Please cite as: T. Winkler, J. Drieseberg, A. Hasenfuß, B. Hammer, and K. Hormann. Thinning mesh animations. In O. Deussen, D. Keim, and D. Saupe, editors, *Proceedings of Vision, Modeling, and Visualization 2008*, pages 149–158, Konstanz, Germany, October 2008. Aka.

Thinning Mesh Animations

Tim Winkler Jens Drieseberg Alexander Hasenfuß
Barbara Hammer Kai Hormann

Department of Informatics
Clausthal University of Technology

Abstract

Three-dimensional animation sequences are often represented by a discrete set of compatible triangle meshes. In order to create the illusion of a smooth motion, a sequence usually consists of a large number of frames. We propose a pre-processing algorithm that considerably reduces the number of frames required to describe the whole animation. Our method is based on *Batch Neural Gas* [11], a new clustering and classification approach that can be used to automatically find the most relevant frames from the sequence. The meshes from the original sequence can then be expressed as linear combinations of these few key-frames with small approximation error. The key-frames can finally be compressed with any state-of-the-art compression scheme. Overall, this leads to improved compression rates as the number of key-frames is significantly smaller than the number of original frames and the storage overhead for the reconstruction weights is marginal.

1 Introduction

Consider an artist who models his story in the style of a flip-book with one drawing per page. When flipping through the pages, the animation sequence is then created by linear interpolation between the frames and in order to create a smooth animation, the flip-book will contain many pages. However, we could still understand the story if we leafed through the pages one by one, and would certainly be less bored if the number of pages was reduced to the most relevant ones: the *key-frames*.

Let us now add a third dimension to the drawings and consider 3D animation sequences that are based on triangle meshes. Again, a smooth motion is described by a lot of meshes which in turn constitute an enormous amount of data. In order to store and transmit this data, it needs to be compressed and plenty of sophisticated compression approaches exist to do so. However, the compression rate can be improved by first reducing the sequence to a smaller number of important key-frames.

In this paper we show how to find these few key-frames and how to faithfully reconstruct the whole animation via linear interpolation.

1.1 Related Work

A very simple approach for compressing dynamic meshes is to use *static mesh compression* (see [28], [4], and [27] for surveys) on each frame of an animation sequence. This approach is particularly useful in case the connectivity of the meshes changes from frame to frame. For a compatible sequence, using a static coder on each frame results in poor compression rates since their spatial and temporal coherence is not exploited.

Dynamic mesh compression techniques instead utilize this information. The prediction-based approaches of Ibarria and Rossignac [16] and Yang et al. [34] compute the new vertex positions based on the change of the neighbouring vertices in the previous frames of the sequence.

Another option are wavelet-based methods. For example, Guskov and Khodakovsky [12] apply wavelets on top of a progressive mesh hierarchy and the resulting wavelet details are encoded in a progressive manner, yielding good results if the input sequence is a rigid-body motion. The goal of

the temporal lifting scheme introduced by Payan and Antonini [26] is to exploit the coherence in the geometry of successive frames to reduce the information needed to represent the original sequence.

Geometry videos, introduced by Briceño et al. [9] use a remeshing step to discard the original connectivity information. The main drawback of this approach is its high computational cost.

Another class of compression schemes are clustering approaches. They divide the mesh into subparts and the motion of these subparts is expressed as a set of rigid-body transformation. This idea was first introduced by Lengyel [20] and Boulfani et al. [8, 7] later combined clustering with wavelets.

From another point of view, the frames of an animation sequence can be interpreted as the observations of a statistical experiment with the mesh vertices as variables. A well known tool to reduce the information in such a data set is the *principal component analysis* (PCA). The first approach that used PCA for mesh compression was proposed by Alexa and Müller [3]. Here the vertex positions in each frame are interpreted as the columns of a matrix. The eigenvectors obtained by singular value decomposition (SVD) of this matrix capture the information inherent to the animation.

The PCA approach is efficient for sequences with small meshes as the matrix size is given by the number of vertices. The main drawbacks of this approach are the computationally challenging SVD and the fact that many animated meshes contain highly non-linear behaviour which is hard to capture globally using this approach. Karni and Gotsman [17] extended the PCA approach by coding the eigenvectors with linear prediction. Lee et al. [18] further showed how to find the optimal number of eigenvectors for a given sequence automatically. Amjoun and Straßer [5] cluster the vertex positions into several local coordinate frames (LCF) and execute a PCA on each LCF with an optimally chosen number of eigenvectors for each LCF. Instead of clustering the vertex positions, Sattler et al. [29] propose to cluster the vertex trajectories in combination with a local PCA. But although this approach scales well with the size of the meshes, it breaks down for extremely long sequences because here the size of the PCA matrix is given by the number of frames.

