



Available at

www.ElsevierComputerScience.com

POWERED BY SCIENCE @ DIRECT®

Computers & Graphics 28 (2004) 25–34

COMPUTERS  
& GRAPHICS

www.elsevier.com/locate/cag

# Compression of soft-body animation sequences

Zachi Karni\*, Craig Gotsman

*Center for Graphics and Geometric Computing, Faculty of Computer Science, Technion, Taub Bld., Room 422, Haifa 32000, Israel*

## Abstract

We describe a compression scheme for the geometry component of 3D animation sequences. This scheme is based on the principle component analysis (PCA) method, which represents the animation sequence using a small number of basis functions. Second-order linear prediction coding (LPC) is applied to the PCA coefficients in order to further reduce the code size by exploiting the temporal coherence present in the sequence. Our results show that applying LPC to the PCA scheme results in significant performance improvements relative to other coding methods. The use of these codes will make animated 3D data more accessible for graphics and visualization applications.

© 2003 Elsevier Ltd. All rights reserved.

*Keywords:* Animation; Compression; Meshes

## 1. Introduction

The most common representation of 3D objects is in polygonal mesh form. These consist of vertices, edges and faces. This representation is widespread because it is supported by the hardware of all the manufacturers of leading graphics rendering cards. 3D animation appears in two forms: rigid-body and soft-body motion. In rigid-body motion, the relative position of each two vertices of the mesh stays fixed and the body moves as one entity. Such motion may be described by six parameters (also known as degrees of freedom): three for the position of the object's center and three for its orientation. The simplicity and compact representation of the rigid motion makes it particularly attractive. However, this model is not strong enough to capture many life-like movements.

Soft-body motion does not impose any restrictions on the relation between an object's vertices (as long as they form a valid mesh). It consists of a separate trajectory for each mesh vertex, which allows capturing of smooth and realistic motion. However, the size of files storing these data is usually very large because each frame is actually a complete 3D object. This is a major factor preventing soft-body motions from becoming popular or

suitable for real-time rendering. They are mostly used in offline rendering scenarios, such as high-quality rendered animation clips for the movie industry.

In this paper, we present a compression scheme for soft-body motion sequences. This scheme reduces the size of the data files significantly and, hopefully, will enable their use in home computing and games as well as enable their transmission over the Internet.

The rest of the paper is organized as follows: Section 2 gives a short survey of previous compression methods; Section 3 describes a few common techniques for generating animation sequences; Sections 4 and 5 describe possible compression schemes; and in Section 6 we compare the performance of selected schemes and conclude in Section 6.6.

## 2. Previous work

The domination of polygonal 3D mesh data in the field of 3D content has led to a proliferation of research on the compression of this type of data. Some deal with compression of the mesh geometry (vertex positions) (e.g. [1–3]), while the rest focus on connectivity (e.g. [4–6]). A few papers are dedicated to multi-resolution methods for progressive transmission (e.g. [7–9]).

In contrast to the large volume of work on compression of static meshes, there is almost none on the

\*Corresponding author. Tel.: +972-48293896; fax: +972-48294906.

*E-mail address:* zachik@cs.technion.ac.il (Z. Karni).

compression of dynamic meshes, with only a few notable exceptions. Lengyel [10] suggested segmenting the mesh into smaller submeshes whose motion can be described as a rigid body. The compression is achieved by specifying only the affine transformations for each submesh instead of the complete trajectory of each vertex. However, the critical segmentation process is difficult and only a heuristic solution is provided. Shamir and Pascucci [11] suggested a multi-resolution representation for the mesh. They solve for the best affine transformation between each frame and the first one (the low-frequency motion) and encode the residual (high-frequency motion) after applying this. Ibarria and Rossignac [12] advocated the use of space–time predictors to capture the correlations in both spaces. They propose to traverse the connectivity graph in a manner similar to [4] and [6], predicting a new vertex position based on its neighbors (already traversed) in the current and previous frames.

To a bystander, the absence of work in this field may indicate a lack of interest in the problem. However, there exists an abundance of techniques showing how to *generate* such animation sequences. Hence, we believe that efficient compression of these type of data will ultimately be a must.

