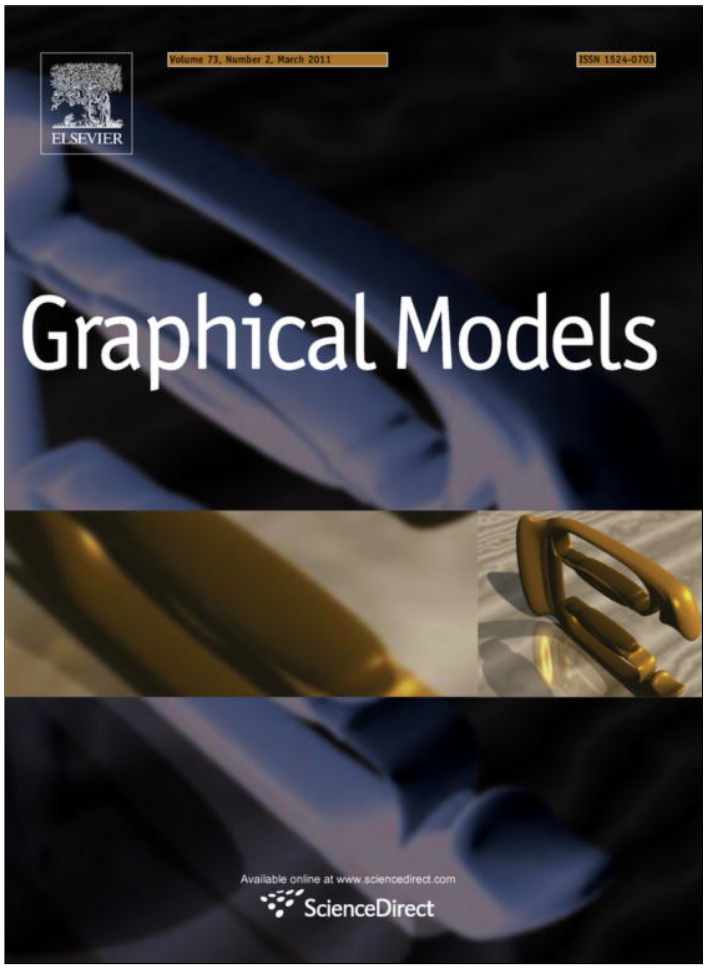

(This is a sample cover image for this issue. The actual cover is not yet available at this time.)

Graphical Models 73 (2011) 218–230

Contents lists available at ScienceDirect

Graphical Models

journal homepage: www.elsevier.com/locate/gmod

Optimised mesh traversal for dynamic mesh compression

Libor Váša

Department of Computer Science and Engineering, Faculty of Applied Sciences, University of West Bohemia, Univerzitní 8, Plzeň, CZ 306 14, Czech Republic

ARTICLE INFO

Article history:
Received 13 April 2010
Received in revised form 7 March 2011
Accepted 22 March 2011
Available online 1 April 2011

Keywords:
Dynamic mesh
Compression
Traversal
Optimisation
Animation
Distribution

ABSTRACT

During the last decade many algorithms for compressing 3D animations represented by sequences of triangular meshes have been proposed. Most of these algorithms are lossy in their nature, i.e. the reconstructed data do not exactly match the algorithm input.

Quite surprisingly, most of the existing algorithms mainly use only general compression techniques, such as entropy coding, quantisation, PCA or wavelet decomposition, while the inherent geometrical properties of the compressed surface remain unexploited. In this paper we focus on geometry specific optimisation: we extend the PCA-based dynamic mesh compression by optimising the order in which the mesh is traversed. By considering the distribution of residuals and optimising the gate selection strategy we achieve data rate reductions by 5.9–29.1% over the existing approaches in the experiments, while the error introduced by compression remains unchanged. This optimisation improves the performance of our encoder above the performance of current state of the art algorithms.

1. Introduction

Recent fast development of hardware has allowed processing of highly precise animated surface meshes. However, most representations of such data are highly demanding for both storage space and bandwidth for transmission. A very convenient representation of animated 3D surface model is the *dynamic mesh* – a series of static triangular meshes, which share a single connectivity, but each frame of the animation has a different geometry (vertex positions). In this representation we need three float or double values for each vertex in each frame, plus one additional set of indices identifying the shared connectivity. The connectivity compression has been thoroughly studied for the case of static meshes, and since it is shared by all the meshes in the sequence we can neglect it's influence on the overall data requirements for storage of the dynamic mesh.

This has motivated development of lossy geometry compression techniques, which can exploit inherent redundancies in the data. Usually the algorithms focus on the temporal coherence, which is much stronger than in the case of video data. In the case of temporal coherence we use the fact that position of a vertex in two subsequent frames is likely to change only slightly, which allows using temporal prediction in order to reduce the entropy of encoded data.

The data however also exhibit a different kind of redundancy, which has been already identified and exploited in the case of static triangular meshes: the spatial redundancy. In this case, we use the fact that the positions of neighbouring vertices (neighbouring in the sense of triangular connectivity) are likely to have similar positions.

This redundancy is in fact exploited by most dynamic mesh compression algorithms, however in most cases it is only exploited partially. The clustering-based algorithms usually build a local coordinate system in order to reduce the entropy of the data, other types of algorithms use PCA and wavelets to exploit the spatial coherence. However, one of the most efficient approaches used in static mesh encoding – the parallelogram prediction [1] – has only been used in few dynamic compression schemes.

Parallelogram prediction uses information about immediate neighbourhood, and therefore it usually outperforms the clustering-based algorithms, where prediction is based on cluster centre which is usually much further away. It

E-mail address: Lvasa@kiv.zcu.cz

1524-0703/$ - see front matter © 2011 Elsevier Inc. All rights reserved.
doi:10.1016/j.gmod.2011.03.005

also uses the information contained in the shared connectivity, which is often neglected by dynamic mesh compression algorithms. Efficiency of parallelogram prediction has been demonstrated for static meshes, however it can be still improved by accurately choosing prediction gates (prediction parallelograms). By changing the traversal order in a way similar to [2,3] it is possible to change the data available for prediction, thus making the prediction residuals smaller and residual entropy lower.

In this paper, we propose using optimised traversal strategy for dynamic mesh compression. Optimising mesh traversal order is in fact more appropriate for dynamic meshes, where the overhead of information needed to specify the traversal order is more easily outweighed by the benefit of reducing the entropy of geometry residuals, which take much larger part of the encoded data than in the case of static meshes.

Our contribution is a novel cost function for selecting the best gate, based on estimating the distribution of residuals. The proposed cost function minimises the residual magnitudes with nonuniform weighting, which leads to an improved performance of the encoding. Using the proposed cost function, we suggest several gate selection strategies and identify the best performing one based on their performance in experiments with all the available data sets. We also describe a simple but efficient method of encoding the traversal order selected by the encoder.

The rest of this paper is organised as follows: Section 2 overviews the related work in the area of dynamic mesh compression and traversal order optimisation. Section 3 will introduce notation used in the rest of the paper and Section 4 gives an overview of the trajectory-space PCA dynamic mesh compression, upon which we will build our extension. Section 5 will introduce a general approach to traversal order optimisation, and Sections 6 and 7 will describe in detail some important steps in the optimisation. Finally, Section 9 will describe the results obtained with the proposed approach and Section 10 will draw conclusions.

2. Related work

In recent years, a multitude of algorithms has been proposed for compression of dynamic meshes. The algorithms use various approaches to exploiting spatial and temporal coherence of the data.

First attempt to dynamic mesh compression has been published in the paper by Lengyel [4], which exploits spatial coherence of the data by subdividing the mesh into clusters in which the movement can be described by a single transformation matrix.

Ibarria and Rossignac [5] later suggested a local 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. [6] in the angle preserving predictor. The position of the new vertex is expressed in a local coordinate system defined by a neighbouring triangle.

