The Journal of Computer Animation and Virtual Worlds, Willey Interscience, Vol.20, No.4., pp.447-456, ISSN 1546-4261, 2009

Combined Compression and Simplification of Dynamic 3D Meshes

Libor Váša

Václav Skala

Department of Computer Science and Engineering,

Faculty of Applied Science,

University of West Bohemia,

Czech Republic

Abstract

We present a new approach to dynamic mesh compression, which combines compression with simplification to achieve improved compression results, a natural support for incremental transmission and level of detail. The algorithm allows fast progressive transmission of dynamic 3D content. Our scheme exploits both temporal and spatial coherency of the input data, and is especially efficient for the case of highly detailed dynamic meshes. The algorithm can be seen as an ultimate extension of the clustering

and local coordinate frame (LCF) based approaches, where each vertex is expressed within its own specific coordinate system. The presented results show that we have achieved better compression efficiency compared to the state of the art methods.

Keywords: dynamic mesh, compression, simplification, animation, PCA, prediction

Introduction

Compression of dynamic 3D meshes is one of the research areas of computer graphics which has received increased attention in the last period. This is mainly due to the fast development of capture and display hardware which is likely to soon allow quick acquisition and high quality display of truly three dimensional moving content.

It is important to realize that dynamic mesh processing is not a simple extension of the static case. On one hand, there are some new problems, which should be addressed, such as avoiding temporal artifacts, while on the other hand we have new opportunities to be exploited during the processing, namely the temporal coherency of the input data.

Dynamic mesh is a common term used for a series of static triangular meshes that represents a development of some surface in time. Usually, two additional assumptions are made about the dynamic mesh:

- every mesh in the series has the same connectivity, i.e. there is an one-to-one correspondence of vertices from frame to frame of the animation
- the animation represents some physical process, i.e. there are no sudden changes in the geometry of the subsequent frames of the animation.

The task is to store the dynamic mesh using as few bits as possible, while preserving the visual properties of the original data.

There are generally two main classes of approaches to this problem. First, there is a large class of compression schemes. These approaches do not change the original topology

of the mesh, and achieve size reduction by sophisticated encoding of the geometry of the mesh. A second class are the simplifications, which reduce the number of primitives in the data, thus achieving smaller size and faster processing.

Our approach, however, consists of both steps. Our primary goal is size reduction for incremental transmission of 3D meshes, and therefore we only use simplification as a tool which allows better compression rates and also adds some nice features to the algorithm. In our scheme, we use eigentrajectory based Principal Component Analysis (PCA) to describe the trajectories of vertices. The decoder first receives the full connectivity, and subsequently decimates it according to decimation order sent from the encoder.

Finally, the encoder incrementally sends the PCA coefficients for the vertices needed by the decoder to refine the coarse version of the mesh. This allows us to use neighborhood average predictor to predict the PCA coefficients, and only transmit the residuals of very low entropy.

The rest of the paper is structured as follows. Section 2 gives a brief overview of the existing approaches to dynamic mesh compression and simplification. Section 3 specifies our scheme, section 4 clarifies some details, and section 5 reports about the testing we have performed to evaluate our approach. Finally, in section 6 we give conclusions and sketch some concepts for future research in the area.

Related Work

A first attempt to dynamic mesh compression has been published in the paper by Lengyel [1], who suggested subdividing the mesh into clusters in which the movement can be described by a single transformation matrix.

Ibarria and Rossignac [2] later suggested the spatio-temporal prediction schemes ELP and Replica, which were used to predict the next vertex position during a mesh traversal using the EdgeBreaker state machine. A similar approach has been used by Stefanoski [3] in his angle preserving predictor. The position of the new vertex is expressed in a local coordinate system defined by a neighboring triangle.

A different approach has been proposed by Zhang and Owen [4], who suggested encoding the frame to frame difference vectors using an octree structure, exploiting the fact that large portions of the mesh are moving in a very similar manner.

The octree based approach has been improved by Mueller et al. [5, 6], who suggested selecting an appropriate predictor for each cell, choosing from mean replacement, trilinear interpolation and direct encoding.

Wavelet theory has been used for dynamic mesh compression in the work of Payan[7], who suggested treating separate vertex trajectories as sampled signal. However, their method did not use the spatial coherence present in the data.

A different class of approaches has been pioneered by Alexa and Mueller[8], who suggested using PCA in the space of frames, expressing each frame as a linear combination of

eigenframes. However, this method had problems with rigid movement, which had to be compensated for in a preprocessing step, where a transformation matrix for each frame has been found using the least squares approach.

The method has been subsequently improved by Karni and Gotsman[9], who suggested exploiting the temporal coherence of the PCA coefficients by encoding them using linear prediction coding (LPC), thus achieving lower entropy of the encoded data. Another improvement has been proposed by Sattler et al.[10], who suggested using PCA in the space of trajectories, and finding clusters of vertices where PCA worked well (Clustered PCA). However, their randomly initialised iterative clustering method reached very different clusterings in every run.

Another addition to the PCA based method has been proposed in 2007 by Amjoun[11], who suggested using trajectory based analysis along with expressing each trajectory in a local coordinate frame defined for each cluster. Additionally, a bit allocation procedure is applied, assigning more bits to cluster where more PCA coefficients are needed to achieve desired precision.

A very efficient compression scheme for dynamic meshes is the Coddyac algorithm by Vasa and Skala [12]. This approach uses trajectory based PCA to exploit the temporal coherence of the data, however in contrast to the clustering prediction used by Sattler [10], a more efficient local paralellogram predictor is used to predict the PCA coefficients associated with each vertex.

Mamou [13] has proposed an approach similar to PCA, called skinning based com-

pression. The mesh is first segmented to parts that move in an almost rigid fashion. Each cluster's movement is expressed by a transformation matrix, and subsequently each vertex is assigned a vector of weights, that defines how to combine the transforms of the neighboring clusters to obtain the movement of the vertex.

A resampling approach has been proposed by Briceno [14] in his work on Geometry Videos. The idea is an extension of the previously proposed Geometry Images [15] concept. The geometry of the object is unwrapped and projected onto a square, which is regularly sampled. The resulting image is encoded using some off the shelf algorithm. The extension to videos solves the problems of finding a single mapping of a moving content onto a square while minimizing the overall tension. Generally, the method is not easy to implement and suffers from some artifacts, especially for objects with complex geometry.

A method on the borderline between compression and simplification has been recently proposed by Stefanoski et al. [16]. The approach utilises a connectivity driven simplification step, which uses a local spatio-temporal predictor where for each vertex a complete neighborhood is available at the decoder. Our method utilises simplification in a similar manner.

