Eurographics/ACM SIGGRAPH Symposium on Computer Animation (2005)
K. Anjyo, P. Faloutsos (Editors)

Simple and efficient compression of animation sequences

Mirko Sattler Ralf Sarlette Reinhard Klein

Institute of Computer Science II, University of Bonn, Germany

Figure 1: Sample frames of the compressed chicken sequence with 10 clusters, each colored differently.

Abstract

We present a new geometry compression method for animations, which is based on the clustered principal component analysis (CPCA). Instead of analyzing the set of vertices for each frame, our method analyzes the set of paths for all vertices for a certain animation length. Thus, using a data-driven approach, it can identify mesh parts, that are "coherent" over time. This usually leads to a very efficient and robust segmentation of the mesh into meaningful clusters, e.g. the wings of a chicken. These parts are then compressed separately using standard principal component analysis (PCA). Each of this clusters can be compressed more efficiently with lesser PCA components compared to previous approaches. Results show, that the new method outperforms other compression schemes like pure PCA based compression or combinations with linear prediction coding, while maintaining a better reconstruction error. This is true, even if the components and weights are quantized before transmission. The reconstruction process is very simple and can be performed directly on the GPU.

Categories and Subject Descriptors (according to ACM CCS): I.3.7 [Computer Graphics]: Animation

1. Introduction

3D objects are most often represented as polygonal meshes, which consist of vertices, edges and faces. This representation is supported by the majority of modeling tools and graphics accelerator boards are optimized to use this form of data for rendering. For life-like soft body animations of this 3D data, e.g. of avatars, relative vertex positions change over time, while the face connectivity stays constant. High-quality soft-body animations are widely used in fields which allow off-line production, like films. On the other hand, this form of representation is not practical for real-time entertainment or transmission over the internet with limited bandwidth, because of the huge amount of data. Essentially, each frame contains the whole 3D scene. Hence, efficient compression schemes are necessary. On the one hand, there should be significant compression. On the other hand, the scheme should allow for reconstruction without visible artifacts and it should be fast enough for real-time applications.

Contributions: in this paper, we present a method to efficiently compress the geometry data of animation sequences. This is achieved by two main features. First, we reorganize the data to form vertex trajectories. Each trajectory is stored separately. We then cluster all trajectories using Lloyd's [Llo82] algorithm in combination with principal component analysis, to segment the mesh into parts which move almost independently. These mesh parts then can be compressed more efficiently with lesser components than using standard principal component analysis on the complete animation as

done in previous approaches. The resulting eigenvectors and weights of the clustered PCA afterwards are compressed in the time domain. All data is further quantized to achieve an even better compression before transmission. Connectivity data is also compressed using standard techniques. The method allows for a fast reconstruction on the GPU, using programmable shaders. Another application of our method is the user-controlled segmentation of an object, based on a sample animation.

The rest of the paper is organized as follows. Section 2 gives an overview of related work; in Section 3 the different stages of our proposed compression method are described; in Section 4 we present results and comparisons to other compression schemes and conclude with a discussion in Section 5.

2. Previous Work

The storage and transmission over bandwidth-limited channels of large data is always a challenging problem. In the context of computer graphics the existing methods can be divided into the geometry compression of static and dynamic meshes. A good overview of geometry compression is given in [TR99]. Most research is done to compress static geometry. Here, methods to compress geometry (vertex positions) [Dee95, TR98, KG00], connectivity [SG98, TG98, Ros99], multi-resolution [AD01, KBG02] and progressive meshes [Hop96, KSS00, PR00] exist. While it is possible to use all static compression schemes also on a frame-to-frame basis for animations, no time and space coherence can be used, which is crucial to achieve higher compression rates for animations.