Although the basic PCA-based approach [3] and its improved version [17] reconstruct the sequence in the optimal way in terms of the L_2-norm, they suffer from the aforementioned drawbacks. Nevertheless, we still advocate that the general idea behind these approaches is promising for a pre-processing step. Consider the flip-book example from the beginning: once we have found the subset of important frames, the key-frames, we can then use again linear interpolation to reconstruct the complete sequence from this subset. And since these key-frames are part of the original sequence, we can also use any of the schemes mentioned above to compress this subset of frames.

A first approach to extract meaningful frames out of captured motion data was presented by Lim and Thalmann [21]. It is based on curve simplification and the meaningful frames of the sequence are in fact part of the sequence. This holds as well for the approach by Huang et al. [14], who propose to solve a constrained matrix factorization problem in a least-squares sense, but since this is an iterative approach it is not very efficient for long sequences. An optimized version was recently presented by Lee et al. [19]. They first simplify each mesh of the sequence and then use a genetic algorithm (GA) to search for representative key-frames. Although their approach is much faster than [14], the number of iterations needed by the GA to converge is the bottleneck. Huang et al. [15] discuss how to extract key-frames from 3D video. They reformulate the key-frame selection process as a shortest path problem in a graph that is constructed from a self-similarity map. Since this approach is developed for complete 3D video sequences, the connectivity of the mesh as well as the vertex positions are allowed to change in each time step. This setting, however, is not the focus of our approach.

1.2 Contribution

We present a fast and simple framework that attacks the problem of reducing the amount of data in mesh animations from a slightly different angle. Our framework automatically clusters the meshes into groups representing similar deformations (Section 2). We show that it is possible to reconstruct the whole sequence from these key-frames with negligible error (Section 3). Since each centre of a cluster is a member of the underlying manifold of meshes, the set of key-frames extracted by our framework fits excellent as input to *any* state-of-the-art compression algorithm mentioned in Section 1.1.

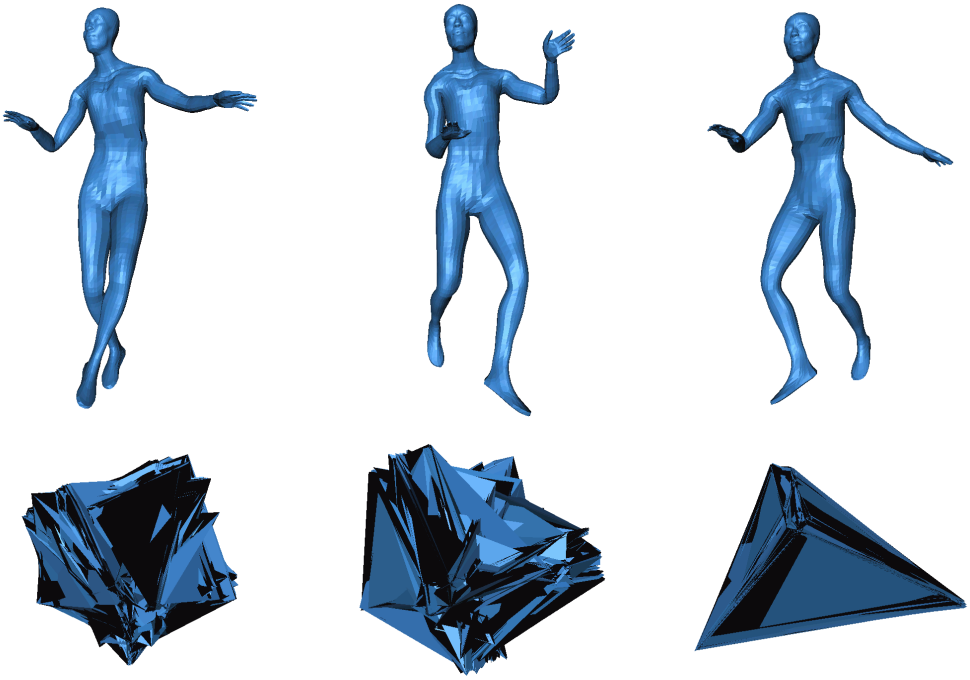

Figure 1: Some of the key-frames extracted from the dance sequence by our Batch Neural Gas approach (top) and some of the PCA eigenvectors (bottom).

2 Framework overview

In this section we explain the different components of our pre-processing framework. Suppose we have a sequence of meshes $m_1, \ldots, m_n$. We assume that all m_i have the same connectivity and only the vertex positions change over time.

In Section 2.1 we explain how to retrieve a set of meaningful key-frames $k_1, \ldots, k_l$ using Batch Neural Gas (BNG). We then use them as a basis for reconstructing the original meshes m_i as linear combinations (Section 2.2). The approximation quality is comparable to the one obtained from PCA (Section 2.3), but it can be improved by using deformation gradients [6, 31] (Section 2.4).