Pure simplification of dynamic meshes has been addressed by only a few papers so far. Mohr and Gleicher [17] extend the quadric-based simplification static mesh simplification scheme by Garland [18]. They suggest finding a global quadric for each vertex, and simplifying the global connectivity. Kircher and Graland [19] propose using a multiresolution representation of the first frame, and then to send connectivity updates (swaps) with each

following frame. This way, they allow connectivity to change without need to send the whole connectivity with each frame.

None of the simplification methods so far proposed considered compression as the necessary following step in dynamic 3D mesh processing.

Algorithm Description

Our scheme consists of following steps:

1. compute the PCA of the vertex trajectories
2. decimate the mesh
3. transmit the PCA coefficients for the non-decimated vertices, using parallelogram predictor
4. transmit the PCA coefficients for decimated vertices, using neighborhood average predictor

We will now describe each step in detail.

We have chosen the trajectory-based PCA as a core algorithm because it allows efficient dimensionality reduction. This first step allows us to express each trajectory as a combination of eigentrajectories, thus utilizing the temporal coherence.

We assume that the number of frames of a typical mesh sequence will not change dramatically in the future, because the length of an animation is dictated by film editing rules.

On the other hand, we do expect the number of vertices to increase, as more detailed meshes are likely to be transmitted in the future. From this point of view, it is better to perform PCA in the space of trajectories, where the number of samples available (which is equal to the number of vertices) is a lot greater than the dimension of the space (which is equal to the trajectory length). This also implies that the PCA itself will be performed with a relatively small correlation matrix of size $f \times f$ where f stands for the number of frames. Also, it can be expected that the PCA basis, which needs to be transmitted with the data, will be considerably smaller than in the case of eigenshapes.

After finding a basis of the space of trajectories, we express each trajectory as a linear combination of the eigentrajectories found. Usually about 80% of the coefficients can be neglected without causing any major distortion of the animation. From this point, we treat the animation as a static mesh, where each vertex has associated a vector of reduced PCA coefficients.

The key observation for the following step is that the PCA coefficients of neighboring vertices are likely to be similar. We can use any spatial predictor used for static meshes. Our suggestion is to use neighborhood average (NA) predictor, which usually provides very robust estimation. The encoder at this point sorts all the vertices according to the accuracy of their prediction by the NA predictor. This ordered set is then used as a priority queue for vertex removal.

The removal of vertices is performed simultaneously at both encoder and decoder. Note that at this point the decoder has no information about the geometry at all, and therefore

the retriangulation of the hole must be performed solely according to connectivity criteria. The only information the decoder receives is the index of the vertex to be removed, however this leaves some control to the encoder - it knows how the decoder will retriangulate, and based on this knowledge it can avoid removing such vertices where the retriangulation would introduce geometrical problems - details on this will be given in section 4.

The strategy of the encoder is the following:

1. set all vertices unlocked, evaluate the decimation costs
2. pick the best predicted vertex v from the head of the priority queue
3. if v is locked, remove v from queue and go to 2
4. simulate the retriangulation after removal of v, if it violates geometrical criteria, then remove v from the queue and go to 2
5. lock the neighbours of v
6. mark the vertex v to be removed by the decoder at the current level of simplification
7. if there are some vertices left then go to 2
8. if further simplification is required then go to 1

The simplification process is repeated several times in order to create multiple simplification levels of the mesh. After each step all the vertices are unlocked (step 1), and the

prediction accuracy (i.e. simplification cost) is reevaluated at each vertex, again allowing all the vertices to be removed in the following simplification step.

After this process, the encoder and decoder share a simplified connectivity, and every removed vertex can be predicted from its neighbors (however, the decoder has still no information about the geometry of any vertex).

Note that there is some overhead associated with the decimation. However, we don't need to send the exact order of vertices to be removed (which would take $v.log_2 v$ bits, v being the number of vertices), the only information the decoder needs is at which level of decimation each vertex should be removed. To transmit this information, we only need $v.log_2 s$ bits, where s is the number of simplification levels, or even less than that when entropy coding is used (simplification levels are not distributed uniformly, most vertices will be removed at first level, less at second etc.).

At this point, the encoder and decoder share a series of increasingly simplified versions of the mesh, where each finer one can be obtained by inverse vertex removal, i.e., we know where each vertex should be placed to reach a finer version of topology. In our experiments, we have usually driven the algorithm to reduce the number of vertices to less than one fifth of their original number.

Now we use another spatial predictor to finally pass some geometry information to the decoder. We traverse the coarsest mesh in EdgeBreaker [20] fashion, however we don't need to send the CLERS string at all, as both sides share the connectivity of the coarsest level. The EdgeBreaker allows us to use the parallelogram predictor to predict the PCA

coefficients at each vertex - the original version which predicted the spatial coordinates can be easily extended to predict a general dimension vector c_i^D:

$$Pred(c_i^D) = c_i^B + c_i^C - c_i^A \tag{1}$$

where c_i^X stands for PCA coefficient vectors at vertices $X = A, B, C$ and D, where A, B and C were already transmitted, and D is a vertex found in a "new vertex" EdgeBreaker operation (see Fig. 1).

Both encoder and decoder perform this prediction, and the only information that is being sent with each vertex is a vector of quantized prediction residuals (except for the initial three vertices, which need to be fully transmitted). This way, we transmit the complete geometry of the coarsest version of the animation.

At this point, the decoder can start playing the animation, while it is still receiving refinement information. The parallelogram predictor is now replaced by the NA predictor, and the decoder continuously refines the mesh by reversing the previous decimation. It is guaranteed, that for each vertex the decoder has all the neighbors available, and therefore it can compute the NA prediction. Again, the encoder only sends the quantized residuals.

Algorithm details

There are two gaps in the scheme that are to be filled - the conditions for selecting the vertices for decimation, and the criteria for retriangulation performed in the decoder. We are suggesting simple approaches for both tasks.

If we were creating only one decimation level, then we generally would not care about the geometry properties of the created holes, however when multiple simplification levels are being created it is advisable to preserve a reasonable quality mesh. On the other hand, the encoder cannot control the way the hole is retriangulated by the decoder, as this would require additional information to be sent.

The retriangulation process must be driven by connectivity only, as this is the only information available at the decoder. Although there are more advanced methods for this task, we have chosen a simple "ear cutting" algorithm[21] for triangulation of a simple hole. The decoder however has no information about the convexity/concavity of the hole border, and therefore it simply considers any vertex to be a candidate ear tip.