To compress complete animations geometry, Lengyel [Len99] proposes the decomposition of the mesh into subparts and the description of these parts as rigid-body motion. For the segmentation process only a heuristic solution is provided. Alexa et al. [AM00] represent animations by using principal component analysis (PCA) to reduce the data amount. Our work is very closely related to these two papers. For example Nishino et al. [KNI01] applied PCA to the reparameterized images of an object viewed from different poses and obtained so-called *eigen-textures* and Matusik et al. [WL02] compressed the pixels of captured reflectance fields applying PCA to 8 by 8 image blocks and Sloan et al. [PPSS03] uses CPCA for efficient radiance transfer computation. Karni et al. [KG04] use the principal component idea and also linear prediction coding to exploit temporal coherence. Briceno et al [BSM*03] use geometry images [GGH02] to compress every frame of an animation. Reconstruction artifacts can occur due to sampling problems. A costly re-meshing process is also involved and reconstruction is only fast for small image sizes. The Dynapack algorithm as introduced by Ibarria and Rossignac [IR03] exploits inter-frame coherence and uses two predictors to encode the mesh motion. Most recently, Guskov et al. [GK04] uses wavelets to exploit parametric coherence in the animation sequences. While a good compression performance is achieved, a multi-resolution transform is needed and only inter-frame coherence is used. Due to the usage of wavelets, the reconstruction may become computationally costly depending on the used filter kernels. We also give a brief overview of automatic mesh segmentation methods, because our method uses segmentation to achieve better compression results. Besides the spectral compression method by Karni et al. [KG00], mesh segmentation was rarely used to compress animations sequences. For static geometry several promising approaches exist. For instance, the algorithm of Katz et al. [KT03] uses geodesic distance and convexity information. Shlafman et al. [STK02] propose decomposition by assigning faces to a patch based on physical and angular distances, to allow the metamorphosis between two meshes. GPU-based mesh segmentation was shown in [JDH04]. Because our segmentation metric is based on the motion of a vertex present in the animation, most of the above methods are not applicable to our problem.

3. Algorithm Overview

This section describes the new compression method in detail. An overview of the data flow is given in Figure 2. In the following we assume a triangle mesh with V vertices and an animation with F frames in length. We further assume constant mesh connectivity. This is true for most animations based on bones models, space warps or simulation results.

3.1. Animation Representation

Several methods exist to create 3D animations in computer graphics. Simple animations can be described with equations which model the trajectories of the vertices. But to achieve highly complex motion, the classical approaches involves human animators, who first model a character or object and then create key-frames, using techniques like inverse kinematic. To achieve more realistic motion, physically based simulation, which describe, for instance, cloth or hair behavior are also incorporated. Based on basic animations and some constraints, new motions can be synthesized. For film and entertainment, motion capturing systems with real actors are used to animate the bones model of a virtual object. Therefore, assuming a constant face connectivity, animations can be thought of a matrix A which stores the vertex positions for each frame in its columns:

$$A = \begin{pmatrix} v_{11} & \cdots & v_{1F} \\ \cdots & \cdots & \cdots \\ v_{V1} & \cdots & v_{VF} \end{pmatrix} = \begin{pmatrix} T_1 \\ \cdots \\ T_V \end{pmatrix} \qquad (1)$$

with V as the number of vertices, F as the number of frames in the animation and T as the vertex trajectories. The main

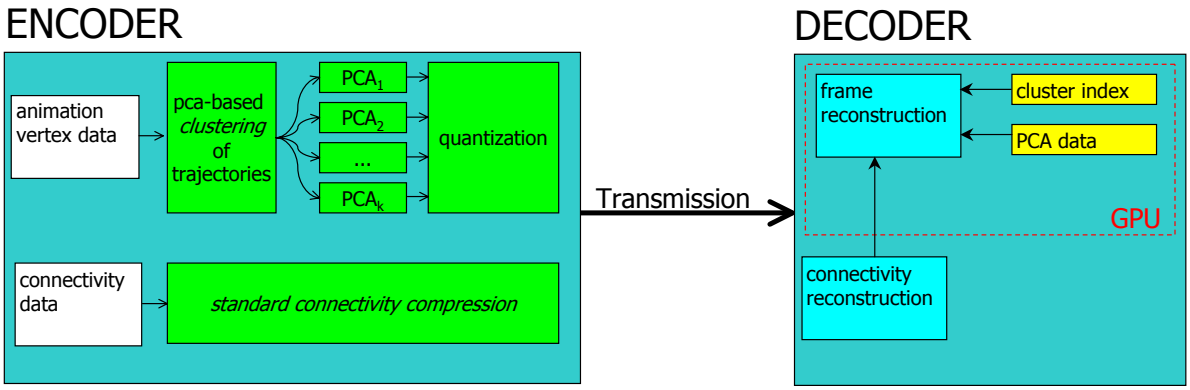

Figure 2: *Simplified data flow of our method.*

goal is to compress this data to save memory and bandwidth. If a lossy compression scheme is used, the reconstruction of the matrix should be done with the least error with regard to the positions of the vertices during the animation.

3.2. Mesh Segmentation and Linear Basis Decomposition

Given an animation e.g. based on physical simulation, the vertex paths through space are different and the relative relationship between the paths might show a highly nonlinear behavior. In this case, dimensionality reduction of the complete animation using PCA, which projects the sequence onto a linear subspace, will not be very efficient. Nevertheless, many high-dimensional data sets show a local linear behavior. That is, some vertex trajectories lie similar in space, meaning that their vertices move similar within the animation. The main idea of our proposed method is to consider the vertex trajectories for compression and not the single frames, using the local linear data assumption. Therefore, instead of clustering the frames (matrix A), we propose to cluster the trajectories (matrix A^T). It should be the goal, to find clusters of local linear data, which is equivalent to a segmentation of the animated mesh into parts. The efficiency of the final compression relies on these clusters of the data.

