Università della Svizzera italiana

Faculty of Informatics

USI Technical Report Series in Informatics

Exploring Compression in Edge Shape Space

Stefano Marras[1], Kai Hormann[1]

[1] Faculty of Informatics, Università della Svizzera italiana, Switzerland

Abstract

It often happens that the 3D representation of a scene or a shape exhibits some static or dynamic content and thus requires to store a large amount of geometric information. A number of techniques to reduce the size of such data have been proposed and most techniques are able to greatly reduce the amount of data by detecting the spatial and temporal relationship between the coordinates of nearby vertices. In this work we present an overview of compression techniques in edge space. We exploit redundancy in this space and propose different methods to encode and compress dynamic geometric data in terms of edges. Our results show that this shape space does not appear to be the best choice for compression, and we discuss the issues and provide some explanation for the poor performance.

Report Info

Published
October 2015

Number
USI-INF-TR-2015-4

Institution
Faculty of Informatics
Università della Svizzera italiana
Lugano, Switzerland

Online Access
www.inf.usi.ch/techreports

1 Introduction

Three-dimensional objects are widely used in a large number of fields and applications, from videogames to computer-aided design and manufacturing, scientific visualization, and entertainment. In most cases, it is sufficient to work on a single *static* shape, but in several applications the shape is characterized by a *motion* or *deformation* over time. The deformation may affect both the connectivity of the shape, which can change over time, and the geometry of the shape, that is, the vertices of the shape change their positions over time. Consequently, when dealing with a deforming *dynamic* shape, we have to take into account the connectivity information, the geometric information, and the temporal dimension in order to correctly store, visualize, and manipulate the whole animated sequence.

High quality models may consist of millions of triangles, which requires a huge amount of data to be stored, and may limit possible applications, such as transmission and visualization on low-capability devices. The solution for this problem is to design and apply *compression* techniques, exploiting the redundancy in the original geometric information and reducing the amount of data to be stored. The so-called *lossless* compression techniques are able to recover the original data without losing any information. However, a certain amount of distortion in the information is sometimes acceptable, therefore there is also a large number of *lossy* compression techniques that find a reasonable tradeoff between stored data and the loss of information.

In this work, we concentrate on lossy compression of both static and dynamic shapes. Particularly, instead of focussing on compression in the commonly used vertex space, we turn our attention to the *edge space* to find the redundancy that may lead to a reduced compression rate.

The rest of the paper is structured as follows. We first recall common techniques for static and dynamic mesh compression in Section 2.1 and give an overview of alternative spaces for mesh representations in Section 2.2. We then describe the motivation behind our approach in Section 3. In Section 4 we present different approaches for compression in the space of edge coordinates, point out their weaknesses and advantages, and evaluate the results, which turn out to be unsatisfactory. We therefore discuss the limitations of this approach in detail in Section 5.

2 Related work

2.1 Mesh compression

One of the main concepts in the field of compression is Shannon's *entropy*, which expresses the amount of information contained in a set of discrete symbols. It also provides a rough estimate of the average number of bits per element needed by the encoder to store the data. Given a random discrete variable X with possible values $x_1, \ldots, x_n$, its entropy $H(X)$ is computed as

$$H(X) = -\sum_{i=1}^{n} P(x_i) \log_2(P(x_i)),$$

where $P(x_i)$ is the probability of X to be equal to x_i. A low entropy implies that x does not contain much information and may be predicted and efficiently encoded. The challenge in data compression is to explore the data space in order to find the redundancy that allows to reduce the entropy and achieve a lower data rate.

Mesh compression aims at reducing the amount of space needed to store or transmit a static or dynamic shape. In case of a single static mesh, we have to compress information about the *connectivity* and the *geometry*. Usually, both parts are encoded and transmitted separately. Connectivity is usually encoded by performing a smart traversal of the mesh on the encoder side and recording some information such as vertex valence or edge traversal order. The decoder receives the information and recovers the exact connectivity of the mesh. A number of different techniques have been proposed over the years, from Edge-breaker [22] to valence-driven encoding [2]. We refer the reader to [20] for a detailed survey.

Geometry compression, on the other hand, is usually carried out separately. In most cases, efficient compression of geometric data relies on *prediction* and *correction*. First, the encoder attempts to predict the next value that is going to be encoded with the highest possible precision, usually exploiting the already encoded data. It then measures the *error* in the prediction and quantizes, encodes, and sends only the corrections to the decoder. The decoder performs the prediction in the same way as the encoder and adjusts the prediction using the correction provided by the encoder. The better the prediction, the smaller the error, which leads to a lower *entropy* and an efficient compression. Several prediction schemes have been proposed. Probably the most used geometry compression technique is the parallelogram predictor [28], but also other techniques such as Laplacian compression [24] or octree-based compression [21] are often employed.

In case of dynamic shapes, we also have to encode the temporal information, that is, how the geometry changes over time. In the common setup, the connectivity of the deforming shape is *fixed*, while its geometry *varies* in time. An animated sequence of deforming shapes is represented as a set of M *poses* (or *frames*) of the input shape. In principle, to compress the sequence, we can simply apply one of the common techniques for static mesh compression and encode each frame separately. However, the temporal coherence of the sequence can be exploited to improve the prediction of the subsequent frames, further reducing data entropy.

In the following, we assume that the fixed connectivity of all poses is compressed once, using one of the techniques for static mesh compression, and we focus on the encoding of the time-varying geometry. The geometric information can be compressed in different ways. Most techniques rely on the principle that close vertices are characterized by the same rigid motion over time. One can take advantage of this relationship to develop a predictor and encode the sequence. The MPEG-4 Animation Framework eXtension AFX [5, 10] predicts the position of a vertex v in frame k as a combination of the positions of its neighbours in frame k and the positions of both v and its neighbours in frame $k-1$. These techniques may be coupled with a segmentation that clusters the vertices characterized by similar trajectories [16], or using a spatial partitioning structure to cluster the vertices of the shape, carrying out the predictions on a per-cell basis [40, 41, 18, 19].

So far, the compression techniques based on *principal component analysis* (PCA) provide the best trade-off between compression rates and distortion in the reconstructed sequence. The first PCA-based approach, proposed by Alexa and Müller [1], which was later revised and extended in [11, 14], uses PCA to find a set of *key frames* which serve as a basis for the whole sequence. The other frames are then expressed as weighted linear combination of these key frames. Each key frame has the same size as the sequence frames. Even though storing only a few key frames plus the coefficients of the other frames significantly reduces the bit rate, further improvements are possible.

For example, the PCA technique may be combined with clustering methods [23, 3] or multi-resolution techniques [25]. The best way to achieve optimal compression rates is the Coddyac algorithm, proposed

by Váša and Skala in [33]. It applies PCA to the trajectories of the vertices to find a number of basis vectors and uses them to encode each frame of the sequence. The experiments show that these vectors belong to a space that has a much lower dimensionality than the vectors computed by Alexa and Müller [1]. The same technique is revised and improved in [35, 34] and used in combination with a purely combinatorial Laplace operator in [31] and with a geometric Laplace operator in [30].