The only criterion the decoder uses for selecting an ear tip is the regularity of degrees of the vertices. It is known that the expected degree of a vertex is six, and good shaped meshes have very regular distribution of vertex degrees. Cutting an ear does not increase the ear tip degree, while it increases the degree of the two vertices next to the tip by one. Therefore, we select the candidate tip where the following expression is maximal:

$$\delta(tip) - \delta(tip_{left}) - \delta(tip_{right}) \tag{2}$$

where $\delta(x)$ stands for the degree of vertex x. The tip is cut, and the remaining hole is again searched for best candidate tip, until it is fully retirangulated. Ambiguous situations are solved by selecting a candidate tip vertex with lowest index, so that both encoder and decoder have the same retriangulation.

There is however no guarantee that the retriangulation is geometrically correct. Therefore, we perform the retriangulation at the encoder as well and check for geometry correctness. If it is not preserved, then the vertex is not decimated. We only evaluate the criteria for the first frame, as it is likely that if geometry is not compromised in the first frame then it will not be changed in the following frames either. Note here that simplification is only a tool for compression, and we do not aim for perfect simplified version of the model, nor is geometry preservation of essential importance for us.

First, we check for normal flips. We compute an average normal of the hole by averaging the normals of all removed triangles. If any of the normals of the new triangles is oriented opposite to the original normal (we use scalar multiplication), then the decimation is not performed.

Second, we check for ”sharp” triangles. We check all the corner angles of the new triangles, and if any angle is lower than some threshold (0.05 rad in our experiments, i.e. cca 1.4°) then the decimation of the vertex is also cancelled.

Finally, we do not consider decimation of all the vertices. We only use the first 80% of the priority queue, the rest is considered to be too badly predicted by the NA predictor. The exact value of this threshold has only low significance, setting it too low results in higher number of simplification levels, while setting it too high (above 95%) results in higher values of prediction residuals.

Evaluation

We have implemented our algorithm and compared it to competing approaches. We have used several datasets of varying properties and density. We have performed testing with the chicken sequence, however we note that the nature of this sequence is not ideal for our approach, as it is a low-poly model (3030 vertices, 400 frames) where the simplification does not provide so substantial improvement as it does in the case of highly detailed dense mesh sequences.

We have used high precision data sets which describe human movement. We have used the dance sequence (7000 vertices, 200 frames), the human jump sequence [22][23](15700 vertices, we have chosen a subset of 200 frames) and a walk sequence (36000 vertices, 187 frames) generated by the animation software SmithMicro Poser (http://graphics.smithmicro.com/go/poser).

For the human motion sequences it can be argued that a skinning approach, where a bone system is sent with the model in some basic pose, can achieve better compression ratios. However, aside from the fact that for some of the sequences a bone system is not known, we have also tested our approach on a sequence of falling cloth (10 000 vertices, 200 frames), where a bone system can be applied only with some difficulties[24], and even then it produces quite high number of bones, higher than the number of eigentrajectories.

Unfortunately, there is no public implementation of the state of the art methods for dynamic mesh compression, and therefore we could only compare our results with the results that are claimed by the authors of the corresponding papers. This also restricts the choice of

quality measures to the approaches used in given papers, which are usually based on MSE computation or per-frame Hausdorff distance evaluation. There are some recent efforts to create a better human-vision based metric for dynamic mesh processing [25], however the area is still waiting for some standard metric which would be followed by the researchers.

We have used two different error measures to compare our results with competing algorithms. First, we have used the measure proposed by Karni and Gotsman [9], denoted KG-error. In this approach an animation is represented by a $3v \times f$ matrix A, where each column represents one frame of the sequence. The error is expressed as

$$KG_{error} = 100 * \frac{\|A - A_{decoded}\|}{\|A - E(A)\|} \tag{3}$$

where E(A) is a matrix where elements at each column have been replaced by the mean value of the column. This way the error metric becomes scale invariant. Please note that the value of this error metric depends on the length of the sequence.

The second metric that we have used is the RMSE used by Mamou [13]. This metric utilises the Hausdorff metric concept, which computes the distances between two surfaces instead of computing point to point distances. The approach employed is to work the sequence in a frame by frame fashion, and in each frame compute the RMS value (mean surface to surface distance, for details see [26]). Finally, the RMS values for each frame are averaged, yielding the RMSE value.

Note that for all the tests we have included all the needed information into a rate measure. The data therefore consists of PCA basis, level grouping information, parallelogram

prediction residuals and NA prediction residuals, and the rate is computed as

$$r = \frac{connectivity + basis + grouping + PP + NA}{fv} \tag{4}$$

where f is the number of frames and v is the number of vertices.

First, we have investigated the influence of the simplification level on the prediction performance. Figure 4 shows the dependency of residual distribution on the decimation level of the human jump sequence. It can be seen, that the entropy drops when the mesh is more decimated, as more residuals become zero. Figure 6 shows the dependency of achieved rate/distortion ratio on used decimation level. It can be seen that using more NA predicted vertices leads to better performance. However, for the precise quantizations (right side of the graph), the parallelogram predictor version provides lower rate with equal distortion. This is caused by the fact that the parallelogram predictor does not require the grouping information.

The effect of the simplification step is illustrated by figure 5, where we show the theoretical amount of data needed to transmit the sequence for various numbers of simplification levels and various bitrates. The height of each column is computed as the number of residuals encoded multiplied by the entropy of the residuals. Generally, the entropy of the paralellogram residuals increases, however the number of these is lower, and the entropy of the NA residuals (whose number increases with the number of levels) is substantialy lower.

The rate/distortion curve for the walk sequence is shown in Figure 7. The rate/distortion curve for the falling cloth sequence is shown in Figure 8, showing that using the proposed

system provides improvement of almost 50%.

The rate/distortion curve for the dance sequence is shown in Figure 9 along with values for the same sequence obtained by competing algorithms. The rate/distortion curve for the chicken sequence is shown in Figure 10 along with values for the same sequence obtained by competing algorithms. These figures show that for the low-poly sequence our algorithm slightly outperforms the competing approaches, however in the case of more detailed mesh the gain is much bigger.

The table 1 gives details about the ratio of types of transmitted data, running times and the amount of data that needs to be transmitted before the decoder can start playing the animation. It can be seen that the compression time, although not real-time, is still acceptable and compared to some competing methods (Geometry Videos, RD-optimized BSP trees) can be considered as very fast. Finally, the table 2 gives a summary of the findings for the data sets used in our experiments.

Conclusions and Future Work

We have shown that it is possible to combine dynamic mesh compression with simplification in a natural matter, which allows improved compression ratios, as well as level of detail displaying and incremental transmission. Our scheme only consists of procedures well known in computer graphics, such as PCA, EdgeBreaker traversal, parallelogram prediction or neighborhood prediction.