Owen and Zhang [7] have proposed exploiting spatial coherence using an octree to subdivide the model and encoding each subdivision cell separately. This approach has been improved by Mueller et al. [8,9]. 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 exploiting temporal coherence in the work by Payan and Antonini [10], which 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 [11], which 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 in a paper by Karni and Gotsman [12], which 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. [13], which suggests 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 in a paper by Amjoun [14,15], which 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 quantisation of the basis matrices.

Mamou [16] 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 neighbouring clusters to obtain the movement of the vertex.

A resampling approach has been proposed by Briceno et al. [17] in the work on Geometry Videos. This idea is an extension of the previously proposed Geometry Images [18]. 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. [19].

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 2009, a second amendment of the MPEG-4 part 16 was published, specifying a new MPEG standard for dynamic mesh compression, the FAMC algorithm [20]. The standard is based on the algorithms of Mamou and Stefanoski, and it includes a specific arithmetic coder based on the CABAC scheme [21]. The algorithm has been shown to outperform all the algorithms available at the time of publication. Quite surprisingly, this algorithm almost neglects the information about connectivity, and apart from a clustering step, it processes the data as a point cloud rather than as a connected mesh.

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 mesh traversal (see Rossignac [22]) and parallelogram prediction [1]. The Coddyac algorithm [23] 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 [24]. 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 [25], which has boosted the performance of the algorithm so that it outperforms the FAMC standard.

Note that in contrast to Alexa et al. [11] 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 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. [26]), 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 min instead of hours needed for the shape space PCA.

We should also note two works on static mesh encoding, where traversal optimisation has been used to improve compression rates. Kronrod and Gotsman [2] have suggested interpreting the task as a minimum spanning tree (MST) problem, however the specifics of the problem do not allow using MST algorithms to find a global solution. In the end, the authors propose an algorithm which can be interpreted as a greedy growth of a processed region, using a cost function defined by lengths of prediction residuals. Later Chen et al. [3] have proposed a modification of this approach, however it is again a greedy strategy which grows a processed region of the mesh by one triangle at the time.

In this paper, we propose a modification of the greedy strategy for traversal optimisation, applied on dynamic mesh encoding. We build on the trajectory-space PCA based method, and we enhance the approach by including traversal optimisation similar to the one proposed by Kronrod and Gotsman, however using an improved cost function and a different gate selection strategy.

3. Used notation

Throughout the rest of the paper we will use following symbols:

F – number of frames in the input animation sequence
V – number of vertices in each frame of the input animation sequence
N – dimension of the reduced trajectory space after applying PCA. Usually set by user, $N \ll 3F$
E_i – ith basis vector of the PCA space, in order of importance (i.e. in order of descending eigenvalue)
$\bar{\xi}$ – denotes the decoded value of ξ (vertex coordinate, component of feature vector, matrix of coordinates, etc.), which may be different from the original value of ξ due to quantisation and other sources of information loss.
$pred(\xi)$ – denotes prediction of the value ξ, which is computed by both encoder and decoder
gate g – information about a potential expansion of the processed part of the mesh through the edge $\langle g.v_1, g.v_2\rangle$. A gate (depicted in Fig. 1) is formed by two triangles: $g.t_1$, which has been already processed, and $g.t_2$, which might not have been processed. The triangle $g.t_1$ has vertices $\langle g.v_3, g.v_1, g.v_2\rangle$ and the triangle $g.t_2$ is formed by vertices $\langle g.v_4, g.v_2, g.v_1\rangle$. The vertices $\langle g.v_1, g.v_4, g.v_2, g.v_3\rangle$ form a quadrilateral which might be used to predict the data assigned with vertex $g.v_4$ from data assigned with vertices $g.v_1$, $g.v_2$ and $g.v_3$.

4. Trajectory space PCA algorithm overview

In this paper we will examine the influence of optimised mesh traversal on the trajectory-space PCA algorithm Coddyac [23], which we will now describe in more detail. The algorithm is based on representing dynamic meshes as a set of vertex trajectories of individual vertices. Trajectory of the ith vertex is described by a vector T_i of

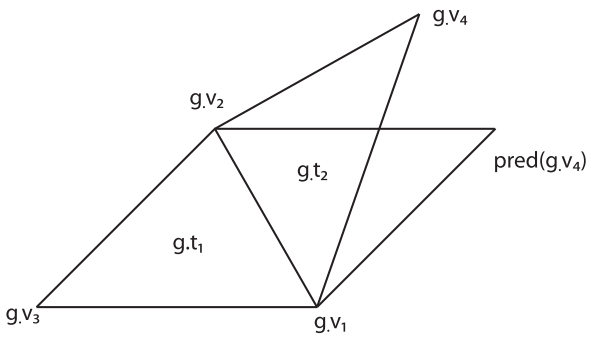

Fig. 1. Vertices and triangles forming a gate structure.

length $3F$, consisting of XYZ coordinates of the given vertex in all the frames. Notice that for dense meshes, it is very likely that trajectory vectors of neighbouring vertices will be similar. In other words, the trajectory vectors are not distributed evenly in the space of dimension $3F$, instead they are roughly located in a subspace of much **lower dimension**. This observation yields the first step of the Coddyac algorithm: finding the subspace and expressing the vertices in this subspace.

A straightforward way to find a subspace of a set of samples is using the PCA tool of linear algebra. We represent the original animation by a matrix B of size $3F \times V$, where the ith column is the trajectory vector associated with the ith vertex. First, we compute an average trajectory vector A (over all vertices), and subtract it from each column of B, obtaining a matrix of samples S. Subsequently, we compute the autocorrelation matrix $Q = S.S^T$ of size $3F \times 3F$. Finally, the eigenvalue decomposition of the autocorrelation matrix Q gives us a set of eigenvectors E_i, $i = 1 \ldots 3F$, and their corresponding eigenvalues.

These eigenvectors form a so-called **decorrelated basis** of the original space, i.e. expressing the original trajectory vectors in this basis reduces significantly the redundancy present in the data. Out of these eigenvectors we select N most important ones (according to their respective eigenvalues), N being a user-specified parameter. The selected eigenvectors form a basis of the subspace, and each trajectory vector can be expressed as:

$$T_i = A + \sum_{j=1}^{N} c_i^j E_j \tag{1}$$

Since the basis is orthonormal it is possible to compute the matrix of combination coefficients c_i^j by matrix multiplication $C = S^T E$, where E is a matrix of size $3F \times N$ in which the ith column is the ith eigenvector E_i. The vector of coefficients $c_i^j, j = 1 \ldots N$, corresponding to the ith vertex (which is a particular row of matrix C), is known as the *feature vector* of the vertex i. Note that the feature vector can be seen as transformed trajectory vector – it describes the trajectory of a vertex, however, its components no longer relate to particular frames or axes. Instead, it determines how to combine the eigen-trajectories in order to obtain the trajectory of a particular vertex.

In order to transmit the dynamic mesh we have to transmit the selected subset of eigenvectors (matrix E of size $3F \times N$), the combination coefficients (matrix C of size $V \times N$) and the vector A representing the average trajectory. Details on how to efficiently encode the matrix of eigenvectors can be found in [25].