2.2 Differential shape spaces

In the common setup, a three-dimensional shape is represented by the spatial positions of its vertices. We refer to that representation as the *vertex shape space* $\mathcal{S}_v$ whose dimension is $3V$ with V equal to the number of vertices. We can think of a shape as a point in $\mathbb{R}^{3V}$. Assuming fixed connectivity, a sequence of different poses of the same shape is simply a curve in $\mathbb{R}^{3V}$ which connects the points in the temporal order corresponding to the poses. However, performing tasks like interpolation between two or more poses in $\mathcal{S}_v$ is usually an unreliable process, resulting in artefacts on the interpolated shape. To overcome this issue, other shape spaces have been proposed.

Sumner and Popović [27] propose an alternative shape space based on the *deformation gradient*. Each face of the mesh is characterized in terms of a linear transformation with respect to a reference triangle. Since the translational part of the transformation is omitted, the representation of a mesh in this shape space $\mathcal{S}_f$ is invariant under translations and the space dimension is $9F$, where F is the number of faces. In order to recover the geometry of the vertices, a projection operator P_f is defined. It requires to solve a sparse linear system that can bre pre-factorized to speed up the computation. However, Kircher and Garland [13] observe that when combining deformation gradients, faces rotate along the shortest path in the embedding space, so interpolation between two or more poses can give unnatural, non-smooth deformations. Similar approaches, which are not affected by these artefacts, are presented in [15, 12, 4].

Winkler et al. [39] introduce the *edge shape space* $\mathcal{S}_e$, in which a mesh is represented in terms of *edge coordinates*. For each edge e_{ij} of the mesh, connecting vertices v_i and v_j, we store its length l_{ij} and the dihedral angle θ_{ij} between the adjacent faces. Since this representation is invariant with respect to both translation and rotation, we need to add six more values to correctly place the mesh in $\mathbb{R}^3$. As a consequence, the dimension of the mesh representation in $\mathcal{S}_e$ is $2E+6$ with E the number of edges, which means that the size of the representation is approximately twice as big as the standard representation. To pass from the representation in $\mathcal{S}_e$ to the equivalent representation in $\mathcal{S}_v$, we must solve a non-linear optimization problem, which can be tackled in a hierarchical fashion [39] or by employing a non-linear global minimization approach [6]. This space has proven to be particularly suitable for performing mesh interpolation or deformation in a natural and physically plausible way. Interpolation and deformation are performed in terms of edge lengths and dihedral angles, and the resulting shape in $\mathcal{S}_e$ is projected back to $\mathcal{S}_v$. In the rest of the paper, we refer to this representation as ELDA (edge length, dihedral angle).

Marras et al. [17] propose a hybrid approach that combines vertex and edge shape spaces. They first classify the parts of a shape as *rigid* or *deforming* and then describe the rigid parts by their vertex coordinates and the deforming parts in terms of edge coordinates. This representation is suitable for a large number of articulated shapes and requires to store approximately $4V+6$ elements per mesh.

3 Motivation

As already stated, static and dynamic shapes in three-dimensional space are usually represented by their spatial coordinates. Hence, the vertex space is the common choice for compression of geometric data, despite the fact that vertex coordinates do not immediately exhibit significant redundancy. However, as seen in the literature, it is not difficult to find redundancy in the vertex behaviour and develop efficient and precise spatial and temporal predictors. Most of the spatial predictors rely on *spatial coherence* and are able to predict the position of a vertex from the positions of its neighbours with reasonable precision, while temporal predictors usually exploit *temporal coherence* by investigating vertex trajectories to understand how the shape deforms and predicting the position of a vertex in the next frames.

However, as described in Section 2.2, there are a number of applications where the vertex representation may not be the best choice, such as interpolation and deformation, and alternative representations have been proposed. It seems reasonable to assume that the spatial and temporal coherence of the vertex representation may be characteristic in other spaces as well. For example, instead of exploiting spatial redundancy to predict

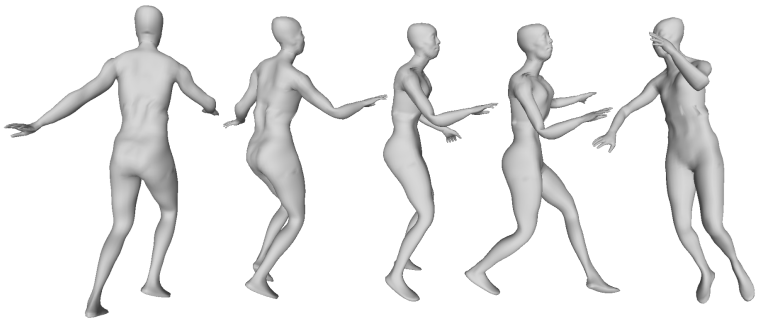

Figure 1: Five frames of the *Dance* sequence. The mesh is characterized by static connectivity and dynamic geometry.

the position of a vertex, one could predict the length of the edges that connect a face to an adjacent vertex using the same spatial coherence.

Relying on the same assumption as before, we propose to shift the focus and work in a different space. When dealing with dynamic meshes, focussing on edge coordinates seems to be an even more reasonable choice. It is well-known that the deformation of a shape can be described in terms of edge *stretching* and *bending* [7], which corresponds to the variation of lengths and dihedral angles, respectively. This is particularly true for *articulated* shapes. The deformation between consecutive frames of a sequence is usually localized in small regions of the shape and consequently only few edges are affected by variation, while the remaining edges are almost static. On the other hand, the same deformation may cause lots of vertex coordinates to change. Edge behaviour suggests that there is an even bigger redundancy in the temporal dimension to exploit for compression purposes.

However, working in edge space presents some major drawbacks. Particularly, the size of the shape space is approximately twice as big as the common vertex space, which is a significant complication in compression, where the aim is to save as many bits as possible. Thus, in order to be competitive, the edge shape space must exhibit an outstanding redundancy, allowing us to massively reduce the entropy of the edge data. The advantage of working in edge space is that a static or dynamic shape can be compressed in a format which has several nice properties, such as invariance under rotation and translation, and it is immediately suitable for interpolation and deformation purposes.

Our goal therefore is to present a set of different approaches to perform compression in edge space, and to give an immediate overview of different ways for tackling the problem by finding and exploiting the redundancy in this shape space.

4 Compression in edge shape space

Before going into detail, we first describe our setup and introduce the notation that is used in the rest of the paper.

