Probabilistic Motion Sequence Generation

Mirko Sattler Ralf Sarlette Reinhard Klein

University of Bonn
Institute of Computer Science II
Computer Graphics
Römerstraße 164, 53117 Bonn, Germany
Email: {sattler,sarlette,rk}@cs.uni-bonn.de

Abstract

Creating long animation sequences with non-trivial repetitions is a time consuming and often difficult task. This is true for 2D images and even more true for 3D sequences. Based upon the idea of video textures we propose a simple algorithm to create new user controlled animation sequences based only on a few key frames by the analysis of velocity and position coherence. The simplicity of the method is achieved by carrying out the calculations on the main principal components of the reference animation, hence reducing the dimensionality of the input data. This also leads to significant compression. Smooth animations are ensured, using one of the two proposed blending schemes. The method is demonstrated with several examples.

1. Introduction

The usage of computer generated images (CGI) and especially animations of humans or creatures in feature films [8, 7, 3] and computer games is steadily increasing. Animating sequences per hand is a difficult and time consuming process. Often only short basis sequences exist and have to be extended to longer sequences (e.g. walk of an avatar) with non trivial repetitions and can not be solved by simple copy and paste. On the other hand creating a large motion database, as it is sometimes done in the gaming industry, is also a costly task. In this paper we extend the idea of video textures as introduced by Schödl et al.[20] to geometry and generalized attributes like vertex normals, velocities or reflection properties. Similar to Lee et al.[16] control of the avatar can be achieved, but in contrast to the latter we are not restricted to motion capture data.
The analysis of a given short basis animations leads to the creation of transition matrices, which store frame-to-frame coherence information and transition probabilities. Because the amount of data of the basis sequence might be impractical to store and to be used during runtime, we achieve compression and computation of the transition matrices by reducing the dimensionality of the input data, using principal component analysis. To ensure the preservation of the motion dynamics we also split geometry related data (vertex positions) and animation related data (vertex velocities) and analyze them independently.
During runtime the algorithm creates endless or fixed length sequences, whereat the user can control the randomness and the direction of the animation. Depending on the quality of the input data we propose two blending schemes to ensure smooth animations.

1.1. Related Work

The basic idea to search for coherence in frames of image sequences was introduced by Schödl et al.[20] with the term video textures. The goal of this work was to create endless or fixed length image sequences with non-trivial repetitions. E.g. candle flames, clocks or a waterfall animations were generated with their method. Modelling the textures as Markov processes simple frame-to-frame distances are computed and mapped to transition probabilities. Simple filtering is used to preserve dynamics and also a method for avoiding dead ends and anticipating the future is introduced. Within this framework they also allow user control, e.g. a mouse-controlled fish. This was achieved by modifying the distance function and therefore the transition probability in such a way, that the valid transitions to certain frames (in the fish case towards the mouse location) are more likely than others (any other locations in the fish tank). This work focuses only on image based animations, not on geometry. Geometry anima-

tion in this field naturally focuses on human motion. The sequences are driven by key frames, rule-based systems [5, 17, 4, 18, 6], control systems and dynamics [21, 15, 10, 9] and motion capture data [11, 16]. Knowledge from biomechanics and motion studies is often involved to insure natural looking output. A lot of recent work is restricted to special animation cases, e.g. diving. Lee et al.[16] uses the method described with video textures to compute coherence between motion capture data to generate new data based on several input methods. They rely on a precomputed database and focus also on different control mechanisms for the avatar. To reduce the dimensionality of the input data, Alexa et al.[1] use principal component analysis of the animation to compress the needed data, but are restricted to a whole given sequence. Bowden[2] uses also PCA to simplify the motion based on feature points of the objects. While Lee et al.[16] is based on motion capture data and Bowden [2] introduces the idea to use dimensionality reduction for the data, we want to combine video textures and PCA to allow the handling of arbitrary geometry animations, including geometry attributes like normals or local reflection properties.

1.2. Algorithm Overview

Motion or animation data is often given in the form of key frames containing all necessary information like vertex positions, normals, connectivity information etc. Our algorithm first does an analysis of the already finished animation A of the length l, which can be modelled by an artist, based on motion capture data, procedural created or obtained using any other form of data generation. In our examples we use animations of short or medium length ($l = 115, 190$). To have a diversity in the animation, the length and hence the data amount is often much higher. But also using only a small l can involve huge data amounts. Therefore we first compress the animation A using a principal component analysis (PCA) [12, 14, 19]. The motion structure is computed, using only the weights of the eigenvectors, hence can be computed efficiently depending only on the number of PCA components used. The main idea is, that similarities between frames are transferred into the PCA weights. As in Schödl et al.[20] we compute a coherence and transition matrix, which can be used to create either infinite random or length controlled loops during rendering, as shown in figure 1. The rest of the paper is organized as follows. In section 2 we describe the data analysis in detail, including dynamics preservation and final jump map generation. This is followed by the results section 3 in which we demonstrate our method with examples and discuss blending between transitions and more user control possibilities.

