Computer Graphics Forum, Vol.29,No.6, pp.1921-1933, ISSN 0167-7055, 2010

Geometry driven local neighbourhood based predictors for dynamic mesh compression

Libor Váša[1] and Václav Skala[1]

[1] Department of Computer Science and Engineering, Faculty of Applied Science, University of West Bohemia, Czech Republic

Abstract

The task of dynamic mesh compression seeks to find a compact representation of a surface animation, while the artifacts introduced by the representation are as small as possible. In this paper we present two geometric predictors, which are suitable for PCA based compression schemes. The predictors exploit the knowledge about the geometrical meaning of the data, which allows a more accurate prediction, and thus a more compact representation. We also provide rate/distortion curves showing that our approach outperforms the current PCA-based compression methods by more than 20%.

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

1. Introduction

Dynamic mesh compression is a topic that has received increased popularity in the last period. The research activities are focused on finding a compact representation for 3D surface animations. An animation is represented by a series of static triangular meshes (frames) of shared connectivity. The geometry of the subsequent frames usually does not change radically, because the temporal distance between two frames is usually 1/25 s or 1/30 s.

The problem of connectivity compression is of minor importance to this case. First because it has been studied intensively for the static case, and the results can be directly applied to the dynamic case, and second because the encoded size of a single frame's connectivity is almost negligible when compared to the geometry data from all the frames.

The problem of dynamic mesh compression has been addressed from many points of view. One of the most promising is the representation of the animation by a series of samples in the space of trajectories, each vertex having one sample. Such representation provides useful properties, such as:

1. possibility of very strong dimensionality reduction using a tool such as the PCA (Principal Component Analysis) to find an optimal basis for expressing the samples,
2. possibility to predict the trajectories of topologically incident vertices by some kind of interpolation or extrapolation of neighbouring sample values,
3. suitability for direct and memory efficient displaying using modern GPUs with programmable vertex processing pipeline,
4. possibility of compact representation without a need for a bone system, which may be unknown, or unfit for transmitting due to intellectual property issues.

In this paper we extend the trajectory space PCA based compression scheme by adding two predictors, which allow higher precision estimation of trajectories, and thus more efficient compression. The predictors are suited to be used in a situation, when the trajectories of a topological neighbourhood of a vertex are already transmitted. This state is achieved by a compression scheme which involves vertex decimation, such as the scalable compression by Stefanoski *et al.* [SLKO07].

In our approach, we update the predictor during the transmission of the coordinates of a single vertex. The nature of the PCA removes the global correlation between the coordinates, and thus we cannot directly determine anything about a subsequent coordinate based on the value of the previous one. However, when the neighbourhood is known, we can make some assumptions on the relations between the neighbourhood and the decoded vertex, allowing a more efficient compression.

Computer Graphics Forum, Vol.29,No.6, pp.1921-1933, ISSN 0167-7055, 2010

2. Related Work

First attempt to dynamic mesh compression has been published in the paper by Lengyel [Len99], who suggested subdividing the mesh into clusters in which the movement can be described by a single transformation matrix.

Ibarria and Rossignac [IR03] later suggested a spatio-temporal prediction schemes ELP and Replica, which were used to predict next vertex position during a mesh traversal using the EdgeBreaker state machine. A similar approach has been used by Stefanoski *et al.* [SO06] in the angle preserving predictor. The position of the new vertex is expressed in a local coordinate system defined by a neighboring triangle.

Owen and Zhang [ZO04] have proposed exploiting spatial coherence of the data using an octree to subdivide the model and encoding each subdivision cell separately. This approach has been improved by Mueller *et al.* [MSK*06,MSK*05]. In their approach they select the best fitting appropriate predictor for each cell, and the cells which are predicted badly are further subdivided.

The wavelet theory has been used for dynamic mesh compression in the work by Payan and Antonini [PA05], 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 [AM00], who suggested using the PCA in the space of frames, expressing each frame as a linear combination of eigen frames. However, this method had problems with rigid movement, which had to be compensated 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 [KG04], who suggested exploiting the temporal coherence of the PCA coefficients by encoding them using linear prediction coding (LPC), thus achieving a lower entropy of the encoded data. Another improvement has been proposed by Sattler *et al.* [SSK05], who suggested using PCA in the space of trajectories, and finding clusters of vertices where the PCA worked well (Clustered PCA). However, their iterative clustering method did not always reach the same clustering because it had been randomly initialised.

Another addition to the PCA based method was proposed in 2007 by Amjoun [Amj07, AS07], who suggested using a 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. This paper also mentions the compression of the PCA basis, however it suggests simple direct encoding without prediction and with uniform quantization of the basis matrices.

Mamou [MZP06] has proposed an approach similar to the PCA, called skinning based compression. The mesh is first segmented into parts that move in an almost rigid fashion. The movement of each cluster is expressed by a transformation matrix, and subsequently each vertex is assigned a vector of weights, that tells how to combine the transforms of the neighboring clusters to obtain the movement of the vertex.

A resampling approach has been proposed by Briceno [BSM*03] in his work on Geometry Videos. This idea is an extension of the previously proposed Geometry Images [GGH02]. 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.

Recently, there are also scalable approaches appearing, such as the scheme proposed by Stefanoski *et al.* [SLKO07]. These approaches allow progressive level of detail transmission of the dynamic mesh, and also achieve better compression ratios by using sophisticated local predictors which use the data from coarser detail levels.

In 2008, a second amendment of the MPEG-4 part 16 was published, specifying a new MPEG standard for dynamic mesh compression, the FAMC algorithm. The standard is based on the algorithms of Mamou and Stefanoski, and it includes a specific arithmetic coder based on the CABAC scheme [MWS03]. The algorithm has been shown to outperform all the algorithms available at the time of publication.

Recently Váša and Skala have published compression schemes based on the trajectory space PCA, suggesting a combination of the PCA step with an EdgeBreaker-like predictor (see Rossignac [Ros99]). The Coddyac algorithm [VS07] predicts the PCA coefficients by the well known parallelogram local predictor, which allows better performance than the clustering-based approaches. Subsequently, they have suggested using vertex decimation as a part of the compression [VS09b]. The main advantage of this approach is that it allows the encoder to partially steer the decimation process according to the accuracy of the used predictors, and therefore their approach is well suited for interchanging predictors. Finally, the authors have presented an algorithm for efficient encoding of the PCA basis [VS09a], which has boosted the performance of the algorithm so that it outperforms the FAMC standard.