Our input is a sequence of N poses $S = (\mathcal{M}_1, \ldots, \mathcal{M}_N)$ representing the same shape $\mathcal{M}$ at different times as pictured in Figure 1. Each pose shares the connectivity with the others, so we implicitly know the exact correspondence between the vertices of different frames and therefore do not need to perform any search for vertex matching. Each pose is described by a set of V vertices, E edges, and F faces. The connectivity of $\mathcal{M}$ is encoded once using one of the commonly used techniques for static mesh compression, such as the Edgebreaker [22]. The goal is to *encode* the original sequence by quantizing and compressing the data of its poses, such that the *decoded* sequence $\hat{S} = (\hat{\mathcal{M}}_1, \ldots, \hat{\mathcal{M}}_N)$ exhibits the lowest possible distortion. Since we focus on edge coordinates, we choose to represent each frame of the sequence in terms of edge lengths and dihedral angles. The whole sequence S is then represented by the two matrices

$$S_l = \begin{pmatrix} l_1^1 & \cdots & l_1^N \\ l_2^1 & \cdots & l_2^N \\ \vdots & \ddots & \vdots \\ l_E^1 & \cdots & l_E^N \end{pmatrix}, \quad S_\theta = \begin{pmatrix} \theta_1^1 & \cdots & \theta_1^N \\ \theta_2^1 & \cdots & \theta_2^N \\ \vdots & \ddots & \vdots \\ \theta_E^1 & \cdots & \theta_E^N \end{pmatrix}. \tag{1}$$

The matrix S_l contains the *lengths* of all edges $e_1, \ldots, e_E$ over the sequence, while S_θ contains the associated dihedral *angles*. The whole sequence is therefore represented by these two matrices. The i-th rows of S_l and

S_θ describe the behaviour of e_i over time, while the j-th columns of S_l and S_θ describe the frame $\mathcal{M}_j$ in $\mathcal{S}_e$. To avoid the ambiguity in spatial positioning, we introduce the matrix

$$S_{RT} = \begin{pmatrix} R^1 & \cdots & R^N \\ T^1 & \cdots & T^N \end{pmatrix},$$

which contains the rotations and translations needed to place the shape in $\mathbb{R}^3$. The total size of this representation is $N \times (2E+6)$. This is roughly equivalent to $N \times (6V+6)$ and about twice as big as the $N \times 3V$ matrix representing the same sequence in $\mathcal{S}_v$. Note that we refer to the edge connecting two vertices v_i and v_j as e_{ij} in the rest of the paper.

Since we observe that lengths, dihedral angles, and rigid transformations appear to be independent from each other, we treat each matrix separately from the other two. This allows us to handle lengths and angles independently, possibly applying different compression techniques to S_l and S_θ. Also, since the rotations and translations are described by only six values per frame, their cost is negligible, so we do not take S_{RT} into account in the evaluation. We assume to encode S_{RT} at high precision, such that the final positioning of the shape is precise enough and does not affect the error computation.

The cost of explicitly quantizing and encoding every entry of S_l and S_θ is obviously prohibitive, since each frame requires twice as much data than compressing the spatial coordinates of the vertices, and the entropy of theses values is not small enough to allow for an efficient compression. However, we can exploit both spatial and temporal coherence in the edge coordinates to significantly reduce the amount of data required to reconstruct the whole sequence. In the following sections we present several approaches and investigate their key ideas, advantages, and weaknesses.

In a generic compression pipeline, the encoder first analyses the sequence and then quantizes and encodes the geometry data *in some way*. Some techniques encode and decode the sequence frame by frame, while others process the whole sequence as a single entity. After the encoding step, the decoder receives the data and decodes them, recovering the geometric data of S_l and S_θ, which are usually affected by some distortion due to the quantization.

In our case, the input data are edge coordinates, so after decoding them we must switch from the representation in $\mathcal{S}_e$ to the one in $\mathcal{S}_v$. In our work, we rely on *multi-scale geometry interpolation* (MSGI), proposed by Winkler et al. [39], which provides a reasonably fast way to project from $\mathcal{S}_e$ into $\mathcal{S}_v$. We employ this technique to pass from the decoded edge coordinates to the corresponding shape in the vertex space. We also tested the global *discrete shell optimization* (DSO), proposed by Fröhlich et al. [6], but its high computational cost, combined with the fact that the results were not significantly different from the ones achieved by MSGI, persuaded us to focus on MSGI.

Input sequences were taken from Sumner's data set [27], Vlasic's data set [38] and the Aim@Shape repository. Compression rates are measured in bits per vertex [bpv] for static meshes and in bits per frame per vertex [bpfv] for dynamic shapes. The distortion of the decoded static mesh is measured in terms of the *dihedral angle metric error* (DAME) [32], while the error of the decoded sequences is measured in terms of *spatio-temporal edge distortion* (STED) [37], which evaluates the distortion in the decoded sequence as a weighted combination of spatial and temporal distortion in relative vertex positions. When possible, we employed the original tools implemented by their authors, such as the MSGI reconstruction tool or the STED measurement. The rest of the tools were implemented as C++ and MatLab routines.

In the rest of the paper, we use the following notation for predicted, quantized and decoded elements. The prediction of v is denoted by $\overline{v}$, $\tilde{v}$ represents the quantized and encoded value, and $\hat{v}$ represents v after the decoding process. Note that $\tilde{v}_i$ and $\hat{v}_i$ are different in case of lossy compression, due to the introduced compression error. A small difference between v and $\hat{v}$ usually results in a better approximation of the original data.

4.1 Parallelogram predictors

Traditional compression techniques work on vertex coordinates. In this setup, the compression is carried out by first predicting the position of the vertex v according to some prediction scheme and then measuring and encoding only the residual error in the prediction. In case of static compression, the prediction scheme usually involves a small set of already encoded neighbours $v_1, \ldots, v_k$, while dynamic compression can also involve the position of v and its neighbours in the previously encoded frames. Our goal is to develop one or more prediction schemes to encode the edge coordinates of a mesh by exploiting both spatial and temporal coherence. To find the redundancy in edge space, we design several *edge-oriented* predictors, whose goal is

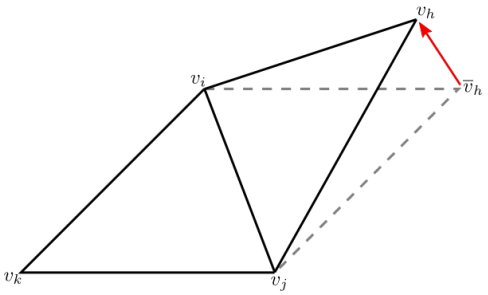

Figure 2: The parallelogram predictor is a simple but very efficient scheme that is widely used for both static and dynamic mesh compression. Due to the geometric nature of the scheme, the extension to predicting edge lengths is straightforward.

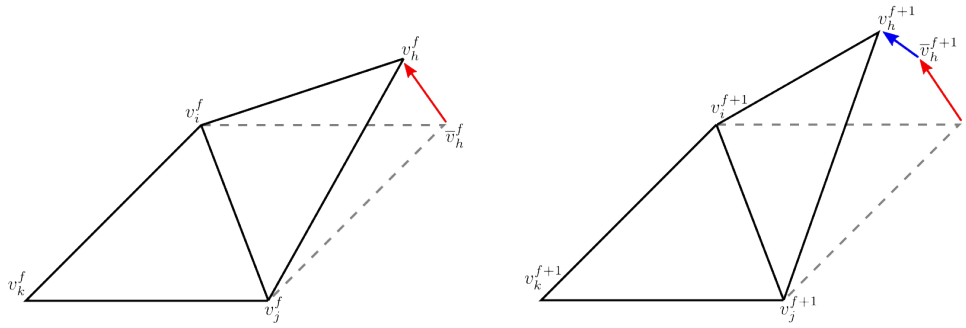

Figure 3: The parallelogram predictor can be extended to include temporal information. The error in the spatial prediction of v_h in frame f (left) is used to correct the prediction of v_h in frame $f+1$ (right). In the same way, the edge length prediction in frame $f+1$ can be corrected using the error in the f-th frame. We also assume that θ_{ij} does not vary significantly between consecutive frames and therefore use the angles predicted at f as prediction for $f+1$.

to predict the length and angle of the edge e according to its neighbours and its behaviour in the sequence. Mesh connectivity is encoded separately, using one of the standard technique mentioned in Section 2.1.