As pointed out in the previous work, Lengyel [Len99] proposes a heuristic solution for the segmentation, but a better segmentation can be obtained using data analysis techniques.

Therefore, we apply a clustered PCA to the animation data, which was introduced by Kambhatla and Leen [NT97] to the machine-learning community in competition to classical non-linear/neural-network learning algorithms. It combines clustering and PCA using the reconstruction error as metric for choosing the best cluster. In contrast to more sophisticated non-linear dimensionality reduction techniques, this method introduces no additional run-time cost to the reconstruction apart from a simple cluster look-up. The clustering in the encoding stage (see Figure 2) of the algorithm can be summarized as follows:

1. Initialize k cluster-centers r_j randomly chosen from the dataset. Assign a collection of c unity basis vectors $e_{i,j}$ to each cluster.
2. Partition the dataset into regions by assigning each data-vector to its closest center. The distance to a center r_j is given by squared reconstruction error:
$$||x-\tilde{x}_j||^2 = ||x-r_j-\sum_{i=1}^{c}\langle x-r_j,e_{i,j}\rangle e_{i,j}||^2$$
where x is the original and $\tilde{x}_j$ the reconstructed data vector.
3. Compute new centers r_j as the mean of the data in the region j.
4. Compute a new set of basis-vectors $e_{i,j}$ per region, i.e. perform a PCA in each region.
5. Iterate steps 2.-4. until the change in average reconstruction error falls below a given threshold.

After this process, we have a cluster index I_m, $m \in [1,\ldots,V]$, which stores for each original vertex trajectory T_m the number of the cluster, it belongs to.

We also get c eigenvectors per cluster, which we will call *Eigentrajectories*. These Eigentrajectories E_{ij}, $j \in [1,\ldots,c]$ were generated in the last step 4 of the clustering process.

Furthermore, The first c Eigentrajectories approximate any of the original trajectories in their cluster i in such a way that the sum of the squares of the projection errors onto the affine subspace spanned by $\{E_{I_m1},\ldots,E_{I_mc}\}$ is minimized

$$T_m \approx \sum_{h=1}^{c} w_{I_mh}E_{I_mh} \qquad (2)$$

The coefficients $w_{I_mh} = \langle T_m, E_{I_mh}\rangle$ are weights for the cluster I_m, were $\langle,\rangle$ denotes the standard scalar product in $\mathbb{R}^{3\times F}$.

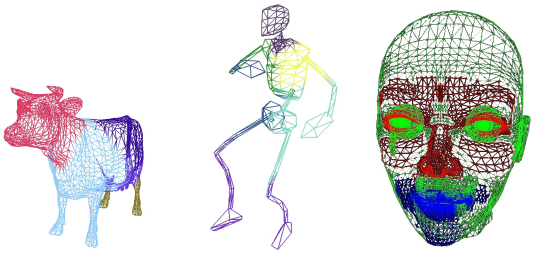

Figure 3: *Clustering based on animation data. Right to left:* ***cow*** *with 5,* ***dance*** *with 6 and* ***head*** *with 3 clusters, each colored differently.*

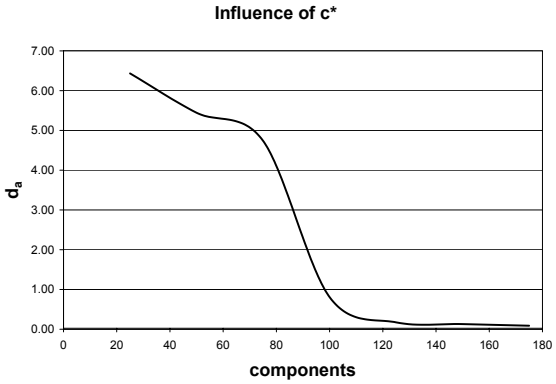

Figure 4: *Influence of different numbers of components c^* on the reconstruction quality d_a for the chicken sequence.*

Using this method, for the given number of clusters, guarantees the best reconstruction error per cluster for a given number of components. The clustering itself heavily depends on the given model and the animation. While, on the one hand, it is possible to let the user choose the number of clusters at the onset, based on visual perception, it is also possible, to have an iterative determination of k. The desired maximal reconstruction error or the available bandwidth for transmission can be used as a stopping criterium for increasing k, starting with $k = 1$, similar to [KT03]. Clustering usually is in the order of minutes (see Tab. 3). Therefore, the computer can search for an optimal k.