Note that in contrast to Alexa [AM00] the schemes of Váša and Skala are based on PCA in the space of trajectories, which is of a much lower dimension than the space of shapes. The dimension still depends on the number of frames in the animation, however the number of frames is dictated by the rules of content editing (see Reisz *et al.* [RMD68]),

which usually state that individual scenes between scene cuts should not be longer than 20 seconds (500 frames). Moreover, a longer sequence can be quite easily split into shorter sequences, and thus the dimension of the space never has to be much larger than about 1500 (three coordinates in each frame). Therefore the schemes based on trajectory-space PCA are fully practical, because the PCA step can be usually performed in 1-2 minutes instead of hours needed for the shape space PCA.

Generally, the area of dynamic mesh compression has matured over the last years, as illustrated by the publication of the second MPEG standard for this task. It is therefore quite difficult to achieve any significant improvement of performance. In this paper we demonstrate that improvement is still possible, at the cost of increasing the computational complexity, which, however, does not compromise real-time decompression of the mesh.

3. Algorithm derivation

3.1. Used notation

In the following paragraphs we will be using following symbols:

F stands for the total number of frames in an animation.

V stands for the total number of vertices in each frame of an animation.

v_i^f stands for the three-component vector of XYZ coordinates of the i-th vertex in the f-th frame.

c_i stands for a vector of PCA coefficients (feature vector) associated with the i-th vertex. c_i^j is the j-th component of the feature vector.

R stands for a number of PCA coefficients used for encoding, usually $R << 3V$.

$n(i)$ stands for a set of indices of vertices in topological neighbourhood of i-th vertex. $n(i,j)$ is the j-th member of the set, i.e. index of the j-th neighbor of the i-th vertex.

$N(i)$ stands for the number of vertices in topological neighbourhood of i-th vertex.

$pred(\xi)$ stands for a prediction of the value ξ (vertex coordinate, PCA coefficient) which can be simultaneously computed by both encoder and decoder.

$\overline{\xi}$ denotes value of ξ (vertex coordinate, PCA coefficient) as decoded by decoder. It might be different from the original value of ξ due to quantization and other causes of data loss in transmission. The value of $\overline{\xi}$ is always known to both decoder and encoder, because encoder can simulate the decoding process to obtain the decoded value.

3.2. Overview of the background algorithms

In order to derive our predictor, we will first describe a general scheme used in some static and dynamic mesh compression algorithms. The idea is targeted on predictive delta coding, i.e. a scheme where the decoder predicts the value (coordinate etc.) being decoded as precisely as possible, the encoder simulates the prediction and only sends the difference between the actual value and the prediction. The quantized residuals generally have much lower entropy than the quantized original values, thus the final code is shorter.

Some algorithms have used an EdgeBreaker-like [Ros99] traversal of the mesh, using the parallelogram predictor to estimate the coordinate values (for static meshes see [Ros99], for dynamic case see [VS07]). The predictor in this case is based on a neighboring triangle, which is available at the decoder, and performs basically an extrapolation of the vertex coordinates of the triangle.

However, a better approach presented in [VS09b] uses interpolation instead of extrapolation. In order to do so, we need to transmit the vertices in such an order that in each step a complete topological neighbourhood of the vertex is available at the decoder. Fortunately, this can be done quite easily: the decoder first receives a full connectivity, which is then simultaneously decimated (see [SZL92, COLR99]) at both the encoder and the decoder, i.e. vertices are removed from the mesh, and the resulting holes are retriangulated.

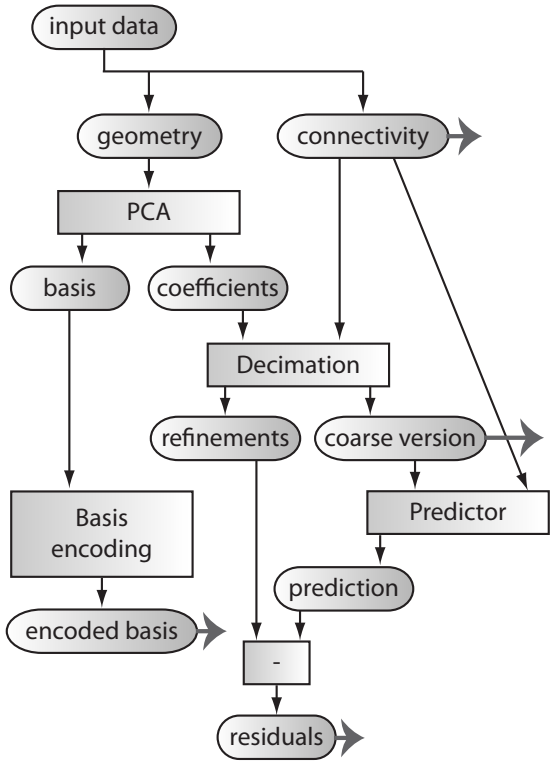

Figure 1: *Block scheme of the encoder. The thick arrows denote data that is sent over to the decoder.*

The decimation must use topology-only criteria, because

geometry is not yet known to the decoder. In this paper, we follow the scheme by Váša and Skala [VS09b] and use preservation of vertex degree close to 6. This way, a majority of vertices are removed, obtaining a very coarse version of the original mesh. The geometry of the coarse mesh is transmitted using the previously described method, and subsequently the inverse of the decimation follows, in each step adding a vertex into a neighbourhood completely known to the decoder. This allows a better estimation of the actual vertex position by simply averaging the coordinates of the neighboring vertices.

$$pred(v_i^f) = \frac{1}{N(i)} \sum_{j \in n(i)} \overline{v_j^f} \tag{1}$$

It has been suggested by in [SSK05] and [VS07] to first perform a decorrelation using the principal component analysis (PCA) in the space of trajectories. The input data is reordered to a matrix M of size $3F \times V$, where each column represents a trajectory of a single vertex. An average value is computed for each row, forming an average trajectory, and this average trajectory is subtracted from each column. The matrix is subsequently decomposed using either the singular value decomposition, or simply the eigenvalue decomposition of MM^T. In both cases a new basis of the column space of M is found, where the combination coefficients are uncorrelated.

Note that at this point, the data is decomposed into feature vectors c_i and a basis, both of which need to be transmitted to the decoder. We will not discuss the encoding of the basis here, because its efficient compression is a different problem which has been addressed by [VS09a].

Each column (vertex trajectory) of M is expressed in the new basis. Experiments show that a vast majority of variance is accumulated in only a few most important coefficients, and therefore preservation of only a subset of length R of the combination coefficients works as a good representation of the original values.

After this process, each vertex has assigned a vector of coefficients c_i (feature vector). This vector determines how to combine the basis trajectories to obtain the trajectory reconstruction. Note that at this point vertices no longer have their XYZ coordinates, and they are only determined by their topological position and by the above mentioned feature vector.