2.1 Batch Neural Gas

We consider a key-frame k_j to be *meaningful* if it lies on the same data manifold as the input meshes m_i or at least close to it, in contrast to the PCA eigenvectors which do not satisfy this property; see Figure 1. Although this restriction gives us less degrees of freedom as compared to PCA, we still need only slightly more key-frames to obtain comparable reconstruction results (Section 2.3).

Key-frames are further considered *optimal* if they allow to reconstruct all meshes of the sequence with minimal global approximation error, measured in the L_2-norm. Optimal key-frames can thus be found by minimizing a quadratic cost function, and without any other constraints this yields exactly the PCA solution. But if we also want the key-frames to be meaningful, we end up with a rather complex optimization problem.

However, we can solve a simpler problem instead and still get reasonable results. Usually the data manifold is highly non-linear and the linear combination of meaningful key-frames can lie outside the manifold. As a consequence, meshes from the original sequence can only be obtained by linear combinations of close-by key-frames. Therefore, the overall approximation error is also going to be small if we minimize instead the standard quantization error [13], i.e. the squared Euclidean distance of all m_i to their respective closest key-frame.

The quantization error constitutes a classical objective of clustering algorithms and one of the most popular clustering algorithms which is directly based on this cost function is the k-means algorithm [13]. For instance, it is used by Park and Shin [25] for example-based motion cloning. However, it is well known that k-means clustering is highly sensitive to initialization and usually finds only local optima of the cost function, i.e. suboptimal key-frames. Therefore, we use an efficient alternative that optimizes the same cost function as k-means clustering in the limit, but does not suffer from the problems of k-means. Our method is based on the Neural Gas (NG) algorithm by Martinetz et al. [23]. NG is a vector quantization technique that aims to represent given data (i.e. meshes) $M \subseteq \mathbb{R}^d$ faithfully by prototypes (i.e. key-frames) $k_j \in \mathbb{R}^d$, $j = 1, \ldots, l$. For a continuous input distribution given by a probability density function $P(m)$ over M, the cost function minimized by NG is

$$E \sim \frac{1}{2} \sum_{j=1}^{l} \int h_\lambda\big(\mathrm{rk}(k_j, m)\big) \|m - k_j\|^2 P(m)\, dm,$$

where $\mathrm{rk}(k_j, m) = \#\{k_i : \|m - k_i\| < \|m - k_j\|\}$ denotes the rank of the key-frame k_j arranged according to the distance from the mesh m, i.e. it indicates whether key-frame k_j is representative for m (corresponding to rank 0) or not (corresponding to a large rank). The parameter $\lambda > 0$ controls the neighbourhood range, i.e. the area of influence of each single point, through the exponential function $h_\lambda(t) = \exp(-t/\lambda)$. This important parameter is initialized with a high value and then quickly driven asymptotically to zero, yielding the characteristic dynamics of NG. For $\lambda \to 0$ the standard quantization error is recovered in the limit. By integrating the differences of the key-frames according to all meshes in the beginning and weighted according to the ranks, NG is not sensitive to initialization and able to overcome local optima, in contrast to the popular k-means algorithm.

Because the final key-frames k_i found by NG lie on the data manifold (it has been shown in [22] that the final NG solution can be extended to a valid Voronoi tessellation of the given manifold under mild conditions on the density of the mesh sequence), they are similar to meshes of the sequence. Due to the cost function E, a further benefit of NG can be observed: it has been shown in [23] that the so-called magnification factor of NG approaches 2/3. Roughly speaking, the magnification factor characterizes the fraction of meshes from the original sequence represented by one key-frame depending on the underlying density of the sequence. Because of this factor, NG focusses on regions which are only sparsely covered, while key-frames in dense regions represent a higher percentage of meshes. Thus, NG is able to adequately capture regions of the mesh sequence with large deformations.

In the original formulation NG optimizes its cost function in an online mode by using a stochastic gradient descent method. That means, that meshes are presented several times in random order to the algorithm and adaptation of the key-frames takes place after every single mesh. For that reason, a huge number of training steps is necessary for convergence [23]. However, for a given finite sequence $M = \{m_1, m_2, \ldots, m_n\}$ the cost function of NG becomes

$$E \sim \frac{1}{2} \sum_{j=1}^{l} \sum_{i=1}^{n} h_\lambda\big(\mathrm{rk}(k_j, m_i)\big) \|m_i - k_j\|^2.$$

For this special setting, Cottrell et al. [11] introduced a fast batch optimization technique that adapts key-frames according to all meshes at once, similar to the original k-means adaptation scheme. The resulting Batch NG algorithm (BNG) determines first the ranks $r_{ji} = \mathrm{rk}(k_j, m_i)$ for fixed key-frames k_j and then new key-frames via the update formula

$$k_j = \sum_{i=1}^{n} h_\lambda(r_{ji}) \cdot m_i \Big/ \sum_{i=1}^{l} h_\lambda(r_{ji})$$

for fixed ranks r_{ji}. BNG shows the same accuracy and behaviour as NG, but its convergence is quadratic instead of linear as for NG.

