Efficient Compression of 3D Dynamic Mesh Sequences

Rachida Amjoun and Wolfgang Straßer
WSI / GRIS
University of Tübingen, Germany
{amjoun, strasser}@gris.uni-tuebingen.de

Figure 1 : Sample frames of the animations used for the analysis. From left to right: dance with 14, dolphin with 9, chicken with 10 and cow with 6 clusters. Each cluster is colored differently and encoded separately.

ABSTRACT

This paper presents a new compression algorithm for 3D dynamic mesh sequences based on the local principal component analysis (LPCA). The algorithm clusters the vertices into a number of clusters using the local similarity between the trajectories in a coordinate system that is defined in each cluster, and thus transforms the original vertex coordinates into the local coordinate frame of their cluster. This operation leads to a strong clustering behavior of vertices and makes each region invariant to any deformation over time. Then, each cluster is efficiently encoded with the principal component analysis. The appropriate numbers of basis vectors to approximate the clusters are optimally chosen using the bit allocation process. For further compression, quantization and entropy encoding are used. According to the experimental results, the proposed coding scheme provides a significantly improvement in compression ratio over existing coders.

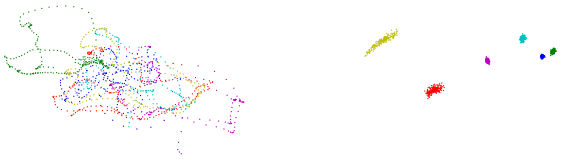
Keywords: 3D animation, animated mesh compression, segmentation, PCA, rate-distortion optimization.

1 INTRODUCTION

Animated meshes are commonly used in computer games, computer generated movies, and many scientific applications. The animations in these applications are often complex, nonlinearly generated and contain large geometric datasets. They often consist of many frames, each of which stores an own mesh. Even if key frame animations are used, they are too voluminous to be stored. Often the meshes differ only slightly between neighboring frames, leading to a large redundancy between frames and between neighboring vertices in the same frame. Therefore, it is important to develop compact representations that significantly reduce the storage space of animated models and facilitate their transmission over networks. Moreover, we need compression algorithms that allow for small compressed representations that maintain good visual fidelity.

Many existing compression schemes are restricted to animated meshes that do not change the topology from frame to frame so that the topology can be compressed once and only the vertex positions need to be compressed for the individual frames. Here, we distinguish between four methods: predictive based methods [JCS02, IR03], PCA based representations [AM00, KG04, SSK05], wavelet based techniques [GK04, PA05] and clustering-based approaches [Len99, ZO04].

In this paper, we present a new PCA-based technique as extension of the work [ASS06]. The advantage of using PCA is that it captures the linear correlations present in the dataset. The set of vertices can be represented by very few components and coefficients depending on the user's desired visual quality. The PCA is a good compressor for rigid motion and provides a more compact representation for temporally-invariant meshes. In many applications, however, animated meshes exhibit highly nonlinear behavior, which is globally difficult to capture using standard PCA. Locally, the neighboring vertices have a strong tendency to behave and to move in a similar way. The nonlinear behavior can therefore be described in a linear fashion by grouping the vertices of similar motion into clus-



(a) (b)

Figure 2: *The position of six different vertices over time (illustrated with different colors) are represented with global coordinates (a) and local coordinates (b)(dance animation).*

ters or by segmenting the mesh into meaningful parts. Then a PCA is performed in each group. The process to construct this representation is called Local Principal Component Analysis.

On the other hand, introducing a local coordinate frame (*LCF*) in each cluster may lead to extra clustering of the coordinates before performing the PCA. If the segmentation or clustering process is efficient then it would be highly probable that these coordinates change very slightly relative to the coordinate frame of their cluster. Of course, the number of clusters/segments will also affect the compression. If the number of clusters is very small, then a cluster might contain vertices that have different behaviors. To overcome this problem one might possible improve on the present approach by automatizing the selection of the number of clusters.

Figure 2 demonstrates the idea of using local coordinate systems. Figure 2 (a) shows the path of six points of a dance animation in the world coordinates. Note the highly nonlinear behavior of the trajectories. Figure 2 (b) shows the path of the points using a local coordinates. Note the relative small changes and the tendency of the trajectory of individual points to cluster.

In our approach, we perform a PCA on the local coordinates rather than the world coordinates. The advantage of combining PCA with the *LCF* is now obvious: if the motion of a group of vertices is rigid in the world coordinates, the positions of the vertices are slightly invariant relative to their *LCF*. Therefore, performing a PCA in these invariant groups of vertices leads to a more compact representation than the original data, and a large number of PCA coefficients are close to zero.

