COMPUTER ANIMATION AND VIRTUAL WORLDS
Comp. Anim. Virtual Worlds 2013; **24**:365–375
Published online 9 May 2013 in Wiley Online Library (wileyonlinelibrary.com). DOI: 10.1002/cav.1522

SPECIAL ISSUE PAPER

Compression of 3D mesh sequences by temporal segmentation

Guoliang Luo[1], Frederic Cordier[2]* and Hyewon Seo[1]

[1] Université de Strasbourg (ICube, UMR 7357, CNRS), Strasbourg, France
[2] LMIA, Université de Haute Alsace (LMIA, EA 3993), Mulhouse, France

ABSTRACT

We describe a compression method for three-dimensional animation sequences that has notable advantages over existing techniques. We first aggregate the frame data by similarity and reorganize them into clusters, which results in the sequence split into several motion fragments of varying lengths. To minimize the number of clusters and obtain optimal clustering, we perform frame alignment, which eliminates the "global" rigid transformation from each frame data and use only "pose" when evaluating the similarity between frames. We then apply principal component analysis for each cluster, from which we get coordinates of corresponding frames in a reduced dimension. Because similar frames are considered, the number of coefficients required for each frame becomes smaller; thus, we obtain better dimension reduction for a given reconstruction error. Further, we perform intracluster compression based on linear coding. Because every motion fragment presents similar frames, conventional linear predictive coding can be replaced by key frame-based linear coding to achieve minimal reconstruction error. Results show that our method can obtain a high compression ratio, with a limited reconstruction error.

KEYWORDS
mesh sequences; compression; temporal segmentation

***Correspondence**
Frederic Cordier, LMIA, Université de Haute Alsace (LMIA, EA 3993), Mulhouse France.
E-mail: frederic.cordier@uha.fr

1. INTRODUCTION

With an abundance of animation techniques available today, efficient compression of animation data has been under intense investigation in Computer Graphics research community. A large portion of the animation data is so called frame data, which are essentially a set of vertex coordinates describing the geometry at each frame, stored as series of consecutive frames. Often, vertex-to-vertex correspondence is assumed to be known among the frames, which is usually the case. And when the data is in the form of animating mesh, face connectivity (topology) is assumed to stay constant. This explains why most of the existing works concentrate on the compression of vertex data only.

In this work, we present an alternative approach to the compression of three-dimensional (3D) animation sequence. Our work is motivated by the fact that animation sequences often tend to exhibit repetition of similar motions. To illustrate this, we provide a two-dimensional representation of a "horse" mesh sequence in Figure 1, which we obtained from the webpage of Sumner and Popović [1]. The projection in the two-dimensional embedded space has been obtained by applying multidimensional scaling [2] to the frame data. We observe that closely located frames in the embedded space are not always temporal neighbours. For instance, frames 8 and 31 exhibit the same pose, as well as frames 18 and 42. In addition, all these frames are closely located in the embedded space, although some of them are 36 frames apart in the frame domain. Consequently, we develop our method that exploits such *postural coherence* that is globally presented in the time domain, instead of searching for local coherences. We do so by clustering the frames according to their *pose similarity* and by removing the *postural redundancy* within each cluster. Despite the nonnegligible overhead of clustering, this allows to obtain high level of sparsity for each frame, leading us towards more effective compression as a whole. This simple, alternative change in notion becomes particularly beneficial (i.e., increased compression ratio) as the number of frames becomes large.

There are three main features of our work.

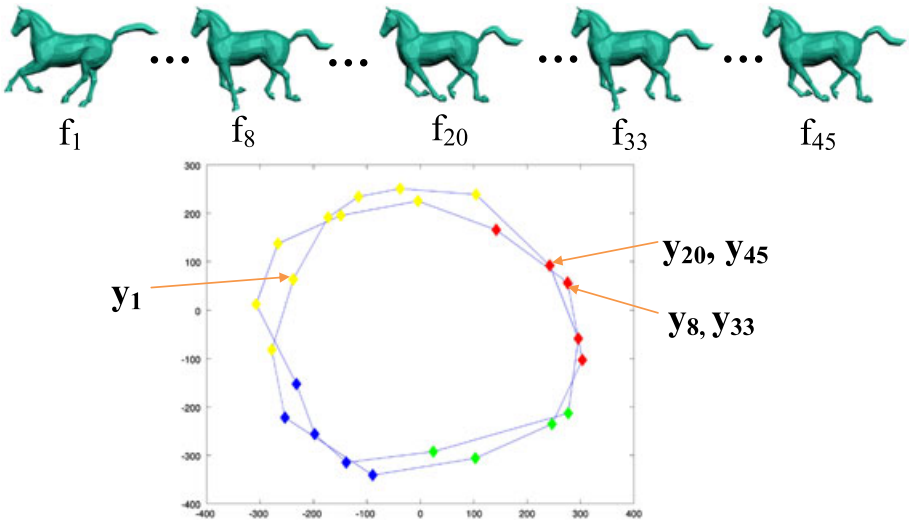

Figure 1. Frame data f_i from an animation sequence is represented as y_i in a reduced space by using multidimensional scaling. Frames that are temporally apart can overlap or come close together, as is the case with frames 8, 20, 33, and 45.

(1) We extend the notion of temporal coherence to postural coherence, which we exploit by frame clustering and by performing per-cluster compression.
(2) Principal component analysis (PCA) and its variants have been successfully used in animation compression, which is also adopted here. Contrary to previous approaches, however, we reorganize the frame data into clusters according to aforementioned pose similarity and apply PCA on each cluster. The animation sequence then can be compressed more efficiently with lesser PC components.
(3) We further perform intracluster compression by exploiting temporal coherence.

Instead of coding each motion fragment independently, we use linear predictive coding (LPC) to code the first motion fragment and reuse the once-specified frames to conduct key frame-based linear coding for the remaining fragments in the cluster. Because fragments belonging to a same cluster exhibit similar poses, the key frame-based linear coding allows us to achieve minimal reconstruction error. This has been justified by our experimental results.

The paper is organized as follows: We review previous works on compression in Section 2. After providing an overview of the proposed method in Section 3, we describe in detail the two main parts of our compression method in Sections 4 and 5. Then, we briefly describe the decoding procedure in Section 6. After demonstrating results as well as comparison with a number of existing works in Section 8, we conclude the paper in Section 9.

2. PREVIOUS WORKS

The research works on compression has been published mostly during last 6 years. We classify them into five categories depending on which technique they use: PCA-based compression, skinning method, scalable compression, connectivity-based compression, wavelet-based compression and octree-based compression.