A useful property of the PCA step is that it does not affect the shape of the predictors in any way, because the PCA can be viewed as a simple change of coordinate system, which has no effect on the result of a linear operation. Thus, for the extrapolation we can use the very same formula which has been used to extrapolate the XYZ coordinates, only this time extrapolating the PCA coefficients, i.e. the components of the feature vectors c_i [VS07]. Equally, we can average

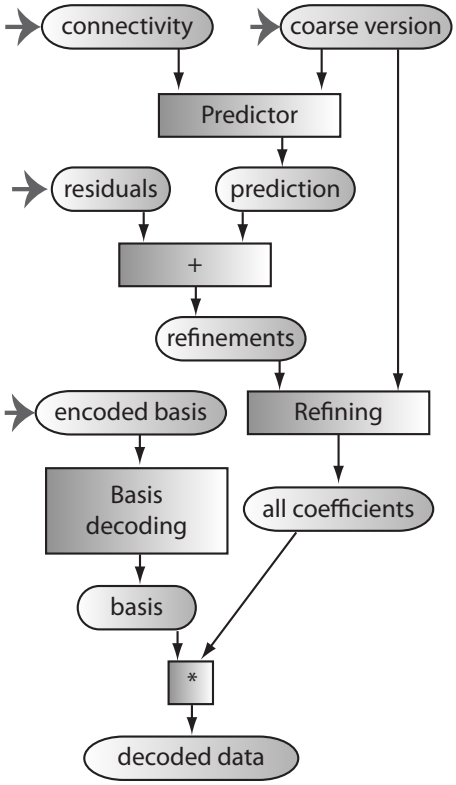

Figure 2: *Block scheme of the decoder. The thick arrows denote data that is received from the encoder.*

the components of the feature vectors during interpolation [VS09b].

$$pred(c_i) = \frac{1}{N(i)} \sum_{j \in n(i)} \overline{c_j} \tag{2}$$

Block schemes of both general encoder and decoder are shown in figures 1 and 2 respectively.

Generally, it might seem inconvenient that in order to start playback of the animation, it is necessary to first decode its full length. It is important to realise that we are dealing with compression of separate scenes usually of length about 500 frames (20 seconds), between which the user can freely navigate. In fact most video codecs also do not allow fully free temporal navigation, because in order to decode a "B frame", it is necessary to first locate and decode its surrounding "P frames", which may in turn require first decoding their respective "I frames" (see Poynton [Poy03]).

Similarly, it might seem inconvenient that the user is not given random access to vertices. However, access to separate vertices is usually only needed for the purposes of LOD playback, which is naturally supported by the incorporation

of simplification into the process. Therefore the user is not allowed to decompress any vertex at any time, however it is possible to simply stop the decompression at any time during the mesh refinement, and play the partially decompressed dynamic mesh.

3.3. Hypothesis

The hypothesis behind our approach is that **each vertex is located at a certain position relative to its neighbors**, and that this **relative position does not change too much during the whole animation**. If the decoder had the information about the relative position, then it would be able to perform a more precise prediction, and thus achieve a better bitrate.

The existing approaches, such as parallelogram extrapolation or neighbourhood average interpolation, actually already use this idea, placing the prediction either to a tip of a parallelogram formed by vertices of an adjacent triangle, or into the average position of neighboring vertices. However, as shown by Isenburg and Alliez [IA02], in most cases **this position is not optimal**.

To support our hypothesis, we have constructed a hypothetical predictor, which knows the relative position of each vertex. The relative position of i-th vertex is expressed as a set of weights $w(i)_j, j = 1..N(i)$ of its neighboring vertices used for interpolation, or by a triplet of weights $w(i)_j, j = 1..3$ for the three vertices of the adjacent triangle for the case of extrapolation (extrapolation is always used to transmit the coarse version of the mesh). These weights have been computed by the least squares optimization, and in our hypothetical scenario they do not increase the bitrate at all. Note that these weights are not barycentric coordinates (sum of weights is allowed to be different form 1), and therefore we are able to express any relative position, even when it does not lie in the plane of the neighboring vertices. The hypothetical predictor uses the following scheme for prediction during interpolation:

$$pred(c_i) = \sum_{j=1}^{N(i)} w(i)_j \overline{c_{n(i,j)}} \qquad (3)$$

Table 1 shows the performance of a PCA-based coder, using parallelogram and neighbourhood average (will be denoted NAVG) predictors, compared to the same experiment using the predictor with the knowledge of weights. It can be seen that we have achieved a decrease of bitrate for all the models, and in some cases we have even decreased the error (for details on how the errors were measured see section 6, we have used the same method throughout the paper). Roughly speaking, this decrease is caused by the improved accuracy of the predictor in the cases when the residual value is quantized to zero - more such cases appear, i.e. the entropy is decreased, and the predicted position is closer to the actual position, which causes the decrease in error, even when the quantization has not been changed.

dataset	extrapolator	interpolator	rate [bpfv]	KG error	Δrate	Δerror
chicken	Parallelogram	NAVG	1,55	0,09		
chicken	Weighted	Weighted	1,26	0,08	19,14%	3,64%
dance	Parallelogram	NAVG	1,05	0,05		
dance	Weighted	Weighted	0,70	0,05	33,36%	8,61%
jump	Parallelogram	NAVG	0,75	0,41		
jump	Weighted	Weighted	0,48	0,40	35,78%	0,43%
cowheavy	Parallelogram	NAVG	2,33	0,31		
cowheavy	Weighted	Weighted	1,70	0,31	27,15%	0,50%
snake	Parallelogram	NAVG	0,97	0,06		
snake	Weighted	Weighted	0,73	0,06	24,68%	-0,98%
cloth	Parallelogram	NAVG	0,58	0,12		
cloth	Weighted	Weighted	0,44	0,12	24,26%	1,89%
				average	**27,40%**	**2,35%**

Table 1: *Hypothetical predictor results.*

This improvement of performance is our goal, however, we cannot achieve it directly. Our hypothetical predictor assumed an exact knowledge of the weights without any impact on the bitrate, which, of course, cannot be achieved in practice. A straightforward idea is to transmit the weights along with the other data, and optimise their quantization to obtain the best rate/distortion ratio.