The other key observation of the Coddyac algorithm is that the PCA step can be interpreted as a simple change of basis, and therefore it should not have any influence on results of linear operators. This feature is employed for prediction of the values c_i^j at the decoder. In static mesh encoding, a very common prediction method is based on the parallelogram rule [1,27]. The idea is that the mesh is traversed progressively by growing an area of processed vertices by adding one adjacent triangle (with one adjacent vertex) at a time. The expansion is always performed through a gate g, which is selected by the used topology compression/traversal scheme. The XYZ coordinates of the new vertex $g.v_4$ are predicted to lie at the top of a projected parallelogram formed by the three known vertices $g.v_1$, $g.v_2$ and $g.v_3$. The coordinate prediction is then expressed as:

$$\begin{aligned} pred(v^X_{g.v_4}) &= \overline{v^X_{g.v_1}} + \overline{v^X_{g.v_2}} - \overline{v^X_{g.v_3}} \\ pred(v^Y_{g.v_4}) &= \overline{v^Y_{g.v_1}} + \overline{v^Y_{g.v_2}} - \overline{v^Y_{g.v_3}} \\ pred(v^Z_{g.v_4}) &= \overline{v^Z_{g.v_1}} + \overline{v^Z_{g.v_2}} - \overline{v^Z_{g.v_3}} \end{aligned} \tag{2}$$

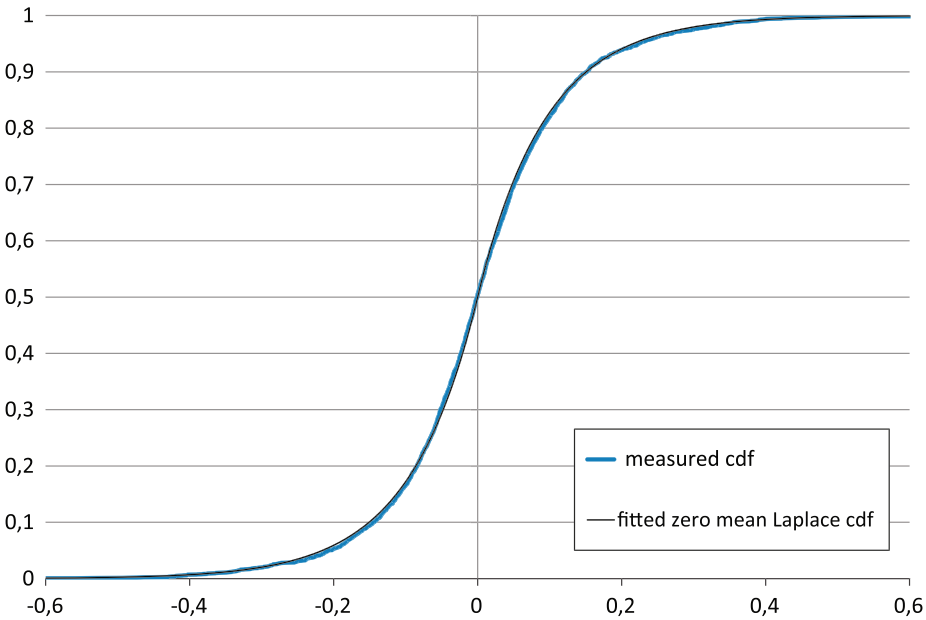

Fig. 2. Example of measured and fitted cumulative distribution function. The data represent the distribution of residuals of eighth basis vector of the chicken run data sequence.

In dynamic mesh compression, these formulae may be applied on each component[1] of the trajectory vectors. However, since the feature vectors are in fact linearly transformed trajectory vectors, it is possible to use the same formula also for the elements of feature vectors:

$$pred(c^j_{g.v_4}) = \overline{c^j_{g.v_1}} + \overline{c^j_{g.v_2}} - \overline{c^j_{g.v_3}}, \quad j = 1 \ldots N \tag{3}$$

The Coddyac algorithm traverses the mesh, adding one triangle at the time, performs the prediction according to Eq. (3) and transmits the quantised prediction residuals. Although any integer coding scheme can be used, it is reasonable to expect an exponential distribution with zero mean of the prediction residuals (see Fig. 2), and therefore it is efficient to use an arithmetic entropy coder together with exp-Golomb coding.

5. General approach

In the case of compressing static meshes, the number of possible gates is about six times larger than the number of vertices, and a traversal order selects one gate for prediction of each vertex. Therefore there is quite large freedom in choosing a particular traversal order.

In our approach, we start from the observations made for the static meshes. It is possible to find a globally optimal traversal order, but such problem can be interpreted as a special case of the Minimal Steiner Tree problem, which is known to be NP-Hard [2]. Therefore, we will use a greedy algorithm to traverse the mesh with a growing region of processed mesh, which is expanded in each step by one triangle, and we will focus on how to select the border edge (gate) for the next expansion step.

The first difference with respect to algorithms for static mesh encoding is that we will not use the mesh traversal to encode the connectivity of the mesh. We assume that the connectivity has been encoded using some efficient approach known for static meshes, and that it is known to both encoder and decoder. The traversal of the mesh is performed solely for the purpose of geometry encoding, and the description of the traversal order is considered a necessary overhead. Later in Section 8 we will show how to encode the traversal description with minimum data.

The general framework of the algorithm works in the following steps:

1. Assign cost $K(g)$ to each potential gate.
2. Select an unprocessed first triangle at random, add its edges into the initial set of gates G.
3. Select the best gate g_b from G.
4. Perform expansion of the processed area through g_b, including following steps:
   - if the vertex $g_b.v4$ has not been processed, then perform its prediction and encode residuals into the output stream,
   - remove g_b from G,
   - add to G any new gates created by expansion through g_b.
5. If G contains any more gates, then proceed with step 3.
6. If the mesh contains any unprocessed triangles, then proceed with step 2.

Note that in the first step, we evaluate all the possible gates of the mesh. The mesh of V vertices contains roughly $3V$ edges, and each edge can be passed by a potential gate in two directions. Therefore there are approximately $6V$ potential gates.

We will now discuss several possibilities of how to

- assign costs to gates,
- select the best gate,
- encode the final traversal order.

6. Assigning costs to gates

The previous approaches to traversal optimisation [2,3] have used sum of lengths of residuals as costs for potential expansion gates. Working with feature vectors, we can compute the cost of traversing through gate g as:

$$K_l(g) = \sum_{j=1}^{N} \| pred(c^j_{g.v_4}) - c^j_{g.v_4} \| \tag{4}$$

This approach, although intuitive, does not follow the actual purpose of traversal order optimisation. In fact, we should not attempt to reduce the sizes of residuals, instead, we should attempt to reduce the size of encoded residuals, which contributes to the length of the final code. Although closely linked, the two criteria are fundamentally different. Knowing a probability p of a symbol, we may express the (possibly fractional) number of bits required to encode it as:

$$D(p) = log_2\left(\frac{1}{p}\right) \tag{5}$$

The symbol might be for example a particular value of a quantised prediction residual. The number of bits required to encode the whole vector of residuals associated with a particular vertex is then expressed as a sum of contributions $D(p)$. This sum should then work as a cost for a particular gate producing the vector of quantised residuals.

The key observation is that the probability of a particular symbol is different for each component of the feature vector residual. The values at low index positions have much higher variance, and therefore higher magnitude residuals are much more probable at these positions than at positions with higher index, where the variance is lower.

For the following derivations we will assume that the residuals (not quantised) have a Laplace distribution with zero mean. Even though the Kolmogorov-Smirnov test shows that the distribution is not exactly Laplace, Fig. 2 shows that Laplace provides a quite good fit. Zero mean Laplace distribution has the following probability density function (pdf):

$$f(x, b) = \frac{1}{2b} exp\left(-\frac{x}{b}\right) \tag{6}$$

The only parameter of the pdf is the scaling parameter b. In order to exploit the differences in distributions, we pro-

[1] In the rest of the paper "component" always refers to PCA component, i.e. element of feature vector, never geometrical component.

pose to build a different statistical model for each component of the residual vectors, yielding N models with N different values of b.