We initially focus on spatial coherence only, looking for meaningful relationships between locally related edges. We thus develop a spatial predictor that mimics the parallelogram predictor [28]. In the original prediction scheme (see Figure 2), the position of v_h is predicted as the fourth vertex of a parallelogram having the already decoded neighbours v_i, v_j and v_k as other vertices and only the residual error is then quantized and encoded as

$$\tilde{v}_h = v_h - (v_i + v_j - v_k).$$

This technique takes advantage of the local smoothness of the surface and has been largely employed and revised over the years, proving to be a reasonable choice in most cases. In fact, the predictor drastically reduces the entropy of vertex coordinates and provides a nice and efficient way to reduce the bit rate of the encoded mesh. Following the same principle and maintaining the notation of Figure 2, we can easily adapt the predictor to guess the length of the edges e_{ih} and e_{jh} from the opposite edges in the parallelogram, encoding only the residual errors for both edges as

$$\tilde{l}_{ih} = l_{ih} - l_{jk}, \qquad \tilde{l}_{jh} = l_{jh} - l_{ik}.$$

However, the parallelogram predictor does not allow to make any assumption about the dihedral angles. The only possible prediction is that θ_{ij} should be equal to zero, since the parallelogram lies on a plane. This prediction, however, does not affect the entropy of the original data and is therefore useless.

The parallelogram predictor is a purely spatial predictor, but it can be extended to take into account the temporal information [9, 5, 16], assuming that the motion of the mesh is *locally rigid*. The rationale is that the prediction error remains almost constant over time. Therefore the correction encoded for frame f can be employed to correct the prediction of the vertex positions in frame $f+1$ (see Figure 3) and to further reduce the entropy of the data. This predictor is sometimes referred to as the *extended* parallelogram predictor. In frame f, the vertex v_h^f is predicted using spatial information, with the residual error $v_h^f - \tilde{v}_h^f$.

	vertex space $\mathcal{S}_v$					edge space $\mathcal{S}_e$				
	size	H_V	$H_{\tilde{V}}$	$d(V,\hat{V})$	time	size	H_E	$H_{\tilde{E}}$	$d(E,\hat{E})$	time
Cow	3×2902	11.78	8.24	0.15	7	2×8700	10.66	10.91	0.15	890
Homer	3×5103	12.38	8.17	0.15	16	2×15303	10.82	10.72	0.15	1132
Horse	3×8431	12.91	7.44	0.08	23	2×25287	11.14	10.65	0.08	1477
Hippo	3×22260	11.48	6.56	0.08	40	2×66774	11.61	12.03	0.08	4089
Elephant	3×42321	14.14	6.39	0.02	74	2×126957	10.71	9.52	0.02	7672
Leaning	3×59727	14.84	7.60	0.09	108	2×179175	10.74	10.01	0.09	9701
Max-Planck	3×99636	13.78	7.47	0.07	190	2×298655	12.02	12.02	0.07	16782

Table 1: Comparison of static mesh compression with parallelogram prediction in vertex and edge space. In all examples the quantization deltas are chosen so that the DAME errors between the original and the reconstructed meshes are similar. While the parallelogram predictor significantly reduces the entropy in vertex space from H_V to $H_{\tilde{V}}$, the entropy reduction in edge space is negligible. The table further reports the sizes of the meshes in vertex and edge space and the times for decoding each mesh in milliseconds.

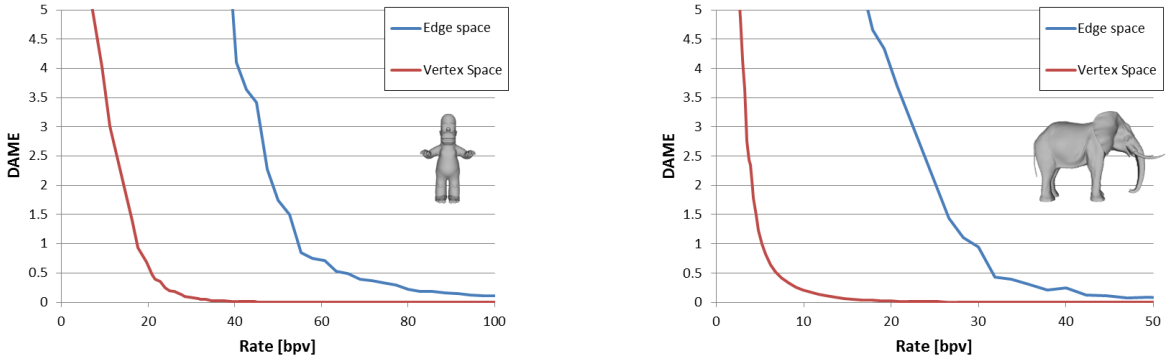

Figure 4: Rate-distortion plots for compression with parallelogram prediction in vertex and edge space, for the Homer (left) and the Elephant (right) static meshes.

When predicting the position of v_h in frame $f+1$, the spatial prediction in enhanced by the residual error of frame f. The encoder compresses only the error with respect to this prediction,

$$\tilde{v}_h^{f+1} = v_h^{f+1} - \left(v_i^{f+1} + v_j^{f+1} - v_k^{f+1} + (v_h^f - v_i^f - v_j^f + v_k^f)\right).$$

As done for static meshes, we revise the edge-oriented parallelogram predictor to include also the temporal information. The revised predictor corrects the prediction of the edge lengths similarly to the vertex-based predictor as

$$\tilde{l}_{ih}^{f+1} = l_{ih}^{f+1} - \left(l_{jk}^{f+1} + (l_{ih}^f - l_{jk}^f)\right),$$
$$\tilde{l}_{jh}^{f+1} = l_{jh}^{f+1} - \left(l_{ik}^{f+1} + (l_{jh}^f - l_{ik}^f)\right).$$

Following the hypothesis of local rigidness of motion, as done by Stefanoski et al. [26], we plug the temporal information into the predictor by assuming that dihedral angles do not change significantly between consecutive frames,

$$\tilde{\theta}_{ij}^{f+1} = \theta_{ij}^{f+1} - \theta_{ij}^f.$$

For each frame of the sequence, we encode the edge coordinates of a starting reference triangle plus the correction values for all other edges. We can therefore recover the whole set of edge lengths and dihedral angles by traversing the mesh. The final mesh is reconstructed using MSGI, which provides a mesh whose edge coordinates are as close as possible to the decoded values. For encoding the next frame we use length and angles of the reconstructed mesh, which may be different from the decoded data.