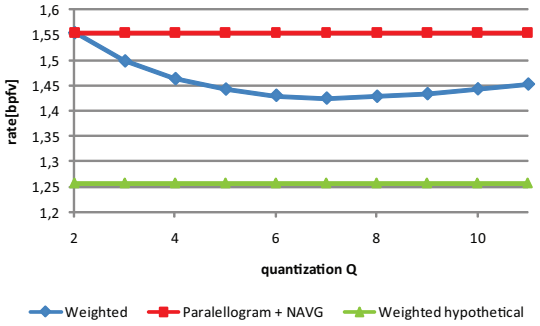

Figure 3: *Performance of the weighted predictor on the chicken run dataset. The quantization parameter Q shown on horizontal axis determines the roughness of quantization, the quantization constant is determined as $1/2^Q$. Bpfv denotes bit per frame and vertex.*

Such an approach has been implemented, and figure 3 indeed shows that for the chicken run sequence we can achieve results better than using the original predictors by using uniformly quantized weights with quantization constant of $1/2^7$. However, this result is only possible to achieve, when meshes of low detail (i.e. low number of vertices, low number of weights to be transmitted) and long duration (i.e. the weights are amortised by many frames) are used.

Although the quantization constant of $1/2^7$ remains opti-

mal, it is shown in figure 4 that for highly detailed sequence it may not be beneficial at all to transmit the weights. As for the remaining datasets, we have found results consistent with this analysis, having significant improvement in two cases (chicken run, cowheavy), slight improvement in two cases (dance and human jump) and no improvement at all in two cases (snake and cloth).

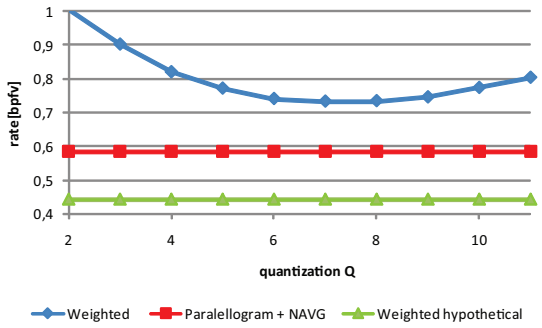

Figure 4: *Performance of the weighted predictor on the falling cloth dataset. The quantization parameter Q shown on horizontal axis determines the roughness of quantization, the quantization constant is determined as* $1/2^Q$.

In the following text we will therefore search for a way to estimate the weights without having to send them to the decoder.

3.4. Least squares prediction (LSP)

In our first real predictor, we will use least squares minimisation to obtain a vector of weights of neighboring vertices that fits the data well.

The task is to insert the i-th vertex, which has a trajectory described by a feature vector c_i of length R.

We assume that $R > N(i)$, because for regular meshes $N(i)$ is usually about 6, while R needs to be about 30-80, depending on the length and character of the animation.

For the first $N(i)$ components of the vector c_i we have to use neighbourhood average prediction in the following form:

$$pred(c_i^j) = \frac{1}{N(i)} \sum_{k \in n(i)} \overline{c_k^j}, j = 1..N(i) \qquad (4)$$

The encoder simulates this prediction, and sends over the corrections, so that after N steps, $\overline{c_{1..N(i)}}$ are known to both encoder and decoder. The prediction so far can be seen as combining the neighboring vectors using weights equal to $\frac{1}{N(i)}$. This would be a perfect predictor if the new vertex was placed in the exact centre of the hole. However, it is usually not the case (not even for regular meshes, because the iterative decimation usually destroys the regularity). Therefore, the decoder will now estimate a better set of weights $w_j, j = 1..N(i)$, using the following system of linear equations:

$$\begin{aligned}
w_1\overline{c^1_{n(i)_1}} + w_2\overline{c^1_{n(i)_2}} + \cdots + w_{N(i)}\overline{c^1_{n(i)_{N(i)}}} &= \overline{c_i^1} \\
w_1\overline{c^2_{n(i)_1}} + w_2\overline{c^2_{n(i)_2}} + \cdots + w_{N(i)}\overline{c^2_{n(i)_{N(i)}}} &= \overline{c_i^2} \\
&\vdots \\
w_1\overline{c^{N(i)}_{n(i)_1}} + w_2\overline{c^{N(i)}_{n(i)_2}} + \cdots + w_{N(i)}\overline{c^{N(i)}_{n(i)_{N(i)}}} &= \overline{c_i^{N(i)}}
\end{aligned} \qquad (5)$$

The matrix of this system of linear equations should be regular (unless two neighbors are located at the same position - this situation may appear due to quantization, and its treatment will be described later). Therefore a solution can be found and used to predict the component $N+1$:

$$pred(c_i^{N(i)+1}) = \sum_{j=1}^{N(i)} w_j \overline{c^{N(i)+1}_{n(i,j)}} \qquad (6)$$

Again, this prediction is computed at both encoder and decoder, and the encoder sends a correction, which makes the value of $\overline{c_i^{N+1}}$ known to the decoder. Thus, the system of equations (5) can be extended by one row. This makes the set overdetermined, however, it can be still solved using the least squares minimisation. Such solution will yield an even more precise estimate of the weights w_j, which will be then used to compute $pred(c_i^{N+2})$. In this manner, with each correction sent to the decoder, a new set of weights is found and used to predict the subsequent component, until the whole vector has been transmitted.

There are two additional circumstances that require treatment:

1. The coordinates are quantized, and therefore it might happen that two or more neighbors have the same values assigned. This may lead to underdetermination of the system of linear equations (5). We solve such case by simply removing the neighbors with equal values assigned from the computation. Note that this state is likely to be corrected when more components are transmitted, as it is unlikely that two vertices shared the same position. If such case occurs, it is an indication that the mesh has been quantized too roughly.
2. The values transmitted are the PCA coefficients, and thus their magnitude is approximately sorted from large to small. The quantized residuals of the large values contribute a big part to the entropy, and thus we should focus on precise prediction of these. Therefore, we transmit the coefficient vectors in the reverse order, first transmitting

the small coefficients, which allows us to have a very precise set of weights at the point when large coefficients are transmitted.

The interpolating predictor presented in this paragraph can be simply transformed into an extrapolating predictor by simply considering the three vertices of a neighboring triangle instead of the vertices of the neighbourhood.

The overdetermined matrix solved during the evaluation of the LSP is of size $R \times N(i)$, where R stands for the number of basis vectors used and $N(i)$ stands for the number of neighbors (which does not change significantly). The least squares solution of this matrix can be obtained in time linear to the number of basis vectors. The solution is computed for each of the R components of each feature vector. Thus the overall computational complexity is linear with respect to the number of vertices and quadratic with respect to the number of used basis vectors.

3.5. RBF based predictor (RBFP)

The second predictor we are presenting is based on the Radial Basis Function (RBF) interpolation (see Duchon [Duc77] and Uhlíř and Skala [US06]). This approach becomes natural if we reinterpret the predictor described in the previous section as an interpolator. The predictor actually interpolates each unknown component of the feature vector (unknown variable), using the already known components, which can be seen as coordinates. The LS predictor of previous paragraph can be thus seen as an interpolator based on generalized barycentric coordinates. However, we have better concepts for interpolation.