### 3. Animation synthesis

Before we discuss how to compress soft-body animation sequences, we elaborate on how they are typically created. This is a long and tedious process. It requires many experienced animator man months to produce a high-quality animation lasting a few minutes. The creation of animation sequences consists of two main stages: creation of the static 3D model and the procedure for bringing it to life by making it move. Models can be animated in one of the two ways, which we describe in the following two subsections.

#### 3.1. Physical-based animation

In physical-based animation, the object motion is described by a differential equation, or more precisely, by the solution of the equation. A very common governing equation that can generate a variety of interesting motions is the second-order linear equation:

$$M\ddot{x}(t) + C\dot{x}(t) + Kx(t) = f(t), \quad (1)$$

where  $M$ ,  $C$  and  $K$  are constant matrices which determine the material properties (density, damping and elasticity) of the object. The vector  $x(t)$  is the 3D object position over time and  $f(t)$  is a vector of external forces acting on the body. The body's motion  $x(t)$  is the solution to (1). Depending on the nature of

$f(t)$ , the solution can be obtained either analytically or numerically.

Most of the work on this type of motion has concentrated on finding accurate and fast numerical solvers [13,14], and on defining the appropriate  $M$ ,  $C$  and  $K$  which result in realistic animation sequences [15].

#### 3.2. Synthetic animation

Animation sequences which are not based on physical phenomena are called synthetic. Tools like 3D Studio-Max, Maya and others, enable the animator to create skeletons and to move them as he likes. Moradoff and Lischinski [16] used one skeleton motion to generate others, by adding random variations to the original.

## 4. New animation representations

The main objective in compression is to find a new representation for a given data set which “costs” less than the original representation. In computer science, and in particular in computer graphics, “cost” means fewer bits to transmit or store. In Section 2, we mentioned a few compression schemes designed for static meshes. When dealing with soft-body animation sequences, each frame can be treated as a static mesh and be compressed using these techniques. However, such an approach does not exploit the correlation between the frames and will most likely to be far from optimal. Indeed, we show in this paper that much more compact representations exist.

An animation is a sequence of vectors  $f_1, f_2, \dots, f_T$  in which  $f_i$  ( $1 \leq i \leq T$ ) represents one static mesh (i.e., frame). An obvious way to code such a sequence is to use the first frame:  $f_1$ , and then the sequence of differences between each two adjacent frames:  $\Delta f_i = f_{i+1} - f_i$  ( $1 \leq i \leq T - 1$ ). It is easy to see that such a representation is “better” than the original when the animation is well behaved. However, most existing animations do not admit such a simple relation between their frames, so better representations are needed.

#### 4.1. Principle component analysis

Alexa and Müller [17] have used principle component analysis (PCA) to construct a compact representation of an animation sequence. They calculated the eigenvectors (and eigenvalues) of the matrix  $AA^T$ , where each column of the matrix  $A$  is the geometry of one frame. Because  $A$  is an  $n \times T$  rectangular matrix ( $n$  is the number of vertices and  $T$  is the number of frames),  $AA^T$  is a symmetric  $n \times n$  matrix (also known as the *covariance* matrix). The eigenvectors of such a matrix are orthonormal and span  $\mathcal{R}^n$ . This computation is sometimes called the Karhunen–Loeve Transform (KLT). Usually,



Fig. 1. A sample of frames from the animation sequences “Dolphin”, “Chicken” and “Face”.

the *singular value decomposition* (SVD) of the matrix  $A$  is used to construct its eigenvector. It is known to be more efficient and stable than the computation of the eigenvector of  $AA^T$ .

When projecting each frame of the animation on the PCA eigenvectors, most of the *energy*<sup>1</sup> will concentrate in the span of the eigenvectors corresponding to the smaller eigenvalues. This means that “most” of the animation sequence is “in” a linear subspace of dimension  $k$  ( $k \ll n$ ), and may be approximated well by a linear combination of those  $k$  eigenvectors. For example, the “chicken” sequence in Fig. 1 (3030 vertices and 400 frames), was reconstructed using only 10 eigenvectors with small distortion and 50 eigenvectors with negligible distortion.

The major drawback of the PCA method is the significant computing resources it requires in processing time and memory usage. For example, computing the eigenvectors of a mesh containing 15,000 vertices will require  $\sim 1$  GB of memory (just to store the covariance matrix) and many hours of computing. Despite the significant computational resources needed, we believe that PCA is appropriate for animation coding, primarily because most animations are produced in studios where such computing resources exist. However, optimization in computing time and memory usage is essential for the decoder, which will ultimately be run on simple home PC. For the PCA method, the decoder’s task is to

receive the eigenvectors, the coefficients for each frame, and by multiplying and summing them, reconstruct the animation. This is a very simple operation.