Table 1 presents a comparison between the impact of the spatial predictor on data entropy in $\mathcal{S}_v$ and $\mathcal{S}_e$ for a single static mesh. It is undeniable that the parallelogram predictor significantly reduces the entropy of vertex coordinates by 30–50%, while it does not have the same impact on edge coordinates. In this case the entropy is only marginally affected and, in some cases, not even reduced. Two examples of the resulting rate-distortion plots are shown in Figure 4. The parallelogram predictor proves again to be a good choice for

	frames	vertex space $\mathcal{S}_v$					edge space $\mathcal{S}_e$				
		frame size	H_V	$H_{\hat{V}}$	$d(V,\hat{V})$	time	frame size	H_E	$H_{\hat{E}}$	$d(E,\hat{E})$	time
Cow	204	3×2904	5.13	4.21	0.03	1.86	2×8706	7.66	5.01	0.03	183.2
Dance	201	3×7061	3.58	2.96	0.03	4.18	2×21177	6.41	4.16	0.03	209.2
Humanoid	154	3×7646	3.98	2.06	0.05	4.91	2×22932	9.05	6.09	0.05	161.5
Samba	175	3×9971	5.11	3.45	0.02	5.82	2×29907	8.47	6.08	0.02	262.4
March2	250	3×10002	5.42	3.74	0.02	8.74	2×30000	9.02	6.32	0.02	381.0
Jump	222	3×15830	4.61	3.53	0.01	11.20	2×47490	10.02	6.98	0.01	453.7

Table 2: Comparison of dynamic mesh compression with extended parallelogram prediction in vertex and edge space. The notation is the same as in Table 1, and the quantization deltas are chosen so that the STED errors between the original and reconstructed sequences are similar. The time for decoding the data is given in seconds.

RMSE = 0.0044, DAME = 14.35 RMSE = 0.2499, DAME = 22.97

Figure 5: Visual comparison between a mesh recovered from parallelogram prediction in vertex space (left) and in edge space (right) at similar compression rates. Uniform quantization in vertex space affects each vertex locally, causing "blocky" artefacts on the surface. Uniform quantization in edge space may have a completely different impact, since the error introduced in a small region can propagate to the adjacent ones, due to the hierarchical approach of the MSGI. As a consequence, the mesh reconstructed in edge space may be characterized by a much larger objective and perceived error.

mesh compression in the common vertex space, but it is not suitable for compression in edge space. The poor entropy reduction, combined with the size of the representation, do not allow to achieve competitive results.

In our experiments we developed a number of variations of the basic parallelogram approach, slightly revising the prediction in order to improve the precision. For example, since the length of one edge may get up to four different predictions (one for each side of the parallelogram), we designed an approach that predicts each edge length as the average of all the possible predictions. Another alternative predictor [29] estimates the interior angles of each face from vertex valences and then uses the interior angles to predict the lengths of the adjacent edges. However, none of these approaches gave any significant advantage in terms of entropy reduction.

Table 2 shows the effects of the spatial and temporal predictors for dynamic mesh sequences. In this case, there is a slight improvement in edge compression: exploiting the temporal coherence results in an entropy reduction of the edge data by 20–30% on average. However, this is not enough to reach a competitive compression rate, since the entropy in vertex shape space is still significantly lower. From the experiments, it seems reasonable to assume that the edge shape space shows a significant redundancy in terms of temporal information, while it does not seems to be characterized by a significant redundancy in spatial information.

Moreover, there are several issues related to the use of the MSGI to recover the original mesh from edge coordinates. The decoding step is quite expensive compared to the simple mesh traversal of the parallelogram predictor. As a consequence, the time required to decode the mesh or mesh sequence is considerably higher than the decoding time of the standard parallelogram predictor, which is a major drawback. In addition, the error introduced by uniform quantization affects vertices only locally, while the local error introduced by uniform quantization in edge space may propagate over the whole surface, as shown in Figure 5, due to the hierarchical nature of the MSGI approach.

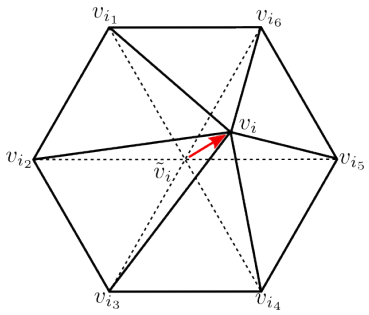

Figure 6: The δ-coordinates of v_i are computed as the difference between its position and the centre of mass $\tilde{v}_i$ of its 1-ring neighbours.

4.2 Laplacian encoding

Before putting aside the compression of static meshes in terms of edge coordinates, let us try to adapt another of the widely used techniques for compression in vertex space. In particular, we focus on *high-pass quantization*, proposed by Sorkine et al. [24]. In this scheme, a vertex is predicted as the centre of mass of its surrounding neighbours. For each vertex v_i we therefore compute

$$\delta_i = v_i - \frac{1}{d_i}\sum_{k=1}^{d} v_{i_k},$$

where d_i is the valence of v_i and v_{i_k} is the spatial position of the k-th neighbour of v_i (see Figure 6). We then quantize and encode this residual vector, which is also referred to as the *delta coordinates* of v_i. The computation of delta coordinates can be written in a more compact form as

$$\boldsymbol{\delta} = L\boldsymbol{v},$$

where $\boldsymbol{v}$ is the $V \times 3$ matrix containing the spatial coordinates of the mesh vertices, $\boldsymbol{\delta}$ is the $V \times 3$ matrix of the δ-coordinates, and L is the *combinatorial Laplacian* matrix, also known as the Kirchhoff Laplacian with entries

$$L_{ij} = \begin{cases} d_i, & \text{if } i = j, \\ -1, & \text{if } v_i \text{ and } v_j \text{ are neighbours,} \\ 0, & \text{otherwise.} \end{cases}$$

Since L is not invertible, the original positions of the vertices cannot be recovered from $\boldsymbol{\delta}$ only. To overcome this limitation, it is necessary to add one or more *anchor points*, that is, vertices whose spatial coordinates are explicitly stored. Having n anchors $a_1, \ldots, a_n$ the Laplacian matrix becomes

$$L^* = \begin{pmatrix} L \\ A \end{pmatrix},$$

where A is a matrix with one row for each anchor and entries $A_{ij} = 1$ if $i = j$ and $A_{ij} = 0$ otherwise. The δ-coordinates are then computed as $\boldsymbol{\delta} = L^*\boldsymbol{v}$. The entropy of the δ-coordinates is significantly smaller than the entropy of the spatial vertex coordinates, which makes them suitable for being efficiently quantized and encoded. From the quantized δ-coordinates $\tilde{\boldsymbol{\delta}}$ we get back the vertex positions by solving the minimization problem

$$\min_{\hat{\boldsymbol{v}}} \|L^*\hat{\boldsymbol{v}}\| - \tilde{\boldsymbol{\delta}}^2,$$

which can be solved efficiently in the least squares sense, taking advantage of the sparsity of L^*. The advantage of this approach is that the local details of the surface and the overall appearance of the shape are usually preserved, despite the quantization error, therefore reducing the perceived error evenly. This approach usually outperforms the parallelogram predictor in terms of perceived distortion, as confirmed by metrics such as the revised KG error [24] or the already mentioned DAME. However, if the number of anchor points is too small the result may be affected by a large objective error, usually measured as RMSE of vertex displacement.