Results for the segmentation for some of the test animations can be seen in Figure 1 and 3. Each of the clusters is coded in a different color. Note the segmentation of the *POSER* head, which is animated by facial expressions. It shows clearly the three main parts (eye and mouth region, rest of the head).

In the case of $F \gg V$, we always rearrange the matrices in a way, that the covariance matrix has the size of the smaller part of (F,V), as shown in [NdS01]. Furthermore, due to the offline nature of this preprocessing step, out-of-core techniques implemented in packages like *LAPACK* [LAP] are used.

3.3. Error Measurement

To compare the performance of our method with regard to compression and reconstruction quality, we introduce a distortion factor d_a similar to [KG04], which measures the quality of the vertex position reconstruction for the complete animation. d_a is defined as follows:

$$d_a = 100 \frac{||B - \hat{B}||}{||B - C(B)||}$$

where B is a matrix with the dimensions $3V \times F$ containing the original animation sequence. $\hat{B}$ is the same matrix after the compression and reconstruction stage. $C(B)$ is a matrix in which each column consists of the average vertex positions for all frames.

We also compute the per-frame distortion d_f which is defined as the L2 norm of all original and reconstructed vertex positions of a frame.

3.4. Compression of Eigentrajectories

Note, that the eigentrajectories E_{ij}, we computed above are still of the length F. To further compress this data, we again apply a PCA on these vectors. This results in c^* new eigenvectors and the corresponding eigenvalues. This is especially useful for long sequences. In contrast to the first step we now compress the time axis. After this second compression step, we quantize the new eigenvectors and weights for transmission, which is discussed in the next subsection. Using more than $c^* = 100$ components gives nearly the same error than without time domain compression ($d_a = 0.03$). Figure 4 shows the influence of different numbers of components c^* on the reconstruction error d_a of the complete chicken animation ($k = 10, c = 20$). Figure 5 shows the per frame distortion d_f.

3.5. Quantization

Before storing or transmitting data, quantization is a common technique to reduce the floating-point data (32 or 64 bits), by using only q bits to represent a float value. This is done via a normalization process, which computes a tight, axis-aligned bounding box for the geometry. In the case of an animation, one first computes the center of gravity of each frame and chooses the largest occurring bounding box for all frames. As pointed out by some authors [KG04, Ros04] $q = 12$ bits can be treated as "lossless" compression with regard to visual quality as seen in the top of Figure 7. But to operate with full precision, we compute the PCA on the original floating-point data and quantize only the components and weights as shown in the dataflow in Figure 2. This is

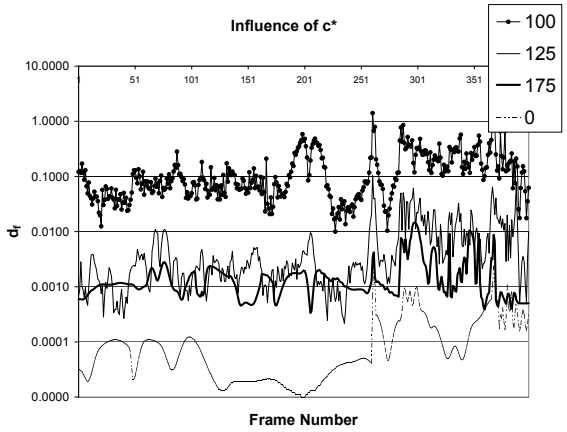

Figure 5: *Influence of different numbers of components c^* on the reconstruction quality per frame d_f for the chicken sequence.*

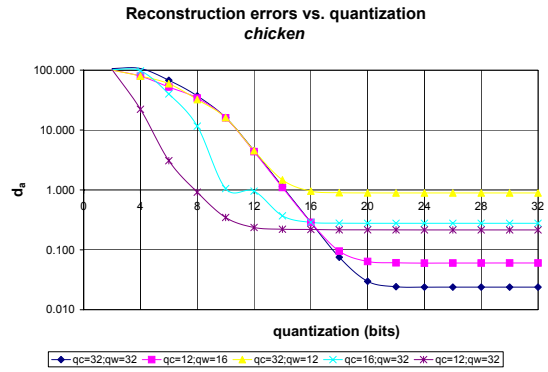

Figure 6: *Reconstruction errors for different combinations of q_c and q_w for the **chicken** sequence.*

done separately with two bit values q_c and q_w for the components and the weights respectively before transmitting them. The bottom of Figure 7 shows the reconstruction samples for different quantization levels (top row) and different values for q_c and q_w (bottom row) for the chicken sequence with 20 clusters and 10 components. As shown in Figure 6 the reconstruction error d_a is more effected by the quantization of the components, than by the quantization of the weights. In practice, we suggest to use at least 18 bits for both q_c and q_w.