The scaling parameter may be estimated for each component by first evaluating residuals for all the possible expansion gates. Having the total of M gates, we have M residual vectors r_i, $i = 1 \ldots M$, each of length N. From this data, we can estimate the variance v_j in each component of the residual vectors:

$$E_j = \frac{1}{M}\sum_{i=1}^{M} r_i^j \tag{7}$$

$$v_j = \frac{1}{M-1}\sum_{i=1}^{M}(r_i^j - E_j)^2 \tag{8}$$

From the variance we can then estimate the scaling parameter b_j, $j = 1 \ldots N$ for each of the N distributions:

$$b_j = \sqrt{\frac{v_j}{2}} \tag{9}$$

The parameter b_j fully specifies a zero-mean Laplace distribution and its pdf. Finally, in order to evaluate the number of bits required for a particular quantised residual value, we need its probability, which is computed by integrating the pdf over the quantisation interval. For a particular component j, quantisation step d and quantised residual $\hat{r}$ we get probability as:

$$p(j,d,\hat{r}) = \int_{\hat{r}*d-d/2}^{\hat{r}*d+d/2} \frac{1}{2b_j} exp\left(-\frac{x}{b_j}\right) dx \tag{10}$$

Finally, the expected required number of bits can be expressed as:

$$D(j,d,\hat{r}) = log_2\left(\frac{1}{p(j,d,\hat{r})}\right) \tag{11}$$

After some algebraic derivations, it can be shown that the cost can be expressed as:

$$D(j,d,\hat{r}) = \begin{cases} \alpha_j & \hat{r} = 0 \\ \beta_j + |\hat{r}| * \gamma_j & \hat{r} \neq 0 \end{cases} \tag{12}$$

$$\alpha_j = -log_2(1 - exp(-d/2b_j))$$

$$\beta_j = \frac{d}{2ln(2)b_j} - log_2(exp(d/b_j) - 1) + 1$$

$$\gamma_j = \frac{d}{ln(2)b_j}$$

Apart from the central value, the number of bits depends linearly on the magnitude of the quantised residual. If there was only a single distribution, then there would be no difference between optimising residual magnitude or the expected required bit count. However, in our case we have to assign a cost to a vector of quantised residuals, each from a different distribution (i.e. with a different scaling factor b_j). Therefore having a cost function based on estimated required number of bits from Eq. (12) allows us to assign a different (more precise) cost to residuals with respect to their position in the residual vector. Overall, the proposed cost function K_e is defined as:

$$K_e(g) = \sum_{j=1}^{N} D\left(j, d, quantise\left(pred\left(c_{g.v_4}^j\right) - c_{g.v_4}\right)\right) \tag{13}$$

6.1. Discussion

The proposed cost function we have described only uses an estimation of actual distributions of residuals. The main limiting factor is that we are using distribution of residuals of all the potential gates, but the gate selection strategy only uses a small subset of these gates (roughly one sixth), which will probably have different (narrower) distribution of residuals. In our experiments, the variance of actually used residuals has been about 10–15% lower than the variance of residuals of all the gates.

It is possible to compute the costs iteratively, using the distributions of previous step to estimate the required number of bits for the following step. We have implemented such iterative algorithm, however the benefit of that approach has been negligible in the experiments.

7. Gate selection strategy

Another key aspect of traversal order optimisation is the selection criterion used for selecting the next gate g from the set of open gates. It is necessary to take the gate cost $K(g)$ into account, however, there are more possible algorithms of how to select a gate based on its assigned cost.

Due to the proposed traversal strategy outlined in Section 5, it is possible that G contains gates where the top vertex $g.v_4$ has already been processed. All the selection strategies we have tested therefore first check the set G for any such gates and process them. During the processing of such gates no data are output into the geometry stream, however new gates may be added into the list of open gates G.

The simplest strategy, which has been used for the case of static mesh encoding, is selecting the gate of lowest cost $K(g)$ from the list of open gates G. This approach is equivalent to the selection strategies proposed by Kronrod and Gotsman [2] and Chen et al. [3]. This strategy S_1 can be expressed by following priorities:

1. If G contains a gate g where $g.v_4$ has been already encoded, then select g.
2. Select a gate g which has the smallest value of $K(g)$ of all gates in G.

Although intuitive, this strategy is not the only option and it can be significantly improved.

It may for example happen that a particular vertex v is difficult to predict from all possible directions, i.e. all the gates g where $g.v_4 = v$ have a relatively high cost. This vertex will, sooner or later, have to be processed, and the strategy S_1 attempts to postpone the processing of this vertex as much as possible, even in the case when it encounters a gate g_h which predicts v with the lowest possible cost. If such case appears, nothing can be lost in terms of global cost (the vertex $g_h.v_4$ will be processed at lowest

possible cost), however something might be gained, because by processing the gate g_h new gates with low cost might open. For an example of such situation see Fig. 3.

Addressing this issue requires computing a minimum cost for each vertex, and using it in the gate selection strategy. We denote $q(i)$ the set of all potential gates g where $g.v_4 = i$. For each vertex we can compute the minimum cost needed to predict it:

$$K_{min}(i) = \min_{g \in q(i)} K(g) \tag{14}$$

In the strategy S_2, we use following priorities for selecting the next gate from the set of open gates G:

1. If G contains a gate g where $g.v_4$ has been already encoded, then select g.
2. If G contains a gate g where $K(g) = K_{min}(g.v_4)$, then select g.
3. Select a gate g which has the smallest value of $K(g)$ of all gates in G.

This approach can be even generalised to following strategy S_3, which attempts to minimise the loss caused by selecting suboptimal gates:

1. If G contains a gate g where $g.v_4$ has been already encoded, then select g.
2. Select a gate g which has of all gates in G smallest value of $K(g) - K_{min}(g.v_4)$.

7.1. Discussion

It is also possible to extend the strategy S_2 differently. We might order the prediction gates in each $q(i)$ with respect to the assigned costs $K(g)$, $g \in q(i)$, and use this ordering as a selection criterion, choosing first the gates where no better prediction of $g.v_4$ exists, followed by gates where one better prediction exists, etc.

We have also implemented this approach, however we did not obtain any significant difference between the results of this approach and the selection strategy S_3.

8. Transmitting the traversal order description to the decoder

In the previous section we have described how gates are selected at the encoder, however in compression we need to know the traversal order at the decoder as well. In traditional approaches from static mesh encoding, the traversal order has been driven by topology encoding strategy (EdgeBreaker, valence-base or any other), and encoded in the data produced by the topology encoder (CLERS string, valence string, etc.).

In our approach, we assume that the topology has been transmitted already, and we only need to transmit the traversal order used for geometry data. One possibility is to encode the index of the selected gate g in the list of open gates G. This approach however requires sending $log_2(\|G\|)$ bits per vertex.

This overhead can however be significantly reduced. In our approach, we use the fact that it is not the traversal **order**, that influences the overall performance. Instead, it is the selection of **which gates are actually used** for prediction. In other words, if the decoder uses the same set of gates, but in a different order, it will reach the same result.

Therefore, all the information the decoder needs is which gates should be used for prediction (one bit per gate). We also have to ensure that the encoder saves the data into the output stream in the same order in which the decoder will read it, however that can be easily achieved by simulating the decoder in encoder and storing the data in the decoding order.

Having a mesh of V vertices, we can estimate that there will be $3V$ edges. In actual compression, each edge can be traversed in only one direction, thus we have $3V$ gates for which we have to encode whether or not the gate is actually used for prediction. Since V vertices have to be predicted, we can estimate the probability of *true* to 1/3 and the probability of *false* to 2/3. The entropy of the stream is therefore approximately 0.92 bits/symbol. The stream has a length of $3V$ symbols, thus we get 2.75 bits/vertex overhead, regardless of the total number of vertices.