1.1 Overview

We propose a new compression algorithm for animated meshes of fixed number of vertices based on LPCA. Our main contribution is to cluster the mesh vertices using the local similarity of trajectories. The original vertex coordinates are transformed into several *LCFs* defined by seed triangles. One *LCF* (one seed triangle) is associated with each cluster. The vertices are then clustered depending on the variation of their local coordinates in each *LCF*. Thus, each vertex is added to the cluster where the vertex coordinates have the smallest variation over time. This automatically "transforms" the nonlinear behavior of the original vertices into the clustering behavior which is very well compressable. The vertex positions will tend to cluster around the same position over time (see Figure 2(b)). Thus, the clusters themselves are almost invariant to any deformation. A PCA is then performed on each cluster such that the local coordinates of the vertices are transformed into another basis which allows for very efficient compression.

Our clustering process produces clusters of different sizes. If one chooses a fixed number of basis vectors for all clusters, then there may be too few eigenvectors to recover the clustered vertices at a desired accuracy and eventually too many eigenvectors for other clusters (which we call *underfitting* and *overfitting*, respectively). Moreover, the number of bits needed to encode the unnecessary basis vectors in *overfitting* cases may be better allocated for other clusters in *underfitting* cases. Therefore the selection of the best number of basis vectors to be extracted from animation data is necessary to properly recover the original data of each cluster with a certain accuracy. We introduce a rate distortion optimization that trades off between rate and the total distortion. We call our approach Relative Local Principal Component Analysis (RLPCA) compression. We use the term *Relative* as the LPCA is performed in local coordinates. Our Algorithm achieves an increased compression performance, is computationally inexpensive (compared to a PCA for the full mesh) and is well suited for progressive transmission.

2 RELATED WORK

Static Meshes A large number of compression techniques have been developed for static meshes. Deering [Dee95] was the first to publish work on geometry compression for triangle meshes. Then, a succession of efficient schemes were proposed for both connectivity and geometry compression [TR98, GS98, TG98, IA02]. Progressive compression techniques [Hop96], which enable a mesh to stream from a server to a client have also been proposed. Recently some comprehensive surveys of the developed techniques have been provided [Ros04, AG05, JPK05].

Animated Meshes Recently, research has started to focus on animated meshes with fixed connectivity. Lengyel [Len99] introduced the first work on animated geometry compression. He partitioned the mesh into submeshes and described the motion of the submeshes by rigid body transformations. The rigid body transformation of a submesh was thereby estimated to best match the trajectories of its vertices. His approach is very effective when large parts of an animated model can be described well by rigid body transformations.

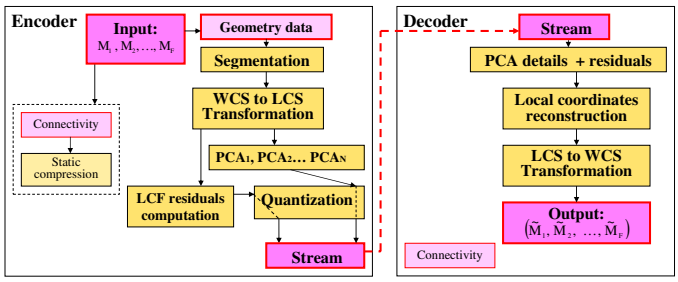


Figure 3: *Overview of the compression / decompression pipeline.*

Jinghua et al. [ZO04] used an octree to spatially cluster the vertices and to represent their motion from the previous frame to the current frame with a very few number of motion vectors. The algorithm predicts the motion of the vertices enclosed in each cell by tri-linear interpolation in the form of weighted sum of eight motion vectors associated with the cell corners. The octree approach is later used by K. Mueller et al. [MSK+05] to cluster the difference vectors between the predicted and the original positions.

Alexa et al. [AM00] used PCA to achieve a compact representation of animation sequences. The PCA coefficients were shown to be well compressable. Karni and Gotsman [KG04] improved this method by applying second-order Linear Prediction Coding (LPC) to the PCA coefficients such that the large temporal coherence present in the sequence is further exploited. Sattler et al. [SSK05] introduced the clustered PCA. The mesh is segmented into meaningful clusters which are then compressed independently using a few PCA components only.

Prediction techniques can also be used to efficiently compress animated meshes. Assuming that the connectivity of the meshes doesn't change, the neighborhood in the current and previous frame(s) of the compressed vertex is exploited to predict its location or its displacement [JCS02, IR03]. The residuals are compressed up to a user-defined error.

Guskov et al. [GK04] used wavelets for a multiresolution analysis and exploited the parametric coherence in animated sequences. The wavelet detail coefficients are progressively encoded. Payan et al. [PA05] introduced the lifting scheme to exploit the temporal coherence. The wavelet coefficients are thereby optimally quantized.

Segmentation Mesh segmentation has recently become useful for many applications in geometry processing. In the context of compression, segmentation is often used to decrease the computational costs as well as to preserve the global shape of the mesh because some compression algorithms (e.g. PCA for a full mesh) can destroy important features of the mesh.

To find the vertices that have similar motion, Lengyel [Len99] proposed that one select a set of seed triangles randomly and compared their trajectories. Triangles with a similar motion are combined. Then the vertices are associated with the triangle whose trajectory best fit theirs. Sattler et al. [SSK05] proposed that one cluster the trajectories of vertices using Lloyd's algorithm in combination with PCA. In the both cases, the segmentation is computationally expensive.