This scheme subsequently determines the responsibility of key-frames for meshes of the sequence by means of the ranks. Afterwards, it calculates new key-frames as the generalized mean of the meshes, weighted according to the responsibilities.

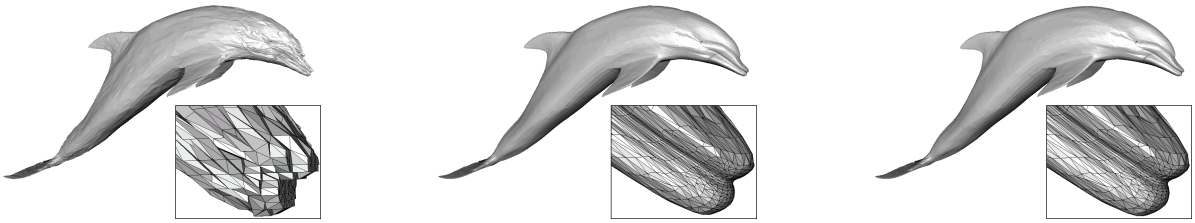

Figure 2: Comparing the KG-Error: quantization using 8 bits (left), original model (middle), reconstruction using the key-frames extracted with our approach (right).

2.2 Finding the interpolation weights

Now that we have found a set of meaningful and optimal key-frames, the next goal is to find for each mesh m_i the *weights* $\lambda = (\lambda_1, \ldots, \lambda_l)$ so that the reconstructed mesh

$$\hat{m}_i = \sum_{j=1}^{l} \lambda_{ij} k_j \tag{1}$$

is as close to the original m_i as possible. Denoting by $K = (k_1 \cdots k_l) \in \mathbb{R}^{n \times l}$ the matrix with key-frames k_j as columns, we have $\hat{m}_i = K\lambda$ and the best set of weights is found by

$$\min_{\lambda} \|m_i - \hat{m}_i\|,$$

which is equivalent to solving the normal equation

$$K^T K \lambda = K^T m_i.$$

2.3 Reconstruction quality

Instead of the d_a-error, which was introduced by Karni and Gotsman [17] and is therefore often referred to as the KG-error, we measure the quality of our reconstruction using the root mean square error [10], which was also used in [9] and [12]. As already noted by Guskov and Khodakovsky [12] the KG-error does not capture the visual quality of the reconstruction well. In Figure 2, the approximations of the dolphin model (left and right) have the same KG-error with respect to the original (middle), but the model on the left shows the well-known staircase artifacts [12] caused by quantization of the vertex positions. We only measure the KG-error in Table 1 to show that although our selected key-frames are not optimal in the sense of the L_2-norm, they still introduce only a slightly bigger error, compared to the PCA result.

Model	# KF (basis vectors)	Our approach KG-error	PCA approach KG-error
Dolphin	100	0	0.024
	50	0.029	0.029
	25	0.094	0.075
	10	0.975	0.607
Face	200	0.062	0.046
	100	0.131	0.081
	50	0.254	0.146

Table 1: Using the key-frames extracted by BNG as reconstruction basis, we get an error similar to the one achieved by PCA [17].

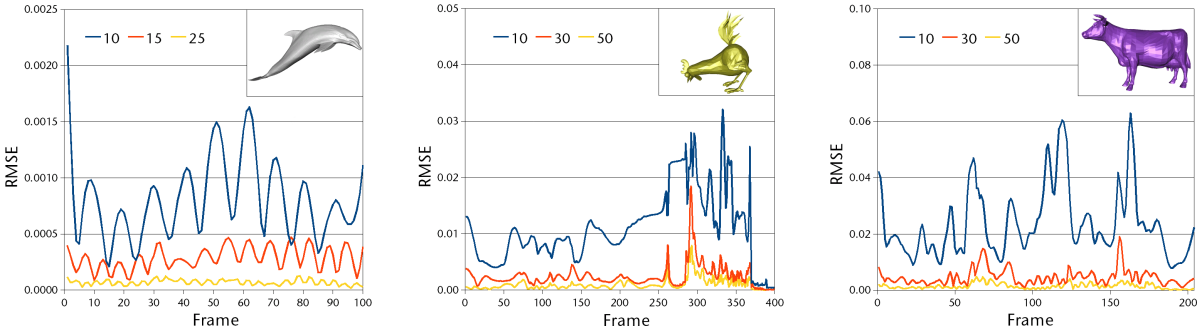

Figure 3: Error distribution relative to the number of key-frames, for the dolphin (left), chicken (middle), and cow sequence (right).