Note that in the final data rate, which is usually expressed in bits per frame and vertex (bpfv), this overhead

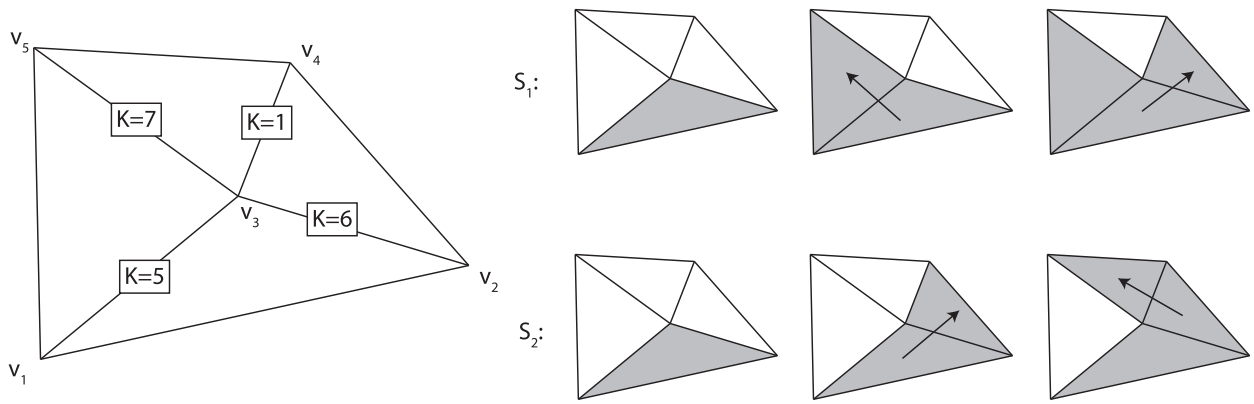

Fig. 3. Example of different behaviour of selection strategies S_1 and S_2. The process starts with triangle (v_1, v_2, v_3) known to the decoder. Strategy S_1 goes for the lowest cost gate and thus predicts position of v_5 at cost 5, and then predicts v_4 at cost 6, yielding the total cost of 11. The strategy S_2 on the other hand identifies that v_4 can be only predicted at cost 6 or 7, and thus there is nothing to loose by first predicting the vertex v_4 at cost 6, because it cannot be predicted at any lower cost. This in turn allows predicting v_5 at cost 1, yielding the total cost of 7.

is almost negligible. For an usual mesh sequence of 200 frames, we get an overhead of approximately 0.014 bpfv.

8.1. Discussion

Since the PCA coefficients are globally uncorrelated, it might be possible that a traversal order which is optimal for one component will be suboptimal for another. In fact our experiments have shown that this is the case for most datasets, and that we can achieve further reduction of data rate by using a different traversal order for each PCA component.

The problem with this approach is the overhead associated with encoding the traversal order. Transmitting data about a single traversal order has been shown to be equivalent to approximately 0.014 bpfv, however transmitting data describing N traversal orders of course requires transmitting N∗0.014 bpfv of additional overhead. In our experiments, this overhead has always exceeded the savings in data rates earned by optimising the traversal order for each component separately.

We have also experimented with various in between setups, such as optimising the traversal order for pairs or larger sets of PCA components. However, for all the datasets available, the gain of these optimisations has never exceeded the overhead of data required for describing more than one traversal order.

9. Experimental results

We have tested all the proposed cost functions and selection strategies and we have compared the results with the original approach, where mesh traversal is driven by the connectivity encoding scheme (EdgeBreaker in our case). Our implementation uses encoding of basis vectors described in [25].

We are evaluating bitrate R in bpfv (bits per frame and vertex), which is obtained from the total number M of bytes required to encode the dynamic mesh as:

$$R = \frac{8 * M}{F * V} \tag{15}$$

The error introduced by compression is evaluated according to [12] by computing:

$$KG_{error} = 100 * \frac{\|B - \overline{B}\|}{\|B - \widetilde{B}\|}[\%] \tag{16}$$

Table 1
Data rates for the main experiment.

Dataset	S_0		K_l			K_e			Best gain (%)
	KG error (%)	Rate (bpfv)	S_1	S_2	S_3	S_1	S_2	S_3	
			Rate (bpfv)						
Jump	0.496	0.554	0.537	0.533	0.535	0.528	0.521	0.525	5.9
Jump	0.244	1.095	1.020	1.011	1.019	1.012	1.001	1.008	8.6
Jump	0.100	2.351	2.173	2.154	2.179	2.155	2.134	2.151	9.3
Dance	0.498	0.292	0.276	0.270	0.270	0.273	0.266	0.267	9.0
Dance	0.008	1.258	1.160	1.128	1.130	1.149	1.118	1.117	11.2
Dance	0.001	2.014	1.905	1.863	1.865	1.898	1.857	1.858	7.8
Cow	0.242	2.169	1.965	1.938	1.952	1.950	1.912	1.924	11.8
Cow	0.068	4.760	4.340	4.278	4.318	4.261	4.206	4.235	11.7
Cow	0.012	9.172	8.540	8.447	8.515	8.382	8.296	8.358	9.6
Chicken	0.055	1.430	1.334	1.326	1.333	1.329	1.319	1.322	7.8
Chicken	0.017	2.045	1.922	1.912	1.919	1.913	1.900	1.907	7.1
Chicken	0.006	2.878	2.731	2.720	2.727	2.720	2.703	2.711	6.1
Humanoid	0.213	0.453	0.379	0.368	0.372	0.374	0.363	0.366	19.8
Humanoid	0.026	0.956	0.780	0.759	0.768	0.773	0.751	0.760	21.4
Humanoid	0.007	1.374	1.151	1.125	1.136	1.140	1.116	1.125	18.7
Camel	0.430	0.730	0.669	0.662	0.670	0.644	0.634	0.645	13.2
Camel	0.058	2.316	1.972	1.951	1.976	1.942	1.918	1.949	17.2
Camel	0.008	5.140	4.397	4.355	4.407	4.320	4.280	4.334	16.7
Elephant	0.279	0.704	0.696	0.677	0.678	0.662	0.643	0.652	8.7
Elephant	0.029	2.356	2.081	2.027	2.046	2.028	1.980	1.998	16.0
Elephant	0.005	4.408	3.899	3.807	3.837	3.772	3.688	3.733	16.3
Horse	0.325	1.275	1.136	1.089	1.090	1.118	1.070	1.076	16.1
Horse	0.138	2.108	1.841	1.764	1.768	1.817	1.737	1.748	17.6
Horse	0.018	4.972	4.318	4.150	4.160	4.233	4.063	4.090	18.3
Dolphin	0.289	0.316	0.248	0.233	0.234	0.246	0.230	0.231	27.4
Dolphin	0.034	1.135	0.865	0.816	0.815	0.860	0.804	0.805	29.1
Dolphin	0.006	3.109	2.444	2.308	2.310	2.394	2.207	2.212	29.0
Mocap	0.366	0.238	0.211	0.202	0.202	0.207	0.198	0.198	16.9
Mocap	0.029	0.896	0.789	0.757	0.758	0.770	0.741	0.746	17.3
Mocap	0.003	2.544	2.333	2.267	2.270	2.259	2.207	2.217	13.3
Walk	0.342	0.154	0.154	0.150	0.150	0.148	0.144	0.145	6.2
Walk	0.012	0.887	0.813	0.789	0.789	0.797	0.777	0.781	12.3
Walk	0.003	1.728	1.593	1.553	1.554	1.558	1.528	1.534	11.6
Gain (%)			10.4	12.6	12.2	11.9	14.2	13.6	

Table 2
Amounts of data required to encode the feature vectors.