3 ANIMATION COMPRESSION

In this section, we describe in detail the core of our compression algorithm for the motion of vertices of animated triangle meshes. An overview of compression and decompression pipeline is illustrated in Figure 3.

Given a sequence of triangle meshes $M_i, i = 1,..,F$ of constant connectivity with V vertices and F frames (meshes), we first construct N *LCFs* in each frame, then group the mesh vertices into N clusters, where each cluster contains $V_i, i = 1,..,N$, vertices.

3.1 Local Coordinate System

Expressing the vertex locations in a *LCF* is an optimal way of exhibiting clustering behavior. It makes the clusters quite invariant over time to any rotation and/or translation. This representation can be very compressable with the PCA. This is the key feature of our algorithm.

Figure 4 illustrates the *LCF* that we use in our algorithm during and after segmentation. We consider that each cluster is initialized with seed triangle $(\mathbf{p_1}, \mathbf{p_2}, \mathbf{p_3})$. Each cluster $\mathscr{C}_i$ has its own *LCF* defined on the seed triangle. The origin $\mathbf{o}$ is the center of one of its three edges (typically $(\mathbf{p_1}, \mathbf{p_2})$), the x-axis (red arrow) points down the edge $(\mathbf{p_1}, \mathbf{p_2})$, the y-axis (green arrow) is orthogonal to the x-axis in the plane of the seed triangle and the z-axis is orthogonal to the x- and y-axis. The transformation of a point $\mathbf{p}$ to its local coordinate system $\mathbf{q}$ can be accomplished by an affine transformation with a translation $\mathbf{o}$ and a linear transformation $\mathbf{T}$ (orthonormal matrix):

$$\mathbf{q} = \mathbf{T}(\mathbf{p} - \mathbf{o})$$

In our algorithm, for each frame f $(1 \leq f \leq F)$ and for each frame cluster $G_i^f \in \mathscr{C}_i$ $(1 \leq i \leq N)$, we computed $\{\mathbf{T}_i^f, \mathbf{o}_i^f\}$ from the points of the seed triangle $(\mathbf{p}_1^{i,f}, \mathbf{p}_2^{i,f}, \mathbf{p}_3^{i,f})$.

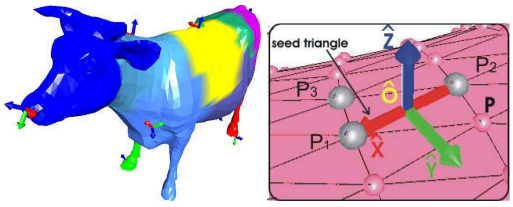


Figure 4: *Illustration of the local coordinate frame*

3.2 Segmentation based on Clustering

Our segmentation algorithm starts with several seed triangles upon which the *LCFs* are constructed. Then the clustering is obtained by assigning the vertices to the seed triangle in whose local coordinate frame they have minimal coordinates variation across the F frames. The clustering process consists of the following steps:

Initialization: Initializes the N cluster S_i, $i = 1,...,N$, to be empty. All vertices are unvisited.

Seed Selection: Selects N seeds using the far distance approach [YKK01]. The first seed is selected as the vertex corresponding to the largest euclidian distance from the geometrical center of all vertices in the first frame. The next seeds are selected sequentially until all N seeds are selected. Each seed is selected to be the vertex with the farthest distance from the set of already selected seeds. We associate with each seed one of its incident triangles and call this triangle the seed triangle. The regions are initialized with their three incident vertices denoted as $(\mathbf{p}_1^{i,f}, \mathbf{p}_2^{i,f}, \mathbf{p}_3^{i,f})$ the three vertices of seed triangle of i-th cluster in the f-th frame.

Local Frame Construction: A local coordinate frame is constructed for each seed triangle (see section 3.1).

Vertex clustering: Given an unvisited vertex $\mathbf{p}_k^f$, we do the following: Transform its world coordinates into the N local coordinate frames constructed in each frame f, so: $\{\mathbf{q}_k^{1,f}, \mathbf{q}_k^{2,f}, ..., \mathbf{q}_k^{N,f}\}$, $(f = 1,...,F)$, compute the total deviation (motion) of the vertex between each two adjacent frames f and $f-1$ in euclidian space:

$$\theta_{k,i} = \sum_{f=1}^{F} \|\mathbf{q}_k^{i,f} - \mathbf{q}_k^{i,f-1}\|^2$$

$\theta_{k,i}$ represents the total motion of the vertex k in the *LCF* associated with the cluster i. A small value means that the vertex position has motion that is similar to $\mathscr{C}_i$. Thus the vertex should belong to the cluster i for which the deviation is very small, note i_{min}:

$$i_{min} := argmin_{1 \le i \le N}\{\theta_{k,i}\}$$

We iterate over all vertices, adding the unvisited vertex whose local coordinates are almost invariant in the *LCF* to the cluster $\mathscr{C}_i$ and store its local coordinates for the next step (compression). The iteration stops if no more candidate vertices exist. When a vertex is added to a cluster, it is marked as visited. We end up with N clusters that have V_i vertices each.