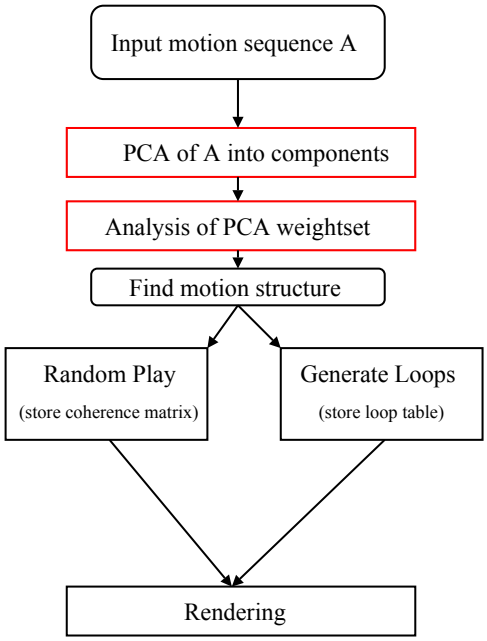

Figure 1. Algorithm Overview. First a principal component analysis is used with the input motion sequence. The motion structure is evaluated using only the weight set. For random play or defined loops coherence and transitions matrices are computed, which are later used for rendering together with the eigenvectors of A.

2. Data Analysis

Given an animation A of l frames length and constant connectivity, we define the vector A_i to contain all vertex positions of the geometry for frame $i \in [0, \ldots, l] \subset \mathbb{N}$. We perform a PCA of these vectors, resulting in a series of c eigenvalues λ_{ik} and eigenvectors F_{ik} $k \in [1, \ldots c] \subset \mathbb{N}$, latter we will call *eigenframes*. We use $\lambda_{i0} = 1$ and the mean of A as F_{i0}. The first $c < l$ eigenframes approximate any of the original frame A_i in such a way that the sum of the squares of the projection errors onto the affine subspace spanned by $\{F_{i0}, \ldots, F_{ic}\}$ is minimized

$$A_i \approx \sum_{k=0}^{c} w_{ik} F_{ik}, \quad i = 0 \ldots l. \tag{1}$$

The coefficients $w_{ik} = \langle A_i, F_{ik} \rangle$ are weights, were $\langle , \rangle$ denotes the standard scalar product in $\mathbb{R}^{3\times N}$, where N is the number of vertices of the geometry.
Following the work of Schödl et al. [20] and Lee et al. [16] we also model the data as a first-order Markov process, hence the transition between states depends only on the current state, which in our case are the given key frames in A. The Markov process is represented as a matrix P_{ij} storing the probability of a transition from frame i to frame j. This matrix is computed, by first computing the frame-to-frame distances D_{ij}, which we define by the differences of the PCA weights w_{ik}:

$$D_{ij} = \sum_{k=0}^{c} |w_{ik} - w_{jk}| \tag{2}$$

As in Schödl et al., the transition probabilities from frame i to frame j are computed using an exponential function P_{ij}, as follows:

$$P_{ij} \approx \exp(-D_{i+1,j}/\sigma) \tag{3}$$

where σ controls the mapping between the distance measure and the probability of the transition. Higher values for σ allow for a greater variety at the cost of poorer transitions. Transitions with high probabilities are the ones, where the successor of i is similar to j, hence $D_{i+1,j}$ is small. To propagate the forward motion, the probability for frame $i + 1$ should be higher than for i itself. All probabilities are normalized per row, hence $\sum_j P_{ij} = 1$.

2.1. Motion Dynamics

To preserve the dynamics of the motion, we also compute all central velocities V_i for all vertices per frame $i, i \in [1, \ldots, l)$. Using the vertex positions p_{in} stored in A_i we compute the velocity v_{in} for vertex n and frame i:

$$v_{in} = ((p_{in} - p_{i-1,n}) + (p_{i+1,n} - p_{in}))/2 \tag{4}$$

where we define $v_{0n} = v_{ln} = 0$ for all n. Computing a separate PCA on V_i and denoting the weights as w^*_{ik}, were $i \in [0, \ldots, l]$ is the number of the frame the velocities are to be reconstructed for and $k \in [0, \ldots, c^*]$ with c^* the number of components we get similar to equation 2:

$$D^*_{ij} = \sum_{k=0}^{c^*} |w^*_{ik} - w^*_{jk}| \tag{5}$$

and finally

$$P^*_{ij} \approx \exp(-D^*_{i+1,j}/\sigma^*) \tag{6}$$

where σ^* is the mapping parameter respectively. We also propose another simple method to propagate the *forward* motion, that is to keep a simple list of the last m visited frames. m should be a fairly small number ($m < 5$), to prevent certain motions, e.g. waving arms up and down or other jittering. While on the one hand according to the system itself this is a valid motion, it might be desired to prevent this for more natural animations.