Figure 3 shows the reconstruction error for the dolphin, chicken, and cow sequence. Since the motion in the dolphin sequence is sine-like, it has a repetitive character and only moves in two dimensions, which fits well into our framework. We thus get almost as good results as [17]; see Table 1.

For the chicken sequence it is obvious that 10 key-frames are insufficient to capture the highly non-linear motion. But using 30 key-frames allows us to halve the error for the chicken at the end of the sequence. This is due to the fact that most of the motion is present at the end of the sequence when the chicken starts to panic and our framework can adapt perfectly to this by selecting more key-frames to represent the end.

The cow sequence shows a different behaviour. Right from the start of the sequence the cow is subject to extreme transformations. Using only 10 key-frames, the three times when the cow is dragged into the air are clearly reflected by the error. The frames in between are already captured well. With 30 key-frames we can also capture these parts well, leading to a reduction of the error by at least two thirds.

All sequences can be found in the accompanying video. Please note that Figure 3 should only emphasize that our framework has the potential to extract well suited key-frames from a dynamic mesh. This is indirectly visible if the number of frames for the chicken and the cow sequence are compared. For both sequences 30 key-frames are sufficient, which is remarkable since the chicken sequence has twice as many frames, but most of the motion happens at the end, leading to the conclusion that our framework extracts suitable key-frames in this case.

A comparison to a state-of-the-art approach for extracting key-frames is given in Table 3.

2.4 Improvements using deformation gradients

It is well known that linear interpolation of vertex positions leads to a visually unpleasant result if the dynamic mesh incorporates a great deal of rotations. This can be avoided by using more key-frames, which is obviously not a good choice in the given setting.

In this section we show how to extend our framework by utilizing deformation gradients [31]. Inspired by the work of Sumner et al. [32] we adapt the idea to interpolate the deformation gradients $T_k \in \mathbb{R}^{3\times 3}$ of the triangles instead of the vertex positions. This requires only minimal changes to our framework.

Since deformation gradients are defined with respect to a reference pose, we compute them using the first mesh of a sequence m_1 as reference. Finally, each deformation gradient is decomposed into a rotation R_k and a scale/shear part S_k, using polar decomposition [30]:

$$T_k = R_k S_k.$$

The S_k are symmetric matrices and their representative six values can be combined linearly. However, for rotations it is better to use the exponential map [24, 2] and interpolate the logarithms of the rotations instead. The logarithm of R_k is a skew symmetric matrix with diagonal elements equal to zero and can

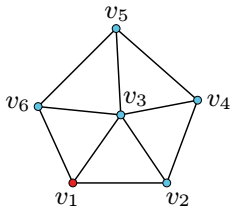

Figure 4: Example mesh for which the edge structure is stored in the matrix A in (2).

thus be represented by three values. For each frame we store the nine values of all triangles into the vector

$$p_i = \begin{pmatrix} \log(R_{i,1}) \\ S_{i,1} \\ \log(R_{i,2}) \\ S_{i,2} \\ \vdots \\ \log(R_{i,f}) \\ S_{i,f} \end{pmatrix}.$$

Compared to the vertex-based approach that we described above with $m_i, k_j \in \mathbb{R}^{3n}$, where n is the number of vertices, we now have $p_i, q_j \in \mathbb{R}^{9f}$, where f is the number of faces. The q_j are the deformation-gradient-based representations of the key-frames k_j.

This representation allows us to measure the L_2-norm on the R_k as well as on the S_k and we can use p_i and q_j as input of the BNG algorithm. We then obtain the optimal weights again according to (1), but this time in terms of the deformation gradient representation. This simply amounts to interpolation of the deformation gradients using the exponential map as shown in [32].

After determining the key-frames, the reconstructed $\hat{p}_i$ as well as the k_j are still in the deformation gradient format, which challenges us to find a way to convert them back to a mesh.

For this step, we closely follow the approach of Xu et al. [33]. Since the $\hat{p}_i$ store the decomposed and interpolated deformation gradients with respect to the reference mesh m_1, the obvious idea is to apply the deformation gradients stored in $\hat{p}_i$ to the triangles of the reference mesh m_1. But since the deformation gradients only affect the triangles locally, this results in a triangle soup.

Fixing the position of one vertex allows us to express the whole mesh in terms of edge vectors. To get back to the mesh representation we simply solve the linear system

$$A\hat{m}_i = u_i,$$

with A storing the edge structure of the mesh and u_i being the new edge vectors generated from the interpolated deformation gradients and the reference mesh m_1.