Our approach can be seen as an ultimate extension of the LCF-based method, which is based on the fact that vertex coordinates are better compressed when expressed in dependency on some other vertices in the vicinity. Our algorithm expresses most of the vertices based on their immediate neighborhood, thus achieving better results and removing the need to create any vertex clustering.

The method is well suited for sequences of highly detailed geometry with hundreds of frames. It is probably not the best possible choice for low-poly meshes or extremely long lasting animations, however dividing a long animation into shorter frame series should fix the problem quite easily.

In the future, the method can be enhanced by replacing some of the trivially solved parts with more rigorous solutions. Especially the retriangulation of holes can be performed using other approaches known in the literature.

We would also like to test our approach along with competing algorithms using different error measures. We see current quality measures as despicably insufficient, and we will focus our research on finding measures that better model human spatio-temporal perception of moving 3D content.

Our implementation of the presented algorithm is publicly available at

`http:\\herakles.zcu.cz\~lvasa\compression`

in the form of modules for the Modular Visualisation Environment. Any comments or comparisons with other compression methods are welcome.

Acknowledgements

This work has been supported by EU within FP6 under Grant 511568 with the acronym 3DTV and by the Ministry of Education, Youth and Sports of the Czech Republic under the research program LC-06008 (Center for Computer Graphics).

The chicken character was created by Andrew Glassner, Tom McClure, Scott Benza and Mark Van Langeveld. This short sequence of connectivity and vertex position data is distributed solely for the purpose of comparison of geometry compression techniques.

References

[1] Jerome Edward Lengyel. Compression of time-dependent geometry. In *SI3D '99: Proceedings of the 1999 symposium on Interactive 3D graphics*, pages 89–95, New York, NY, USA, 1999. ACM Press.

[2] Lawrence Ibarria and Jarek Rossignac. Dynapack: space-time compression of the 3d animations of triangle meshes with fixed connectivity. In *SCA '03: Proceedings of the 2003 ACM SIGGRAPH/Eurographics symposium on Computer animation*, pages 126–135, Aire-la-Ville, Switzerland, Switzerland, 2003. Eurographics Association.

[3] Nikolce Stefanoski and Joern Ostermann. Connectivity-guided predictive compression of dynamic 3d meshes. In *Proc. of ICIP '06 - IEEE International Conference on Image Processing*, number 0, oct 2006.

[4] Jinghua Zhang and Jinsheng Xu. Optimizing octree motion representation for 3d animation. In *ACM-SE 44: Proceedings of the 44th annual Southeast regional conference*, pages 50–55, New York, NY, USA, 2006. ACM Press.

[5] Karsten Muller, Aljoscha Smolic, Matthias Kautzner, Peter Eisert, and Thomas Wiegand. Rate-distortion-optimized predictive compression of dynamic 3d mesh sequences. *SP:IC*, 21(9):812–828, October 2006.

[6] Karsten Muller, Aljoscha Smolic, Matthias Kautzner, Peter Eisert, and Thomas Wiegand. Predictive compression of dynamic 3d meshes. In *ICIP05*, pages I: 621–624, 2005.

[7] Frederic Payan and Marc Antonini. Wavelet-based compression of 3d mesh sequences. In *Proceedings of IEEE ACIDCA-ICMI'2005*, Tozeur, Tunisia, November 2005.

[8] Marc Alexa and Wolfgang Müller. Representing animations by principal components. *Computer Graphics Forum*, 19(3), 2000.

[9] Zachi Karni and Craig Gotsman. Compression of soft-body animation sequences. In *Computers & Graphics 28, 1*, pages 25–34. Elsevier, 2004.

[10] Mirko Sattler, Ralf Sarlette, and Reinhard Klein. Simple and efficient compression of animation sequences. In *SCA '05: Proceedings of the 2005 ACM SIGGRAPH/Eurographics symposium on Computer animation*, pages 209–217. ACM Press, 2005.

[11] Rachida Amjoun. Efficient compression of 3d dynamic mesh sequences. In *Journal of WSCG*, volume 15 of *Journal of WSCG*. University of West Bohemia, January 2007.

[12] Libor Váša and Václav Skala. Coddyac: Connectivity driven dynamic mesh compression. In *3DTV-CON, The True Vision - Capture, Transmission and Display of 3D Video*, 2007.

[13] Khaled Mamou, Titus Zaharia, and Francoise Preteux. A skinning approach for dynamic 3d mesh compression: Research articles. *Comput. Animat. Virtual Worlds*, 17(3–4):337–346, 2006.

[14] H. Briceno, P. Sander, L. McMillan, S. Gortler, and H. Hoppe. Geometry videos: A new representation for 3d animations. In *ACM Symposium on Computer Animation 2003*, pages 136–146. Eurographics Association, 2003.

[15] Xianfeng Gu, Steven J. Gortler, and Hugues Hoppe. Geometry images. In *Proceedings of the 29th annual conference on Computer graphics and interactive techniques*, pages 355–361. ACM Press, 2002.

[16] Nikolce Stefanoski, Xiaoliang Liu, Patrick Klie, and Jörn Ostermann. Scalable linear predictive coding of time-consistent 3d mesh sequences. In *3DTV-CON, The True Vision - Capture, Transmission and Display of 3D Video*, May 2007.

[17] Alex Mohr and Michael Gleicher. Deformation sensitive decimation. Technical report, University of Wisconsin Graphics Group, 2003.

[18] Michael Garland. Quadric-based polygonal surface simplification, 1998.

[19] Scott Kircher and Michael Garland. Progressive multiresolution meshes for deforming surfaces. In *SCA '05: Proceedings of the 2005 ACM SIGGRAPH/Eurographics symposium on Computer animation*, pages 191–200. ACM Press, 2005.

[20] Jarek Rossignac. Edgebreaker: Connectivity compression for triangle meshes. *IEEE Transactions on Visualization and Computer Graphics*, 5(1):47–61, 1999.

[21] G.H. Meisters. Polygons have ears. *American Mathematical Monthly*, pages 548–551, June 1975.

[22] Nizam Anuar and Igor Guskov. Extracting animated meshes with adaptive motion estimation. In *VMV*, pages 63–71, 2004.

[23] Peter Sand, Leonard McMillan, and Jovan Popovič. Continuous capture of skin deformation. In *SIGGRAPH '03: ACM SIGGRAPH 2003 Papers*, pages 578–586, New York, NY, USA, 2003. ACM Press.

[24] Ladislav Kavan, Steven Collins, Jiri Zara, and Carol O'Sullivan. Skinning with dual quaternions. In *2007 ACM SIGGRAPH Symposium on Interactive 3D Graphics and Games*, pages 39–46. ACM Press, April/May 2007.