The RBF is a tool for interpolating scattered data in a general dimension space. The interpolation is formed as a superposition of radial functions centred at the points of known values. Each radial function is multiplied by a weight λ_i, which is found so that the interpolation function passes through the known values. Additionally, there should be a polynomial function that improves the fitting.

For our purposes we will only be able to use zero order polynomial, i.e. a constant. Thus our interpolation function has the following form:

$$f(x) = \sum_{i=1}^{N} \lambda_i \phi(\|x - x_i\|) + a \tag{7}$$

where x_i are the locations of N known points, $\|.\|$ denotes Euclidean norm and $\phi(r)$ is some function (the function we have used will be discussed later). The values λ_i are unknown, and are determined from the values at the known points. The equation (7) should have the correct value y_i at the given points x_i, thus we get the system of linear equations 8.

$$\begin{aligned}
\lambda_1\phi(\|x_1 - x_1\|) + \cdots + \lambda_N\phi(\|x_1 - x_N\|) + a &= y_1 \\
\lambda_1\phi(\|x_2 - x_1\|) + \cdots + \lambda_N\phi(\|x_2 - x_N\|) + a &= y_2 \\
&\vdots \\
\lambda_1\phi(\|x_N - x_1\|) + \cdots + \lambda_N\phi(\|x_N - x_N\|) + a &= y_N
\end{aligned} \tag{8}$$

This gives us N equations for $N+1$ unknowns ($\lambda_{1..N}$ and a), thus we need to add one more equation to obtain a determined system. This equation usually takes the following form:

$$\lambda_1 + \lambda_2 + \cdots + \lambda_N = 0 \tag{9}$$

For exact derivation of this additional equation see [Duc77]. This gives us a determined system of linear equations (10), which has a symmetric matrix.

Now that we have described how the RBF interpolation works, we can apply it in a quite straightforward way on our case. The first component of the vector assigned to a vertex is estimated from its neighbors by simply averaging them, i.e. using the formula (4). The encoder simulates this prediction and sends a correction, and thus the first component is now known to the decoder.

We will now use RBF for the prediction of the second component. We will treat the first component of the feature vectors as spatial coordinate in a 1-dimensional space. The second component is known only for the neighboring vertices, and RBF is used to interpolate this value to the location of the added vertex. Again, the encoder simulates this prediction and sends a correction, so that the first two components of the vector become known to the decoder.

The subsequent steps are a straightforward repetition of the last step. The known K components of the transmitted feature vector are treated as spatial coordinates, and the first unknown component (of index $K+1$) is treated as a value which is interpolated using the RBF. The dimension of the interpolation space increases, but the size of the system of linear equations remains unchanged. We construct and solve the system of linear equations (11), and the solution is used to compute the interpolant described by equation (12).

$$\begin{pmatrix} \phi(\|x_1-x_1\|) & \phi(\|x_1-x_2\|) & \dots & \phi(\|x_1-x_N\|) & 1 \\ \phi(\|x_2-x_1\|) & \phi(\|x_2-x_2\|) & \dots & \phi(\|x_2-x_N\|) & 1 \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ \phi(\|x_N-x_1\|) & \phi(\|x_N-x_2\|) & \dots & \phi(\|x_N-x_N\|) & 1 \\ 1 & 1 & \dots & 1 & 0 \end{pmatrix} \begin{pmatrix} \lambda_1 \\ \lambda_2 \\ \vdots \\ \lambda_N \\ a \end{pmatrix} = \begin{pmatrix} y_1 \\ y_2 \\ \vdots \\ y_N \\ 0 \end{pmatrix} \tag{10}$$

$$\begin{aligned} \sum_{j=1}^{N(i)} \lambda_j \phi\left(\sqrt{\sum_{k=1}^{K}\left(\overline{c^k_{n(i,1)}} - \overline{c^k_{n(i,j)}}\right)^2}\right) + a &= \overline{c^{K+1}_{n(i,1)}} \\ \sum_{j=1}^{N(i)} \lambda_j \phi\left(\sqrt{\sum_{k=1}^{K}\left(\overline{c^k_{n(i,2)}} - \overline{c^k_{n(i,j)}}\right)^2}\right) + a &= \overline{c^{K+1}_{n(i,2)}} \\ &\vdots \\ \sum_{j=1}^{N(i)} \lambda_j \phi\left(\sqrt{\sum_{k=1}^{K}\left(\overline{c^k_{n(i,N(i))}} - \overline{c^k_{n(i,j)}}\right)^2}\right) + a &= \overline{c^{K+1}_{n(i,N(i))}} \\ \sum_{j=1}^{N(i)} \lambda_j = 0 \end{aligned} \tag{11}$$

$$pred(c_i^{K+1}) = \sum_{j=1}^{N(i)} \lambda_j \phi\left(\sqrt{\sum_{k=1}^{K}\left(\overline{c^k_i} - \overline{c^k_{n(i,j)}}\right)^2}\right) + a \tag{12}$$

4. Basis function selection

One question that remains to be answered is the choice of the function $\phi(r)$. The RBF theory does not offer any ultimate function that would give the best results in any situation. Generally any function can be used, and the efficiency of the functions is difficult to predict.

We have performed experiments with various functions suggested by the RBF literature, such as the thin plate spline (TPS) of form $\phi(r) = r^2 log(r)$ or compactly supported functions such as $\phi(r) = e^{-sr^2}$, however, the most accurate predictions have been obtained by using power function

$$\phi(r) = r^{\beta}, \tag{13}$$

where the value of β is little less than 2, usually about $1.7-1.9$.

We do not have a derivation of the β parameter value, and it varies slightly depending on the dataset used, however, it can be seen that using $\beta = 1, \phi(r) = r^1 = r$ is not a good choice, because this function has a non-zero derivative for zero argument, i.e. the radial function is not smooth. A simple function with zero derivative at zero is obtained for $\beta = 2, \phi(r) = r^2$, unfortunately this function cannot be used, because it makes the system of equations (10) degenerate. Therefore, a function $\phi(r) = r^{1.8}$ is a compromise which produces a solvable system of equations, while the derivative at zero is zero and the shape of such function is close enough to the shape of $\phi(r) = r^2$.

Also note that the prediction algorithm is used not only in the reconstruction step, but also in the simplification step, where the prediction error is used as decimation cost. This leads to a slightly different decimation strategy. In the case of neighbourhood average, the algorithm removed vertices that were close to the centre of their neighbourhood, while our predictors prefer vertices that lie in the plane of their neighbourhood, even when they are slightly off-centre. In other words, the simplification is now more precise in removing vertices that carry little geometrical information. This effect is positive on one hand, because the coarse version of the mesh which can be extracted from the bitstream already contains much of the geometrical detail. On the other hand, this effect causes an increase in the entropy of extrapolation residuals.