2.2. Generate Jump map

To create the final probability map $\hat{P}_{ij}$, which we call jump map we use position and velocity coherence as follows:

$$\hat{P}_{ij} = P_{ij} P^*_{ij} \tag{7}$$

To allow different emphasis on either the position coherence (w), the velocity coherence (w^*) or both, we propose the following extension to equation 7:

$$\hat{P}_{ij} = [q + (1 - q)P_{ij}][q^* + (1 - q^*)P^*_{ij}] \tag{8}$$

where $q, q^* \in [0, 1]$ control the emphasis. Note that values between $0, 1$ create a base probability for either q and/or q^* to ensure that $\hat{P}_{ij} \in [0, 1]$.
The resulting map now can be used either to generate infinite random or fixed length looped sequences. Besides the vertex positions we also include the vertex normals in the vectors A_i. The reconstructed normals later can be used in the rendering step for proper shading. Figure 3 shows examples for D_{ij},P_{ij},D^*_{ij},P^*_{ij} and $\hat{P}_{ij}$.

2.3. Adding vertex attributes

Besides the vertex normals, additional information could be stored with the vector A_i, e.g. reflections properties. For numerical stability it then might be necessary to introduce another vector R_i to store these attributes, to obtain optimal results from the principal component computations.

2.4. Blending Transitions

Because we allow also transitions from i to j which have larger differences, slight discontinuities can occur. Instead of jumping directly from frame i to j we blend with a chosen amount b of blending frames $i^\diamond$:

$$i \rightarrow i^\diamond_1 \rightarrow i^\diamond_2 \rightarrow \ldots \rightarrow i^\diamond_b \rightarrow j \tag{9}$$

For our examples we found $b = 3$ to yield smooth animations. The blending itself is done on the PCA weight set. Let w_{ik} be the weights for the c PCA components for frame i and w_{jk} for frame j, respectively. The blending weights

Figure 2. Two models used in this paper. On the left side, the AVATAR model consists of three parts and has 20948 vertices. The SKELETON model with another shirt on the right side consists of 12427 vertices.

$w_{i_t^\diamond k}$ for frame $i_t^\diamond, t \in [1 \dots b]$ are linear interpolated between the weights for i and j:

$$w_{i_t^\diamond k} = w_{ik} + \frac{w_{jk} - w_{ik}}{b(b-t+1)} \tag{10}$$

$\forall k, k \in [0 \dots c]$.

2.4.1. Variable Length Transitions In contrast to transitions with constant length we also propose transitions with dynamic length, depending on the Euclidean distance between the frames i and j. This is done by introducing a smoothness parameter s, which defines the minimal distance which should be covered by a single transition step. A good starting point for s is the mean distance $s = \bar{d}$ between the frames in the animation sequence A. From this it follows that the parameter b in equation 9 is computed for each transition as follows:

$$b = \frac{\left\| \tilde{A}_i - \tilde{A}_j \right\|_2}{s} \tag{11}$$

where $\tilde{A}_i$ is the reconstructed geometry for frame i.

2.5. Motion Control

If the random or looped sequence does not satisfy the users needs, we propose a simple but efficient way for