Our algorithm provides a simple and effective way of efficiently clustering mesh vertices. The results of the segmentation technique can be seen in figure 1.

3.3 Compression

Once the mesh vertices are clustered, their coordinate systems need to be encoded using PCA. In order to be able to transform back to the world coordinates during the decoding step, we also have to encode the world coordinate of the points of seed triangles (used to construct the transformations). The affine transformation should then be correctly computed (at decoding) without loss of information. Therefore, we propose the seed triangle points be encoded separately with delta coding.

Delta coding

Given the sequence of the seed triangle points $(\mathbf{p}_1^{i,f}, \mathbf{p}_2^{i,f}, \mathbf{p}_3^{i,f})$, we first encode their world coordinates in the first frame. Then, the differences between each two adjacent frames in the sequence are computed. To avoid error accumulation during animation, these residuals are computed between the coordinates of the point $\mathbf{p}_j^{i,f}$ in the current frame and their recovered coordinates $\tilde{\mathbf{p}}_j^{i,f-1}$ in the previous frame: $\delta_j^{i,f} = \mathbf{p}_j^{i,f} - \tilde{\mathbf{p}}_j^{i,f-1}$, $(j = 1,2,3)$ where $i = 1,...,N$ and $f = 1,...,F$.

Principal Component Analysis

Principal Component Analysis (PCA) is a statistical technique that can reduce the dimensionality of a dataset. It determines linear combinations of the original dataset which contain maximal variation and represents them in an orthogonal basis. PCA reconstructs the original dataset optimally in the mean square-error sense. If we have F frames of $3V$ dimension each, PCA produces a reduced number $L \ll F$ of principal components that represent the original dataset.

We now consider how a cluster evolves over the frames of the animation. Let G_i^f be the i-th cluster in the f-th frame, $i = 1,...,N$ and $f = 1,...,F$. A single cluster $\mathscr{C}_i$ thus consists of F clusters (one for each frame) $\mathscr{C}_i = \{G_i^1, G_i^2, ..., G_i^F\}$ where G_i^f represents the vector with the geometry of the cluster i in frame f

$$G_i^f = (\mathbf{q}_4^{i,f}, \mathbf{q}_5^{i,f}, .., \mathbf{q}_{V_i}^{i,f})^t,$$

whose elements are the local coordinates of corresponding vertices (except the coordinate of the seed triangle). All these vectors $\mathbf{G}_i^f$ have the same length $3(V_i - 3)$, and construct a geometric matrix $\mathbf{A}_i$ with $3V_i - 9$ rows and F columns.

$$\mathbf{A}_i = \left[\ G_i^1 G_i^2 ... G_i^F\ \right]$$

A singular value decomposition on $\mathbf{A}_i$ is

$$\mathbf{A}_i = \mathbf{U}_i \mathbf{D}_i \mathbf{V}_i^t$$

where $\mathbf{U}_i$ is a $(3V_i - 9) \times F$ column-orthogonal matrix that forms an orthogonal basis and contains the eigenvectors of the $\mathbf{A_i A_i}^t$. $\mathbf{D}_i$ is a diagonal matrix whose nonzero elements represent the singular values and are sorted in decreasing order. Thus $\mathbf{D}_i = diag\{\lambda_1, \lambda_2, ..., \lambda_F\}$. $\mathbf{V}$ is a $F \times F$ orthogonal matrix.

To reduce the dataset, we pick only the first L eigenvectors (L is a user specified number). So, $\mathbf{U}_i^{'} = \{\mathbf{u}_{i,l}, l = 1, ..., L\}$ contains the most important principal components $\mathbf{u_i}$ that correspond to the largest eigenvalues $\lambda_1, ..., \lambda_L$. Then each cluster G_i^f is projected into the new basis $\mathbf{U}_i^{'}$ to get a new matrix of coefficients $\mathbf{C}_i^{'}$ of size $L \times F$.

$$\mathbf{C}_i^{'} = \mathbf{U}_i^{'t} \mathbf{A}_i$$

After performing the PCA for all N clusters $\mathscr{C}_i$, we get N new sets $\{\mathbf{U}_1^{'}, \mathbf{U}_2^{'}, ..., \mathbf{U}_N^{'}\}$ and coefficient matrices $\{\mathbf{C}_1^{'}, \mathbf{C}_2^{'}, ..., \mathbf{C}_N^{'}\}$ with different sizes.

Quantization and Arithmetic Coder

For further compression, the floating-point values (32 or 64 bits) are often quantized to a user specified number of bits per coordinate relative to the maximum extend of the bounding box of the model. The quantized values are encoded with an arithmetic coder [WNC87].