3.6. Decompression

After transmission, we first decompress the Eigentrajectories. After this, we have k clusters, each with c PCA components and the corresponding weights, which we obtained after the clustering process. Now we are able to reconstruct

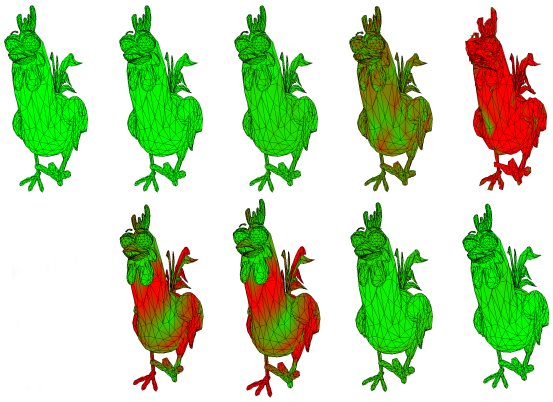

Figure 7: ***Top row:*** *Different quantization levels (q=32,12,10,8,6 bits) of the original data for the chicken sequence . **Bottom row:** Reconstruction results using different quantization for q_c and q_w (q_c=12,q_w=12/q_c=12,q_w=32/q_c=32,q_w=12/q_c=16,q_w=16). Color-coded error to original 32bit frame. See text for details.*

```
for all vertices j do
    vec4 newVertexPos;
    float clusterIdx = texture2D(texClusterIdx, texCoord[j]);
    vec4 weights = texture2D(texWeights, texCoord[clusterIdx]);
    newVertexPos.xyz = texture3D( texClusterMean, texcoord[clusterIdx]);
    for i = 1 to NumComponents do
        newVertexPos +=
        weights[i]*texture3D(texClusterComponents, texcoord[clusterIdx]);
    end for
    glPosition = glModelViewProjectionMatrix * newVertexPos;
end for
```

Figure 8: *Pseudocode to compute new vertex positions in a shader program for decompression*

each trajectory of the animation, hence each frame. In addition, we also have the cluster index I_m.

As shown in Figure 2 the reconstruction can easily be done on the GPU, because all data can be stored in graphics memory and only simple matrix multiplications have to be computed to reconstruct a certain frame. This is easily be done within a shader program [ARB].

Figure 8 gives an overview of the GPU shader program written in GLSL pseudocode for decompression. The cluster index and the weights are stored in 2D textures, while the PCA data is stored in 3D Textures. After several lookups the new vertex position is calculated.

name	vertices V	triangles T	frames F
chicken	3030	5664	400
cow	2904	5804	204
dance	452	570	1733
dolphin	6179	12337	101
face	539	1042	10001
head	8172	15974	500

Table 1: *Used animation sequences.*

4. Results

The following section describes the data used and introduces some error measurements, followed by an evaluation of the method and the comparison to other compression schemes.

4.1. Data Sets

All of the following data sets are animations of polygonal meshes and the number of faces and their connectivity does not change over time. The animation was done either by hand or by the usage of motion capture data or pre-generated basic facial expressions. Table 1 shows basic information for all animations sequences. Some of the animations can be viewed in the accompanying video and sample frames can be seen in Figure 11.

4.2. Compression & Reconstruction Results

For the evaluation of our method we use the reconstruction error metrics described in Section 3.3 and the *bpvf* (bits per vertex per frame) unit for bandwidth usage measurements. Because we have to transmit the face connectivity in beforehand as *payload*, we distribute the data over the whole animation for comparison reasons. Note, that we use the *Edgebreaker* [Ros99] method with freely available source code, to compress the connectivity information. In the worst case we then need $6V$ bits to store the data. Table 2 summarize the compression results for different bpvf for the used animations. Note, that c^* is used only for the long sequences.

4.3. Timings

The compression was done using an AMD Athlon64 XP 3200+ with Windows 2000. As graphics adapter an ATI Radeon X800pro was used. Table 3 shows compression and decompression timings for several test animations. Compression times are in seconds for the clustering, principal component analysis and saving. Frames per second (FPS) for display while reconstructing. It seems, that the performance heavily depends on the number of clusters used.

name	bpvf	d_a	c^*	c	k
chicken	8.7	0.002		60	2
	4.7	0.076		20	10
	2.8	0.139		20	5
cow	7.4	0.16		40	5
	3.8	0.50		20	5
	2.0	1.47		10	5
dolphin	7.1	0.024		20	2
	4.1	0.033		10	4
	2.1	0.168		10	2
head	7.4	0.003		40	10
	5.3	0.003		60	2
	2.5	0.083		20	5
dance	6.1	0.21	-	5	10
	4.9	0.25	40	5	10
	4.3	0.48	35	5	10
	2.5	0.87	-	2	10
	1.9	2.06	15	2	10
	1.3	4.57	10	2	10
face	10.0	0.013	-	5	20
	5.1	0.023	50	5	20
	5.0	0.029	-	5	10
	2.5	0.038	25	5	10
	1.5	0.058	15	5	10