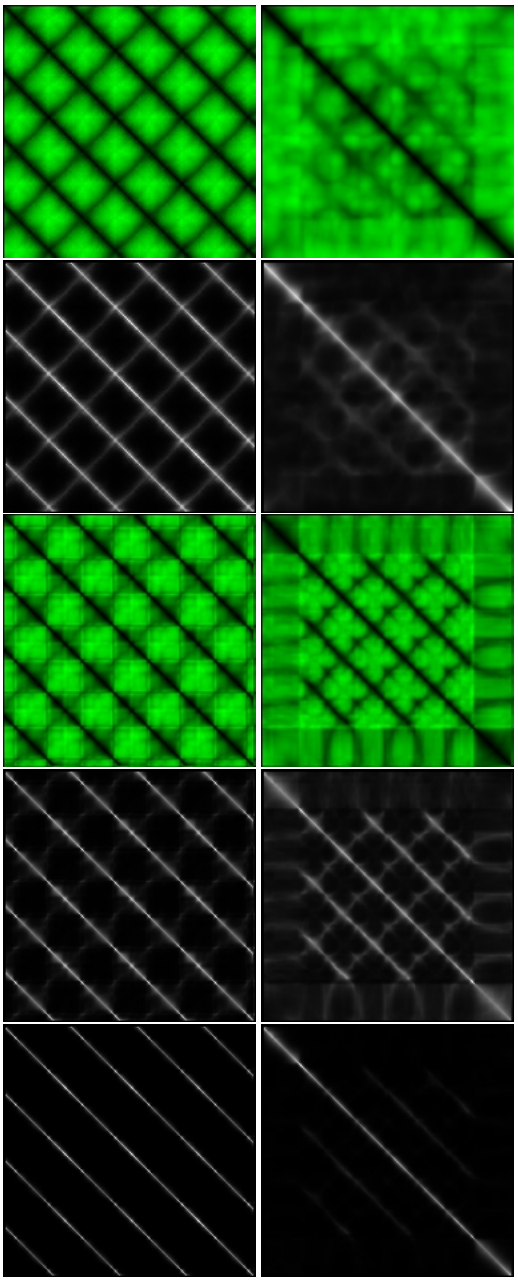

Figure 3. Difference matrices and jump maps for the AVATAR (left) and SKELETON (right) sequence. From top to bottom: D_{ij},P_{ij},D^*_{ij},P^*_{ij} and $\hat{P}_{ij}$. In the distance matrices black is zero distance and green high distance. In the probability matrices stands white for high probabilities and black for low ones. The usage of both geometry and velocity data for P^*_{ij} prevents wrong jumps.

more user control. Instead of one input sequence, several small sequences have to be created, each containing only a specific motion *subtype*, e.g. *move forward*. The user then can select a target cell (blue), as illustrated in figure 4 using e.g. a joystick. The algorithm now searches for coherence frames between the originating cell (red) and the target cell, interrupts the random jumping and continues with it, when blending to an appropriate frame within the target cell. The target cell now becomes the new originating cell. This method should work for arbitrary cell contains, as long as there is no change of the geometry connectivity. Other control interfaces based on choice, sketching or vision as described in Lee at al.[16] could also be applied.

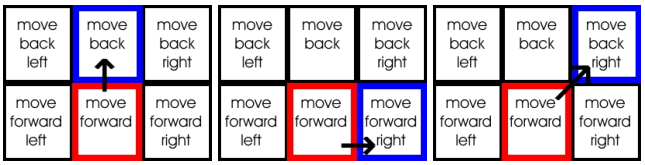

Figure 4. Three examples for user control of the motion. Each cell contains PCA components for a certain motion, e.g. *move forward*. Only red (currently in) and blue (target) cells are valid. The blue cell is chosen through user input, e.g. via a joystick.

3. Results

We implemented our method under Windows 2000 on a 1.5GHz Athlon with a state-of-the art graphics accelerator. All computations besides the rendering are done by the CPU. The accompanying video clips show two examples of initial animations A and two new long sequences generated out of it. The rendering is done with OpenGL and Phong shading, using the reconstructed vertex normals.

3.1. PCA Reconstruction Error Analysis

The error of a reconstructed geometry depends on the number of PCA components used. Because the usable number might be limited through speed and/or memory restrictions we introduce two errors E and E^*, corresponding to the position and to the velocity with a given number of components c and c^*, respectively. They are defined as follows using N as the number of vertices in the geometry:

$$E(c) = \sum_{i=1}^{l-1} \frac{\sum_{n=0}^{N} \|p_{in} - \tilde{p}_{in}\|_2}{\sum_{n=0}^{N} \|p_{in}\|_2} \tag{12}$$

$$E^*(c) = \sum_{i=1}^{l-1} \frac{1}{N} \sum_{n=0}^{N} \frac{\|v_{in} - \tilde{v}_{in}\|_2}{\|v_{in}\|_2} \tag{13}$$

where E is a relative error for the geometry of the object and E^* is expressed as a relative velocity error per vertex. p_{in} is the position in world coordinates and v_{in} the central velocity of the vertex n, respectively. $\tilde{p}_{in}$ and $\tilde{v}_{in}$ are the reconstructed vertices and velocities. Note, that this reconstruction error makes only sense to have a quality indicator for the newly generated sequence, therefore, in which amount the PCA reconstruction error determines the similarity between the original and the new sequence. These errors are discussed in section 3.3. Figure 5 shows the eigenvalues for the avatar and skeleton data sets decreasing with the number of components. We took $c = 10$ for the geometry and $c = 5$ for velocity reconstruction.