In the case of an animation, the quantization is often performed according either to the tight axis-aligned bounding box for each frame or to the largest bounding box for all frames. Since we have to encode the basis vector values and the coefficients rather than the vertex coordinates, we use two different encoding contexts. The first concerns the matrices and the second the delta vectors. The basis matrix $\mathbf{U}_i^{'}$ and the coefficient matrix $\mathbf{C}_i^{'}$ of each cluster $\mathscr{C}_i$ are truncated using a fixed number of bits q_u and q_c respectively (typically $q_u = q_c$). We first compute the minimum and the maximum values $(u_{min,i}, u_{max,i})$, $(c_{min,i}, c_{max,i})$ of $\mathbf{U}_i^{'}$ and $\mathbf{C}_i^{'}$ respectively. Then integer values are straightforwardly derived according to

$$\begin{aligned} u_{iq}(m,j) &= \lfloor u_i(m,j)/u_{max,i} - u_{min,i} \cdot 2^{q_u} + 1/2 \rfloor \\ c_{iq}(j,f) &= \lfloor c_i(j,f)/c_{max,i} - c_{min,i} \cdot 2^{q_c} + 1/2 \rfloor \end{aligned}$$

where $1 \leq m \leq 3V_i - 9$, $1 \leq j \leq L$ and $1 \leq f \leq F$.

The resulting signed integer values of the matrices are encoded with an adaptive arithmetic coder and sent with the extreme numbers.

For delta vectors, the coordinates are encoded according to the bounding box of each frame. Using a fixed number of bits q_Δ, the coordinates of the delta vectors are mapped into integers which are then encoded separate from PCA details with an arithmetic coder.

We assume that the quantization errors of PCA details are negligible up to 12 bits quantization. Note that the total number of bits needed for storing delta vectors is very small. It ranges between 0.01 and 1 bit per vertex per frame when the quantization ranges between 12 and 16 bits depending on the number of eigenvectors, the level of quantization, and the number of clusters.

3.4 Rate-Distortion Optimization

In LPCA-based techniques often PCA is performed using a fixed number of components per cluster, neglecting the fact that whole mesh sequences are often not rigid and the different parts can have different behavior (i.e. their motion is not similar). Thus, using a fixed number of components per cluster may results in an insufficient number to represent a given cluster at the desired accuracy while having too many for the representation of other clusters.

To improve the PCA based compression and avoid this overfitting and underfitting, we introduce the Rate-Distortion Optimization (RDO) which is also called the bit allocation. The objective is to find the best tradeoff between the bitrate and the distortion of coordinates of the vertices.

Given N clusters $\mathscr{C}_i$, that we have to encode separately, and a set of eigenvectors $I = \{l_0, l_1, ..., l_L\}$. For each cluster $\mathscr{C}_i$, let $(R_i^l, D_i^l,)$ denote the rate-distortion point for each number $l \in I$, (typically $l = 1, ..., 40$ components). The rate R_i^l represents the number of bits required to encode the basis vector values and the coefficients. The distortion D_i^l is the root square error between the original and the reconstructed coordinates of all vertices in the cluster.

Let R_{target} be the given total bit rate for all clusters. Then the optimization problem is to find the best number of components l_i for the cluster i, $(i = 1, ..., N)$ that minimize $D = \sum_{i=1}^{N} D_i^{l_i}$ subject to the constraint $\sum_{i=1}^{N} R_i^{l_i} \leq R_{target}$.

In our coding, we introduce an R-D optimization which is based on an incremental computation of the convex hull [WS00]. For simplicity, and since the number of bits increases with the size of the basis vectors, we define the rate R as the number of basis vectors rather than the number of bits. Briefly, we define the optimization algorithm in the following:

1. For each cluster $\mathscr{C}_i$ we compute:
   - The number of components l_i that corresponds to the smallest rate;
   - The number of components k_i that corresponds to the next RD point on the lower convex hull;
   - The slope λ_i between the points $(R_i^{l_i}, D_i^{l_i})$ and $(R_i^{k_i}, D_i^{k_i})$.
2. We compute the total rate $R_t = \sum_{i=1}^{N} R_i^{l_i}$
3. As long as $(\sum_{i=1}^{N} R_i^{l_i} \leq R_{target})$ is verified, we:
   - Select the cluster S_n whose λ_n is minimal;
   - Update R_t
   - Modify l_i by k_i;

Figure 5: *Reconstructed chicken.* ***Top raw****: Frame 314.* ***Bottom raw****: Frame 400.* ***From left to right****: Original, optimized RLPCA, RLPCA, and LPCA performed in world coordinates (*10 *clusters;* 10 *components).*

- Determine k_i that corresponds to the next RD point on the lower convex hull;
- Compute λ_n.