Dataset	S_0	K_l			K_e			Residuals gain (%)
		S_1	S_2	S_3	S_1	S_2	S_3	
	C (MB)							
Jump	1.641	1.541	1.528	1.535	1.509	1.489	1.500	9.3
Jump	3.277	2.981	2.951	2.979	2.954	2.919	2.942	10.9
Jump	6.985	6.345	6.281	6.363	6.283	6.213	6.271	11.1
Dance	0.299	0.259	0.252	0.253	0.255	0.246	0.248	17.6
Dance	1.399	1.247	1.206	1.208	1.234	1.192	1.192	14.8
Dance	2.279	2.114	2.059	2.060	2.104	2.050	2.052	10.1
Cow	0.845	0.722	0.706	0.715	0.713	0.692	0.698	18.1
Cow	1.893	1.647	1.612	1.635	1.603	1.571	1.588	17.0
Cow	3.727	3.361	3.308	3.347	3.271	3.223	3.258	13.5
Chicken	1.047	0.928	0.920	0.927	0.923	0.911	0.915	13.0
Chicken	1.550	1.400	1.388	1.397	1.390	1.375	1.383	11.3
Chicken	2.164	1.988	1.974	1.982	1.975	1.955	1.964	9.7
Humanoid	0.461	0.356	0.344	0.349	0.352	0.339	0.343	26.3
Humanoid	0.978	0.759	0.736	0.745	0.751	0.727	0.737	25.6
Humanoid	1.412	1.140	1.112	1.124	1.129	1.102	1.111	21.9
Camel	0.691	0.570	0.564	0.571	0.545	0.535	0.546	22.5
Camel	2.222	1.818	1.797	1.823	1.788	1.765	1.795	20.6
Camel	4.991	4.187	4.146	4.198	4.110	4.071	4.125	18.4
Elephant	1.309	1.184	1.152	1.152	1.116	1.082	1.099	17.3
Elephant	4.442	3.802	3.702	3.737	3.699	3.607	3.642	18.8
Elephant	8.366	7.272	7.097	7.156	7.025	6.864	6.951	18.0
Horse	0.442	0.366	0.348	0.349	0.359	0.341	0.343	22.9
Horse	0.747	0.621	0.592	0.593	0.612	0.581	0.586	22.2
Horse	1.797	1.522	1.458	1.461	1.490	1.424	1.435	20.8
Dolphin	0.173	0.116	0.108	0.108	0.115	0.106	0.106	39.1
Dolphin	0.625	0.449	0.419	0.419	0.446	0.412	0.412	34.1
Dolphin	1.711	1.299	1.218	1.219	1.269	1.158	1.161	32.3
Mocap	0.745	0.606	0.576	0.578	0.595	0.562	0.563	24.6
Mocap	2.829	2.402	2.289	2.293	2.335	2.230	2.249	21.2
Mocap	7.938	7.133	6.897	6.910	6.871	6.681	6.719	15.8
Walk	0.868	0.781	0.760	0.760	0.746	0.721	0.725	16.9
Walk	5.149	4.589	4.439	4.441	4.492	4.368	4.390	15.2
Walk	10.057	9.107	8.859	8.866	8.890	8.697	8.735	13.5
Gain (%)		14.7	17.1	16.5	16.4	18.9	18.3	

where B is a matrix of original coordinates of all vertices in all frames, $\overline{B}$ is the matrix of decoded coordinates and $\widetilde{B}$ is a matrix of the same dimensions, where the values have been replaced by per-frame averages. Also note that the original definition of this metric does not define which matrix norm is used for the computation. We are using the Frobenius norm, which is generally used by other papers dealing with dynamic mesh compression.

This error metric has been chosen so that we can compare our results with competing algorithms. However, this norm suffers from a number of drawbacks, such as poor correlation with human perception of mesh distortion. In the future, we would like to test our approach using a more elaborate error metric, such as the STED metric [28].

Table 1 summarises our main experiment with 11 datasets,[2] each tested at low, medium and high data rates. We are only showing bitrates for each combination of cost function and gate selection strategy, because the difference in introduced error has been found negligible (average difference of 0.0116% for all the experiments, maximum difference of 0.173%). We denote the selection by connectivity encoder S_0, and we use it as a reference.

The table shows that we are achieving a reduction of data rate by 5.9–29.1% with respect to selection by connectivity encoder. The average data rate reduction is 14.2%. The cost function K_e outperforms the cost function K_l by up to 5.3% (1.92% on average), and in all cases K_e provides better results than K_l. Also the selection strategy S_2 clearly outperforms the selection strategy S_1, up to 7.8% reduction of data rate is achieved (2.65% on average).

Quite surprisingly, the most sophisticated selection strategy S_3 does not bring any further improvement, and in some cases it in fact degrades the performance. We have evaluated the sum of "losses" $K(g) - K_{min}(g.v_4)$, for both strategies S_2 and S_3. We have found that in fact the selection strategy S_2 produces results with slightly lower sum of losses, even though S_3 is directly focused on minimising it. Our explanation is that neither of the strategies guarantees to globally minimise its target, and the strategy S_2 wins by not making mistakes (i.e. accepting edges where no better prediction exists) and using a more courageous approach when the cost is low, as opposed to strategy S_3, which plainly "plays safe".

[2] most of the datasets can be downloaded from http://compression.kiv.zcu.cz.

Table 3
Timings for various settings and various datasets.

	S_0 (ms)	S_1				S_2			
		K_l		K_e		K_l		K_e	
		(ms)	(%)	(ms)	(%)	(ms)	(%)	(ms)	(%)
Dance	11,131	11,508	3.4	11,833	6.3	11,229	0.9	11,698	5.1
Cow	7658	7720	0.8	8731	14.0	7628	−0.4	8642	12.9
Jump	25,907	32,053	23.7	33,933	31.0	27,091	4.6	29,690	14.6

The Table 2 shows the results of the same experiment (i.e. with the same distortions as in Table 1), however this time only considering the amount of data needed to encode the feature vectors, i.e. without the other necessary data, such as connectivity information or description of the PCA basis. This table reflects more precisely the efficiency of the optimised traversal. The amount of data needed to encode the feature vectors is reduced by 19% on average.

We have also tested the amount of improvement achieved by the traversal optimisation in relation with the roughness of quantisation. The quantisation is usually expressed as a number of bits per volume diagonal, however in our experiments we have used a more illustrative parameter k based on average edge length. Having the average length of all edges in all frames of the animation $E(l)$, we obtain a quantisation step from the user-specified parameter k as:

$$Q = \frac{E(l)}{2^k}. \tag{17}$$

Using this definition, k can be interpreted as number of bits per average edge length, which gives a quite good idea on the effect of a particular value. Note that k can be negative, and that low values of k, such as 0 or 1, do not indicate a too rough quantisation, because the quantisation step Q is used to encode feature vectors, and therefore the introduced quantisation error is spread over all the frames of the animation. Figs. 7 and 8 show the improvement achieved by different versions of traversal optimisation on two datasets with respect to the quantisation parameter k. Note that together with quantisation parameter k we have also adjusted the number of used basis vectors N in order to obtain an optimal configuration for each measurement [29], and therefore the figures are also influenced by the varying number of basis vectors. This is however necessary, since using a constant number of basis vectors with varying quantization would produce meaningless results, suboptimal in all but one point.

The slowdown caused by increased complexity of mesh traversal is shown in Table 3. The table shows that using the most efficient combination of selection strategy S_2 and cost function K_e we obtain a slowdown of 5.1–14.6% for compression. The cost function K_e is generally slower, due to the more complex computation of estimated distributions. On the other hand, the selection strategy S_1 works in our implementation slower than S_2, because in S_1 we need to select a best gate from the whole list of open gates, whereas in S_2 a gate might be selected before the whole list has been processed. The strategy S_1 could be optimised by employing a better data structure, however for practical cases the more efficient strategy S_2 is a better choice, because its slowdown is relatively low (in one case it is in fact faster than the reference S_0, due to smaller residuals which are in turn processed faster by the arithmetic coder). The decompression times are only affected by the addition of decoding and applying the particular traversal order selected by the encoder, and therefore it is independent on the used selection strategy and cost function. In our experiments the slowdown of decompression has been negligible (<0.1% for all the tested cases).