In contrast to the LSP, the RBF based predictor cannot be used for extrapolation because the RBF generally performs very poorly in extrapolation tasks. The absolute value of the extrapolated data quickly grows with growing distance from the known points, and this effect cannot be reliably controlled. Therefore we use the LSP extrapolation to transmit the coarse version of the mesh, and subsequently we use the RBFP interpolation to add the previously removed vertices.

The RBF predictor involves constructing a matrix of size $N(i) \times N(i)$ in each step, where each element of the matrix is an Euclidean distance in a R-dimensional space. This matrix can be constructed incrementally, adding one term of the Euclidean distance in each step. Therefore the computational complexity is linear with respect to the number of basis vectors and linear with respect to the number of vertices.

5. Discussion of the performance of the new predictors

When evaluating the expected performance of the proposed predictors with respect to the original NAVG prediction, we may do the following reasoning for both LSP and RBFP: For each predictor there is a particular input (*ideal input*), which is predicted perfectly (with zero residuals) by the predictor algorithm. Other inputs are predicted imperfectly, and the

performance is closely related to the distance of the given input to the *ideal input*.

For the NAVG predictor, the *ideal input* is a vector which is the arithmetic average of the neighbourhood upon which the prediction is performed. This input is also an *ideal input* for the LSP (each neighbour will be assigned equal weight and the predictor will predict the values perfectly) and for the RBFP. However, apart form the initialisation stage, there are other *ideal inputs* for both LSP and RBFP. For example, for each input which can be expressed as linear combination of neighbours LSP will produce zero residuals.

This argument is difficult to completely formalise due to the complexity of the prediction algorithms. However, it is intuitive that a predictor with a single possible *ideal input* will be in most cases outperformed by predictors which have in fact infinite number of additional possible *ideal inputs*.

6. Testing

We have performed a series of experiments with the proposed predictors to test their efficiency. Although dynamic mesh compression has been studied for quite a while now, there is currently no consensus on which error measure should be used to evaluate distortion caused by compression. Therefore, we focus on a case when distortion is almost fixed and we compare the data rate results for the given distortion.

In order to show results comparable with known algorithms, we give the error measure values computed according to Karni and Gotsman [KG04] (KG-error). They suggest to reorder the data into matrices, where each column represents a frame of the animation. The original data will be reordered into matrix A, and the decoded version into A'. The error is then expressed as:

$$KG_{error} = 100 \cdot \frac{\|A - A'\|}{\|A - E(A)\|} \tag{14}$$

where $E(A)$ is an average matrix, i.e. a matrix where each X value has been replaced by average X value in the given column and similarly for Y and Z. We are using Frobenius norm and per-frame averages for the $E(A)$ matrix.

We are aware that KG error is not an ideal measure for mesh distortion, however we are using it because the results of competing algorithms have been measured by this error metric. Also note that there is currently no error metric shown to correlate with human perception of distortion, and most metrics (KG error, MSE, DA error, Hausdorff distance) correlate strongly with each other in the usual cases of compression error evaluation.

In our implementation we have used an implementation of arithmetic coding similar to the CABAC encoder used by Marpe *et al.* [MWS03]). We have used 80 basis vectors for the PCA step. Subsequently we have quantized the residuals with precision of 12 bits for the main diagonal length of the first frame of the animation, i.e. we have used quantization with step δ:

$$\delta = \frac{d}{2^{12}} \tag{15}$$

where d is the main diagonal length of the first frame bounding box.

We have used several datasets with similar results. Generally, the algorithm works better for highly detailed meshes (human jump: 15830 vertices, 222 frames, see Sand *et al.* [SMP03] and Anuar and Guskov [AG04]), where the simplification step removes larger amount of vertices. On the other hand, even for quite simple meshes, such as the chicken run sequence (3030 vertices, 400 frames), we achieve a significant improvement.

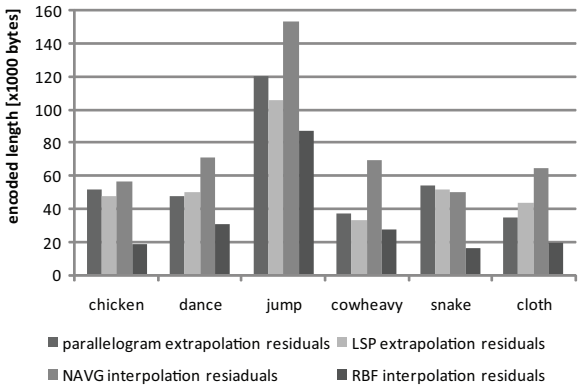

Figure 5: *Encoded sizes of prediction residuals.*

Figure 5 shows the encoded sizes of prediction residuals. The figure demonstrates that by using the RBF interpolation we can reduce the encoded length of interpolation residuals by 40-70%. The encoded size of extrapolation residuals remains almost unchanged, however, without employing our least squares predictor it would have actually grown due to a different simplification order introduced by the new RBF predictor.

The overall results of our algorithms are shown in figure 6. The benefit is relatively smaller compared to the 40-70% reduction in figure 5 because the final data stream also contains the compressed PCA basis, which is not affected by our improvement. Still, the results show that we have achieved an improvement of 7-25% of the bitrate. The figure also shows the result of the standard FAMC algorithm.

We are also including four full rate-distortion (RD) curves for the models dance, cloth, cow and snake (figures 7, 8, 9 and 10). The RD curves for other models we have tested show a similar behaviour, where original algorithm based on

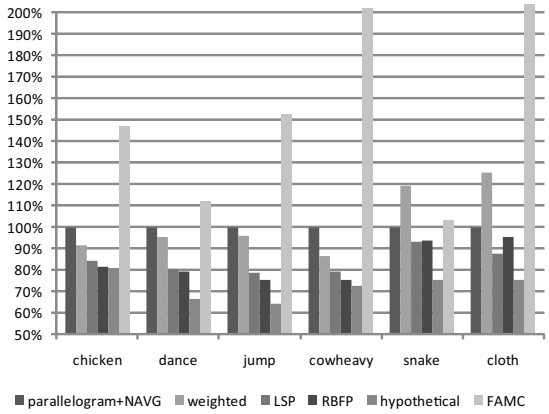

Figure 6: *The relative encoded sizes of the testing datasets. The PCA-based method with parallelogram extrapolation and neighbourhood average interpolation is taken as a reference, and the relative results of the proposed methods and of the standard FAMC method are shown.*

neighbourhood average prediction competes with the FAMC algorithm, but both are outperformed by the newly proposed RBF predictor.