#### 4.2. The spectral basis

Karni and Gotsman [2] presented a compression technique for static meshes, building on a spectral basis which has properties similar to the PCA eigenvectors. They also exploited the fact that only a small portion of the spectral basis is needed for good reconstruction, so the code is just a small number of coefficients. Spectral coding is designed for static mesh compression, where it is not possible to use PCA or any other method based on the geometries of multiple meshes. The spectral basis is constructed from the mesh connectivity alone, using the so-called Laplacian matrix. If all the frames in an animation sequence have identical connectivity, and this is usually the case—spectral coding may also be applied to animation sequences, frame by frame.

We examined the performance of PCA on a few animation sequences and compared it to that of spectral coding. The graphs in Fig. 2 shows that the amount of energy in the lowest  $k$ -dimensional subspace reaches its maximum much faster for PCA-based methods than for the spectral-based ones. Hence the PCA basis is to be preferred in general. This is not surprising, since the PCA basis reflects the geometry of the specific sequence, and the spectral basis reflects the geometry of a much wider class of meshes with a common connectivity.

<sup>1</sup>By *energy* we mean the  $L_2$  norm of a vector.

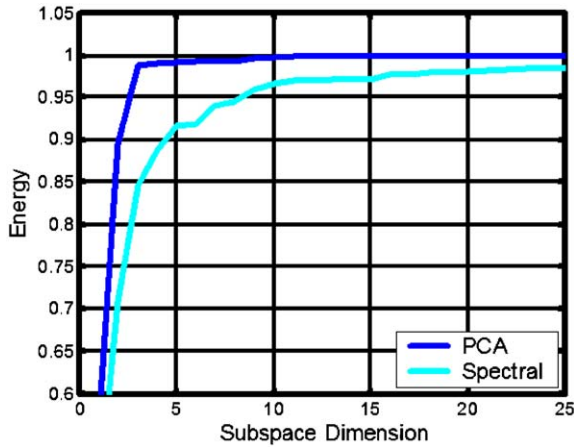


Fig. 2. Energy vs. subspace dimension for one frame from “face”. The PCA coefficients approach the maximal 1 much faster than the spectral coefficients.

## 5. Linear prediction coding

As the name implies, linear prediction coding (LPC) predicts the  $t$ th element in a series as a linear (actually affine) function of the preceding  $m$  elements in the sequence:

$$x_t = a_0 + \sum_{j=1}^m a_j x_{t-j} \quad \forall k < t \leq T. \quad (2)$$

The real coefficients  $a_0 \dots a_m$  are the weights given to each preceding element in determining the current element. Given a sequence  $x$ , optimal values for these may be computed by least squares on a rectangular  $(T - m) \times (m + 1)$  linear system ( $T$  is the total number of elements in the sequence).

$$\begin{pmatrix} 1 & x_m & \dots & x_1 \\ 1 & x_{m+1} & \dots & x_2 \\ \vdots & \vdots & \ddots & \vdots \\ 1 & x_{T-1} & \dots & x_{T-m} \end{pmatrix} \begin{pmatrix} a_0 \\ a_1 \\ \vdots \\ a_m \end{pmatrix} = \begin{pmatrix} x_{m+1} \\ x_{m+2} \\ \vdots \\ x_T \end{pmatrix}. \quad (3)$$

### 5.1. PCA+LPC coder

Using LPC to compress animation sequences is straightforward. The geometry of each vertex in a frame is predicted as a linear combination of the geometries of the corresponding vertices in the preceding frames. The resulting code is the first  $m$  frames and the weights  $a_0 \dots a_m$  for each vertex. This compression scheme is lossy. The least-squares method which we used to calculate the weights guarantees only that the sum of the squared distortions over all the frames will be

minimal. To make the scheme lossless, the residuals  $\Delta x_t$  of each vertex in each frame must be incorporated into the code,

$$\Delta x_t = x_t - \left( a_0 + \sum_{j=1}^m a_j x_{t-j} \right). \quad (4)$$

However, using LPC on the animation sequence directly will capture only the temporal coherence present in the trajectory of each individual vertex. It will not capture spatial correlations between the vertices. This may be achieved by transforming the frames into a space which decorrelates the vertex coordinates.