3.5 Compression Parameters

The compression parameters define the desired amount of compression. In our approach, there are three parameters that govern the compression ratio:

- *The number of basis vectors/rate L*: If this number is fixed for all clusters, then the user defines it (depending on the desired accuracy). The larger this number is, the better reconstruction will be (at the expense of less compression). If the RDO is used, then we will need only to specify the amount of compression (rate) or the maximum number of basis vectors that are to be used to approximate each cluster as we do in our coding. The number of vectors in each cluster is then optimally selected such that the total rate is below the given user-specified rate (or the total number of vectors is below the given user-specified maximum number of vectors).
- *The number of clusters N*: If this number is very small, then the cluster may contain vertices of different behavior and their local coordinates will have a large variation over time. However, it is difficult to find a linear space that efficiently represents these coordinates using PCA. In the future, we want to automatize the selection of this number. Typically, in motion capture based animation the number of clusters should be equal to the number of joints.
- *The reconstruction error*: This error presents the deviation of the reconstructed positions from the original one. It is measured using L2-norm or the metric which we call KG_{error} [KG04]. Moreover, it controls the compression during the RDO. This number should increase with decreases in the number of clusters or the number of eigenvectors.

4 DECOMPRESSION

Figure 3 illustrates the decoding process. After receiving the sequences of the PCA details and the delta vectors, we decode and undo quantization of delta vectors, we reconstruct the points of the seed triangles of each cluster in each frame, then reconstruct the *LCFs*. In the second stage, we undo the quantization of all basis vector values and coefficients, we reconstruct the local coordinates of all vertices in each cluster, and transform them back to world coordinates. Finally, we collect all clusters to reconstruct the sequence of meshes.

5 RESULTS

In order to see the performance of our scheme, RLPCA, we measured the number of bits per vertex per frame (bpvf), and as most other recently proposed methods for animated geometry coding, we used the KG_{error} metric to measure the distortion in the reconstruction animation with regard to the original animation. We also computed the distortion per frame using the L2 norm of all reconstructed vertex positions relative to the original positions of each frame. We compare the compression performance of our algorithm against AWC [GK04], TLS [PA05], PCA [AM00], KG [KG04] and CPCA [SSK05].

RLPCA vs. LPCA We want to find the influence of the segmentation and the local coordinates on the rate and on the reconstruction of animation. We performed LPCA in the world coordinate system as well as in the local coordinate systems for a given numbers of clusters, components and bits of quantization N, L and q_c respectively. Furthermore, we compared LPCA with the standard PCA.

Figure 6 (a) shows the reconstruction results relative the original frame using $q_c = q_u = 12$ and $L = 10$ when

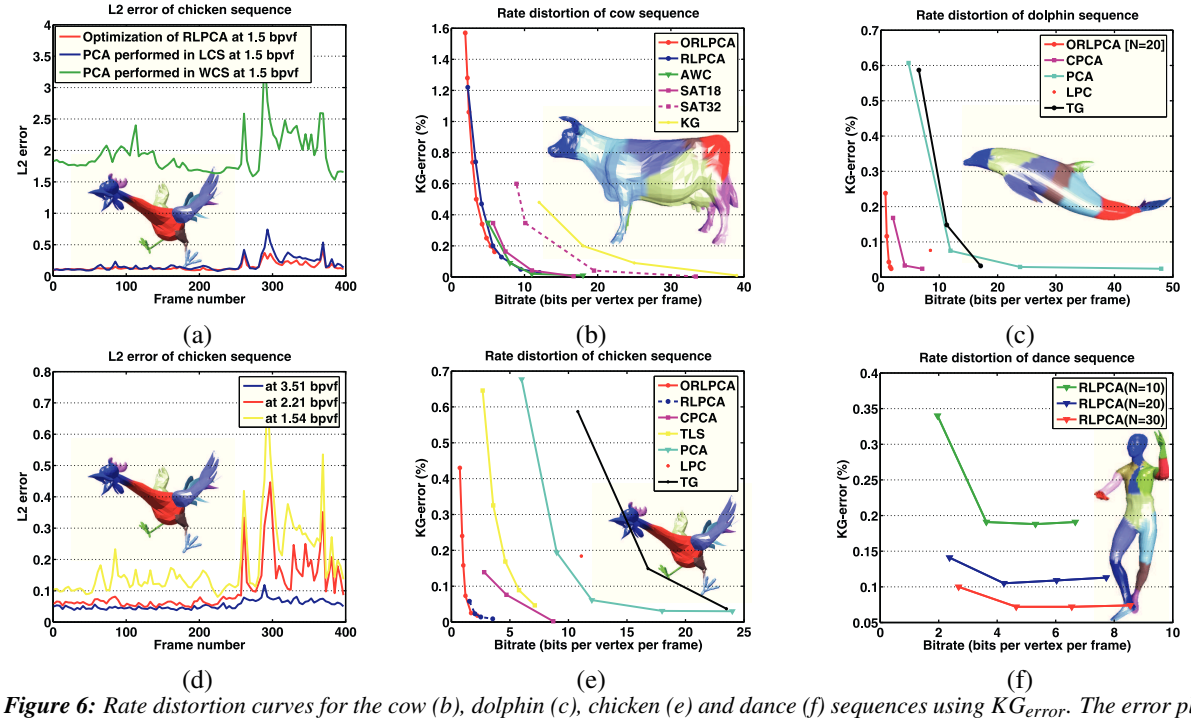


Figure 6: *Rate distortion curves for the cow (b), dolphin (c), chicken (e) and dance (f) sequences using KG_{error}. The error plot for the chicken sequence: (a) using LPCA in the world and the local coordinates and using the RD-optimization (*10 *clusters,*10 *components) and (d) using* 10*,* 10 *and* 20 *clusters and* 20*,* 10 *and* 20 *components.*