An example for the reconstruction is given in Figure 4. For the illustrated mesh, we have the matrix

$$A = \left(\begin{array}{cccccc} -1 & 0 & 1 & 0 & 0 & 0 \\ -1 & 1 & 0 & 0 & 0 & 0 \\ 0 & -1 & 1 & 0 & 0 & 0 \\ 0 & -1 & 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & -1 & 0 & 0 \\ 0 & 0 & 0 & -1 & 1 & 0 \\ 0 & 0 & 1 & 0 & -1 & 0 \\ 0 & 0 & 0 & 0 & -1 & 1 \\ 0 & 0 & 1 & 0 & 0 & -1 \\ 1 & 0 & 0 & 0 & 0 & -1 \\ \hline 1 & 0 & 0 & 0 & 0 & 0 \end{array}\right), \tag{2}$$

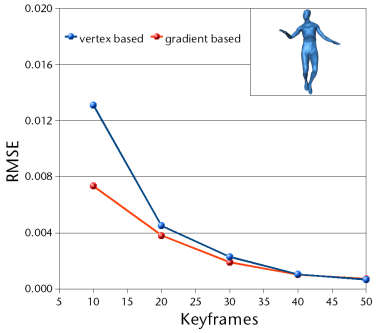

Figure 5: The influence of deformation gradients (red) compared to vertex-based key-frames (blue). Using deformation gradients, the sequence can be reconstructed with less key-frames.

the vector

$$u = \begin{pmatrix} v_3 - v_1 \\ v_2 - v_1 \\ v_3 - v_2 \\ v_4 - v_2 \\ v_3 - v_4 \\ v_5 - v_4 \\ v_3 - v_5 \\ v_6 - v_5 \\ v_3 - v_6 \\ v_1 - v_6 \\ \hline v_1 \end{pmatrix}$$

contains the edge vectors, and

$$\hat{m} = \begin{pmatrix} \hat{v}_1 \\ \hat{v}_2 \\ \hat{v}_3 \\ \hat{v}_4 \\ \hat{v}_5 \\ \hat{v}_6 \end{pmatrix}$$

represents the new vertex coordinates for the mesh.

The last lines of A and u encode the additional constraint of fixing the vertex $\hat{v}_1 = v_1$. This coordinate is taken from the original mesh m and must be saved for the reconstruction process.

The effect of using the deformation-gradient-based representation is illustrated in Figure 5 for the dance sequence. We have chosen this sequence, since it is the longest sequence with the strongest non-linear behaviour. For a small number of key-frames the positive influence of the deformation gradients is clearly visible. But for more than 20 key-frames the difference between both variants becomes negligible and the additional overhead does not pay off.

Model	Vertices	Triangles	Frames
Chicken	3030	5664	400
Cow	2904	5804	204
Dance	7061	14118	201
Dolphin	6179	12337	101
Face	539	1042	10001

Table 2: Details of the sequences that were used throughout the paper.

Model	# KF	Our approach					GA approach [19]				
		Min. PSNR	Max. PSNR	Avg. PSNR	# Iter.	Time (sec.)	Min. PSNR	Max. PSNR	Avg. PSNR	# Gen.	Time (sec.)
Chicken	50	42.09	140.55	64.46	49	40	43.60	131.45	64.75	12238	1486
	40	38.44	102.36	60.25	46	27	39.75	127.04	59.35	9450	1249
	30	34.71	100.68	55.06	38	21	32.98	126.46	54.62	4578	711
	20	33.94	75.29	49.48	26	12	28.48	127.72	46.67	8180	1468
	10	29.88	67.96	41.30	27	10	21.58	135.80	40.06	2302	1123
Dance	50	52.56	78.54	65.65	46	55	49.89	69.30	62.14	5808	886
	40	50.14	73.76	60.89	42	39	46.27	65.45	56.99	9135	1390
	30	44.61	69.17	53.97	40	30	41.99	62.14	50.78	4414	1078
	20	38.96	57.01	47.54	42	19	36.65	54.71	43.41	4210	1855
	10	30.86	49.04	38.48	19	6	24.50	48.05	33.23	1702	547

Table 3: The PSNR error captures the logarithmically scaled ratio between signal and noise. Our approach extracts better key-frames and is 16 to 97 times faster for the dance sequence and 37 to 112 times faster for the chicken sequence.

3 Results

We compare our framework to the recently introduced approach of Lee et al. [19] since we consider it an extension of [14], which is basically a comparison between the most promising algorithms for extracting key-frames. We also use the peak signal-to-noise ratio

$$\mathrm{PSNR}(m_i, m_j) = 20 \log_{10} \left(\frac{\max(\mathrm{Diag}(m_i), \mathrm{Diag}(m_j))}{\mathrm{RMSE}(m_i, m_j)} \right)$$

which measures the quality of a signal that is subjected to noise. It is most commonly used as a measure for the reconstruction quality in 2D image compression.