[25] Libor Váša and Václav Skala. A spatio-temporal metric for dynamic mesh comparison. In *AMDO Lecture Notes on Computer Graphics*, number 4096, pages 29–37. Springer-Verlag Berlin Heidelberg, 2006.

[26] Paolo Cignoni, Claudio Rocchini, and Roberto Scopigno. Metro: Measuring error on simplified surfaces. *Computer Graphics Forum*, 17(2):167–174, 1998.

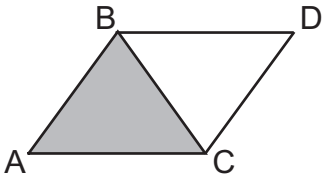

Figure 1: Parallelogram prediction.

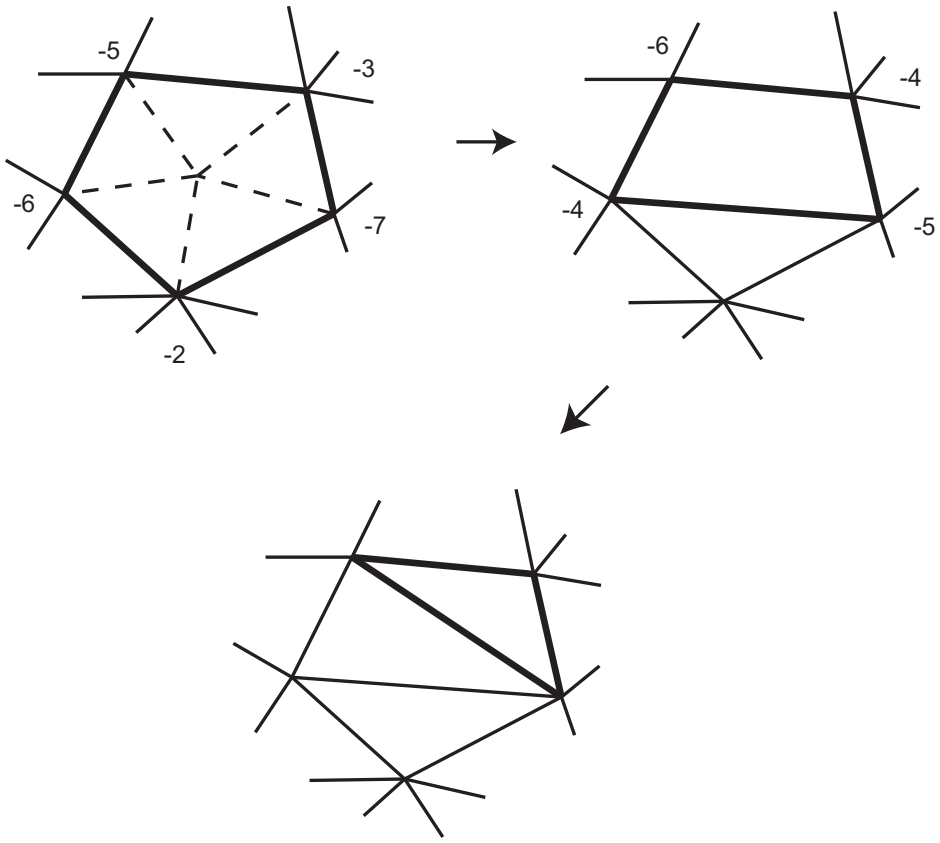

Figure 2: An example of the hole retriangulation process.

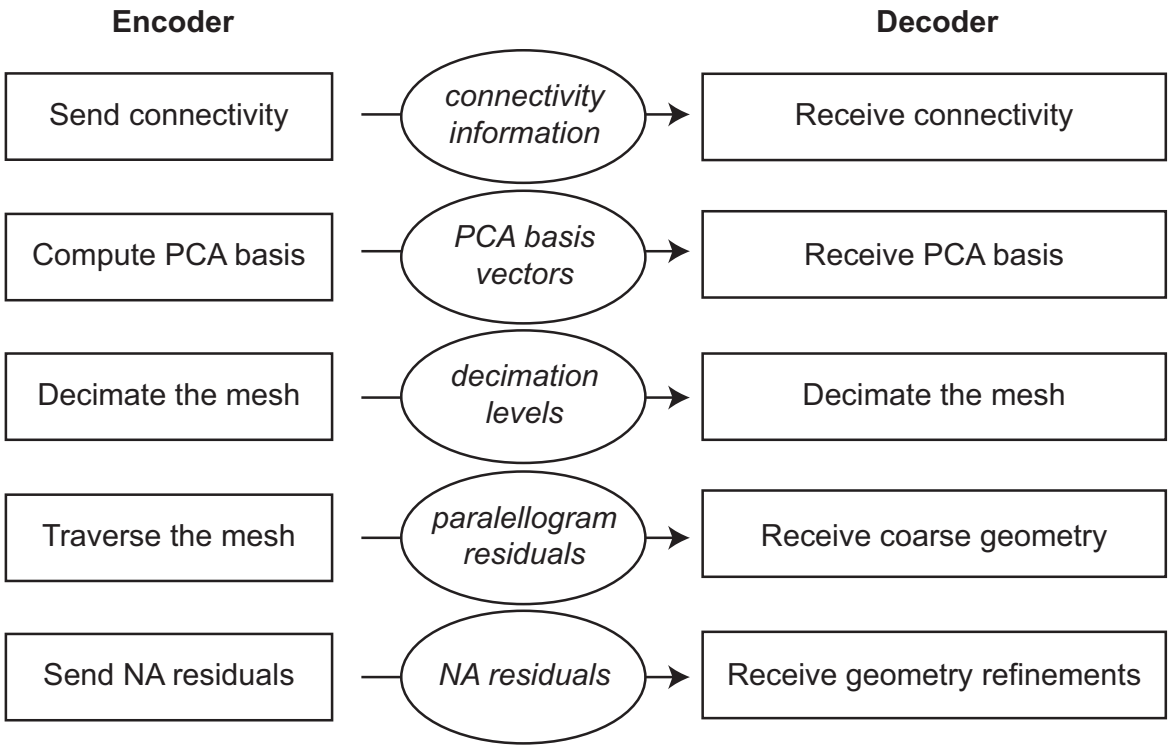

Figure 3: The data flow between the encoder and the decoder.