Table 2: *Compression results.*

name	clusters k	components c	t (sec)	FPS
chicken	2	60	258	105
chicken	10	20	206	214
chicken	5	20	395	215
cow	5	40	75	145
cow	5	20	59	218
cow	5	10	55	284
face	10	40	2730	648
face	2	40	979	654
face	5	20	1320	865

Table 3: *Compression/Decompression timings for several animations.*

4.4. Comparison

In comparison to other approaches our new method performs very well. We compare our method to the wavelet (AWC) [GK04] and linear prediction coding (KG) [KG04]. The values were obtained from the corresponding papers. Figure 10 shows a graphical result for the cow sequence. Depending on the quantization level we used (32 or 18 bits for the PCA Components), we come close or even outperform all other methods. The right side of Figure 10 uses fixed bpvf to visualize d_f. We used k=5, c=40; k=10, c=20 and k=5, c=20 for d_f=7.4, 5.7 and 3.8 respectively. Besides this, our method outperforms the KG method for long sequences also, as can be seen in Figure 9.

To calculate the bpv per frame (bpvf) we used the following equation:

$$bpvf = \frac{6V + q_c kc3F + q_w cV + 5V}{FV} \tag{3}$$

with V and F as the number of vertices and frames of the animation. The first part of the equation encodes the connectivity, the second part the PCA data (with k as the number of clusters and c as the number of components), the third part the PCA weights and the last part represents the cluster index encoded with 5 bits. The cluster index can be avoided, by rearranging the data in a way, that all trajectories are ordered by the cluster number. Then, equation 3 becomes:

$$bpvf = \frac{6V + q_c kc3F + q_w cV}{FV} \tag{4}$$

using some *stopbits* to indicate the cluster change. Note, that the latter will not allow for random access the trajectories, which might become to costly during decompression. With compression of the Eigentrajectories for long sequences 4 becomes:

$$bpvf^* = \frac{6V + q_c kcc^* + q_w 3Fc^* + q_w cV}{FV} \tag{5}$$

using $c^* < F$.

5. Discussion and Conclusions

To achieve efficient compression on the 3D data of a soft-body animation sequence, the exploitation of the spatial and time correlation of the data is crucial. Our new method uses clustered principal component analysis to separate the given geometry into coherent parts, in regard to the animation. We also do not treat the animation as a series of static meshes and frames, but rearrange the data to analysis the trajectory of each vertex individually. Compression is then done on the trajectories of all vertices of a cluster. Compression rates of the sub-meshes outperform simple principal component compression or combination with linear prediction encoding and for some animation types even wavelet-based methods. If the number of frames is much higher than the number of vertices in the animation, further PCA-based compression in the time domain of the Eigentrajectories is performed. Our method requires no meta knowledge besides the geometry data, e.g. no bones model information. Given sufficient animation data, it allows for automatic segmentation of the mesh. Memory problems which might can occur during the principal component analysis, can be handled using out-of-core methods. Reconstruction is easily done on the GPU. We also want to combine our method with linear prediction coding, which might lead to even higher compression rates. As shown in the result section, using a different number of clusters in combination with the distortion error, the method might be able to "predict" a meaningful number of parts for the segmentation of the object. In this version of the algorithm the number of components per cluster is fixed. Given a global distortion error, we also want to change the number of components individually per cluster and apply a cluster-wise different quantization on a local basis. Additionally, we want to perform a complexity analysis with more animations to evaluate how our approach scales.

Acknowledgements

The authors would like to thank their colleagues Gabriel Zachmann and Gero Müller for fruitful discussions and help with coding and Jutta Adelsberger and Michael Kusak for helpful hints. The ***chicken*** sequence is property of Microsoft Inc. and was kindly provided by John Snyder. *POSER*® was used for the creation of the dancing sequence with motion capture data and the face animation with expressions. The ***cow*** animation was created by Matthias Müller (ETH Zürich). The ***dolphin*** and ***face*** sequences were kindly provided by Zachi Karni.

References

[AD01] ALLIEZ P., DESBRUN M.: Progressive compression for lossless transmission of triangle meshes. In *Proceedings of the 28th annual conference on Computer graphics and interactive techniques* (2001), ACM Press, pp. 195–202.