the LPCA is performed in the world coordinates (green) and in the local coordinates (blue) and when the R-D optimization is introduced (red) at the same number of bit per vertex per frame. We can see that the local coordinates are more compressable than the the original coordinates.

Figure 6 (d) and Figure 6 (f) shows the effect of the number of clusters and the component on the frame reconstruction for the chicken animation using $(N,L) = \{(20,10);\ (20,20);\ (10,10)\}$ and on the rate-distortion curves for the dance animation using 10, 20 and 30 clusters. In Figure 6 (a), the improvement in the second curve (blue) is due to the transformation of the original coordinates into local coordinates which forces the coordinates of a vertex to cluster around one point (see Figure 2). This improvement increases (red) when the optimization were introduced.

Figures 5 shows the reconstructed two frames in the chicken sequence when the world and the local coordinates are used and when the optimization is introduced using 10 components and 10 clusters.

Comparison to other coders Figure 6 also illustrates the comparison to other methods as rate-distortion curves for the cow (b), dolphin (c) and chicken (e) animations. At first glance, we can see that our approach achieves a better rate distortion performance than the standard PCA, LPC and TG for the three models. This result is obvious since the animation coding based on static techniques only exploit the spatial coherence and the linear prediction coding only uses the temporal coherence. Furthermore, the standard PCA only approximates the global linearity and is less effective for non-linear animation.

For the CPCA and AWC algorithms, we achieve better or similar results. Figure 6 (b) shows that for the cow animation our method is significantly better than the method of Karni and Gostman and than the CPCA. And it comes close to AWC. For the dolphin and the chicken sequences our method performs better than all the above methods. This improvement is due to the segmentation of the model into meaningful parts (whose vertices move quit similarly) as well as to the use of local coordinates rather than world coordinates. On the other hand, the RLPCA performs well for the models of large number of vertices in contrast to KG. Therefore, by combining RLPCA with LPC, we might achieve a better compression ratio. Figure 6 also demonstrates that the rate distortion optimization we introduce in our algorithm (ORLPCA) is important for achieving better compression performances especially when the number of vertices is large and the animation is complex.

From the computational viewpoint, PCA is computational expensive but in combination with LPC [KG04], it gives a better compression performance, particularly for a long sequence of just a few number of vertices. CPCA [SSK05] outperforms both methods since they explore a robust segmentation which is based on a data analysis technique but remains expensive. In contrast, our RLPCA uses a simple clustering and transformations and achieves a better compression ratio.

Table 1: *Comparison compression and decompression timings with CPCA.*

Models	vertices	triangles	frames	CPCA bpvf	CPCA d_{KG}	CPCA $t^{enc}_{(sec)}$	CPCA $t^{FPS}_{(sec)}$	RLPCA bpvf	RLPCA d_{KG}	RLPCA N	RLPCA L	RLPCA $t^{enc}_{(sec)}$	RLPCA $t^{dec}_{(sec)}$
chicken	3030	5664	400	4.7	0.076	206	214	3.5	0.008	20	20	120	69
				2.8	0.139	395	215	2.2	0.043	20	10	115	69
				2.8	0.139	395	215	1.5	0.057	10	10	110	47
cow	2904	5804	204	7.4	0.16	75	145	6.8	0.128	30	20	82	46
				3.8	0.5	59	218	4.1	0.470	30	20	40	50
				2.0	1.47	55	284	2.2	1.220	10	10	70	23
dolphin	6179	12337	101	7.1	0.024	-	-	3.9	0.016	20	10	74	40
				4.1	0.033	-	-	2.1	0.018	20	5	78	32
				2.1	0.168	-	-	1.9	0.066	10	5	39	25

Timings: Table 1 shows the timings in seconds of the coding (t^{enc}) and decoding (t^{dec}) processes (without optimization) for the three animations with a comparison to CPCA (t^{FPS} for display while decoding). We observe that for the chicken and cow animations, our coder is much faster and performs better than CPCA. Our timing results are measured on Pentium 4 with 2.53 GHz and CPCA on AMD Athlon64 XP 3200+.