Using LPC on the significant (low frequency) PCA vectors will give us the best of both worlds: from the PCA, by applying it only to  $k$  coefficients instead of  $n$  vertices ( $k \ll n$ ); and from the LPC: by using a small number of coefficients to capture the temporal behavior of each PCA vector.

Section 3.1 showed that many physical-based animations may be represented by second order PDEs. This implies that second order LPC ( $m = 2$ ) will probably suffice to capture well, the temporal correlation within the modes. Indeed, for both the “chicken” and the “dolphin” models we used in our tests, second order LPC gained the best rate–distortion ratio, while for the “face” models third order LPC was optimal. Note that increasing the order of the LPC always reduces the distortion. However, the total code size might increase due to the additional coefficients, which in turn will worsen the rate–distortion ratio (especially in short animation sequences).

Our compression scheme works as follows: Given an animated 3D mesh that consists of  $n$  vertices and  $T$  frames, we build a  $3n \times T$  matrix  $A$ . Each column of the matrix represents one frame organized in a column-stack ( $X$  coordinates of all the vertices, followed by the  $Y$  coordinates and finally the  $Z$  coordinates). We apply the singular value decomposition (SVD) to  $A$  in order to calculate the eigenvectors (and eigenvalues) of  $AA^T$ . These are stored in the columns of the matrix  $U$ . The original animation is then decomposed by projecting it onto the new basis yielding a matrix of coefficients  $C = U^T A$  ( $C$  has the same size as  $A$ ). Trimming the high-frequency part (by zeroing all vector elements after the  $k$ th in each frame) results in a new coefficients matrix  $C_1$  of dimension  $k \times T$ . We now solve  $k$  second-order LPC problems each of size  $T$ . The code is the  $k$  eigenvectors of length  $3n$  used for the decomposition together with the  $k$  triples of LPC coefficients and the  $k \times (T - 2)$  LPC residuals.

The reconstruction is done first by using the LPC coefficients and residuals to retrieve the matrix  $C_1$  (lossless up to quantization). Then the animation sequence is reconstructed using the eigenvectors:  $A_R = UC_1$ . The distortions in the reconstructed animation are

due only to the elimination of the high-frequency components (as in [17]).

## 6. Experimental results

We have implemented the PCA and the LPC methods and compared between them and the Dynapack method of Ibarria and Rossignac [12]. For the comparison we used three animation sequences: “Dolphin” (6179 vertices, 12,337 faces and 101 frames), “Chicken” (2916 vertices, 5454 faces and 399 frames) and “Face” (539 vertices, 1042 faces and 10,001 frames). A sample of each of the animations is shown in Fig. 1. We have already mentioned that the number of frames should be much larger than the number of vertices. However, two of the three animation sequences we used do not follow this rule, as they are only a small snippet from a complete sequence, which we do not have.

The mesh coordinates are usually represented in floating-point format (using 32 or 64 bits). Before storing or transmitting their values it is customary to quantize them using a fixed number of bits— $q$ . The quantization process is equivalent to bounding the mesh within a cube, sampling each dimension into  $2^q$  grid vertices and snapping the vertices to the grid points. This process moves the vertices from their original position and distorts the mesh geometry (increasing the number of quantization bits will refine the grid and reduce the distortion). Fig. 3 shows the dolphin mesh together with the 12, 10 and 8 bit quantized versions. It is clear that

12 bit quantization does not introduce any visible distortion, hence may be considered as a lossless representation and can serve as a reference for comparing code lengths.

To compare different methods, we used the following distortion measure:

$$e = 100 \frac{\|A - \tilde{A}\|}{\|A - E(A)\|}, \quad (5)$$

where  $A$  is a  $3n \times T$  matrix containing the original animation sequence.  $\tilde{A}$  is the same animation after the compression and reconstruction stages.  $E(A)$  is an average matrix in which a column  $t$  is

$$(\bar{X}_t[1 \ \dots \ 1], \bar{Y}_t[1 \ \dots \ 1], \bar{Z}_t[1 \ \dots \ 1])^T, \quad (6)$$

where  $\bar{U}$  denotes the mean of the a vector  $U$ .