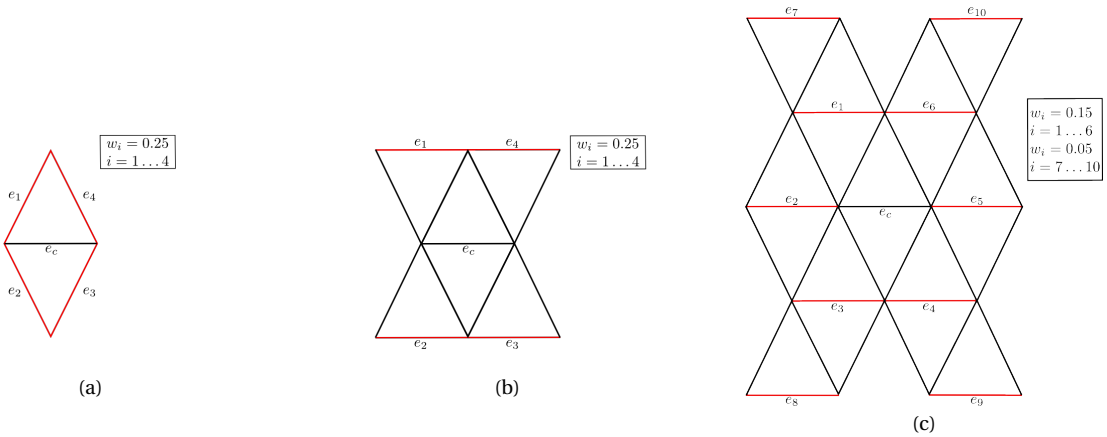

Figure 7: Three different edge prediction stencils. Stencil (a) predicts edge e_c as the average of its adjacent edges. In stencil (b), e_c is the average of its supposedly parallel neighbours. Stencil (c) extends (b) by adding other edges to the scheme, assigning them a weight related to their distance from e_c.

The key factor behind the success of this approach is the correlation between delta coordinates and visual error. The quantization of delta coordinates preserves the *high-frequency* details of the surface, which are the perceptually most relevant ones, while losing information only in the unimportant regions. Unfortunately, there is no such operator as the discrete Laplacian in edge shape space that can be used to predict the characteristics of an edge from its neighbours. Several edge-based Laplace operators have been proposed [8], but they are mostly defined in terms of vertex coordinates and interior angles, which makes them unsuitable for our purposes.

We therefore tried to mimic the general idea of predicting one element as a weighted average of its nearest neighbours, with the aim of exploiting the redundancy in edge space in a different way. The process requires to build a matrix similar to L^*, which allows us to handle the reconstruction process in the same way as in the original approach by recomputing lengths and angles from the encoded data in the least square sense and then applying MSGI to get the resulting shape. To define the entries of L we designed a number of different *stencils* (see Figure 7) used to predict length and angle of a central edge from the surrounding ones. Each row of L^* is defined accordingly to the selected stencil.

Our first stencil in Figure 7 (a) is inspired by the parallelogram predictor and predicts the central edge as the simple average of the edges of its incident faces. Unsurprisingly, it fails to reliably correlate the edges, since there is no fixed pattern in the distribution of edges. It makes sense to use such predictor only for very regular meshes, but in general, it does not give any benefit in terms of prediction accuracy. A deeper analysis suggests that, given an edge, the neighbours characterized by similar length and angle are generally not directly adjacent to it, tending instead to be more or less *parallel* to it. We plug this information into the design of our stencils by extending the support of stencils as shown in Figure 7 (b,c). We noticed that extending the stencils in order to include a larger number of edges does not produce any significant improvement in the prediction, since the most relevant ones are closer to the central edge. As a consequence, our stencils usually do not involve more than 10 nearby edges, which helps keeping the computational cost low. Experimental results support our choice: better predictions are carried out from a small number of edges, usually 4. Increasing the stencil size to 6 or 12 edges as shown in Figure 7 (b,c) does not affect significantly the results. Each predictive scheme identifies a subset of edges and assigns a weight to each of the edge related to its distance from the centre of the stencil: the closer the edge, the higher the weight. We compute the associated *edge δ-coordinates* in terms of length and angles as

$$\tilde{l}_c = l_c - \sum_{i=1}^{k} w_i l_i, \qquad \tilde{\theta}_c = \theta_c - \sum_{i=1}^{k} w_i \theta_i$$

where $e_1, \dots, e_k$ are the k edges of the stencil and $w_1, \dots, w_k$ the associated weights. We quantize and encode only the prediction residual in the same way as done in [24]. Notice that all the schemes are purely combinatorial, in the sense that they depend on the connectivity only, and as a consequence there is no need to encode any information about the stencil itself. From the edge δ-coordinates plus at least one explicitly stored *anchor edge* it is straightforward to recover the whole set of edge coordinates following the approach

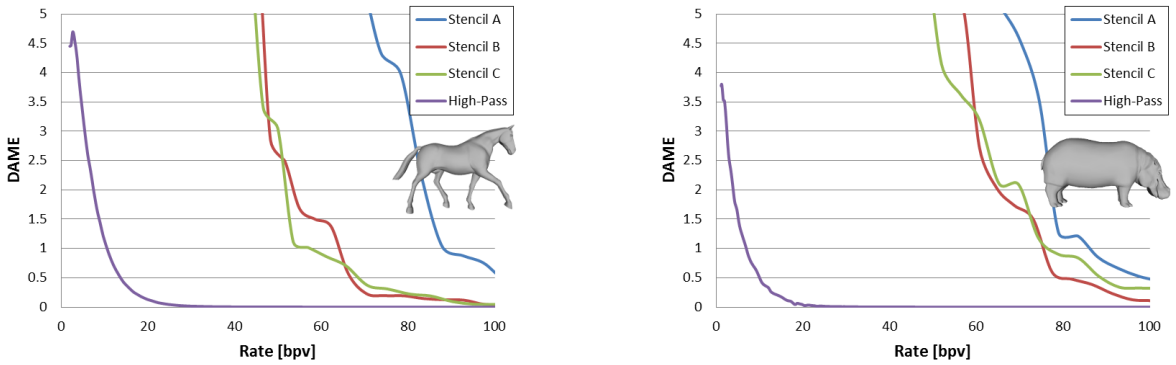

Figure 8: Rate-distortion plots for the stencils in Figure 7 for the horse mesh (left) with 8k vertices and the hippo mesh (right) with 22k vertices, compared to the original high-pass quantization. The irregular behaviour of the plots, due to the instability of the process, and the poor performances in terms of compression rate are clearly visible.