[AM00] ALEXA M., MÜLLER W.: Representing animations by principal components. *Computer Graphics Forum 19*, 3 (2000).

[ARB] ARB: http://oss.sgi.com/projects/ogl-sample/registry/.

[BSM*03] BRICENO H. M., SANDER P. V., MCMILLAN L., GORTLER S., HOPPE H.: Geometry videos: a new representation for 3d animations. In *Proceedings of the 2003 ACM SIGGRAPH/Eurographics Symposium on Computer Animation* (2003), Eurographics Association, pp. 136–146.

[Dee95] DEERING M.: Geometry compression. In *Proceedings of the 22nd annual conference on Computer graphics and interactive techniques* (1995), ACM Press, pp. 13–20.

[GGH02] GU X., GORTLER S. J., HOPPE H.: Geometry images. In *Proceedings of the 29th annual conference on Computer graphics and interactive techniques* (2002), ACM Press, pp. 355–361.

[GK04] GUSKOV I., KHODAKOVSKY A.: Wavelet comrpession of parametrically coherent mesh sequences. *Eurographics Symposium on Computer Animation 2004* (2004).

[Hop96] HOPPE H.: Progressive meshes. In *Proceedings of the 23rd annual conference on Computer graphics and interactive techniques* (1996), ACM Press, pp. 99–108.

[IR03] 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* (2003), Eurographics Association, pp. 126–135.

[JDH04] JESSE D. HALL J. C. H.: Gpu acceleration of iterative clustering. *Manuscript accompanying poster at GP2̂: The ACM Workshop on General Purpose Computing on Graphics Processors, and SIGGRAPH 2004 poster* (2004).

[KBG02] KARNI Z., BOGOMJAKOV A., GOTSMAN C.: Efficient compression and rendering of multi-resolution meshes. In *Proceedings of the conference on Visualization '02* (2002), IEEE Computer Society, pp. 347–354.

[KG00] KARNI Z., GOTSMAN C.: Spectral compression of mesh geometry. In *Siggraph 2000, Computer Graphics Proceedings* (2000), Akeley K., (Ed.), ACM Press / ACM SIGGRAPH / Addison Wesley Longman, pp. 279–286.

[KG04] KARNI Z., GOTSMAN C.: Compression of soft-body animation sequences. *Computer and Graphics 28* (2004), 25–34.

[KNI01] K. NISHINO Y. S., IKEUCHI K.: Eigen-texture method: Appearance compression and synthesis based on a 3d model. *IEEE Trans. on Pattern Analysis and Machine Intelligence 23*, 11 (2001), 1257–1265.

[KSS00] KHODAKOVSKY A., SCHRÖDER P., SWELDENS W.: Progressive geometry compression. In *Proceedings of the 27th annual conference on Computer graphics and interactive techniques* (2000), ACM Press/Addison-Wesley Publishing Co., pp. 271–278.

[KT03] KATZ S., TAL A.: Hierarchical mesh decomposition using fuzzy clustering and cuts. *ACM Trans. Graph. 22*, 3 (2003), 954–961.

[LAP] LAPACK: http://www.netlib.org/lapack/.

[Len99] LENGYEL J. E.: Compression of time-dependent geometry. In *Proceedings of the 1999 symposium on Interactive 3D graphics* (1999), ACM Press, pp. 89–95.

[Llo82] LLOYD S. P.: Least square quantization in pcm. *IEEE Transactions on Information Theory 28*, 2 (1982), 129–137.

[NdS01] NAVARRETE P., DEL SOLAR J. R.: Eigenspace-based recognition of faces: Comparisons and a new approach. In *Proceedings of the 11th International Conference on Image Analysis and Processing* (2001), p. 42.

[NT97] N.KAMBHATLA, T.K.LEEN: Dimension reduction by local pca. *Neural Computation, 9* (1997), 1493–1516.

[PPSS03] PETER-PIKE SLOAN JESSE HALL J. H., SNYDER J.: Clustered principal components for precomputed radiance transfer. *ACM Transactions on Graphics 22*, 3 (2003), 382–391.

[PR00] PAJAROLA R., ROSSIGNAC J.: Compressed progressive meshes. *IEEE Transactions on Visualization and Computer Graphics 6*, 1 (2000), 79–93.

[Ros99] ROSSIGNAC J.: Edgebreaker: Connectivity compression for triangle meshes. *IEEE Transactions on Visualization and Computer Graphics 5*, 1 (/1999), 47–61.

[Ros04] ROSSIGNAC J.: *Surface simplification and 3D geometry compression; Chapter 54 in Handbook of Discrete and Computational Geometry*, 2nd ed. Editors: J. E. Goodman and J. O'Rourke, 2004.

