

choice of databases, we tested two different APIs: Oracle Berkeley DB and Tokyo Cabinet [5]; both engines led to approximately similar results. In all our tests a hash map was used as an index structure to recover the data.

4.1 Raw Database

4.1.1 Global Access Performances

We first tested the global performance on the entire database configured with the *Full Access* mode; i.e., one key–one motion. The total database size including the index was 5 GB. The following procedure was applied: for each pairing of number of joints and number of frames, we picked 20 random keys and measured the total time to get the information back in the CPU memory. Averaging those values finally gives the different results presented in Figure 4. As expected, increasing the number of joints and the number of frames at the same time, leads to an increase in the total *Get Time*. For instance, getting 4000 frames for one joint took on average 20 ms, while getting 4000 frames of the entire skeleton (38 joints) was done in approximately 140 ms. In practice, the search time in the database index is rather small compared to the time spent in allocating the resources in the CPU memory. This seems to be an explanation of the fact that the different databases which we tested behaved roughly the same.

4.1.2 Random Access on a Single Large File

In the second experiment, we tested random access on a random number of frames to a single long motion. This motion was a long sequence of "tai chi" motions (file ID 12_04) in the CMU mocap database, which is made of more than 18000 frames. Three access modes were tested: *Full Access*, *byFrame:10* and *byFrame:200*. In our tests, the first loads of the motion took more time because the database cache was filled with data. The cache size was set up so that the entire motion could be present in the cache. This explains why, in the case of *byFrame Access* modes, other loads were needed before the whole motion was present in the cache. As expected in the case of *Full Access* mode, the general trend was linear: whatever the number of frames required in the query, the whole motion is reconstructed back in the CPU memory and then cut according to the query. In the two cases of the *byFrame Access* modes, both decompositions led approximately to the same results: a linear increase with respect to the query number of frames. In all cases, it is interesting to note that for each access mode, it is possible to bound the access times.

[5] http://tokyocabinet.sourceforge.net

4.2 Semantic Database

Data stored in the semantic database was retrieved with the use of a query. For that matter, we performed two types of queries to evaluate the performance of our indexation method. The first type of query is a one-condition *PhaseQuery*, where the condition operates on a specific phase of one channel (associated to a part of the body). The second type is a two-condition *PhaseQuery* where the conditions correspond to two phases, each phase being associated to a specific channel. For instance, the query used to search for all the phases of forward basketball dribbles ("*ForwardD*") present in the semantic database is a one-condition *PhaseQuery* corresponding to the following:

for $a in ("SemanticDatabase") where $a/@[../attribute="ForwardD"] return $a/@start, $a/@end, $a/@key*

The results returned by the above *PhaseQuery* consist of a set of 79 pairs of time stamps that begin and end each of the forward basketball dribble segments, as well as the key corresponding to the annotated file. It should be noted that multiple condition queries involve a multi-layer search process in the hierarchical semantic database.

Regarding the two modes of processing a query we mentioned in section 3.2, we executed nine queries in order for us to compare the performance time of the two modes: three one-condition *PhaseQueries* and six two-condition *PhaseQueries*. Moreover, the number of iterations indicates the number of candidates, resulting from the query, used for the selection process. Table 1 show that it is preferable to use the *Lazy* query evaluation mode for real-time application.

4.3 Overall Architecture

Overall, in Table 1 we evaluate the entire path of the *PhaseQueries* through the databases. First, the *PhaseQuery*

Table 1. Querying the Databases

Condition(s)	Num. of Results	*Eager* Query(ms)	Num. of Iterations	*Lazy* Query(ms)	Total Time(ms)
ForwardD	79	133	7	106	220
ForwardD & Rhand	52	3410	8	908	1047
ForwardD & Lhand	27	3342	3	626	694
BackwardD	24	125	4	110	178
BackwardD & Rhand	21	1150	4	339	438
BackwardD & Lhand	3	1129	1	935	1052
StillD	26	128	8	116	256
StillD & Rhand	24	1248	8	515	687
StillD & Lhand	2	1220	1	1225	1235

D stands for Dribble, R for Right, and L for Left

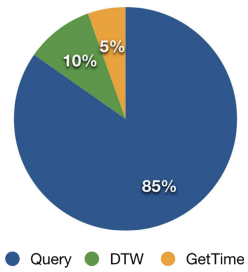

Figure 5. Overall Time Evaluation

is executed, then the selection algorithm is applied, and finally the frames corresponding to the selected candidate are retrieved. The overall process time (expressed by the query, the DTW, and the *Get Time* computational time) is presented in Figure 5, which shows that the retrieving time is much smaller than the query process time.