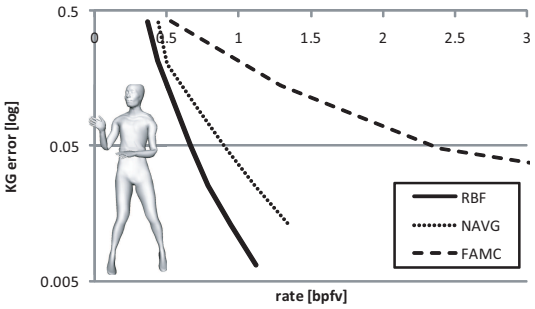

Figure 7: *Rate-distortion curve for the dance model.*

We also present the dependency of the residual entropy on the choice of the parameter β of the radial basis function. The results of this test for the human jump sequence are shown in figure 11, other sequences have produced equivalent results.

We have also tested the compression and decompression speed. The decompression consists of coefficient restoration, from which the original trajectories are reconstructed. Our version of the coefficient restoration algorithm is currently capable of achieving about real-time decompression for moderately complex mesh sequences. The chicken run sequence (cca 16 second animation) can be decompressed in about 4.5 seconds on a 2.53GHz Core2 Duo PC, however, the more detailed meshes take longer to decompress. On the other hand, the algorithm still leaves a quite large space for

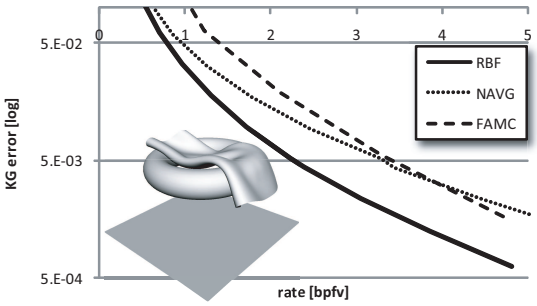

Figure 8: *Rate-distortion curve for the cloth model.*

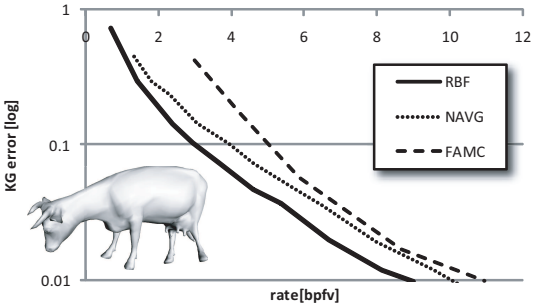

Figure 9: *Rate-distortion curve for the cow model.*

optimization. Additionally, the solution of the LSP equation set is likely to change only slightly with added components, and therefore the solution from previous step could be used in a following step to speed the computation up.

The compression takes at least the same time as decompression, as the compressor emulates the decompressors predictions in order to produce correct residuals. In addition to that the encoder must perform the PCA step, which usually takes about two minutes. Compression and decompression

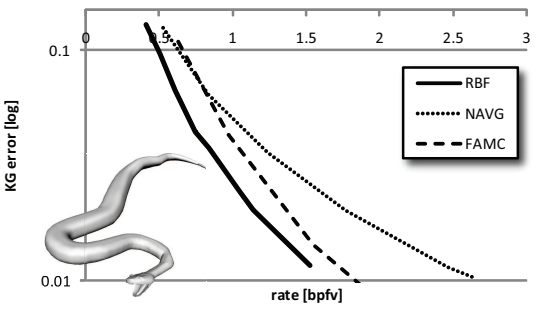

Figure 10: *Rate-distortion curve for the snake model.*

			Intel Pentium Core2 Duo (2.53GHz)				Intel Pentium i7 920 (2.67GHz)			
			NAVG (original)		RBF (proposed)		NAVG (original)		RBF (proposed)	
model	vertices	frames	Compress	Decompress	Compress	Decompress	Compress	Decompress	Compress	Decompress
chicken	3 030	400	64 030	3 268	72 088	4 524	64 225	2 761	69 826	3 214
dance	7 061	201	26 777	5 000	48 360	9 828	20 077	4 290	33 587	5 226
humanoid	7 646	154	19 032	4 774	39 671	7 768	14 071	4 321	28 330	5 086
jump	15 830	222	58 758	8 556	106 751	24 975	43 384	7 972	73 117	13 057
cloth	9 987	200	32 206	4 867	56 222	13 010	22 651	4 290	39 109	6 989
snake	9 179	134	17 129	4 906	47 143	9 984	13 213	4 586	33 119	6 022
walk	35 626	187	95 441	17 402	196 514	41 854	68 266	17 207	136 154	24 695

Table 2: *Compression and decompression times [ms].*

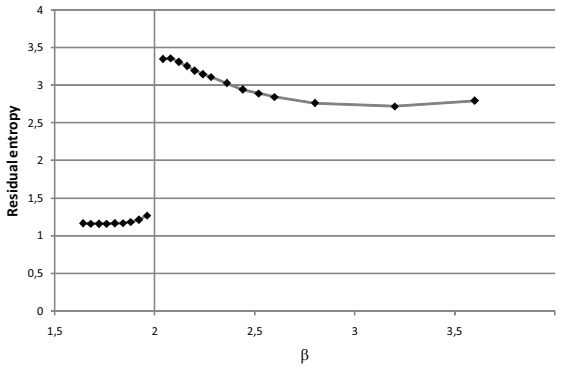

Figure 11: *Dependency of the residual entropy on the choice of parameter β (RBF exponent), see section 4 for additional details.*

times for several models using a mainstream and a high-end computer are shown in table 2.

7. Conclusions and future work

We have presented two novel predictors that can be used for dynamic mesh compression. Our main contribution is the **continuous update of the predictor being used, during the transmission of data related to a single vertex**. The update is not time dependant, i.e. it does not only use information from previous frame of the sequence. Instead the update continuously refines the knowledge about the vertex position relative to its neighbours. By using the framework of RBF interpolation, which naturally works in higher dimensional space, we exploit the character of the PCA represented animations, where vertices have many coordinates which describe their trajectory.

In our experiments we have achieved a reduction of residual entropy of over 40% and for the chicken run sequence we have reached more than 60% reduction. However, the overall efficiency gain depends also on the ratio of interpolated and extrapolated vertices, used entropy coding scheme and the size of the PCA basis. In our implementation, we have achieved bitrate reduction of about 20%.

In the future, we intend to test different basis functions for the RBF in order to find a better interpolation. The decoder could also benefit from the advanced methods of evaluating the RBF interpolation.