### 6.1. The raw data

To compare the compression ratios for each method, it is customary to use the uncompressed raw data size as a baseline reference. The raw data size of the animation sequence in bits is:  $\#quantization\text{-bits} \cdot \#coordinates \cdot \#vertices \cdot \#frames$ . The compression ratios are relative to the 12 bit quantization case. Table 1 summarizes the compression ratio and distortion incurred for each of the animation sequences quantized to 8, 10 and 12 bits/coordinate. The numbers in brackets in the “Size” column represents the normalized-size as *number of bits per vertex per frame*.

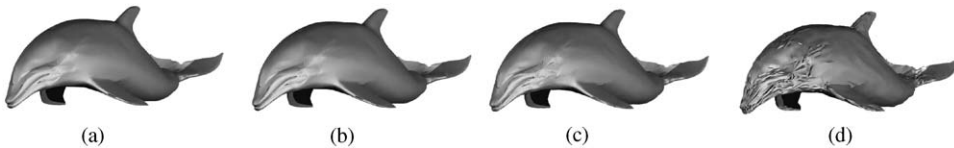


Fig. 3. Quantized Dolphin model: (a) original, (b) 12 bits, (c) 10 bits and (d) 8 bits quantization.

Table 1  
Compression performance using only quantization

Model	Quantization-bits	Size (KB)	Compression ratio	Distortion (%)
Dolphin	12	2742 (36)	1:1.00	—
	10	2285 (30)	1:1.20	0.099
	8	1829 (24)	1:1.50	0.387
Chicken	12	5120 (36)	1:1.00	—
	10	4260 (30)	1:1.20	0.125
	8	3409 (24)	1:1.50	0.488
Face	12	23689 (36)	1:1.00	—
	10	19741 (30)	1:1.20	0.134
	8	15793 (24)	1:1.50	0.521

## 6.2. Comparison to TG coding

Touma and Gotsman [6] described a method for the compression of static meshes. They based the geometry component of their encoder on the fact that in most triangle meshes, two adjacent triangles form an approximate parallelogram. When a face is encountered by the decoder, its neighbors are predicted using this rule. The code for this scheme is the vector difference between the prediction and the true vertex position.

We used the TG method, as implemented in public-domain code, to encode each frame separately. Table 2 presents the total code sizes for each of the animation sequences, the compression ratios together with the loss, again compared to the 12-bit quantized raw data.

## 6.3. LPC on vertex trajectories

It is possible to use second order LPC (as described in Section 5) to predict each mesh vertex position in a frame based on the corresponding positions in the two preceding frames (we predict each coordinate separately). Table 3 shows the results for each of the animation sequences for the case of 12-bit quantization. It is easy to see that the coding efficiency was doubled for all the three models. This supports the claim that the temporal coherence should not be ignored when compressing animation sequences, together with the fact that second order LPC is sufficient to capture most of the temporal coherences.

The differences in the distortions between this method and TG are because, TG quantizes the spatial data while this method quantizes the temporal data.

## 6.4. PCA

We calculated the PCA basis vectors for each of the animation sequences and reconstructed them using only a small number of basis vectors, as discussed in Section 3. Table 4 shows the results for the face animation using 300, 200, 100 and 50 basis vectors of a total 1617 ( $1617 = 3 \cdot \#vertices$ ). It took 250 s to calculate all the 1617 eigenvectors using a P4-2.4 GHz processor with 4 GB of memory. “Payload” is the total size of the basis vectors needed for decoding the animation, together with the quantization parameters. These basis vectors are quantized to 16 bits in order to reduce reconstruction artifacts and are transmitted as the prefix of the code. “Size” is the entropy of the quantized coefficients (in bits) and “Total Size” is the total code length (Payload + Size). As before, the numbers in brackets are *number of bits per vertex per frame*. It is evident that the PCA yields better compression ratios than the LPC or TG methods for similar distortions.

The PCA compression of “dolphin” and “chicken” yielded very poor results (Table 5). This is because the number of frames is much smaller than the number of vertices, so the size of the payload dominates the total code size. Clearly, PCA is not suitable for such cases and we present these results only for completeness sake.