The first category includes all the methods based on the PCA. The purpose of this method is to transform the data into a new basis with eigenvectors and where the coordinates are linearly independent. These eigenvectors form a new basis in which the vectors are linearly uncorrelated. The compression is achieved by removing the eigenvectors whose eigenvalues are negligible compared with the other ones.

In the seminal work by Alexa *et al*. [3], the PCA has been used to build compact representation of an animation sequence. The PCA of a matrix A where each column of the matrix is the geometry of one frame, that is, mesh-vertices are directly stacked into a single vector of length $3V$, where V is the number of vertices in the mesh, yields a set of eigenvectors (eigenframes).

The PCA method has been reused multiple times in combination with other compression techniques. Whereas one of the earliest works by Alexa *et al*. [3] uses PCA on frame data to exploit spatial coherence, most of the existing techniques apply the PCA on the vertex trajectories (temporal coherence). The result is a set of eigentrajectories with their corresponding principal component (PC) coefficients. Karni *et al*. [4] proposed to combine the PCA with the LPC to compress the PC coefficients. Vása *et al*. [5] have worked on the compression of the eigenvectors.

A large amount of work has also been carried out on the spatial clustering combined with the PCA. The idea is to cluster the vertices such that those belonging to the same cluster have similar trajectories. The PCA is then applied on each cluster independently; the number of PCs in each cluster required for describing the motion is usually much smaller than the number of PCs without clustering. This technique greatly improves the efficiency of the PCA

DOI: 10.1002/cav

compression. Sattler and his colleagues [6] were one of the first to propose the clustered PCA. Later, Ramanathan *et al*. [7] have worked on methods to find the optimal clustering that gives the best compression ratio.

The driving idea of skinning-based compression [8,9] is to take advantage of the fact that some of vertices have rigid motion. The mesh is clustered into segments whose motions are described with a 3D affine transformation matrix. This technique has been combined with the context-based adaptive binary arithmetic coding [10] approach to provide high compression ratio. This method has been promoted within the MPEG-4 standard [11]. The main advantage of this method is that it is particularly efficient for compressing dynamic meshes that represent articulated shapes such as human bodies or animals. On the other hand, they perform less efficiently for highly flexible mesh such as a flowing flag.

Some researchers [12] have worked on scalable compression by using a multiresolution scheme. The idea is to relate the geometry and the connectivity to create different levels of resolution. This set of methods makes use of the mesh topology, and they are useful in some specific context where different levels of resolution are required. For instance, the same compressed file could be sent to a mobile phone for low resolution display and a high-end computer where high resolution is preferred.

Ibarria *et al*. [13], Stefanoski *et al*. [14], Amjoun *et al*. [15] and Váša *et al*. [16] have worked on the connectivity-based compression. The driving idea is to exploit the connectivity of the mesh. In particular, these methods find the motion redundancy among vertices by looking at their connectivity. Unlike these approaches, our method does not require the mesh connectivity. It can be used to compress animated point clouds.

3. OVERVIEW

3.1. Animation Data

An animating mesh is a sequence of vectors $\mathbf{f_1}, \ldots, \mathbf{f_F}$ in which $\mathbf{f_t}$ represents one static mesh, often called "frame" or "pose", at time t ($1 \leq t \leq$F). A frame vector $\mathbf{f_t}$ is composed of the vertices coordinates of the mesh, which we represent as a column vector(x-, y-, and z-coordinates of all the vertices in an order of their indices):

$$\mathbf{f_t} = \begin{bmatrix} v^t_{1,x} \\ v^t_{1,y} \\ v^t_{1,z} \\ \vdots \\ v^t_{V,z} \end{bmatrix}$$

The animation of the dynamic mesh is represented with a matrix $\mathbf{A}$ whose columns are the frames of the animation. Let V be the number of vertices of the mesh and F the number of frames in the animation. The matrix $\mathbf{A}$ that contains the entire animation is defined as follows:

$$\mathbf{A} = [\mathbf{f}_1 \ldots \mathbf{f}_F]$$

$\mathbf{A}$ is composed of $3V$ rows and F columns. Note that this matrix does not contain any information about the vertex connectivity. The purpose of our method is to compress the geometry only; the connectivity compression should be carried out separately using an existing approach [17]. Thus, our method is equally suitable for vertex animation data, such as marker data acquired from motion capture.

3.2. Main Idea

One of the main ideas of this paper is to cluster the frames according to their *pose similarity* and compress each cluster by using PCA in search of a smaller number of eigenbasis required (i) for each cluster and (ii) as a whole. In some sense, this idea can be seen as a temporal version of clustering-based compression [6]. To our knowledge, no previous work has addressed such an approach.

Later in this paper, experimental results show that our method is particularly beneficial when the number of frames is larger than the number of vertices and when the animation data contain repetitive motion over the sequence. We remark that this is usually the case, if not always, with most of the physically based or synthetic animation that are commonly practiced in various applications.

3.3. The Proposed Approach

Figure 2 illustrates an overview of our approach. The encoder is composed of three main parts: pose-similarity clustering followed by intracluster compression and frame coding.

Pose-similarity clustering: our technique is to group the frames whose poses are similar. Two frames are said to be similar if the Euclidean distance between the two frame vectors is small. A motion data is typically composed of a rigid motion and a pose deformation. To avoid the rigid motion to interfere in the clustering, we process the motion data so as to separate the rigid motion from the pose before the clustering step. Note that the frames belonging to the same cluster may not be contiguous. The detail description of the pose-similarity clustering is given in Section 4.

Intracluster compression: the second step of our method is to compress the animation data inside each cluster. Similarly to many previous works, our compression makes use of the PCA. The PCA method uses an orthogonal transformation to convert a set of observations of possibly correlated variables into a set of values of uncorrelated variables called PCs. One advantage of the PCA is to provide a compact representation of multidimensional data.