3.2. PCA Compression Ratio

The number of used eigenvectors for the reconstruction directly effects the compression ratio of the input basis sequence. Table 1 shows the ratio dependent of the used number of components c for the avatar and skeleton sequence. Compare also table 2 for the error dependence.

c	Avatar [MB]	Ratio	Skeleton [MB]	Ratio
-	138	1:1	75,2	1:1
1	2.6	1:53	1.4	1:54
2	4.0	1:34	1.9	1:40
4	4.6	1:30	2.8	1:27
6	6.2	1:22	3.8	1:20
8	7.7	1:18	4.7	1:16
10	9.2	1:15	5.6	1:13

Table 1. Compression ratios of the used sequences for several numbers of PCA components. We found $c > 8$ to result in visible good reconstructions, hence a compression ratio of >1:15 is achievable.

3.3. Generated Sequences

We have tested the algorithm with two animation sequences, namely AVATAR (115 frames) and SKELETON

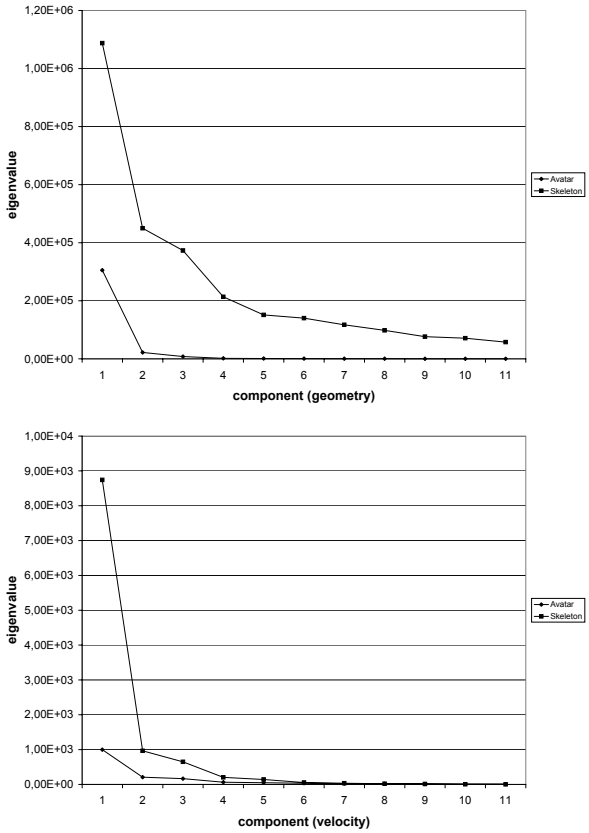

Figure 5. Eigenvalues versus PCA component number for geometry (top) and velocity (bottom).

(190). Figure 9 shows sample frames of an animation with several hundred frames. See also the accompanying video for the new animations. Figures 6 and 8 show the influence of the parameter σ and $\sigma^*$for the two basis sequences. Normally, we used $\sigma = \sigma^*$. It is clearly visible, that if σ is too low or too high, instability occurs and only a specific range results in proper new animations. For the avatar sequence we found $\sigma \approx 0.05$ to generate natural animations and for the skeleton $\sigma \approx 0.30$, respectively. The influence of the parameters q and q^* is shown for the AVATAR sequence in figure 7. If only one criterium is considered, either velocity $q = 0.0, q^* = 1.0$ or position $q = 1.0, q^* = 0.0$, the first 100 or 140 frames are sufficient to find good transitions for constant σ. If both criteria are considered, valid transitions are restricted to an extend, that the algorithm needs all frames of the input sequence or even more. Table 2 shows the reconstruction errors E and E^* for the animations depending on the number of PCA components used.