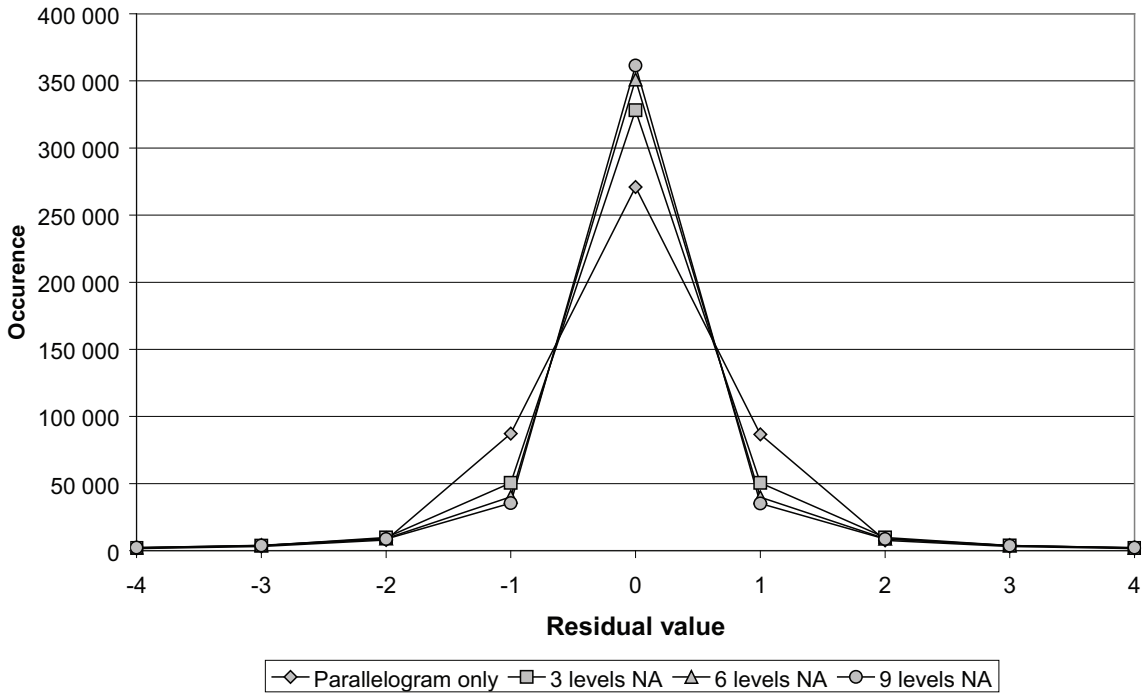

Figure 4: Prediction residual distribution for various decimation levels. Each curve shows histogram of most common residuals when given number of simplification steps is performed. The occurence of zero increases with the number of simplification levels, while the occurence of 1 and -1 decreases, which overall leads to a reduction of the entropy.

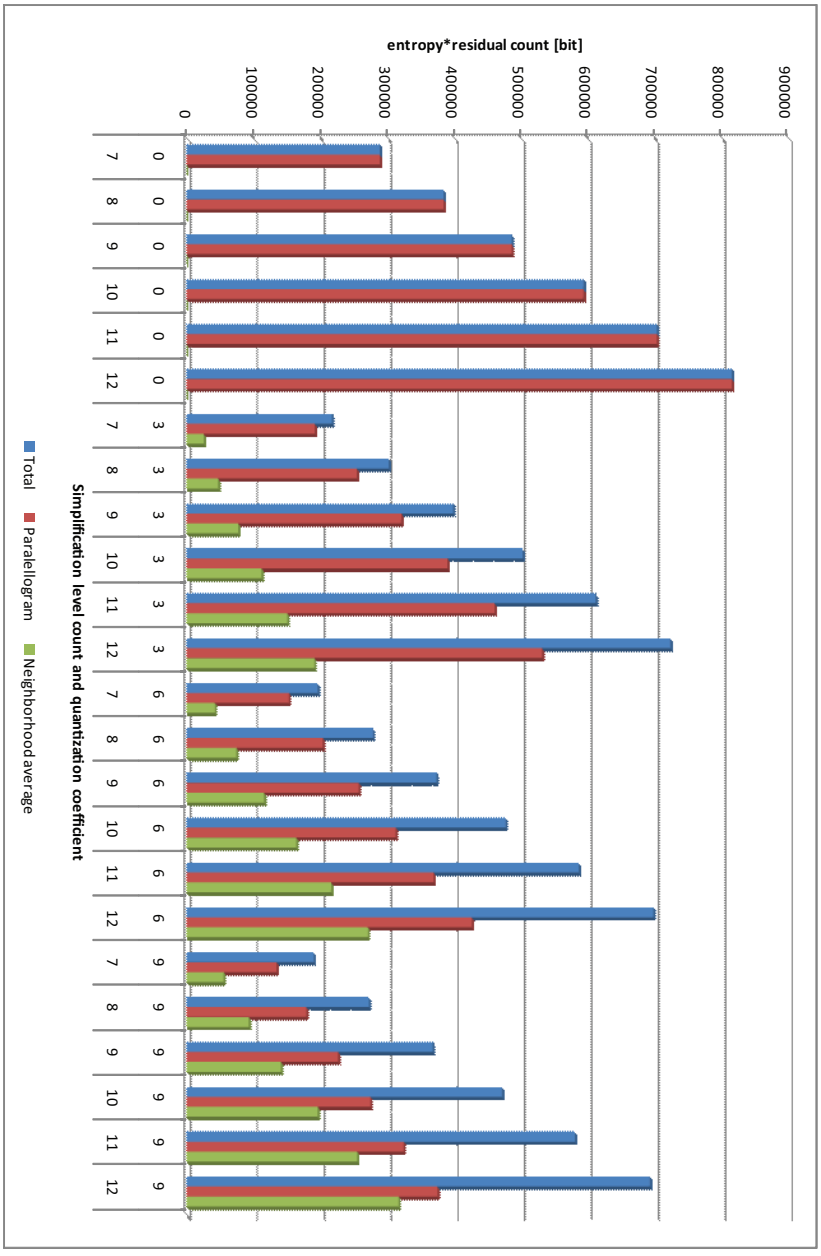

Figure 5: Lengths of encoded residuals for various simplification amounts and quantizations of the chicken sequence.