[SG98] S. GUMHOLD W. S.: Real time compression of triangle mesh connectivity. *ACM SIGGRAPH '98 Proceedings* (1998), 133–140.

[STK02] SHLAFMAN S., TAL A., KATZ S.: Metamorphosis of polyhedral surfaces using decomposition, 2002.

[TG98] TOUMA C., GOTSMAN C.: Triangle mesh compression. In *Graphics Interface* (June 1998), pp. 26–34.

[TR98] TAUBIN G., ROSSIGNAC J.: Geometric compression through topological surgery. *ACM Transactions on Graphics 17*, 2 (1998), 84–115.

[TR99] TAUBIN G., ROSSIGNAC J.: 3d geometry compression. *Siggraph Course Notes*, 21 (1999).

[WL02] W.MATUSIK H.-P.PFISTER A. P. R., L.MCMILLAN: Image-based 3d photography using opacity hulls. *ACM Transactions on Graphics 21*, 3 (2002), 427–437.

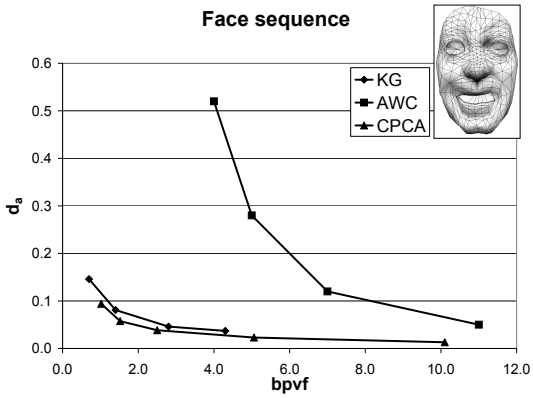

Figure 9: . *Comparison of our method with other algorithms for the face sequence.*

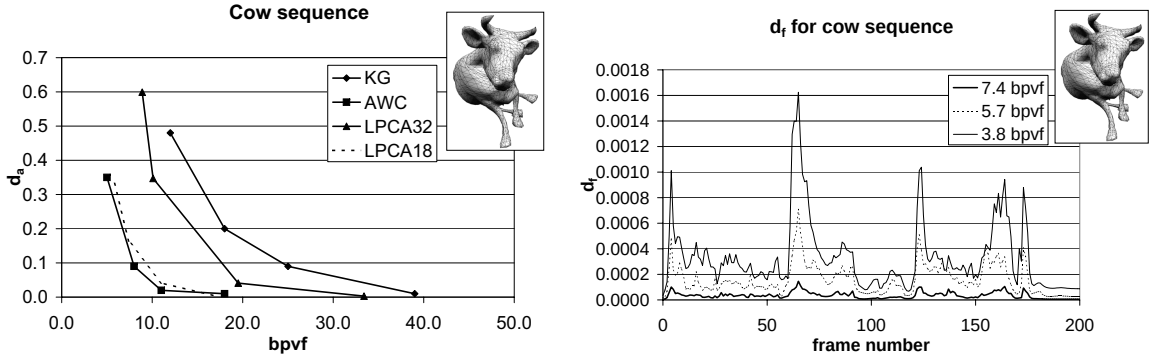

Figure 10: . *Left: Cow sequence comparison to other methods. Our CPCA method with 18 bit PCA quantization yields very good results. Left: d_f for the cow sequence with different bpvf settings.*

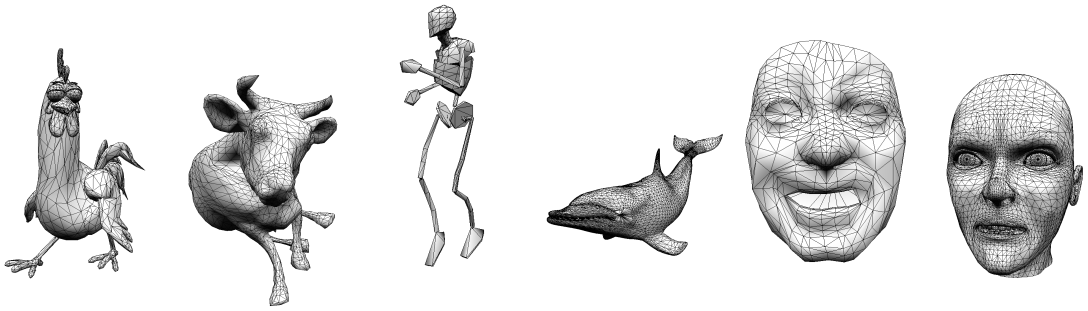

Figure 11: *Sample frames of the used animations. From left to right:* ***chicken***, ***cow***, ***dance***, ***dolphin***,***face*** *and* ***head***.