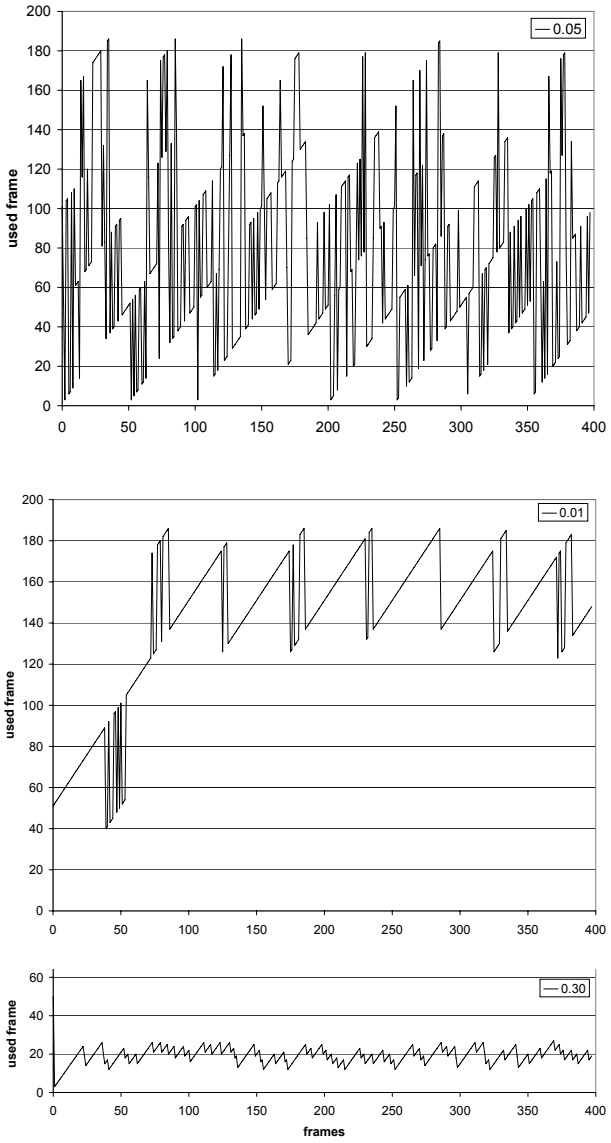

Figure 6. Influence of σ and σ^* for the AVATAR sequence. From top to bottom: $\sigma = 0.05$, $\sigma = 0.01$ and $\sigma = 0.30$. Note the changing dynamics of the generated sequence.

4. Conclusions

We have presented a method to handle arbitrary animated geometry to generate new endless or fixed length sequences with non-trivial repetitions. The algorithm needs only a small basis animation and little preprocessing time to generate transition matrices, which are based on frame-to-frame position and velocity coherence of principal component reconstruction weights. Using the PCA also deliv-

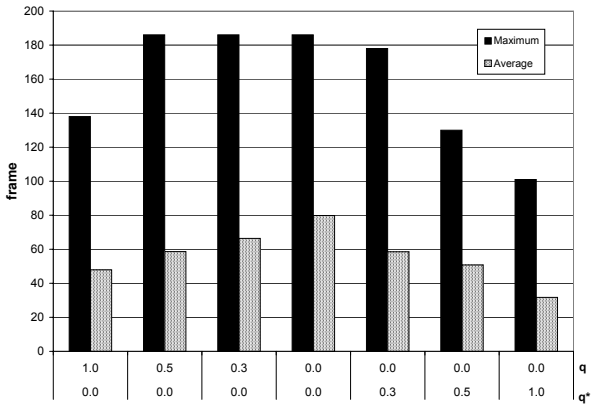

Figure 7. Influence of different combinations of q and q^*, when equation 8 is used. See text for details.

c	Avatar(E)	Avatar(E^*)	Skeleton(E)	Skeleton(E^*)
0	25.06	188.55	31.79	172.14
2	4.96	121.25	16.19	151.95
4	1.89	62.85	10.61	119.52
6	1.63	48.49	8.042	92.76
8	1.24	31.04	6.29	82.66
10	0.99	26.09	5.09	73.82

Table 2. E and E^* errors computed after equation 13. $c = 0$ equals to only use the mean for reconstruction. Similar to the eigenvalues the errors drop with increasing c.

ers a good compression of the input animation for free. Depending on the number of components used and the maximum error allowed, the complete reconstruction and evaluation of the jump map can be done at interactive to real-time frame rates on consumer hardware. This allows the easy usage of this technique in the entertainment field, e.g. for computer games. In practice, only the jump maps, the eigenvectors and the weights for the reconstruction besides the connectivity and texture coordinate information is needed on the client computer.

4.1. Future Work

For practical usage it is worthwhile to use the increased possibilities of modern graphics adaptors. In our case it should be possible to do the PCA reconstruction of the mesh directly on the GPU to avoid bus load. Also the random sequence generation should be possible on the GPU using texture lookups, while storing the coherence matrices as textures. To achieve even better compression results and to reduce the number of needed components we want to evaluate the usage of local principal component analysis (LPCA), introduced by Kambhatla et al.[13].

4.2. Acknowlegdements

We like to thank the university of Tuebingen for the avatar sequence which was created in the focus of the Virtual Try-On project. We also would like to acknowledge the use of $POSER^{®}$ for the creation of the skeleton sequence.

References

[1] M. Alexa and W. Müller. Representing animations by principal components. *Computer Graphics Forum*, 19(3), Aug. 2000.

[2] R. Bowden. Learning statistical models of human motion. *ICVPR2000, IEEE Workshop on Human Modelling, Analysis and Synthesis*, 2000.

[3] W. Bros. The matrix trilogy, 2003.

[4] A. BRUDERLIN and T. CALVERT. Knowledge-driven, interactive animationof human running. *Graphics Interface 96*, page 213221, 1996.

[5] A. BRUDERLIN and T. W. CALVERT. Goal-directed, dynamic animation ofhuman walking. *Proceedings of SIGGRAPH 89*, 23:233242, 1989.

[6] C. M. Z. L. CHI, D. M. and N. I. BADLER. The emote model for effort and shape. *SIGGRAPH 20009*, page 173182, 2000.