It is difficult to compare the processing time with the FAMC algorithm, since its authors do not publish any timings. Our reimplementation of FAMC however works at least an order of magnitude slower than the times reported in Table 3. On the other hand, our method is about three

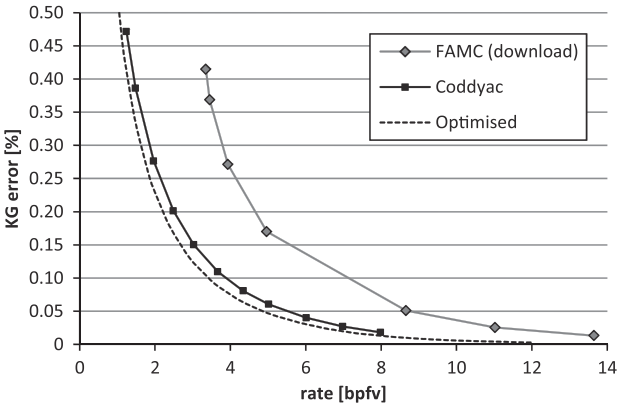

Fig. 4. Rate-distortion curve for the **cow** sequence.

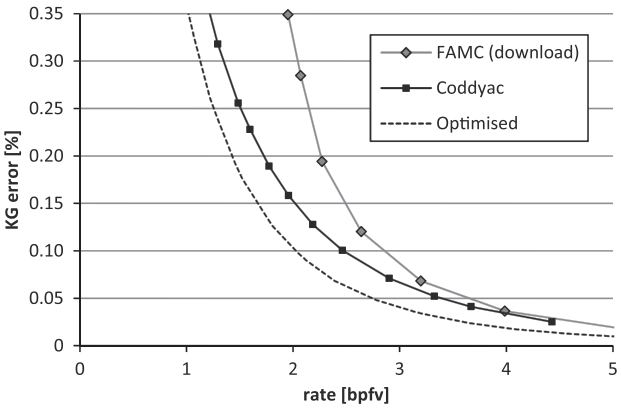

Fig. 5. Rate-distortion curve for the **horse gallop** sequence.

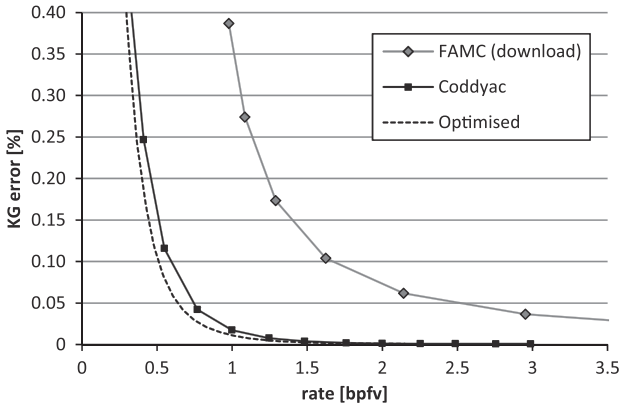

Fig. 6. Rate-distortion curve for the **dance** sequence.

times slower than the SPC algortihm [30], which however offers a much worse rate-distortion ratio.

Finally, we have compared the rate-distortion results with the state of the art algorithm, the MPEG-4 standard FAMC (we are comparing against the "download" version of the algorithm, which should provide the best achievable performance in the terms of rate-distortion ratio). Figs. 4–6 show that our algorithm outperforms the standard significantly. Also note that the results of the FAMC algorithm have been taken over from [30].

10. Conclusion

We have presented an extension of dynamic mesh encoding algorithm based on PCA in the space of trajectories.

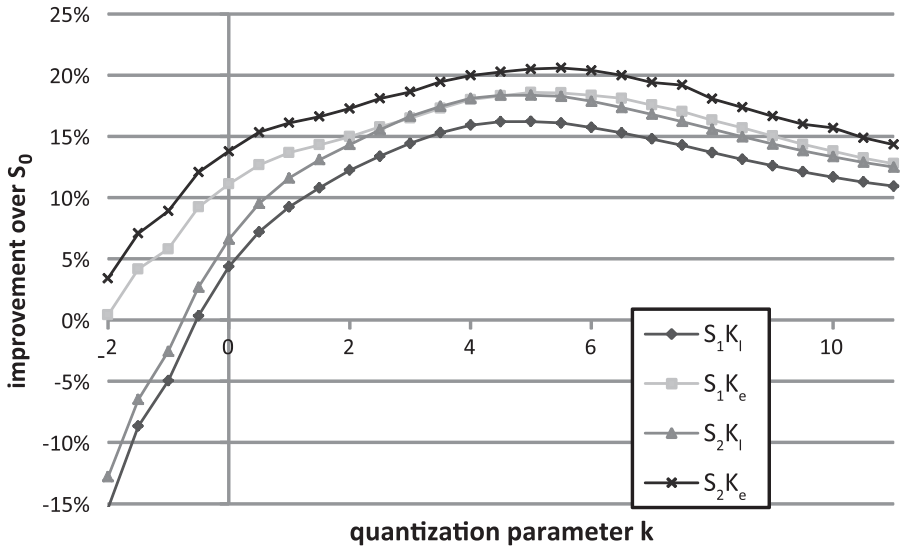

Fig. 7. Compression improvement with respect to quantisation for the Elephant dataset. Note that for low values of k traversal optimisation actually worsens the performance unless the cost function K_e is used. Similar behaviour has been observed with other datasets as well.

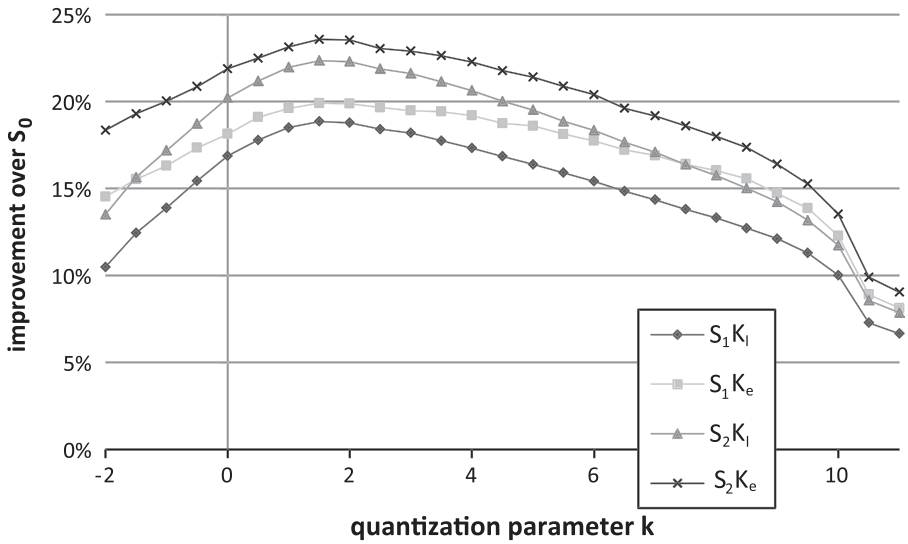

Fig. 8. Compression improvement with respect to quantisation for the Mocap dataset.

Our main contribution is showing that traversal order optimisation provides a significant reduction of required data. We have shown that region growing algorithm can be modified using various selection strategies and cost functions, and we have proposed an improved selection strategy and an improved cost function, that provide significantly better results than using the straightforward approach of selecting the gate with the smallest residuals.

Our improvement reduces the required data rate by 5.9–29.1% by reducing the amount of data required to encode feature vectors by 9.3–39.1%. Best results are obtained when processing irregular meshes, where each vertex can be predicted from multiple directions with varying accuracy. We expect that most meshes where storage/transmission data rate is an issue will have this property, since in such case the data will probably first undergo some kind of simplification process, which usually produces irregular meshes.