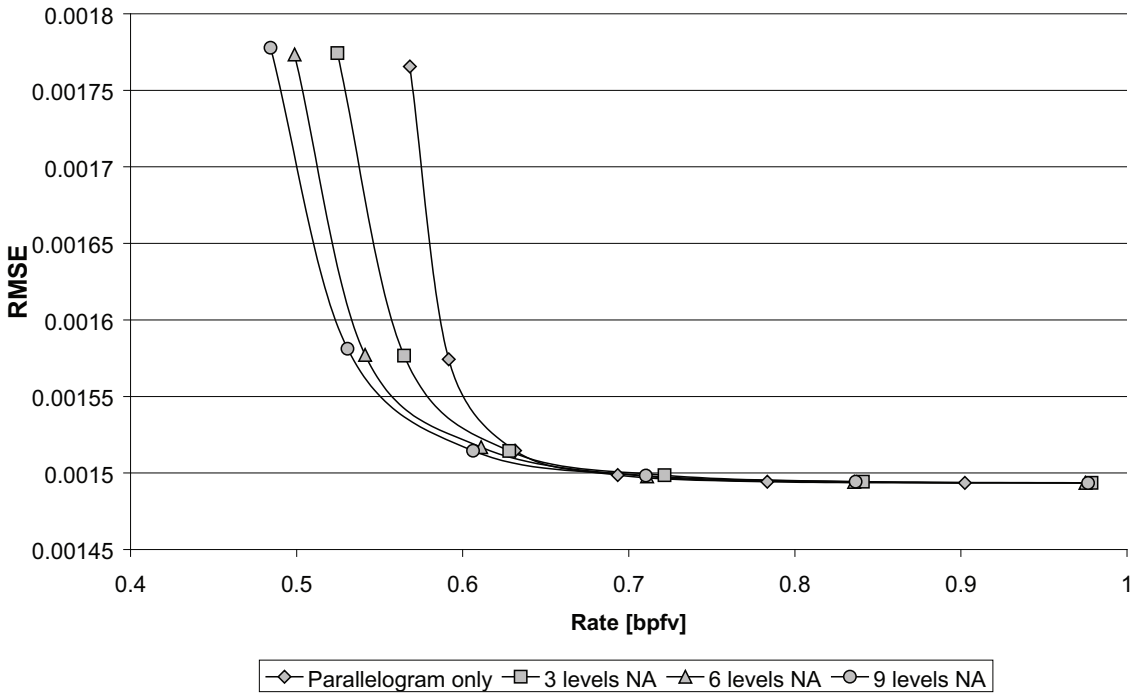

Figure 6: Rate/distortion ratios for various decimation levels of the human jump sequence.

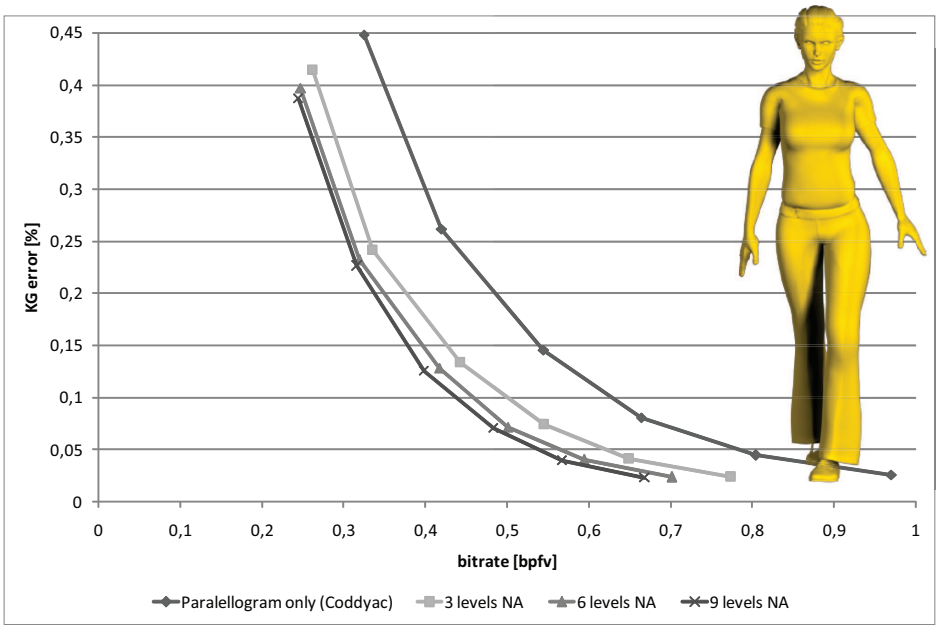

Figure 7: Rate/distortion ratio for the walk sequence. Each curve represents different quantizations with given number of basis vectors.

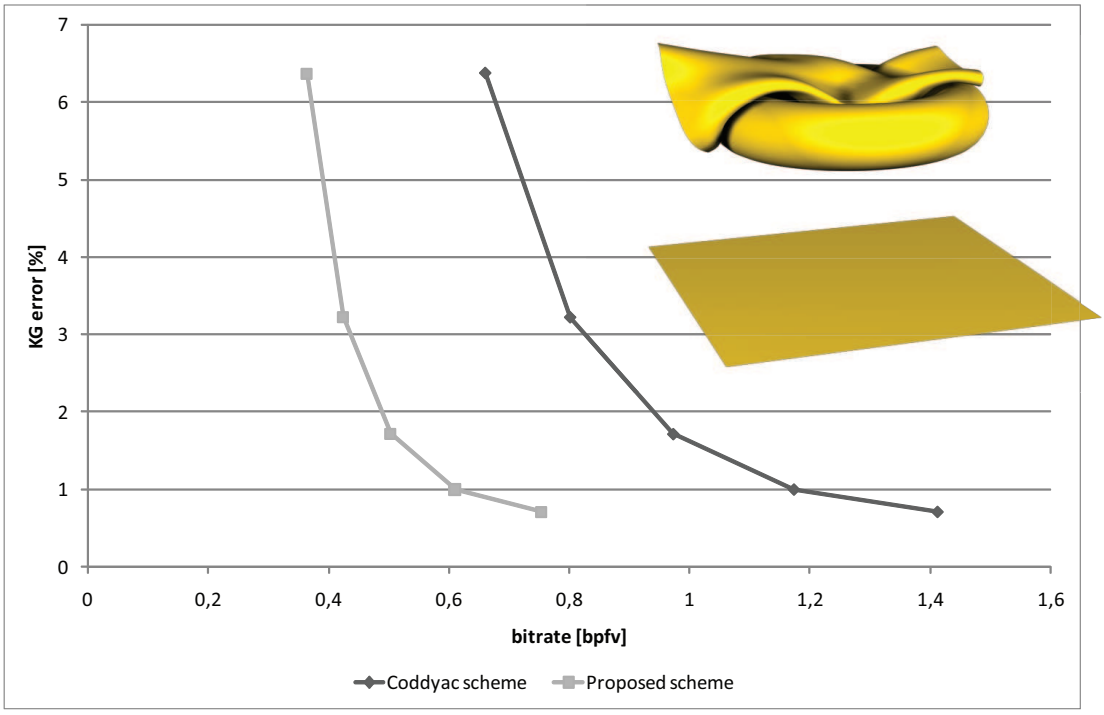

Figure 8: Rate/distortion ratio for the falling cloth sequence. Each curve represents different quantizations with given number of basis vectors.