Table 2  
Compression performance using the TG method

Model	Quantization-bits	Size (KB)	Compression ratio	Distortion (%)
Dolphin	12	1309 (17.1)	1:2.10	0.032
	10	862 (11.3)	1:3.18	0.148
	8	497 (6.52)	1:5.51	0.587
Chicken	12	3335 (23.5)	1:1.53	0.037
	10	2385 (16.8)	1:2.14	0.149
	8	1536 (10.8)	1:3.33	0.587
Face	12	18540 (28.3)	1:1.27	0.039
	10	13664 (20.8)	1:1.73	0.152
	8	9358 (14.2)	1:2.53	0.595

Table 3  
Compression performance using the LPC method

Model	Quantization-bits	Size (KB)	Compression ratio	Distortion (%)
Dolphin	12	645 (8.5)	1:4.25	0.076
Chicken	12	1588 (11.1)	1:3.22	0.184
Face	12	9101 (13.8)	1:2.60	0.032

Table 4  
Compression performance on the “face” sequence using the PCA method

Basis vectors	Payload (KB)	Quantization-bits	Size (KB)	Total size (KB)	Compression ratio	Distortion (%)
300	950	12	4000	4950 (7.5)	1:4.8	0.037
		10	3270	4220 (6.4)	1:5.6	0.038
		8	2535	3485 (5.3)	1:6.8	0.050
200	633	12	2685	3318 (5.0)	1:7.1	0.046
		10	2197	2830 (4.2)	1:8.4	0.047
		8	1707	2340 (3.5)	1:10.1	0.057
100	317	12	1355	1672 (2.5)	1:14.1	0.081
		10	1110	1427 (2.1)	1:16.6	0.082
		8	866	1183 (1.8)	1:20.0	0.087
50	160	12	684	844 (1.3)	1:28.1	0.146
		10	562	722 (1.1)	1:32.8	0.147
		8	440	600 (0.9)	1:39.5	0.149

Table 5  
Compression performance on the “dolphin” and “chicken” sequences using the PCA method

Model	Basis vectors	Payload (KB)	Compression ratio	Distortion (%)
Dolphin	100	3620	1:0.75	0.024
	50	1810	1:1.51	0.029
	25	905	1:3.02	0.075
	10	362	1:7.57	0.607
Chicken	200	3417	1:1.50	0.030
	150	2563	1:2.00	0.031
	100	1708	1:3.00	0.061
	75	1281	1:4.00	0.194
	50	855	1:6.00	0.677

### 6.5. PCA+LPC

We applied second order LPC on the PCA coefficients from the previous section (we used only the “face” sequence because “chicken” and “dolphin” are not suitable). As in Section 6.3 we quantized the temporal vector of each coefficient and then entropy code it. Table 6 shows the results of this method and Fig. 4 shows the rate–distortion graph for the PCA and the PCA + LPC methods. The superiority of the PCA + LPC method over the others is evident. For example, for a distortion of  $\sim 0.035\%$  (essentially lossless) PCA + LPC yields 6.5 times better compression ratio than TG, 2.2 times better than simple LPC and 1.7 times better than PCA alone.

Fig. 5 shows the reconstructions of three frames from the “Face” sequence using 300 and 50 basis-vectors (out of 1617). Although minor defects exist, it is very hard to tell the difference between the reconstructions and the

original. In case more distortion is tolerable, higher compression ratios can be achieved.

### 6.6. Dynapack

Ibarria and Rossignac [12] presented a method which predicts the position of a vertex based on the position of its (already decoded) spatial neighbors (similar to TG) and its position in previous frames (similar to LPC). Table 7 presents the code length generated by Dynapack for the face sequence.

Although the superiority of the PCA + LPC method is evident, the runtime of the Dynapack encoder is significantly shorter, while the decoding time is comparable. As we already emphasized, our lengthy encoding time is not a critical drawback. Animations are created by powerful computers which can easily handle offline PCA calculations.