After applying the PCA on each cluster separately, a cluster is coded with a set of eigenvectors and a sparse representation (PC coefficients) of all the frames belonging to it. Because the clustering of frames based on pose similarity suggests a high degree of correlation among the

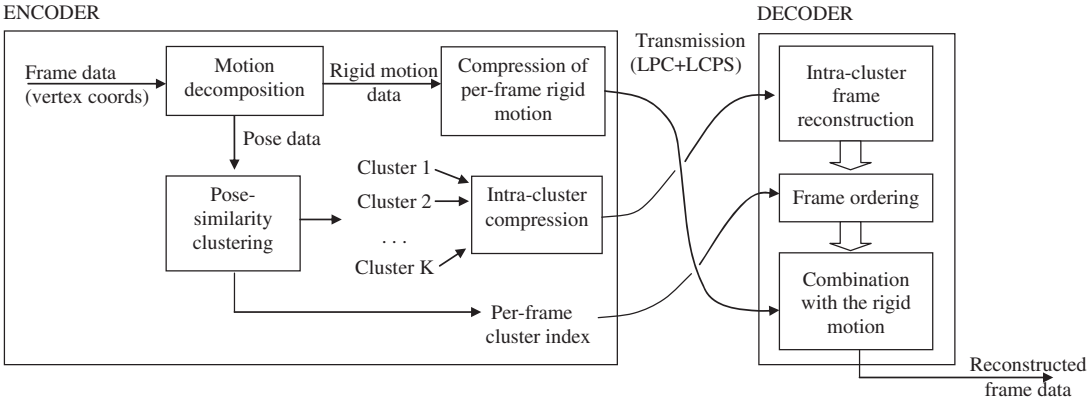

Figure 2. Overview of the method.

frames within cluster, the frames can be represented with a small subset of PC coefficients without degrading much the vertex trajectories. This step is described in Section 5.

Linear coding and linear coding based on pose similarity (*LCPS*): to further compress the data inside each cluster, we propose a linear coding based on pose similarity. The driving idea is to exploit the pose similarity that exists among frames within a cluster. The first fragments are compressed using existing techniques, and the subsequent fragments are represented as a weighted combination of the first frames. This step is described in Section 6.

Finally, we note that our method does not depend on the mesh connectivity. It can be used to compress any motion data with repetitive patterns (point clouds, rotation values, colors values, etc.).

4. POSE-SIMILARITY CLUSTERING

Our algorithm, instead of looking for coherencies in the local neighbourhood like many others, searches for pose coherence independently from temporal coherency. Intuitively, we would like to aggregate frames of similar pose into a cluster and compress them separately. Thanks to the within-cluster pose similarity, we expect to reduce the number of basis vectors required for each cluster and thus obtain better compression ratio. To measure the postural similarity among frames, we want to adopt a simple method based on vertex distances. However, rigid motion of the object may make this method obsolete in practice. Consider a character' walking animation sequence, for instance. The walking trajectory may contain arbitrary turns and orientation changes, making the global animation very complex. Now, if we can separate the global positions and orientations of the character from the articulated motion or "pose," we can easily see that the animated poses are more or less cyclic. For this reason, we propose to conduct frame alignment prior to clustering, by finding the rigid motion (translation and rotation transformation) that aligns each frame with the reference (first) one. As a result, we can increase the within-cluster similarity and thus, achieve highest compression ratio.

4.1. Frame Alignment

For every frame t, we compute the rotation $\mathbf{R}_t$ and the translation $\mathbf{T}_t$ with respect to the first one by using the method by Goryn and Hein [18]. All vertex coordinates of the frame, $\mathbf{v}_t$, are then transformed with $\mathbf{R}_t$ and $\mathbf{T}_t$, as follows:

$$\begin{bmatrix} p^t_{1,x} \\ p^t_{1,y} \\ p^t_{1,z} \end{bmatrix} = \mathbf{R}_t \cdot \begin{bmatrix} v^t_{1,x} \\ v^t_{1,y} \\ v^t_{1,z} \end{bmatrix} + \mathbf{T}_t$$
$$\cdots$$
$$\begin{bmatrix} p^t_{V,x} \\ p^t_{V,y} \\ p^t_{V,z} \end{bmatrix} = \mathbf{R}_t \cdot \begin{bmatrix} v^t_{V,x} \\ v^t_{V,y} \\ v^t_{V,z} \end{bmatrix} + \mathbf{T}_t$$

After the rigid transformation, all frames are placed in geometric centroid of the first one, that is, all frames share the same centroid. In addition, they have similar orientations. At this point, a frame at time t is represented by

$$\mathbf{f}_t = \begin{bmatrix} \boldsymbol{p}^t_{1,x} \\ \boldsymbol{p}^t_{1,y} \\ \boldsymbol{p}^t_{1,z} \\ \vdots \\ \boldsymbol{p}^t_{V,z} \end{bmatrix}$$

These rigid transformations are represented with six coefficients, three for the translation and three for the rotation. These coefficients are compressed separately from the vertices using the LPC.

4.2. Frame Clustering

In case of cyclic or repetitive motion, the animation data tend to exhibit a similar sequence of deformations several times. Subsequently, similar frames can be found at different time ranges, and they are not necessarily adjacent to

DOI: 10.1002/cav

one another. Our strategy is to cluster the frames such that frames with similar pose are put into the same cluster. Simple Euclidian distance $\|\mathbf{f}_j - \mathbf{f}_i\|$ is used to measure the pose similarity between two frame vectors $\mathbf{f}_i$ and $\mathbf{f}_j$.

We use *k*-means algorithm [19] to cluster the frame data into a number of clusters, so that the within-cluster variation is minimized. Given a set of frames ($\mathbf{f}_1$, $\mathbf{f}_2$, ..., $\mathbf{f}_F$), where each frame is a 3V-dimensional real vector, the *k*-means clustering aims to partition the F frames into k sets ($k \le F$) $\mathbf{S} = \{S_1, S_2, \ldots, S_k\}$ so as to minimize the within-cluster sum of squares:

$$\arg\min_{S} \sum_{i=1}^{k} \sum_{f \in S} \left\| \mathbf{f}_j - \boldsymbol{\mu}_i \right\|,$$

where $\boldsymbol{\mu}_i$ is the mean of frame vectors in S_i. The initialization method of our implementation of the *k*-means clustering is the random partition. The choice of the number of clusters is an important factor of the compression ratio. This is explained in Section 8.2.

We have tried other clustering methods, such as clustered PCA as proposed by Sattler *et al*. [6] and the mean-shift clustering. The main idea is to cluster the frames so that the frame coordinates in each cluster are calculated with the smallest possible number of PCs. Its main drawback is its large computation time because the method requires computing the PCA a multiple times. Moreover, the clustered-PCA method did not give noticeably better compression results, although it gave slightly smaller KG error. *k*-means and mean-shift clustering show similar computation time and performance. However, its computation time is about 100 times that of the *k*-means clustering. Because the *k*-means algorithm provides easier control on the number of clusters than the mean-shift clustering, it has been finally used in this work.