[7] N. L. Cinema. Lord of the rings trilogy, 2003.

[8] Disney/Pixar. Finding nemo, 2003.

[9] P. Faloutsos, M. van de Panne, and D. Terzopoulos. Composable controllers for physics-based character animation. In *SIGGRAPH 2001 Proceedings*, pages 251–260, 2001.

[10] V. D. P. M. FALOUTSOS, P. and D. TERZOPOULOS. The virtual stuntman: dynamic characters with a repertoire of autonomous motor skills. *Computers & Graphics 25*, 6:933953, 2001.

[11] M. GLEICHER. Comparing constraint-based motion editing methods. *Graphical Models 63*, 2:107123, 2001.

[12] I. Jolliffe. *Principal Component Analysis*. Springer-Verlag, 1986.

[13] N. Kambhatla and T. K. Leen. Dimension reduction by local pca. *Neural Computation, 9*, pages 1493–1516, 1997.

[14] M. Kendall. *Multivariate Analysis*. Charles Griffin Co, 1975.

[15] V. D. P. M. LASZLO, J. F. and E. L. FIUME. Limit cycle control and its application to the animation of balancing and walking. *Proceedings of SIGGRAPH 96*, page 155162, 1996.

[16] J. Lee, J. Chai, P. Reitsma, J. Hodgins, and N. Pollard. Interactive control of avatars animated with human motion data, 2002.

[17] K. PERLIN and A. GOLDBERG. Real time responsive animation with personality. *IEEE Transactions on Visualization and Computer Graphics 1*, page 515, 1995.

[18] K. PERLIN and A. GOLDBERG. Improv: A system for scripting interactiveactors in virtual worlds. *Proceedings of SIGGRAPH 96*, page 205216, 1996.

[19] W. Press, S. Teukolsky, W. Vetterling, and B. Flannery. *Numerical recipes in C - The art of scientific computation*. ISBN 0-521-43108-5. Cambridge University Press, 2nd edition, 1992.

[20] A. Schödl, R. Szeliski, D. H. Salesin, and I. Essa. Video textures. In K. Akeley, editor, *Siggraph 2000, Computer Graphics Proceedings*, pages 489–498. ACM Press / ACM SIGGRAPH / Addison Wesley Longman, 2000.

[21] W. L. WOOTEN and J. K. HODGINS. Animation of human diving. *Computer Graphics Forum 15*, 1:314, 1996.

5. Appendix

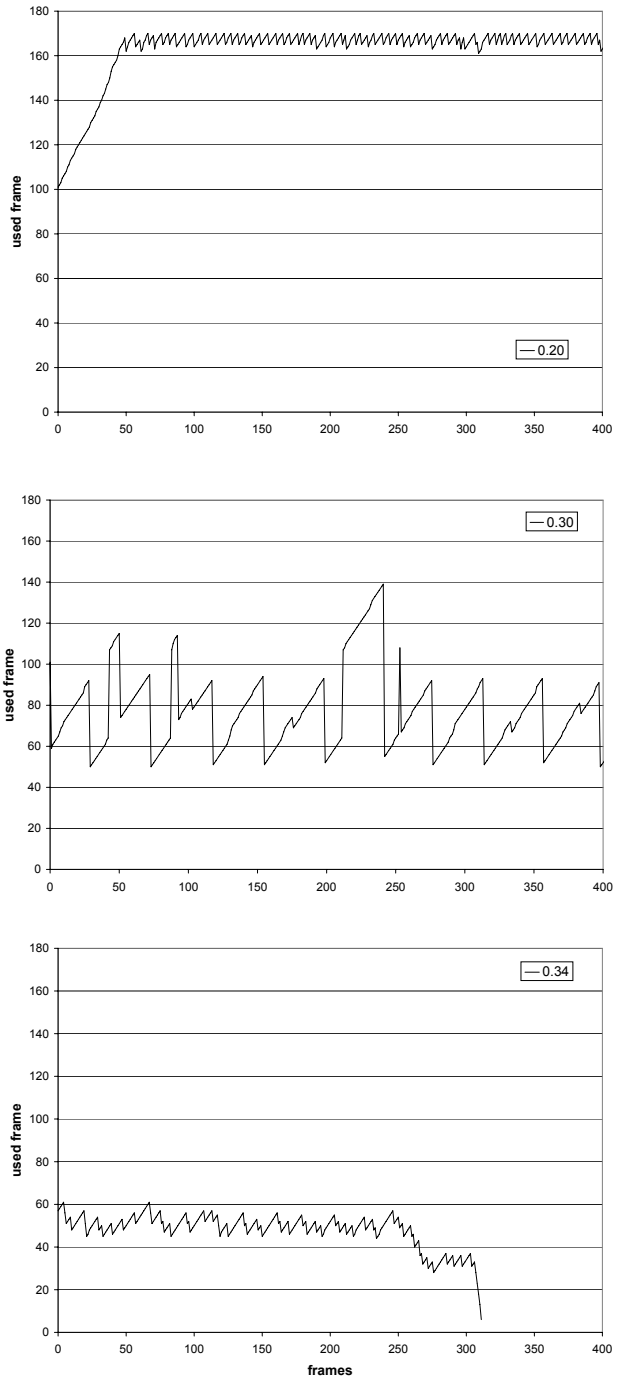

Figure 8. Influence of σ and σ^* for the SKELETON sequence.