6 CONCLUSION

We introduced a new compression technique for the animated meshes which is based on LPCA. The mesh vertices are clustered using the motion in the *LCF* . Then, the world coordinates of each cluster are transformed into local coordinates. This step enables the algorithm to compress an animated mesh efficiently. It exploits the "local" behavior of the local coordinates. Finally, an LPCA is performed in each cluster with the rate distortion optimization. Our approach is simple, fast and achieves a better performance than other current existing compression techniques. It is applicable to meshes and point-based models. It performs well for animations with a large number of vertices. For very long sequences, we suspect that the motion of a local coordinates also becomes complex and non-linear. Therefore, we want to combine our method in the future with LPC which is good for long sequences or split the sequences into small clips. Furthermore, we plan to develop an adaptive segmentation over time and encode the clusters with different quantization levels. The number of clusters can also be chosen automatically.

Acknowledgements We would like to thank Zachi Karni and Hector Briceño for providing us the animated meshes and Mirko Sattler, Igor Guskov and Frédéric Payan for the results of their methods. The Chicken sequence is property of Microsoft Inc.

REFERENCES

[AG05] P. Alliez and C. Gotsman. *Recent Advances in Compression of 3D Meshes*. Elsevier Science Inc., 2005.

[AM00] Marc Alexa and Wolfgang Müller. Representing animations by principal components. *Comput. Graph. Forum*, 19(3), 2000.

[ASS06] R. Amjoun, R. Sondershaus, and W. Straßer. Compression of complex animated meshes. volume 4035, pages 606–613, 2006. Computer Graphics International 2006 Conference.

[Dee95] M. Deering. Geometry compression. In *SIGGRAPH '95 Conference Proceedings*, pages 13–20, 1995.

[GK04] I. Guskov and A. Khodakovsky. Wavelet compression of parametrically coherent mesh sequences. In *Proceedings of the ACM SIG./Eurog. sympo. on Comput. anim.*, 2004.

[GS98] S. Gumhold and W. Straßer. Real time compression of triangle mesh connectivity. In *SIGGRAPH '98 Conference Proceedings*, pages 133–140, 1998.

[Hop96] Hugues Hoppe. Progressive meshes. In *Proceedings of the 23rd annual conference on Computer graphics and interactive techniques*, pages 99–108. ACM Press, 1996.

[IA02] M. Isenburg and P. Alliez. Compressing polygon mesh geometry with parallelogram prediction. In *IEEE Visualization '02 Conference Proceedings*, pages 141–146, 2002.

[IR03] L. Ibarria and J. Rossignac. Dynapack: space-time compression of the 3d animations of triangle meshes with fixed connectivity. In *ACM SIG./Eurog. Symp. on Comput. Anim.*, 2003.

[JCS02] Yang J.H., Kim C.S., and Lee S.U. Compression of 3-d triangle mesh sequences based on vertex-wise motion vector prediction. *Cir. Sys Video*, 12(12):1178–1184, December 2002.

[JPK05] C-S Kim J. Peng and C-C.J Kuo. Technologies for 3d mesh compression : A survey. *ELSEVIER Journal of Visual Communication and Image Representation*, 16(6):688–733, 2005.

[KG04] Zachi Karni and Craig Gotsman. Compression of soft-body animation sequences. *Comput.& Graph.*, 28:25–34, 2004.

[Len99] J. E. Lengyel. Compression of time-dependent geometry. In *Proc. of ACM sympo. on Interactive 3D graphics*, 1999.

[MSK+05] K. Muller, A. Smolic, M. Kautzner, P. Eisert, and T. Wiegand. Predictive compression of dynamic 3d meshes. In *IEEE International Conference on Image Processing*, 2005.

[PA05] F. Payan and M. Antonini. Wavelet-based compression of 3d mesh sequences. In *Proceedings of IEEE ACIDCA-ICMI'2005*, Tozeur, Tunisia, november 2005.

[Ros04] J. Rossignac. *Surface simplification and 3D geometry compression*. Chapter 54 in Handbook of Discrete and Computational Geometry 2004.

[SSK05] M. Sattler, R. Sarlette, and R. Klein. Simple and efficient compression of animation sequences. In *ACM SIG./Eurog. sympo. on Comput. anim.*, pages 209–217, 2005.

[TG98] C. Touma and C. Gotsman. Triangle mesh compression. In *Graphics Interface'98*, pages 26–34, 1998.

[TR98] G. Taubin and J. Rossignac. Geometric compression through topological surgery. *ACM Trans. on Graph.*, 17(2), 1998.

[WNC87] I. H. Witten, R. M. Neal, and J. G. Cleary. Arithmetic coding for data compression. *Communications of the ACM*, 30(6):520–540, 1987.

[WS00] M. Wagner and D. Saupe. Rd-optimization of hierarchical structured adaptive vector quantization for video coding. In *Proceedings of IEEE on Data Compression*, page 576, 2000.

[YKK01] Z. Yan, S. Kumar, and C. C. Jay Kuo. Error-resilient coding of 3-d graphic models via adaptive mesh segmentation. *IEEE Trans. Circ. Syst. Video Tech.*, 11(7):860–873, 2001.

[ZO04] Jinghua Zhang and Charles B. Owen. Octree-based animated geometry compression. In *Proceedings of IEEE on Data Compression*, pages 508–517, 2004.