The number of clusters has been fixed manually, between two and four. As we will see in Section 8, the number of clusters is one factor of the compression performance. Automatically finding the optimal number of clusters is left as a future work.

After the clustering, the cluster indices of each frame in the animation sequence are stored ("per-frame cluster index" in Figure 3). This index is used to correctly reorder the frames at the decoding phase.

5. INTRACLUSTER COMPRESSION

Now that the sequence has been segmented into a number of clusters, we now focus on the problem of compressing the frame vectors in each cluster, which is based on PCA.

5.1. Principal Component Analysis on the Frame Vectors

As a result of the pose-similarity clustering, the original animation sequence is decomposed into a number of motion fragments, each belonging to a frame cluster. A motion fragment is a set of frames that are contiguous and belong to the same cluster.

From each frame cluster c ($c = 1, \ldots, K$) that consists of l_c frames, we build a $3V \times l_c$ matrix $\mathbf{F}_c$. We apply singular value decomposition to $\mathbf{F}_c$ to compute the eigenvectors and eigenvalues of $\mathbf{F}_c \cdot \mathbf{F}_c^T$. The original frame vectors are then represented by their PC coefficients that are obtained by projecting them onto their new basis, yielding a matrix of coefficients $\mathbf{W}_c = \mathbf{B}_c^T \mathbf{F}_c$, where $\mathbf{B}_c^T$ is composed of columns of eigenvectors (PCs), sorted in a decreasing order of corresponding eigenvalues. Trimming both the coefficients of $\mathbf{W_c}$ and the eigenvectors of $\mathbf{B}_c^T$ allows us to effectively reduce the number of floating-point numbers required to represent each frame. That is, by taking only the first r_c PCs, we represent each cluster by $\widetilde{\mathbf{B}}_c^T$ and $\widetilde{\mathbf{W}}_c$. $\widetilde{\mathbf{B}}_c^T$ is the set of the first r_c PCs and $\widetilde{\mathbf{W}}_c$ is a matrix $r_c \times l_c$ whose columns are the first r_c PC coefficients of each frame belonging to the cluster c.

$$\widetilde{\mathbf{W}}_c = [\widetilde{\mathbf{w}}_1, \ldots, \widetilde{\mathbf{w}}_t, \ldots, \widetilde{\mathbf{w}}_{l_c}] = \begin{bmatrix} w_1^1 & w_t^1 & w_{l_c}^1 \\ \vdots & \vdots & \vdots \\ w_1^r & w_t^r & w_{l_c}^r \end{bmatrix}$$

Consequently, each frame vector $\mathbf{f}_t$ is coded by a set of PC coefficients $\widetilde{\mathbf{w}}_t$.

Note that the number of PCs r_c differs from one cluster to another. We determine the number of PCs (eigenframes) in such a way that the variation of the frame vectors inside a cluster covered by the PC's is more than a threshold (99%, for instance). This threshold value is adjusted according to the user-specified error (Section 8 for the definition of error).

5.2. Cost Analysis

The main drawback of PCA-based methods is the significant computation time required for computing the PCs and the PC coefficients. In our Matlab [20] implementation of PCA, we calculate the eigenvectors (and eigenvalues) of the covariance matrix AA^{T}, which is a $m \times m$ symmetric matrix. Given the eigenvectors, sparse representation of each frame is obtained by projecting itself onto each of them. The total number of operations for these calculations is:

$$\mathbf{O}(m^2 n) = k \cdot m^2 n + k' \cdot n^3 \quad (1)$$

When the sequence is clustered into c clusters, the number of operations summed over all c clusters is given by

$$c\left(k \cdot m^2 \left(\sum n_i\right) + k' \sum n_i^3\right) < k \cdot m^2 n + k' \cdot n^3$$

We see that the number of operations is slightly reduced with clustering, but only by some constant factor. In Table 2, we show experimental results with different number of

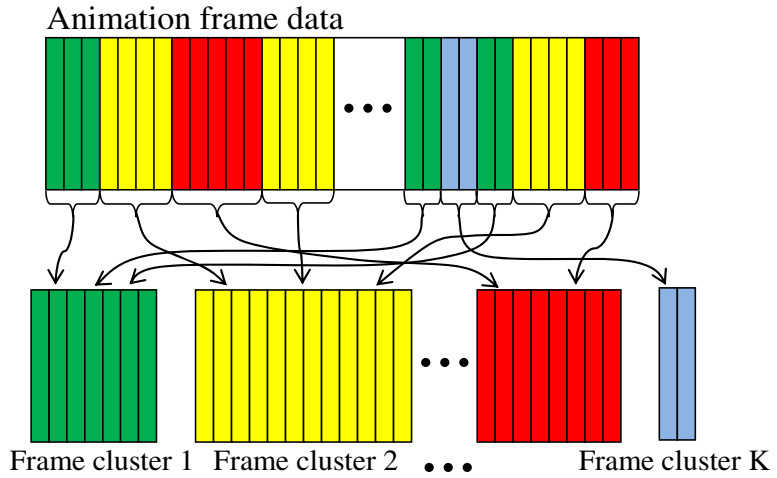

Figure 3. In pose-similarity clustering, frames with similar poses are clustered together.

clusters and PCs chosen, along with the corresponding compression ratio and the reconstruction error.

Finally, we use a uniform quantization to encode the selected PCs of each cluster. Quantization is the process of mapping a large set of input values to a smaller set. The number of possible output values is defined by the number of quantization bits q. The output values are generated by regularly sampling by 2^q times the input range. Smaller number of quantization bits reduces the size of the compressed data but increases the quantization error. In our experiments, the value q is between 8 and 16.

6. PREDICTIVE CODING

Now that we have built a set of pose clusters and the sparse representation of frame data for each cluster, we now look at ways to further exploit the pose redundancy that is present among the frames belonging to the same cluster. Our predictive coding scheme works as follows: given the sparse representation of motion fragments that are clustered by the pose similarity, we compress the first motion fragment by using the LPC. Then, each mesh frame in the subsequent motion fragments is encoded as a linear combination of a number of previously encoded frames (Figure 4). More specifically, each frame in the subsequent motion fragments is encoded as a linear combination of k frames from the first one. Because the mesh frames in the first fragment are similar to the subsequent fragments belonging to the same cluster, the representation based on linear combination is a good choice for reducing the reconstruction error without additional cost.