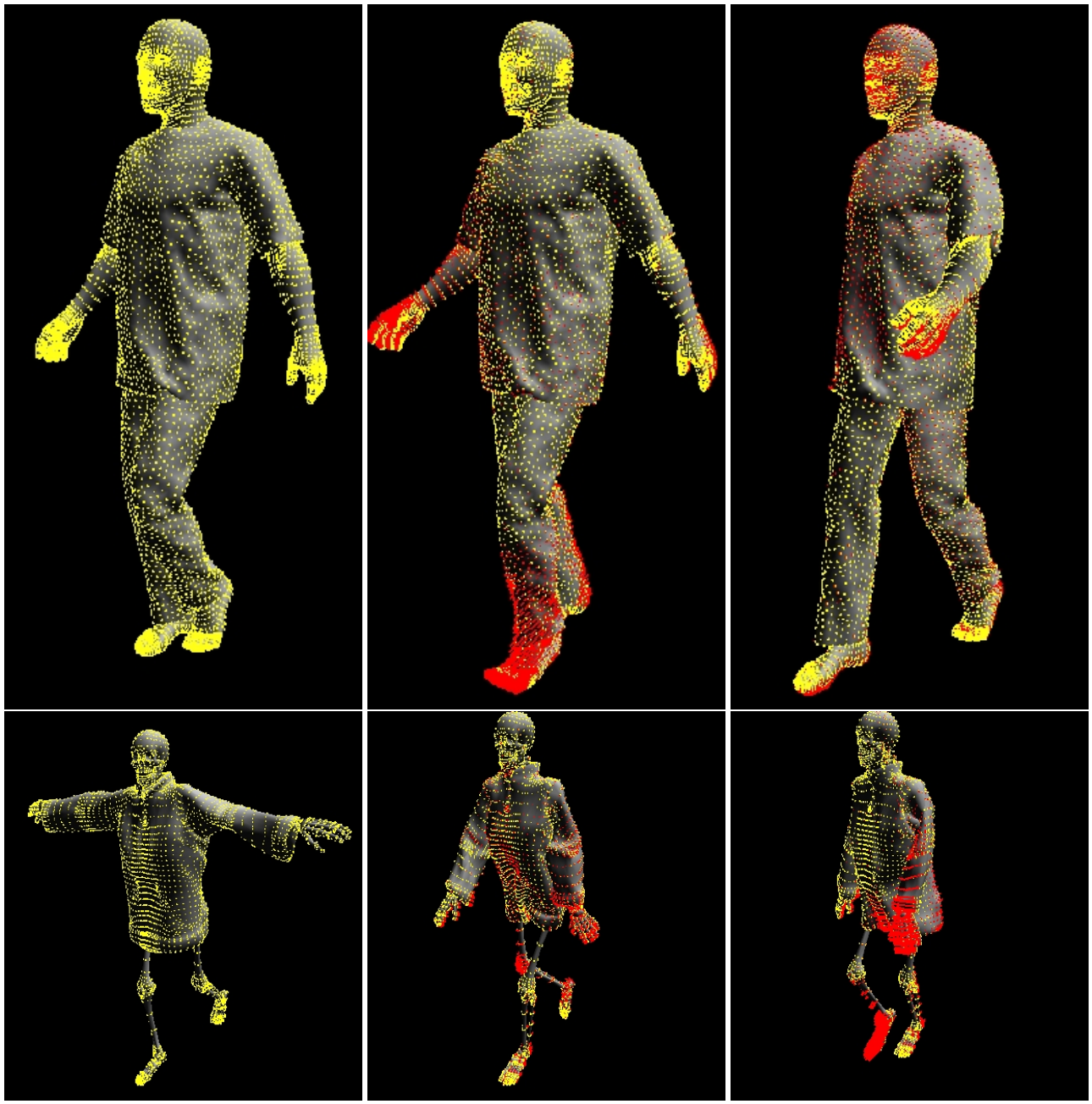

Figure 9. Sample frames of the AVATAR (top) and the SKELETON (bottom) sequence. See also the accompanying video. Yellow dots are vertex positions, red lines are velocity vectors.