The application of the proposed optimisation is not limited to the example examined in this paper. Equivalent optimisation can be used for static mesh encoding, where we expect a good improvement using the proposed selection strategy S_2. Also dynamic mesh encoding using mesh simplification as a means to achieving scalability [24,19] can benefit from traversal order optimisation, especially because simplification usually reduces regularity of the triangles, which is the defining factor for the efficiency of the proposed optimisation.

In the future we would like to experiment with other cost functions, as well as other error metrics. We would also like to experiment with other, possibly non-symmetric predictors, because since our approach does not attempt to interpret the task as a minimum spanning tree problem, it is possible to work with non-symmetric predictors in a straightforward way.

Another interesting topic might be extending the traversal order optimisation to the multi-gate prediction schemes, such as the Dual Parellelogram Prediction scheme by Sim et al. [31], Average Parallelogram Prediction by Cohen-Or et al. [32] or their generalisation in the form of Taylor prediction by Courbet and Hudelot [33].

Acknowledgments

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. We would also like to express our gratitude to the anonymous reviewers, who have largely contributed to the improvement of this paper.

References

[1] C. Touma, C. Gotsman, Triangle mesh compression, in: Graphics Interface, 1998, pp. 26–34.

[2] B. Kronrod, C. Gotsman, Optimized triangle mesh compression using prediction trees, in: Proceedings of 8th Pacific Graphics 2000 Conference, 2000.

[3] D. Chen, Y.-J. Chiang, N. Memon, X. Wu, Optimized prediction for geometry compression of triangle meshes, in: DCC '05: Proceedings of the Data Compression Conference, IEEE Computer Society, Washington, DC, USA, 2005, pp. 83–92. doi:http://dx.doi.org/10.1109/DCC.2005.68.

[4] J.E. Lengyel, Compression of time-dependent geometry, in: SI3D '99: Proceedings of the 1999 Symposium on Interactive 3D Graphics, ACM Press, New York, NY, USA, 1999, pp. 89–95. doi:http://doi.acm.org/10.1145/300523.300533.

[5] L. Ibarria, J. 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, Eurographics Association, Aire-la-Ville, Switzerland, 2003, pp. 126–135.

[6] N. Stefanoski, J. Ostermann, Connectivity-guided predictive compression of dynamic 3d meshes, in: Proceedings of ICIP '06 – IEEE International Conference on Image Processing, 2006.

[7] J. Zhang, C.B. Owen, Octree-based animated geometry compression, in: DCC '04: Proceedings of the Conference on Data Compression, IEEE Computer Society, Washington, DC, USA, 2004, pp. 508–517.

[8] K. Müller, A. Smolic, M. Kautzner, P. Eisert, T. Wiegand, Rate-distortion-optimized predictive compression of dynamic 3d mesh sequences, SP:IC 21 (9) (2006) 812–828.

[9] K. Müller, A. Smolic, M. Kautzner, P. Eisert, T. Wiegand, Predictive compression of dynamic 3d meshes, in: ICIP05, 2005, pp. 621–624.

[10] F. Payan, M. Antonini, Wavelet-based compression of 3d mesh sequences, in: Proceedings of IEEE ACIDCA-ICMI'2005, Tozeur, Tunisia, 2005.

[11] M. Alexa, W. Müller, Representing animations by principal components, Computer Graphics Forum 19 (3) (2000) 411–426.

[12] Z. Karni, C. Gotsman, Compression of soft-body animation sequences, in: ''Computers & Graphics 28, 1'', 2004, pp. 25–34.

[13] M. Sattler, R. Sarlette, R. Klein, Simple and efficient compression of animation sequences, in: SCA '05: Proceedings of the 2005 ACM SIGGRAPH/Eurographics Symposium on Computer Animation, ACM Press, 2005, pp. 209–217.

[14] R. Amjoun, Efficient compression of 3d dynamic mesh sequences, Journal of the WSCG (2007).

[15] R. Amjoun, W. Straßer, Encoding animated meshes in local coordinates, in: CW '07: Proceedings of the 2007 International Conference on Cyberworlds, IEEE Computer Society, Washington, DC, USA, 2007, pp. 437–446. doi:http://dx.doi.org/10.1109/CW.2007.30.

[16] K. Mamou, T. Zaharia, F. Preteux, A skinning approach for dynamic 3d mesh compression, Computer Animation and Virtual Worlds 17 (3–4) (2006) 337–346. doi:http://dx.doi.org/10.1002/cav.v17:3/4.

[17] H. Briceno, P. Sander, L. McMillan, S. Gortler, H. Hoppe, Geometry videos: a new representation for 3d animations, in: ACM Symposium on Computer Animation 2003, 2003.

[18] X. Gu, S.J. Gortler, H. Hoppe, Geometry images, in: SIGGRAPH '02: Proceedings of the 29th Annual Conference on Computer Graphics and Interactive Techniques, ACM Press, New York, NY, USA, 2002, pp. 355–361. doi:http://doi.acm.org/10.1145/566570.566589.

[19] N. Stefanoski, X. Liu, P. Klie, J. Ostermann, Scalable linear predictive coding of time-consistent 3d mesh sequences, in: 3DTV-CON, The True Vision – Capture, Transmission and Display of 3D Video, IEEE Computer Society, Kos, Greece, 2009.

[20] MPEG4 part 16 AMD2: frame-based animated mesh compression, ISO/IEC JTC1/SC29/WG11, 2007.

[21] D. Marpe, T. Wiegand, H. Schwarz, Context-based adaptive binary arithmetic coding in the h.264/avc video compression standard, IEEE Transactions on Circuits Systems for Video Technology 13 (7) (2003) 620–636.

[22] J. Rossignac, Edgebreaker: connectivity compression for triangle meshes, IEEE Transactions on Visualization and Computer Graphics 5 (1) (1999) 47–61.

[23] L. Váša, V. Skala, Coddyac: connectivity driven dynamic mesh compression, in: 3DTV-CON, The True Vision - Capture, Transmission and Display of 3D Video, IEEE Computer Society, Kos, Greece, 2007.

[24] L. Váša, V. Skala, Combined compression and simplification of dynamic 3d meshes, Computer Animation and Virtual Worlds 20 (4) (2009) 447–456.

[25] L. Váša, V. Skala, Cobra: compression of the basis for the pca represented animations, Computer Graphics Forum 28 (6) (2009) 1529–1540.

[26] K. Reisz, G. Millar, T. Dickinson, The Technique of Film Editing, second ed., Focal Press, London, 1968.
[27] M. Čermák, V. Skala, Polygonization of implicit surfaces with sharp features by edge-spinning, Visual Computer 21 (4) (2005) 252–264.
[28] L. Váša, V. Skala, A perception correlated comparison method for dynamic meshes, IEEE Transactions on Visualization and Computer Graphics 17 (2) (2011) 220–230.
[29] O. Petřík, L. Váša, Finding optimal parameter configuration for a dynamic triangle mesh compressor, AMDO Lecture Notes on Computer Graphics, vol. 5098, Springer-Verlag, Berlin Heidelberg, 2010, pp. 31–42.
[30] N. Stefanoski, J. Ostermann, SPC: fast and efficient scalable predictive coding of animated meshes, Computer Graphics Forum 29 (1) (2009) 101–116.
[31] J. Sim, C. Kim, S. Lee, An efficient 3d mesh compression technique based on triangle fan structure 18 (1) (2003) 17–32.
[32] D. Cohen-Or, R. Cohen, R. Irony, Multi-way geometry encoding, Tech. rep., Tel Aviv University, 2002.
[33] C. Courbet, C. Hudelot, Taylor prediction for mesh geometry compression, Computer Graphics Forum 30 (1) (2010) 139–151.