6.1. Linear Predictive Coding

The principle of LPC is to predict a value of a frame using the m immediate preceding frames. More precisely, the PC coefficients in a frame are predicted as a linear combination of those of the preceding frames. Thus, the LPC algorithm consists of coding (i) the first m frames of the motion fragment, (ii) the weights $a_1,, \ldots, a_j$ for each PC coefficient, and (iii) the residuals. The weights a_j are given to each of m preceding frames in determining the current frame, as given by

$$\boldsymbol{w}_t = \sum_{j=1}^{m} \mathbf{a}_j \boldsymbol{w}_{t-j}, \quad \boldsymbol{m}+1 \leq \boldsymbol{t} \leq \boldsymbol{l}_1,$$

$\boldsymbol{w}_{t-j}$ is a PC coefficient of the frame vector $\tilde{\mathbf{w}}_{t-j}$ and l_1 is the number of frames in the first motion fragment in the cluster. Note that the m weights are calculated for the entire fragment and for each PC coefficient by using the least-squares method. To increase the precision of the predictor, the residual (the difference between the predicted value and the actual value) is calculated for each frame and each PC coefficient. This value is then encoded using the uniform quantization.

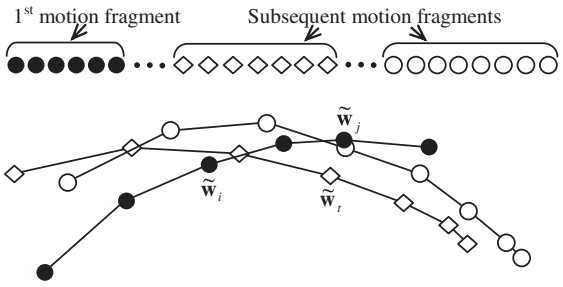

Figure 4. Motion fragments are represented in the principal component coefficient space. In our intracluster compression, the first motion fragment in the cluster is encoded by using the classical linear predictive coding. Subsequent fragments use previously encoded frames as interpolating frames. $\tilde{\mathbf{w}}_t$ is represented as a weighted combination of $\tilde{\mathbf{w}}_i$ and $\tilde{\mathbf{w}}_j$.

DOI: 10.1002/cav

6.2. Linear Coding Based on Pose Similarity

Suppose that we have the first motion fragment encoded with the LPC. The next step is to construct a predictor that estimates each frame in subsequent motion fragments by using those of the first one. More precisely, the PC-coefficient vector of each frame in the subsequent motion fragments is encoded as a linear combination of the PC-coefficient vectors of the k frames belonging to the first fragment. Unlike the LPC method, the interpolating frames are not those immediately preceding the frame to interpolate; they are chosen in a way that the error of the predictor is the smallest possible. In addition, the weights are calculated for each frame to predict.

The algorithm to select the k interpolating frames from the first fragment works as follows. Let S_{FG} be the set of frames of the first fragment and S_{I} be the interpolating frames selected from S_{FG}; S_{I} is initially empty. We first choose the frame of S_{FG} and whose coefficient vector is the most parallel to the one we want to code. This vector is put into S_{I}. Next, we compute the weight of the vector in S_{I} such that their weighted summation is as close as possible to the frame to code. We compute the residual vector, which is the difference between the interpolated vector and the vector to encode. This vector is used the guide the selection of the next interpolating frame at the next iteration, that is, we choose the next interpolating frame that is the most parallel to the residual vector. The number of iteration is fixed to two (Figure 5).

The pseudo-code of the algorithm is as follows:

The key point is to choose the interpolating frames and their weight in a way that the norm of the residual vector $\mathbf{d}$ becomes the smallest possible over the iterations (steps 2 and 3). This algorithm works best when the animation is composed of repetitive frames. If it is not the case, it could happen that the norm of $\mathbf{d}$ does not reduce over the iterations; the dot product of $\mathbf{d}$ with any vector of $S_{\mathrm{FG}} = \{\tilde{\mathbf{w}}_1, \ldots, \tilde{\mathbf{w}}_{l_1}\}$ is equal to 0. The consequence is a degraded compression ratio.

Contrary to the LPC method in which the interpolating frames are determined implicitly, our method requires explicit coding of the indices of the frames that are used for the predictor. One may think that these additional indices may degrade the compression. In fact, these frame indices can be encoded efficiently. Let be l_1 the number of frames in the first fragment. The number of bytes required to store a frame index is $\lceil \log_2(l_1) \rceil / 8$ For instance, 7 bits are sufficient for clusters whose first fragment is composed of 128 frames.

Similarly to the LPC method, we compute the residual values, which are the differences between the predicted values. We then encode these residual values by using uniform quantization.

The main advantage of LCPS over the LPC is that the residual values are much smaller than those of the LPC; the number of bits needed for the quantization of these values is much smaller. This results into a better compression ratio.

7. RECONSTRUCTION

We now focus our discussion on how to reconstruct the original frame data from the encoded data. First, all clusters are decoded separately, according to the following steps: For each cluster, the LPC coefficients and the corresponding residual values are used to compute the PC coefficients of the first motion fragment. Then, the PC coefficients of the remaining frames in each cluster are reconstructed using the first fragment and the coefficients with the residuals of the LCPS. Once all the PC coefficients have been calculated, we compute the frame coordinates by using these PC coefficients and the PCs (eigenframes). This process is carried out separately for each cluster. Next, we reconstruct the original motion by rearranging the frames by increasing order of their

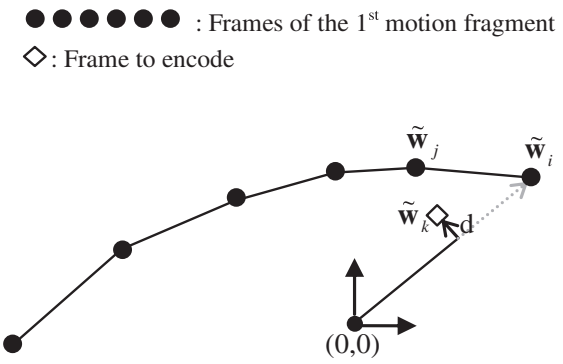

Figure 5. To encode $\tilde{\mathbf{w}}_t$, we first choose $\tilde{\mathbf{w}}_i$ and compute its coefficient b_i such that $\|\tilde{\mathbf{w}}_t - b_i\tilde{\mathbf{w}}_i\|^2$ is minimized. We then compute the residual $d = \tilde{\mathbf{w}}_t - b_i\tilde{\mathbf{w}}_i$ and choose $\tilde{\mathbf{w}}_j$, which is the most parallel to the residual vector. Finally, we compute the coefficients such that $\|\tilde{\mathbf{w}}_t - (b_i\tilde{\mathbf{w}}_i + b_j\tilde{\mathbf{w}}_j)\|^2$ is minimized.