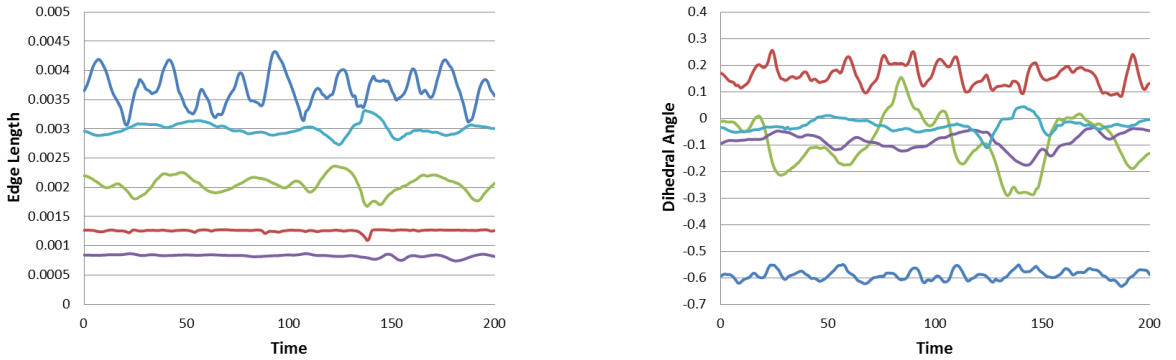

Figure 9: Trajectories of edge lengths (left) and dihedral angles (right) for different edges of the *Dance* sequence. Some of the edges are relatively stable over time, while others are characterized by larger deformations. The red line shows an example of an edge where small deformations in length correspond to large deformations in angles and vice versa for the blue line.

of [24]. Remember that, in our case, we do not aim to preserve some relevant feature of the shape, since our stencils are not characterized by the same properties of the discrete Laplace operator. Instead, we are pursuing the goal of investigating the possible relationships between a small subset of locally close edges that may result in the reduction of the data entropy without compromising the recovery of the original shape.

Unfortunately, none of the proposed predictive scheme seems to be accurate enough to significantly reduce the data entropy on a static mesh. Figure 8 presents the rate-distortion plot in the case of encoding a single static mesh, compared with the results achieved by traditional Laplacian compression. The results are quite unsatisfactory in terms of tradeoff between rate and distortion. And this is not the only drawback, since the aforementioned problems (inefficiency and error propagation) still affect this process.

4.3 Trajectory approximation

The poor performance of the spatial predictors in terms of entropy reduction persuaded us to switch our perspective and to focus on exploiting temporal coherence. As we already pointed out, length and angle of an edge vary over time. Particularly, the edge length varies as the edge is *stretched* while the angle varies as the edge is *bent*. In principle there is no guarantee that the two deformations are correlated, so one edge may be stretched without being bent and vice versa. In fact, experimental results show that the correlation coefficient between the variation in edge length and the variation in dihedral angle between pairs of consecutive frames is usually close to zero, which means that these two quantities are not linearly correlated.

We therefore pursue our goal by compressing lengths and angles separately. Instead of treating each edge as a moving point in a 2-dimensional space, we consider each edge as describing two different *trajectories* in time, one describing the variation of its length and one for the variation of its angle. Figure 9 shows an

example of such a trajectory in both spaces. Referring to (1), the rows of S_l and S_θ represent the trajectories in the length space and in the angle space respectively. Our aim now is to efficiently encode each edge trajectory. As mentioned above, we focus on the prediction of the temporal component, without taking into account the spatial relationship between edges.

As shown in Figure 9, the edges of a mesh can behave quite differently. Some are pretty stable and describe a smooth trajectory, while others, usually corresponding to the parts of the object where deformation occurs the most, are characterized by a large and sometimes abrupt deformation. In most cases the significant deformation is distributed across a small number of frames, so it is reasonable to assume that, knowing the behaviour of an edge in the last w frames, we can guess how the edge is going to deform in the next frame. More formally, we predict the behaviour of an edge at frame k from the trajectories described by the edge in frames $k-w,\dots,k-1$, where w is the size of the temporal window. As usual, we encode only the error in the prediction.

To predict the edge at frame k we approximate the trajectories of the last w encoded frames using *polynomial* interpolation. Given the edge lengths $l_i^{k-w},\dots,l_i^{k-1}$ of e_i in the w-sized temporal window of interest, we look for the coefficients $a_0,\dots,a_{w-1}$ of the polynomial

$$p(x)=a_0+a_1x+a_2(x^2)+\cdots+a_{w-1}x^{w-1}$$

of degree $w-1$ that interpolates the prescribed trajectory,

$$p(t)=l_i^{k-w+t},\qquad t=0,\dots,w-1.$$

For each edge, the set of coefficients can be found by solving a linear system of size $w\times w$, and we can then predict l_i^{k-w+t} as $p(t)$ and encode only the prediction error $l_i^{k-w+t}-p(t)$. There is no need to encode these coefficients, since they are computed directly from the previously encoded frames, and the same can be done during decoding. The procedure is applied in the same way to predict the dihedral angles, possibly with a different size of the temporal window.

This approach significantly improves the result achieved by the parallelogram predictor, both in terms of rate and distortion, but it is still affected by several issues. First, it requires to encode the first $w-1$ frames of the sequence in a different way, in order to have enough data to predict the w-th frame. Unfortunately, for small values of w, such as $w=1$ or $w=2$, the prediction of the trajectories is not accurate enough, since it gets better only for $w=5$ or $w=6$. As a consequence, we need to encode the data related to the first 4 or 5 frames with a different approach (for example, using a spatial predictor), introducing an overhead in the total bit rate. Second, we still have to encode a large number of values, which leads to a huge amount of data to be compressed. Third, the trajectories of a relevant subset of edges are characterized by peaks and erratic behaviour, which makes them difficult to predict. At the same time, if their length and angles are not encoded at reasonable precision, the decoded frame can be affected by a severe deformation error. In general, despite a significant entropy reduction with respect to the previously described approaches, the bit rate is still way too high to achieve a competitive result. These limitations will be discussed in more detail in Section 5.

In an attempt to further reduce the amount of data required to efficiently encode the sequence, we consider the whole sequence as a single entity and try to approximate the whole trajectory of each edge by means of high-degree polynomial interpolation. Instead of considering a small temporal window we consider the global trajectory of the i-th edge $l_i^1,\dots,l_i^N$, and we search for an approximating polynomial p_i of a fixed degree k with $k<N$, which approximates the trajectory in the least squares sense,

$$\min_{p_i}\sum_{t=1}^{N}\|p_i(t)-l_i^t\|^2.$$

For each edge, the coefficients $a_0^i,\dots,a_k^i$ of p_i can again be found by solving a linear system. We then group all coefficients into a single matrix

$$P_l=\begin{pmatrix} a_0^1 & a_1^1 & \dots & a_k^1\\ a_0^2 & a_1^2 & \dots & a_k^2\\ \vdots & \vdots & \ddots & \vdots\\ a_0^E & a_1^E & \dots & a_k^E\end{pmatrix},$$

and encode only P_l and the corresponding matrix P_θ for dihedral angles. The overall size of the data is reduced, since we are not encoding any residual error. To keep the number of coefficients reasonably small and avoid artefacts due to ill-conditioned high-degree polynomials, we usually do not exceed $k=N/4$.

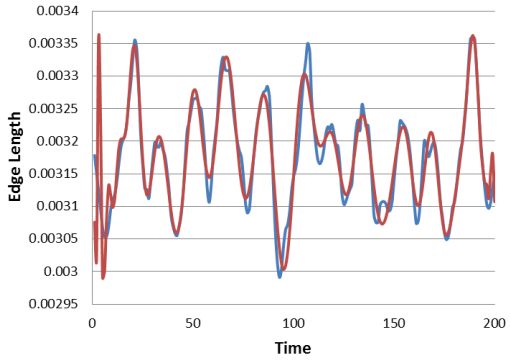

Figure 10: Approximating the edge length trajectory of the original edge (blue) by a polynomial of degree 50 (red) captures the overall behaviour, but significant errors are visible.