Table 6  
Compression performance on the “face” sequence using the PCA+LPC method

Basis vectors	Payload (KB)	Quantization-bits	Size (KB)	Total size (KB)	Compression ratio	Distortion (%)
300	950	12	1888	2838 (4.3)	1:8.3	0.037
		10	1504	2454 (3.7)	1:9.6	0.038
		8	1096	2046 (3.1)	1:11.6	0.050
200	633	12	1240	1873 (2.8)	1:12.6	0.046
		10	982	1615 (2.4)	1:14.7	0.047
		8	708	1341 (2.0)	1:17.7	0.057
100	317	12	606	923 (1.4)	1:25.7	0.081
		10	476	793 (1.2)	1:29.9	0.082
		8	338	655 (1.0)	1:36.2	0.087
50	160	12	298	458 (0.7)	1:51.7	0.146
		10	232	392 (0.6)	1:60.4	0.147
		8	162	322 (0.5)	1:73.4	0.149

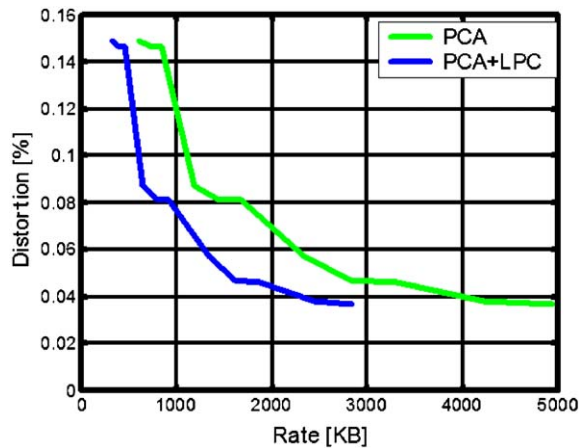


Fig. 4. Rate-distortion graphs for the “dolphin” sequence using PCA and PCA+LPC.

Another advantage of the Dynapack method is its lossless nature. The error is due only to quantization, which for 12 bits is negligible. The PCA+LPC method incurs other types of loss. However, the error is very small and the good compression ratios make the method very attractive.

## 7. Discussion and conclusions

The key to efficient compression of soft-body animation sequences is an exploitation of both the spatial and temporal correlation present in the sequence. The spatial correlations are captured using PCA, and the temporal correlations using LPC on the resulting coefficients. When applying this, we are able to achieve 0.05% distortion with bit rate of 2.45 bits per vertex per frame.

This means that 3D soft-body animation may be transmitted and played in real time.

The PCA basis, which is based on the geometries of all the frames in the sequence, is known to be optimal for that specific sequence, as opposed to the spectral basis, which is based on (the fixed) connectivity information alone. Recently, Ben-Chen and Gotsman [8] have shown that by making some natural assumptions on the distribution of the geometries associated with a fixed connectivity, the spectral basis is actually optimal for that distribution in the same PCA sense.

PCA and its generalizations have long been used to reduce the dimensionality of a data set, in particular in computer vision recognition applications [18,19]. However, its combination here with causal temporal analysis seems to be new.

One other possible combination of PCA and temporal analyses was proposed by Pentland and Williams [14]. They show that if the animation sequence satisfies a differential equation such as (1), a possible basis are the so-called free vibration modes, which may be derived by solving a generalized eigenvector problem related to the matrices  $M$ ,  $C$  and  $K$ . The connection between these and the PCA basis for the sequence is not entirely clear, but there is a chance that these modes might be better for compression purposes.

However, given just the animation sequence, it seems difficult to “reverse-engineer” it to derive the matrices  $M$ ,  $C$ ,  $K$  and from them the modes. So this remains an open problem.

## Acknowledgements

This research was supported by the Technion Vice President for Research Fund - Lowenengrat Research

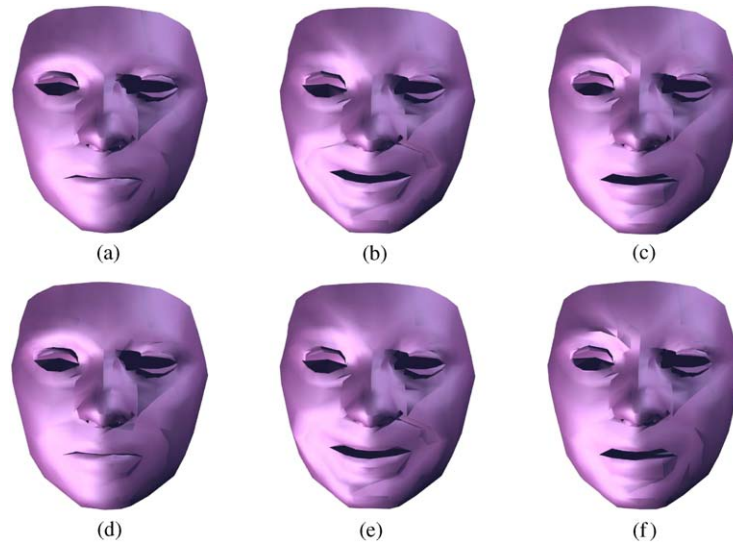


Fig. 5. A sample of frames from the “Face” animation, reconstruct using the PCA + LPC method using 300 (a–c) and 50 (d–f) basis vectors.