While the RMSE is the cumulative error between the reconstructed and the original mesh, PSNR is a measure of the peak error. A lower value for RMSE means smaller overall error, and as seen from the inverse relation between RMSE and PSNR, this translates to a high value of PSNR. Logically, a higher value of PSNR is good because it means that the ratio of signal to noise is higher. Here, the "signal" is the original mesh and the "noise" is the reconstruction error.

For the chicken as well as the dance sequence we outperform [19] in terms of quality and in terms of computation time. We achieve better PSNR for both sequences because the key-frames extracted by our framework are not necessarily frames from the sequence but have a similar shape. Since they are the averages of the clusters they represent, we get better reconstructions of the whole sequence. The second important difference relates to the GA used in [19], which needs a well-defined stopping criterion. Moreover, GA converges very slowly because the mutation phase of any GA introduces a random element. Our BNG approach converges quadratically (see Section 2.1) and therefore yields better key-frames in a matter of seconds.

Table 2 summarizes the statistics for the sequences used throughout the paper. The timings in Table 3 were measured on an Intel Core2 Duo E6400 with 2GB of RAM.

4 Conclusion

We introduced a fast and very simple pre-processing framework for animated meshes. It follows the general idea of PCA to extract the meaningful information from an animation sequence as proposed in [3] and [17], but instead we use a recently introduced algorithm from machine learning for this task.

PCA for an animated mesh is optimal in the L_2-norm, but this optimality comes at a certain cost. The PCA eigenvectors cannot be interpreted as part of the sequence of meshes anymore.

Our framework clusters an animated mesh into frames representing similar motion. This gives us a certain set of frames similar to the PCA vectors, which we call key-frames. Although these frames are not part of the original mesh sequence, linear interpolation between them allows us to reconstruct the whole animation sequence.

This approach introduces a small error compared to PCA, but the main advantage is that our key-frames can be represented as meshes, which allows us to use *any* appropriate state-of-the-art encoder for animation sequences on this subset. The overhead introduced by our framework is marginal, because for l extracted key-frames we only need to store l additional weights per frame in the final compression step.

Although the framework is already very efficient, there still exists some potential for improvement. The NG approach has recently been extended to the so-called *Patch Clustering* for streaming data [1]. This fits perfectly well for streaming compression, since it allows to compute the key-frames on the fly for a streaming sequence, thereby providing a valid clustering (key-frames) at any given time step.

Acknowledgements

This research project was financially supported by the State of Lower-Saxony and the Volkswagen Foundation, Hannover, Germany.

References

[1] N. Alex and B. Hammer. Parallelizing single pass patch clustering. In *Proceedings of the European Symposium on Artificial Neural Networks*, pages 227–232, Bruges, Apr. 2008.

[2] M. Alexa. Linear combination of transformations. *ACM Transactions on Graphics*, 21(3):380–387, July 2002. Proceedings of SIGGRAPH.

[3] M. Alexa and W. Müller. Representing animations by principal components. *Computer Graphics Forum*, 19(3):411–418, Sept. 2000. Proceedings of Eurographics.

[4] P. Alliez and C. Gotsman. Recent advances in compression of 3D meshes. In N. A. Dodgson, M. S. Floater, and M. A. Sabin, editors, *Advances in Multiresolution for Geometric Modelling*, Mathematics and Visualization, pages 3–26. Springer, 2005.

[5] R. Amjoun and W. Straßer. Efficient compression of 3D dynamic mesh sequences. *Journal of the WSCG*, 15(1–3):32–46, 2007.

[6] A. H. Barr. Global and local deformations of solid primitives. *Computer Graphics*, 18(3):21–30, July 1984. Proceedings of SIGGRAPH.

[7] Y. Boulfani-Cuisinaud and M. Antonini. Motion-based geometry compensation for DWT compression of 3D mesh sequences. In *Proceedings of the IEEE Conference on Image Processing*, volume 1, pages 217–220, San Antonio, TX, Sept. 2007.

[8] Y. Boulfani-Cuisinaud, M. Antonini, and F. Payan. Motion-based mesh clustering for MCDWT compression of 3D animated meshes. In *Proceedings of the 15th European Signal Processing Conference*, pages 2105–2109, Posnań, Poland, Sept. 2007.

[9] H. M. Briceño, P. V. Sander, L. McMillan, S. Gortler, and H. Hoppe. Geometry videos: a new representation for 3D animations. In *Proceedings of the 2003 ACM SIGGRAPH/Eurographics Symposium on Computer Animation*, pages 136–146, San Diego, CA, July 2003.

[10] P. Cignoni, C. Rocchini, and R. Scopigno. Metro: Measuring error on simplified surfaces. *Computer Graphics Forum*, 17(2):167–174, June 1998.

[11] M. Cottrell, B. Hammer, A. Hasenfuß, and T. Villmann. Batch and median neural gas. *Neural Networks*, 19(6–7):762–771, July–Aug. 2006.