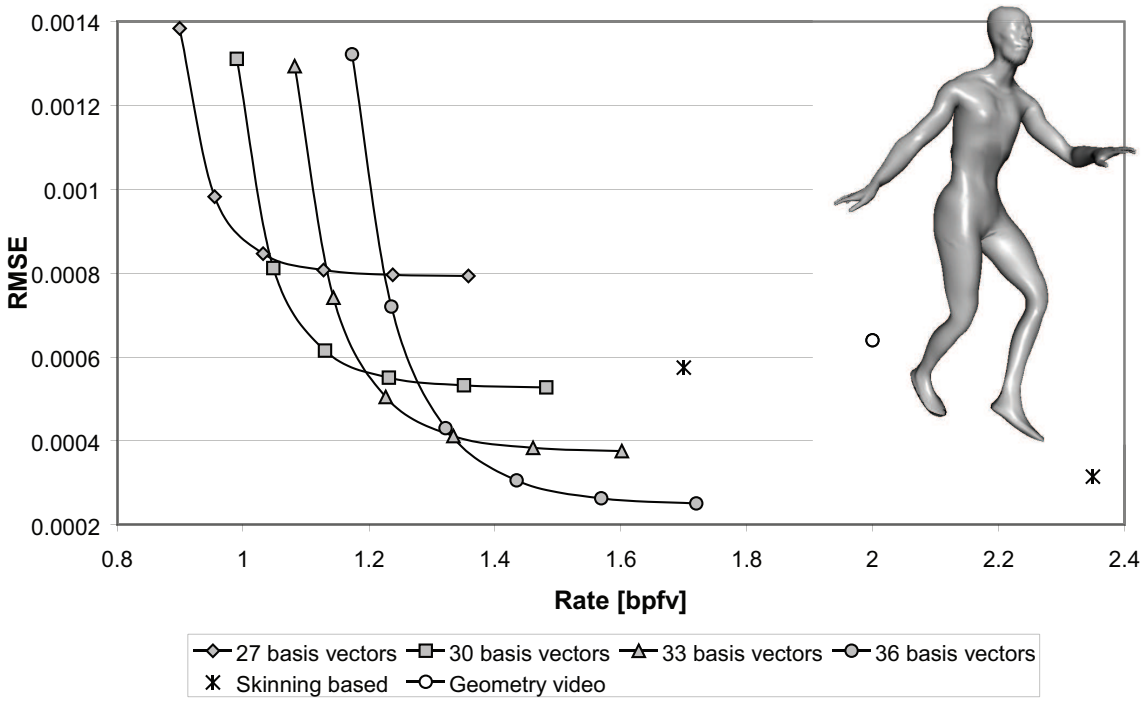

Figure 9: Rate/distortion ratio for the dance sequence. Each curve represents different quantizations with given number of basis vectors. The values for other methods were taken from the corresponding publications.

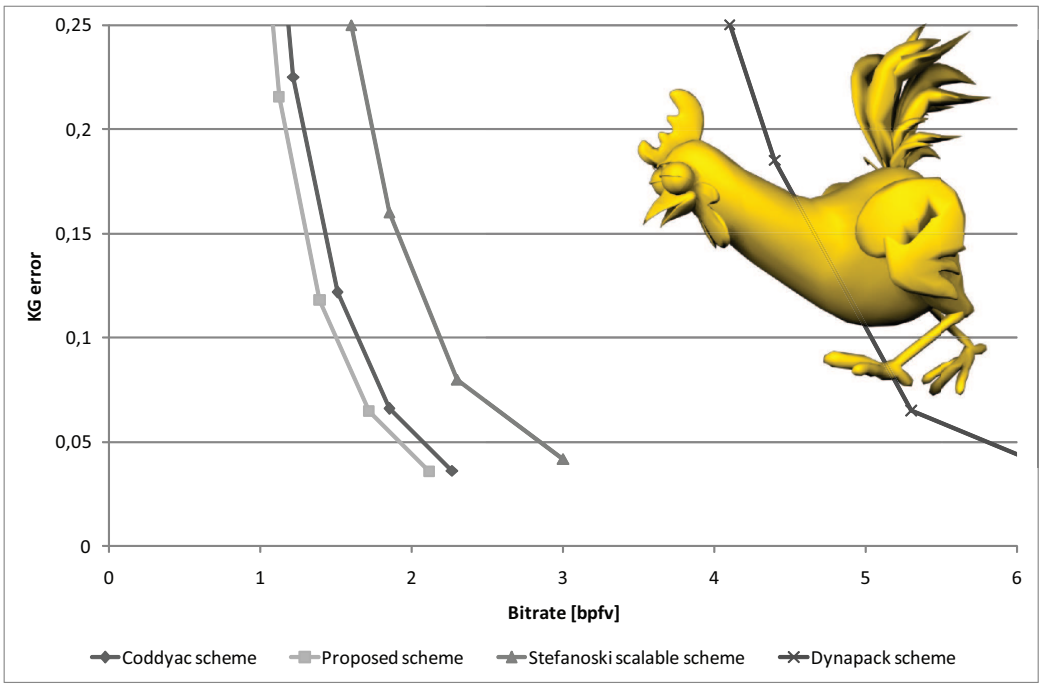

Figure 10: Rate/distortion ratio for the chicken sequence. The values for other methods were taken from the corresponding publications.

dataset	human jump		dance	
vertices	15830		7036	
frames	200		200	
basis vectors	50		33	
original length [kb]	24734		10994	
basis size [kb]	234.38	20.0%	158.4	26.6%
total residual count	791350		232089	
total residual entropy	1.204		1.913	
total residual size [kb]	930.62	79.5%	433.47	72.9%
PP residual count	139850		81840	
PP residual size [kb]	164.46	14.0%	152.85	25.7%
NA residual count	651500		150249	
NA residual size [kb]	766.16	65.4%	280.62	47.2%
vertex grouping size [kb]	5.80	0.5%	2.58	0.4%
total encoded size [kb]	1170.79	100.0%	594.45	100.0%
intital data [kb]	404.63	34.6%	313.83	52.8%
compression time [s]	123		228	

Table 1: Size ratios and running times.

sequence	method	frames	vertices	basis vectors	encoded data length [B]	bitrate [bpfv]	KG error [%]
chicken	coddyac	350	3030	48	179915	1,357209	0,145279
chicken	proposed	350	3030	48	165109	1,245518	0,142822
walk	coddyac	187	35626	40	453334	0,544377	0,145839
walk	proposed	187	35626	40	331304	0,39784	0,125729
cloth	coddyac	200	9987	40	242860	0,972705	1,713824
cloth	proposed	200	9987	40	125645	0,503234	1,713795

Table 2: Proposed method compared to Coddyac scheme.