Table 7  
Dynapack compression results on the face sequence

Quantization-bits	Size (KB)	Compression ratio	Distortion (%)
12	6760 (10.2)	1:3.5	—
10	4541 (6.9)	1:5.2	0.134
8	3090 (4.7)	1:7.6	0.521

Fund. The “chicken” sequence is the property of Microsoft Inc and the “face” sequence was kindly generated by Demetri Terzopoulos. The results of the Dynapack method were supplied by Lawrence Ibarria.

## References

- [1] Deering M. Geometry compression, Proceedings of SIGGRAPH 1995; ACM Press, 1995. p. 13–20.
- [2] Karni Z, Gotsman C. Spectral compression of mesh geometry. SIGGRAPH 2000, Computer Graphics Proceedings, ACM Press, 2000. p. 279–86.
- [3] Taubin G, Rossignac J. Geometric compression through topological surgery. ACM Transactions on Graphics 1998;17(2):84–115.
- [4] Gumhold S, Straßer W. Real time compression of triangle mesh connectivity, SIGGRAPH 1998, Computer Graphics Proceedings, ACM Press, 1998. p. 133–40.
- [5] Rossignac J. Edgebreaker: connectivity compression for triangle meshes. IEEE Transactions on Visualization and Computer Graphics 1999;5(1):47–61.
- [6] Touma C, Gotsman C. Triangle mesh compression. In Proceedings of the 24th annual graphics interface conference (GI); 1998. p. 26–34.
- [7] Alliez P, Desbrun M. Progressive encoding for lossless transmission of 3D meshes. SIGGRAPH 2001, Computer Graphics Proceedings, ACM Press, 2001. p. 195–202.
- [8] Ben-Chen M, Gotsman C. On the Optimality of spectral compression of mesh geometry. Preprint, Technion Computer Science, 2003.
- [9] Karni Z, Bogomjakov A, Gotsman C. Efficient compression and rendering of multi-resolution meshes. In Proceedings of IEEE Visualization’02, 2002;347–54.
- [10] Lengyel JE. Compression of time-dependent geometry. In Proceedings of ACM Symposium on Interactive 3D Graphics; 1999. p. 89–96.
- [11] Shamir A, Pascucci V. Temporal and spatial level of details for dynamic meshes. Proceedings of Virtual Reality Systems and Techniques, 2001. p. 423–430.
- [12] Ibarria L, Rossignac J. Dynapack: space-time compression of the 3D animation of triangle meshes with fixed connectivity. Proceedings of the Eurographics ACM SIGGRAPH 2003 Symposium on Computer Animation, 2003. p. 126–135.
- [13] Debunne G, Desbrun M, Cani MP, Barr A. Adaptive simulation of soft bodies in real-time. In Proceedings of Computer animation 2000. p. 133–144.
- [14] Pentland A, Williams J. Good vibration: modal dynamics for graphics and animation. Computer Graphics 1989;23(3):207–14.
- [15] Terzopoulos D, Platt J, Barr A, Fleischer K. Elastically deformable models. Computer Graphics 1987;21(4): 205–14.
- [16] Moradoff S, Lischinski D. Synthesis of textural motion with hard constraints. In Proceedings of the 4th Israel-Kored 6i-national conference on geometric modelling and computer graphics; 2003. p. 123–128.

- [17] Alexa M, Müller W. Representing animations by principle components. *Computer Graphics Forum* 2000;19(3):411–8.
- [18] Turk MA, Pentland AP. Eigenfaces for recognition. *Journal of Cognitive Neuroscience* 1991;3(1):71–86.
- [19] Vasilescu MAO, Terzopoulos D. Multilinear analysis of image ensembles: tensor-faces. *Proceedings of the Seventh European Conference on Computer Vision (ECCV'02)*, Copenhagen; May 2002.