Algorithm LCPS

Input: $\tilde{\mathbf{w}}_t$, the frame to predict and $S_{FG}= \{\tilde{\mathbf{w}}_1, \ldots, \tilde{\mathbf{w}}_{l_1}\}$, the frames of the first fragment.
Output: S_I, the set of k interpolating frames selected for the predictor and their corresponding weights $\{b_1, \ldots, b_k\}$
Step1: Let $\mathbf{d}=\tilde{\mathbf{w}}_t$ and $S_I=\phi$.
Step 2: Find a vector $\tilde{\mathbf{w}}_i \in S_{FG}$ such that the dot product $\left\| \frac{\mathbf{d}}{\|\mathbf{d}\|} \cdot \frac{\tilde{\mathbf{w}}_i}{\|\tilde{\mathbf{w}}_i\|} \right\|$ is maximal. Add $\tilde{\mathbf{w}}_i$ to S_I.
Step 3: Compute the weights such that $\left\| \tilde{\mathbf{w}}_t - \sum_{\tilde{\mathbf{w}}_i \in S_I} (b_i \tilde{\mathbf{w}}_i) \right\|^2$ is solved in the least squares sense. $\mathbf{d}$ is assigned to a new value $\mathbf{d} = \tilde{\mathbf{w}}_t - \sum_{\tilde{\mathbf{w}}_i \in S_I} (b_i \tilde{\mathbf{w}}_i)$.
Step4: Iterate steps 2-3 until the k best interpolating frames are found.

frame index. The last step is to decompress the six coefficients of the rigid motion and apply this rigid motion on the frames.

8. RESULTS

In this section, we report on experiments with the proposed compression method. All the algorithms, including other methods we have chosen for the comparison, have been implemented as Matlab scripts [20].

8.1. Animation Data

For the comparison, we used three different types of animation sequences: "galloping horse," "face," "flag," and "woman" (Table 1). The horse animation has been generated using the skeleton-driven deformation and the flag with a physics-based simulator. The face data is from a motion capture system [21] and contains relatively fewer number of vertices (markers) and high number of frames because of the very high frame rate (120Hz).

8.2. Compression and Reconstruction

Table 2 shows the compression and reconstruction results with different numbers of clusters. For each animation, the compression ratio has been calculated with varying numbers (one to five) of clusters while keeping constant KG error. We observe that the compression of the "horse" and "flag" animations with four clusters returns the best compression ratio. For the "face" and "women" animations, the best compression ratio is obtained with three clusters. Our experiments show that there is an optimal number of clusters for each data. As we decrease the number of clusters, a higher number of PCs is required to maintain the same KG error, which in turn degrades the compression ratio. Similarly, increasing the number of clusters degrades the compression ratio. This is because clusters require a minimum number of PCs, whatever is the size of the cluster. This implies that a compression with a higher number of clusters implies higher number of PCs, resulting into a degraded compression ratio. Note that the compression is not only a function of the number of PCs but also the number of frames belonging to each cluster. This explains why we obtain better compression ratio on the face with three clusters whose total number of PCs (138) is larger than that of one cluster (68).

The timings in Table 2 include the runtime for frame clustering, PCA, quantization, and LCPS. Our method has implemented using Matlab running on Intel Core 3.40 GHz PC with 16.0 GB RAM. Detailed timings are listed in Table 3.

Table 1. Models used in our experiments.

Name	# of vertices	# of triangles	# of frames	Creation method
Horse	1000	1990	200	Synthetic
Face	80	139	1001	Motion capture
Flag	121	200	1068	Physics-based animation
Woman	1580	3156	558	Motion capture

8.3. Comparison with Other Methods

For the comparison, we have chosen the methods of same class, that is, PCA-based methods. They are the spatial segmentation method by Sattler *et al.* [6] and the PCA combined with the LPC method by Karni and Gotsman [4]. Like ours, they concentrate on the compression of frame data and do not perform compression on the connectivity data. Compared with other methods, the main advantage of PCA-based methods is that they do not use the mesh connectivity for the compression.

Sattler *et al.* [6] presented a compression method that clusters vertices with similar trajectories. With clustered PCA, it segments the given mesh surface into segments so that average of per-cluster reconstruction error is minimized. Once the clustering is done, each cluster is encoded with a number of eigenvectors and PC coefficients corresponding to each vertex trajectory. The animation

Table 2. Compression performance with different number of clusters.

Name	# of clusters	KG error (%)	# of PCs	Total # of PCs	Compression ratio (%)	Timing (in seconds)
Horse	1	0.0290	{23}	23	0.2331	2.38
	2	0.0379	{9, 13}	22	0.1824	2.01
	4	0.0259	{4, 3, 4, 9}	20	0.1617	1.22
Face	1	0.0809	{68}	68	0.3513	41.88
	3	0.0768	{58, 39, 41}	138	0.3351	19.29
	5	0.0771	{28, 38, 35, 50, 41}	192	0.3972	15.06
Flag	1	0.1884	{159}	159	0.5869	10.10
	3	0.1837	{105, 118, 58}	281	0.5493	15.72
	4	0.1815	{111, 79, 36, 88}	314	0.5416	13.45
Woman	1	0.0535	{55}	55	0.5168	12.86
	2	0.0520	{55, 5}	60	0.3812	15.58
	3	0.0538	{51, 10, 15}	76	0.4596	14.30

DOI: 10.1002/cav

Table 3. Computation time (in seconds) of the three methods with two models.

	Our method					Sattler *et al.*	Karni *et al.*
	Clustering (# of clusters)	PCA	Quantization	LCPS	Total		
Horse	0.20 (4)	0.14	0.09	0.16	0.59	236	3.02
Face	0.06 (3)	0.18	0.17	3.20	3.61	185	2.42
Flag	0.18 (4)	0.32	6.56	3.16	10.22	826	24.21
Woman	0.57 (2)	1.74	1.48	0.58	4.27	2904	13.05

The KG_{error} was set to 0.4%.

data are further compressed by applying the PCA on the eigentrajectories and the uniform quantization.

Karni *et al.* [4] have presented another compression method in which the PCA method is applied on the trajectories of vertices. Unlike the method by Sattler *et al.* and ours, they do not use any clustering. The vertex trajectories are further compressed by encoding them using the LPC method.

In all methods, the uniform quantization has been used to encode the PCs, the PC coefficients, and the residual values.