We would also like to implement and test the decompression algorithms using a programmable GPU. Our preliminary results show that we are able to perform the final multiplication of basis vectors and feature vectors on the GPU, thus saving both main memory and communication between CPU and GPU.

8. Acknowledgements

This work has been supported 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.

We would like to thank our colleagues, who provided useful hints and ideas during the development of this work, especially Petr Vaněček, who provided the insight to the RBF interpolation, and Martin Janda for his comments.

References

[AG04] ANUAR N., GUSKOV I.: Extracting animated meshes with adaptive motion estimation. In *VMV* (2004), pp. 63–71.

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

[Amj07] AMJOUN R.: Efficient compression of 3d dynamic mesh sequences. In *Journal of the WSCG* (Feb. 2007).

[AS07] AMJOUN R., STRASSER W.: Encoding animated meshes in local coordinates. In *CW '07: Proceedings of the 2007 International Conference on Cyberworlds* (Washington, DC, USA, 2007), IEEE Computer Society, pp. 437–446.

[BSM*03] BRICENO H., SANDER P., MCMILLAN L., GORTLER S., HOPPE H.: Geometry videos: A new representation for 3d animations. In *ACM Symposium on Computer Animation 2003* (2003).

[COLR99] COHEN-OR D., LEVIN D., REMEZ O.: Progressive compression of arbitrary triangular meshes. In *VIS '99: Proceedings of the conference on Visualization '99* (Los Alamitos, CA, USA, 1999), IEEE Computer Society Press, pp. 67–72.

[Duc77] DUCHON J.: Splines minimizing rotation-invariant seminorms in sobolev spaces. In *Constructive Theory of Functions of Several Variables*, Schempp W., Zeller K., (Eds.). Springer-Verlag, Berlin-Heidelberg, 1977, pp. 85–100.

[GGH02] GU X., GORTLER S. J., HOPPE H.: Geometry images. In *SIGGRAPH '02: Proceedings of the 29th annual conference on Computer graphics and interactive techniques* (New York, NY, USA, 2002), ACM Press, pp. 355–361.

[IA02] ISENBURG M., ALLIEZ P.: Compressing polygon mesh geometry with parallelogram prediction. In *VIS '02: Proceedings of the conference on Visualization '02* (Washington, DC, USA, 2002), IEEE Computer Society, pp. 141–146.

[IR03] IBARRIA L., ROSSIGNAC J.: 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* (Aire-la-Ville, Switzerland, Switzerland, 2003), Eurographics Association, pp. 126–135.

[KG04] KARNI Z., GOTSMAN C.: Compression of soft-body animation sequences. In *"Computers & Graphics 28, 1"* (2004), pp. 25–34.

[Len99] LENGYEL J. E.: Compression of time-dependent geometry. In *SI3D '99: Proceedings of the 1999 symposium on Interactive 3D graphics* (New York, NY, USA, 1999), ACM Press, pp. 89–95.

[MSK*05] MÜLLER K., SMOLIC A., KAUTZNER M., EISERT P., WIEGAND T.: Predictive compression of dynamic 3d meshes. In *ICIP05* (2005), pp. I: 621–624.

[MSK*06] MÜLLER K., SMOLIC A., KAUTZNER M., EISERT P., WIEGAND T.: Rate-distortion-optimized predictive compression of dynamic 3d mesh sequences. *SP:IC 21*, 9 (October 2006), 812–828.

[MWS03] MARPE D., WIEGAND T., SCHWARZ H.: Context-based adaptive binary arithmetic coding in the h.264/avc video compression standard. *IEEE Trans. Circuits Syst. Video Techn. 13*, 7 (2003), 620–636.

[MZP06] MAMOU K., ZAHARIA T., PRETEUX F.: A skinning approach for dynamic 3d mesh compression. *Comput. Animat. Virtual Worlds 17*, 3–4 (2006), 337–346.

[PA05] PAYAN F., ANTONINI M.: Wavelet-based compression of 3d mesh sequences. In *Proceedings of IEEE ACIDCA-ICMI'2005* (Tozeur, Tunisia, november 2005).

[Poy03] POYNTON C.: *Digital Video and HDTV Algorithms and Interfaces*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 2003.

[RMD68] REISZ K., MILLAR G., DICKINSON T.: *The technique of film editing*, 2nd enl. ed. ed. Focal Press, London, 1968.

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

[SLKO07] STEFANOSKI N., LIU X., KLIE P., OSTERMANN J.: Scalable linear predictive coding of time-consistent 3d mesh sequences. In *3DTV-CON, The True Vision - Capture, Transmission and Display of 3D Video* (Kos, Greece, May 2007), IEEE Computer Society.

[SMP03] SAND P., MCMILLAN L., POPOVIČ J.: Continuous capture of skin deformation. In *SIGGRAPH '03: ACM SIGGRAPH 2003 Papers* (New York, NY, USA, 2003), ACM Press, pp. 578–586.

[SO06] STEFANOSKI N., OSTERMANN J.: Connectivity-guided predictive compression of dynamic 3d meshes. In *Proc. of ICIP '06 - IEEE International Conference on Image Processing* (oct 2006).

[SSK05] SATTLER M., SARLETTE R., KLEIN R.: Simple and efficient compression of animation sequences. In *SCA '05: Proceedings of the 2005 ACM SIGGRAPH/Eurographics symposium on Computer animation* (2005), ACM Press, pp. 209–217.

[SZL92] SCHROEDER W. J., ZARGE J. A., LORENSEN W. E.: Decimation of triangle meshes. In *SIGGRAPH '92: Proceedings of the 19th annual conference on Computer graphics and interactive techniques* (New York, NY, USA, 1992), ACM Press, pp. 65–70.

[US06] UHLÍŘ K., SKALA V.: Radial basis function use for the restoration of damaged images. In *Computer vision and graphics* (2006), Springer, pp. 839–844.

[VS07] VÁŠA L., SKALA V.: Coddyac: Connectivity driven dynamic mesh compression. In *3DTV-CON, The True Vision - Capture, Transmission and Display of 3D Video* (Kos, Greece, May 2007), IEEE Computer Society.

[VS09a] VÁŠA L., SKALA V.: Cobra: Compression of the basis for the pca represented animations. *Computer Graphics Forum 28*, 6 (2009), 1529–1540.

[VS09b] VÁŠA L., SKALA V.: Combined compression and simplification of dynamic 3d meshes. *Comput. Animat. Virtual Worlds 20*, 4 (2009), 447–456.

[ZO04] ZHANG J., OWEN C. B.: Octree-based animated geometry compression. In *DCC '04: Proceedings of the Conference on Data Compression* (Washington, DC, USA, 2004), IEEE Computer Society, pp. 508–517.