5 Conclusion

The work described here stems from the problem confronted when retrieving motions in real-time from large motion capture databases. We proposed a new methodology, simultaneously using a semantic and a raw motion database, taking into account the way motion data are indexed in these two databases. We evaluated the efficiency of the indexation by computing the database access and retrieval time, and finally, we proposed an overall architecture combining the different processes, from the querying of the semantic and raw database to the data selection process. The experimental results show that our proposed architecture is suitable for real-time applications. It is important to note that we are not evaluating the quality of the animation system but the retrieval efficiency in real-time. In future work, we hope to implement different motion retrieval techniques, produce real-time motion animations, and evaluate these animations from a quantitative as well as from a qualitative point of view.

References

[1] O. Arikan, D. A. Forsyth, and J. F. O'Brien. Motion synthesis from annotations. volume 22, pages 402–408, USA, 2003. ACM.

[2] S. Basu, S. Shanbhag, and S. Chandran. Search and transitioning for motion captured sequences. In *Proc. of VRST '05*, pages 220–223, USA, 2005. ACM.

[3] S.-P. Chao, C.-Y. Chiu, S.-N. Yang, and T.-G. Lin. Tai chi synthesizer: a motion synthesis framework based on key-postures and motion instructions: Research articles. *Comput. Animat. Virtual Worlds*, 15(3-4):259–268, 2004.

[4] H.-S. Chung and Y. Lee. Mcml: motion capture markup language for integration of heterogeneous motion capture data. *Computer Standards and Interfaces*, 26:113–130, 2004.

[5] K. Forbes and E. Fiume. An efficient search algorithm for motion data using weighted PCA. In *Proc. of SCA '05*, pages 67–76, USA, 2005. ACM.

[6] A. Heloir and S. Gibet. A qualitative and quantitative characterization of style in sign language gestures. In *Proc. of GW 2007*, LNCS, 2007.

[7] A. Kendon. Gesticulation and speech: Two aspects of the process of utterance. In *The Relation Between Verbal and Nonverbal Communication*, pages 207–227, 1980.

[8] E. Keogh, T. Palpanas, V. B. Zordan, D. Gunopulos, and M. Cardle. Indexing large human-motion databases. In *VLDB '04: Proceedings of the Thirtieth international conference on Very large data bases*, pages 780–791. VLDB Endowment, 2004.

[9] S. Kita, I. van Gijn, and H. van der Hulst. Movement phase in signs and co-speech gestures, and their transcriptions by human coders. In *Proceedings of the International Gesture Workshop on Gesture and Sign Language in Human-Computer Interaction*, pages 23–35, UK, 1998. Springer-Verlag.

[10] J. Lee, J. Chai, P. S. A. Reitsma, J. K. Hodgins, and N. S. Pollard. Interactive control of avatars animated with human motion data. *ACM Trans. Graph.*, 21(3):491–500, 2002.

[11] Y. Lin. Efficient human motion retrieval in large databases. In *Proc. of GRAPHITE '06*, pages 31–37, USA, 2006. ACM.

[12] G. Liu, J. Zhang, W. Wang, and L. McMillan. A system for analyzing and indexing human-motion databases. In *Proc. of SIGMOD '05*, pages 924–926, USA, 2005. ACM.

[13] C. R. Morales. Development of an xml web based motion capture data warehousing and translation system for collaborative animation projects. In *WSCG*, pages 168–173, 2001.

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

[15] J. Xiang and H. Zhu. Ensemble HMM learning for motion retrieval with non-linear PCA dimensionality reduction. *IIHMSP*, 1:604–607, 2007.