[12] I. Guskov and A. Khodakovsky. Wavelet compression of parametrically coherent mesh sequences. In *Proceedings of the 2004 ACM SIGGRAPH/Eurographics Symposium on Computer Animation*, pages 183–192, Grenoble, Aug. 2004.

[13] J. A. Hartigan. *Clustering Algorithms*. Wiley Series in Probability and Mathematical Statistics. John Wiley & Sons, New York, 1975.

[14] K.-S. Huang, C.-F. Chang, Y.-Y. Hsu, and S.-N. Yang. Key probe: a technique for animation keyframe extraction. *The Visual Computer*, 21(8–10):532–541, Sept. 2005.

[15] P. Huang, A. Hilton, and J. Starck. Automatic 3D video summarization: Key frame extraction from self-similarity. In *Proceedings of the Forth International Symposium on 3D Data Processing, Visualization and Transmission*, Atlanta, GA, June 2008.

[16] L. Ibarria and J. Rossignac. 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*, pages 126–135, San Diego, CA, July 2003.

[17] Z. Karni and C. Gotsman. Compression of soft-body animation sequences. *Computers & Graphics*, 28(1):25–34, Feb. 2004.

[18] P.-F. Lee, C.-K. Kao, J.-L. Tseng, B.-S. Jong, and T.-W. Lin. 3D animation compression using affine transformation matrix and principal component analysis. *IEICE Transactions on Information and Systems*, E90-D(7):1073–1084, July 2007.

[19] T.-Y. Lee, C.-H. Lin, Y.-S. Wang, and T.-G. Chen. Animation key-frame extraction and simplification using deformation analysis. *IEEE Transactions on Circuits and Systems for Video Technology*, 18(4):478–486, Apr. 2008.

[20] J. E. Lengyel. Compression of time-dependent geometry. In *Proceedings of the 1999 Symposium on Interactive 3D Graphics*, pages 89–95, Atlanta, GA, Apr. 1999.

[21] I. S. Lim and D. Thalmann. Key-posture extraction out of human motion data. In *Proceedings of the 23rd Annual International Conference of the IEEE Engineering in Medicine and Biology Society*, volume 2, pages 1167–1169, Istanbul, Oct. 2001.

[22] T. Martinetz and K. Schulten. Topology representing networks. *Neural Networks*, 7(3):507–522, 1994.

[23] T. M. Martinetz, S. G. Berkovich, and K. J. Schulten. "Neural-gas" network for vector quantization and its application to time-series prediction. *IEEE Transactions on Neural Networks*, 4(4):558–569, July 1993.

[24] R. M. Murray, Z. Li, and S. S. Sastry. *A Mathematical Introduction to Robotic Manipulation*. CRC Press, Boca Raton, FL, 1994.

[25] M. J. Park and S. Y. Shin. Example-based motion cloning. *Journal of Visualization and Computer Animation*, 15(3–4):245–257, July 2004.

[26] F. Payan and M. Antonini. Wavelet-based compression of 3D mesh sequences. In *Proceedings of the Second International Conference on Machine Intelligence*, Tozeur, Tunisia, Nov. 2005.

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

[28] J. Rossignac. Surface simplification and 3D geometry compression. In J. E. Goodman and J. O'Rourke, editors, *Handbook of Discrete and Computational Geometry*, chapter 54, pages 1209–1240. CRC Press, Boca Raton, FL, second edition, 2004.

[29] M. Sattler, R. Sarlette, and R. Klein. Simple and efficient compression of animation sequences. In *Proceedings of the 2005 ACM SIGGRAPH/Eurographics Symposium on Computer Animation*, pages 209–217, Los Angeles, CA, July 2005.

[30] K. Shoemake and T. Duff. Matrix animation and polar decomposition. In *Proceedings of Graphics Interface '92*, pages 258–264, Vancouver, BC, May 1992.

[31] R. W. Sumner and J. Popović. Deformation transfer for triangle meshes. *ACM Transactions on Graphics*, 23(3):399–405, Aug. 2004. Proceedings of SIGGRAPH.

[32] R. W. Sumner, M. Zwicker, C. Gotsman, and J. Popović. Mesh-based inverse kinematics. *ACM Transactions on Graphics*, 24(3):488–495, July 2005. Proceedings of SIGGRAPH.

[33] D. Xu, H. Zhang, Q. Wang, and H. Bao. Poisson shape interpolation. In *Proceedings of the 2005 ACM Symposium on Solid and Physical Modeling*, pages 267–274, Cambridge, MA, June 2005.

[34] J.-H. Yang, C.-S. Kim, and S. U. Lee. Compression of 3-D triangle mesh sequences based on vertex-wise motion vector prediction. *IEEE Transactions on Circuits and Systems for Video Technology*, 12(12):1178–1184, Dec. 2002.