To compare the reconstruction error of the different methods, we use the KG error measure in percentage proposed by Karni *et al.* [4]:

$$KG_{\text{error}} = 100 \cdot \frac{\left\|\mathbf{A} - \widetilde{\mathbf{A}}\right\|}{\|\mathbf{A} - \mathbf{E}(\mathbf{A})\|'}$$

$\mathbf{A}$ is a $3n \times F$ matrix containing the original animation sequence, n being the number of vertices and F the number of frames. $\widetilde{\mathbf{A}}$ is the same animation after the compression and decompression stages. $E(\mathbf{A})$ is a matrix of the same dimensions as $\mathbf{A}$, in which the values have been replaced by per-frame averages. The reconstruction error is calculated with Frobenius norm.

To measure the compression efficiency, we compute the ratio (in percentage) between the size of the compressed data and the size of the uncompressed data.

The horse model is an animated mesh with exact repetitive motion. In other words, the same sequence of deformation is repeated several times. Our algorithm performs the best for high quality compression. As shown in Figure 6, for a KG_{error} of 0.27%, the compression ratio is 22% better than the method by Sattler *et al.* This means that our method could successfully identify the redundancy among frames and use this information to obtain a better compression ratio. For the compression of the "walking woman," our method tends to perform better compared with the two others for small KG_{error} (Figure 7).

Figure 8 shows the compression results for the face model; the compression ratio of our method is approximately four times better that that of Karni *et al.* for a KG_{error} equal to 0.25%. Unlike the horse model, the "face" and the "flag" sequences do not contain any two identical frames; all the frames have different shape.

As shown in Figure 9, the compression ratio of the flag is lower than the two other models. This is because this animation sequence is much more complex. The movement of the flag is highly dynamic because of the wind. For a KG_{error} of 0.27%, the compression ratio of our method is approximately 30% better than that of the two other methods. Similarly to the "face" animation sequence, this sequence does not contain any two identical frames.

8.4. Computation Time

Table 3 summarizes the computation times of our method along with the two other methods.

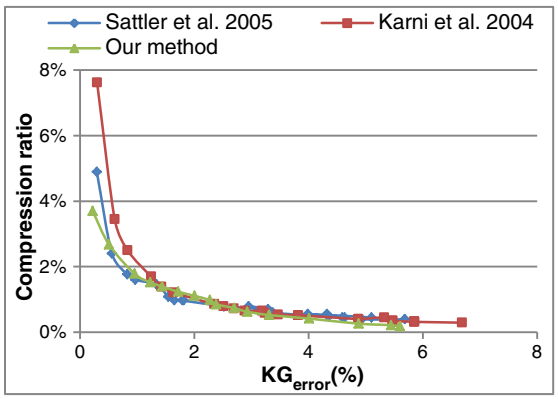

Figure 6. Comparison of performance for the "horse" model.

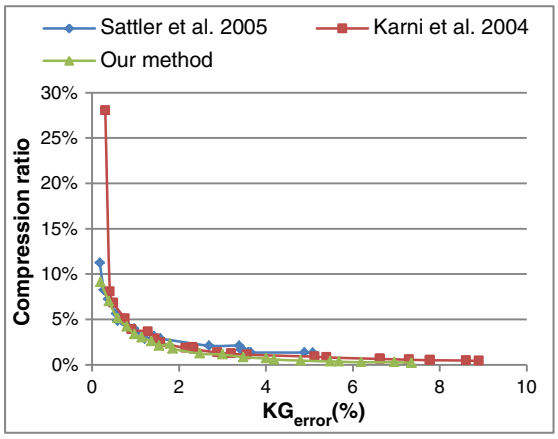

Figure 7. Comparison of performance for the "walking woman" model.

As shown in Table 3, our method has lower computation time compared with the two others, except for the face model. There are two factors that intervene in the computation time. The first one is obviously the length of the animation; longer animations require more computation time than the shorter ones. The second factor is the value of the KG error. Compression with small KG error requires using larger number of PCs, which in turn makes the quantization and LCPS steps slower because of the size of the data.

One may notice that the timing shown in Table 3 are large compared with those provided by Karni *et al*. and Sattler *et al*. This is because all our algorithms have been implemented as Matlab scripts. Because these scripts are not compiled but interpreted, the execution time is much longer than that of the same code written in C++. The purpose of this table is to compare the computation times rather than to provide absolute timing.

8.5. Length of the Animation Sequence

As expected, our method shows best performance for long animation sequences, where the number of frames is much larger than the number of vertices; it outperforms the two other methods when the number of frames is large enough so that the animation exhibits repetitive motions. On the other hand, the performance of our method is lower than the two others for animation sequences with no repetitive motion.

To show the effect of the animation sequence length, we have compared the compression algorithms for the horse sequence truncated at 5%. The truncated animation sequence contains the first 10 frames of the original horse animation. As shown in Figure 10, our method does not perform any better than the two others for the KG_{error} equal to 0.27%.

Our approach has several other limitations. It could sometimes produce discontinuous frames, at boundaries of motion fragments. Such artefact can be observed more often with high compression ratio. Thus, future work could focus on reducing this problem of possible frame discontinuity. Another limitation is the number of frame clusters that need to be defined manually. Future work could be on automatically determining the optimal number of frame clusters towards best compression ratio with minimal error.

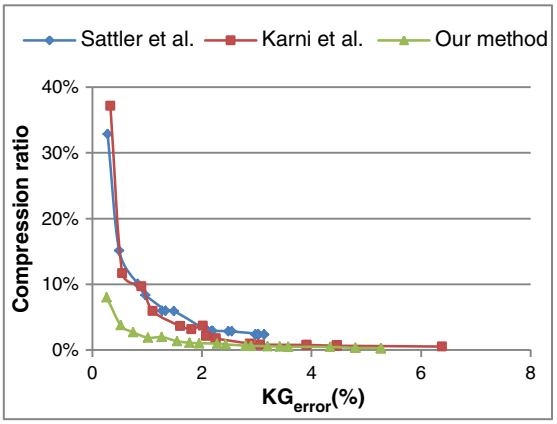

Figure 8. Comparison of performance on the "face" model.

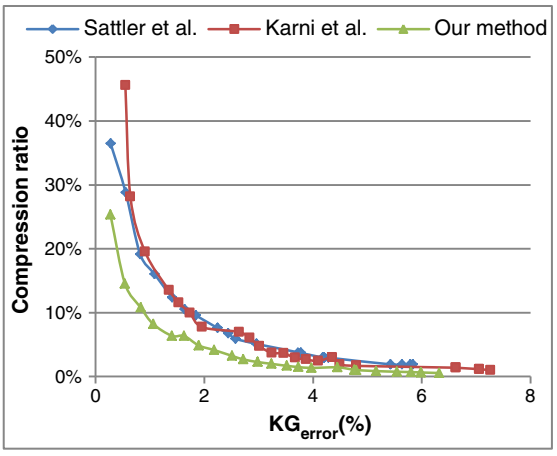

Figure 9. Comparison of performance for the "flag" model.

9. CONCLUSION

In search of good capture of spatio-temporal coherency, we have introduced a new alternative approach to the compression of 3D animation data. The key to efficient compression is the aggregation of similar poses into frame clusters, which allows us to reduce the number of PCs required for each frame.

Further, we perform intracluster compression based on linear coding. Because every motion fragment within a cluster exhibits similar poses, conventional LPC can be replaced by key frame-based linear coding, to achieve minimal reconstruction error. Subsequently, we obtain better compression ratio for a given reconstruction error than other comparable methods.

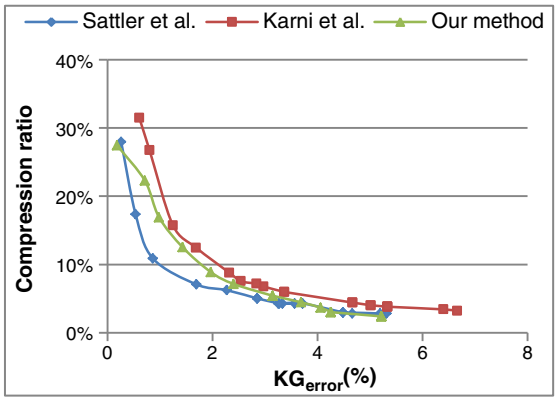

Figure 10. Comparison for the horse model whose animation sequence has been truncated at 5%.

Comp. Anim. Virtual Worlds 2013; **24**:365–375 © 2013 John Wiley & Sons, Ltd.
DOI: 10.1002/cav

ACKNOWLEDGEMENTS

This work has been supported by the project SHARED (No. 10-CHEX-014-01) funded by the French National Research Agency.

REFERENCES

1. Sumner RW, Popovic J. Mesh data from deformation transfer for triangle meshes. *ACM Transactions on Graphics* 2004; **23**(3): 399–405.
2. Kruskal JB, Wish M. Multidimensional Scaling. Sage: Beverly Hills, California, 1978.
3. Alexa M, Muller W. Representing animations by principal components. *Computer Graphics Forum* 2000; **19**(3): 411–418).
4. Karni Z, Gotsman C. Compression of soft-body animation sequences. *Computers and Graphics* 2004; **28**(1): 25–34.
5. Váša L, Skala V. COBRA: compression of the basis for PCA represented animations. *Computer Graphics Forum* 2009; **28**(6): 1529–1540.
6. Sattler M, Sarlette R, Klein R. Simple and efficient compression of animation sequences. In Proceedings of the ACM SIGGRAPH/Eurographics Symposium on Computer Animation, 2005, Anjyo K, Faloutsos P (eds). ACM press: New York, 2005; pp. 209–217.
7. Ramanathan S, Kassim AA, Tan T-S. Impact of vertex clustering on registration-based 3D dynamic mesh coding. *Image and Vision Computing* 2008; **26**(7): 1012–1026.
8. Mamou K, Zaharia T, Preteux F. A skinning approach for dynamic 3d mesh compression. *Computer Animation and Virtual Worlds* 2006; **17**(3-4): 337–346.
9. Mamou K, Zaharia T, Preteux F. Famc: the mpeg-4 standard for animated mesh compression. *15th IEEE International Conference on Image Processing*, 2008; 2676–2679.
10. Marpe D, Wiegand T, Schwarz H. Context-based adaptive binary arithmetic coding in the h.264/avc video compression standard. *IEEE Transactions on Circuits Systems for Video Technology* 2003; **13**(7): 620–636.
11. MPEG4 Part 16 AMD2: frame-based animated mesh compression. ISO/IEC JTC1/SC29/WG11, 2007.
12. Stefanoski N, Ostermann J. SPC: fast and efficient scalable predictive coding of animated meshes. *Computer Graphics Forum* 2010; **29**(1): 101–116.
13. Ibarria L, Rossignac J. Dynapack: space-time compression of the 3d animations of triangle meshes with fixed connectivity. In Proceedings of the 2003 ACM SIGGRAPH/Eurographics Symposium on Computer Animation, Eurographics Association, Aire-la-Ville, Switzerland, 2003; 126–135.
14. Stefanoski N, Ostermann J. Connectivity-guided predictive compression of dynamic 3d meshes. In Proceedings of ICIP'06 – IEEE International Conference on Image Processing, 2006; 2973–2976.
15. Amjoun R, Straßer W. Single-rate near lossless compression of animated geometry. *Computer-Aided Design* 2009; **41**(10): 711–718.
16. Váša L, Skala V. Geometry-driven local neighbourhood based predictors for dynamic mesh compression. *Computer Graphics Forum* 2010; **29**(6): 1921–1933.
17. Gumhold S, Straßer W. Real time compression of triangle mesh connectivity. SIGGRAPH 1998, Computer Graphics Proceedings, ACM Press, 1998; 133–140.
18. Goryn D, Hein S. On the estimation of rigid body rotation from noisy data. *IEEE Transactions on Pattern Analysis and Machine Intelligence* 1995; **17**(12): 1219–1220.
19. Kanungo T, Mount DM, Netanyahu NS, Piatko CD, Silverman R, Wu AY. An efficient k-means clustering algorithm: analysis and implementation. *IEEE Transactions on Pattern Analysis and Machine Intelligence* 2002; **24**(7): 881–892.
20. Matlab, http://www.mathworks.fr/
21. Vicon motion capture system. http://www.vicon.com

Authors' biographies:

Guoliang Luo is a PhD candidate in computer science at the University of Strasbourg, France. He obtained his master's degree in computer science from the Uppsala University, Sweden. His current research interests include computer graphics, segmentation of mesh sequences, and compression of 3D animations.

Frederic Cordier is an associate professor at the Université de Haute Alsace, France. His research interests include 3D modeling and texturing, human-computer interaction, and physics-based simulation. He holds a PhD in computer science from the University of Geneva, Switzerland.

Hyewon Seo is a CNRS researcher at the University of Strasbourg, France. Her research interests include imaging, visual simulation, human–computer interaction, and VR. She has graduate degrees in computer science from the University of Geneva and KAIST.