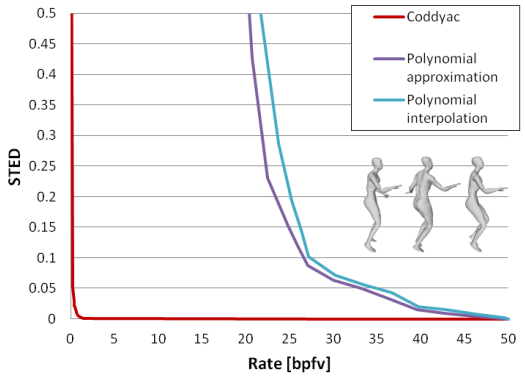

Figure 11: Rate-distortion plot comparing the edge-based polynomial prediction techniques with two Coddyac-based approaches [31, 30] for the Dance data set.

However, this approach also has its flaws. First, we explicitly decided not to store any correction value for the edges, in order to reduce the size of the problem, but there is still a residual error which significantly affects the reconstruction, since not every trajectory is approximated with sufficient precision (see Figure 10). Second, the quantization process of the coefficients further affects the error in the reconstruction, due to a loss of precision in the coefficients. Therefore, the coefficients must be quantized with a reasonable level of precision, but this inevitably affects the entropy of the matrices, so a tradeoff between precision and number of coefficients is required to keep the bit rate as small as possible. The results, as shown in Figure 11, are not very different from the ones achieved by polynomial prediction, but in both cases they are worse than the results achieved by techniques working in vertex space. Variations of the proposed approaches based on rational polynomials and discrete cosine approximation did not produce any substantial improvement in the results.

4.4 Principal Component Analysis

In Section 4.3 we introduced the concept of edge trajectories over time. In the common setup, the concept of a trajectory is related to the spatial motion characterizing the vertices of the mesh. In that case, the trajectories have a purely geometric meaning: each of them is in fact a curve in $\mathbb{R}^3$ passing through the vertex positions according to their temporal order. Nearby vertices are usually characterized by similar trajectories, therefore enforcing both spatial and temporal coherence. As a consequence, the number of significant trajectories is actually much smaller than the number of vertices.

Following this observation, the Coddyac algorithm [33] reduces the size of the problem by finding a set of significant *base trajectories* and then expressing each vertex trajectory as their linear combination. The key to find the base trajectories is to use *principal component analysis* (PCA). The original sequence is represented as a $V \times 3N$ matrix, where the i-th row contains the spatial coordinates (x_i, y_i, z_i) of v_i in each frame $f_1, \ldots, f_N$. PCA is performed on the matrix in order to decompose the trajectories and to find the h most relevant $\mathbb{R}^{3N}$ row-vectors. Each vector represents a base trajectory of the sequence and is quantized

and encoded. To encode the original trajectories of the vertices, each row of the matrix is then decomposed into a linear combination of the h basis vectors, quantizing and encoding only the coefficients. No residual is encoded, hence the quality of the decoded sequence depends mainly on the number of basis vectors and on the quantization level. The number of base trajectories is an input parameter of the process and is usually estimated experimentally. In principle, more basis vectors result in a better approximation of the trajectories and a smaller overall error.

Trajectories and coefficients can further benefit from ad-hoc prediction schemes [34, 36, 30] that further reduce the entropy, achieving a significant gain in terms of compression rate. There are several advantages to this approach: particularly, the overall number of values to be encoded, whose entropy is reduced by the predictive scheme, is significantly smaller than with other techniques. This method achieves the best results so far in terms of both compression rate and perceived distortion.

Inspired by the same principle, we investigated the application of PCA to the concept of edge trajectories. As done before, we work on length and angles separately. The aim is to exploit the presence of significant edge trajectories by applying PCA separately to the matrices S_l and S_θ in (1). Selecting h_l vectors from S_l and h_θ vectors from S_θ, with h_l possibly different from h_θ, then results in the submatrices

$$B_l = \begin{pmatrix} b\,l_1^1 & b\,l_1^2 & \dots & b\,l_1^N \\ b\,l_2^1 & b\,l_2^2 & \dots & b\,l_2^N \\ \vdots & \vdots & \ddots & \vdots \\ b\,l_{h_l}^1 & b\,l_{h_l}^2 & \dots & b\,l_{h_l}^N \end{pmatrix},$$

$$B_\theta = \begin{pmatrix} b\,\theta_1^1 & b\,\theta_1^2 & \dots & b\,\theta_1^N \\ b\,\theta_2^1 & b\,\theta_2^2 & \dots & b\,\theta_2^N \\ \vdots & \vdots & \ddots & \vdots \\ b\,\theta_{h_\theta}^1 & b\,\theta_{h_\theta}^2 & \dots & b\,\theta_{h_\theta}^N \end{pmatrix}.$$

To decompose the i-th trajectory from S_l and S_θ, we compute two sets of h_l and h_θ coefficients

$$C_i^l = B_l \begin{bmatrix} l_i^1 \\ l_i^2 \\ \vdots \\ l_i^N \end{bmatrix}, \qquad C_i^\theta = B_\theta \begin{bmatrix} \theta_i^1 \\ \theta_i^2 \\ \vdots \\ \theta_i^N \end{bmatrix}.$$

In order to find the coefficients, we multiply each row vector of the original matrix with the basis vectors. Due to the orthogonality of the basis vectors, this is in fact equivalent to solving the linear systems $B_l^T C_i^l = [l_i^1, \dots, l_i^N]^T$ and $B_\theta^T C_i^\theta = [\theta_i^1, \dots, \theta_i^N]^T$. Each edge is therefore characterized by $h_l + h_\theta$ coefficients. We group all the coefficients into two matrices C_l and C_θ. Finally, we quantize and encode only B_l, B_θ, C_l and C_θ, for a total size of $(h_l + h_\theta) \times (N + E)$ data values, which is much smaller than the original size of the problem. As in Coddyac, no residual is encoded, and the quality of the reconstruction depends only on the number of basis vectors and on the precision of encoding the basis vectors and the coefficients. The decoder recomputes S_l from $\tilde{B}_l$ and $\tilde{C}_l$ and likewise S_θ from $\tilde{B}_\theta$ and $\tilde{C}_\theta$ and then rebuilds each frame of the sequence using MSGI.

In our experiments we observed that the quality of the results is mostly driven by the precision in the encoding of edge lengths. As a consequence, we compress dihedral angles aggressively using few basis vectors and investigate more bits to encode the edge length matrix. For sequences with about 200 frames choosing $b_L = 30$ and $b_A = 10$ provides good results in most cases. We also point out the fact that, while the Coddyac-based approaches take advantage of spatial coherence to reduce the original coefficients entropy, our approach does not exploit the local proximity of edges, which instead appears to affect the entropy value only marginally. The results of this technique are the best we achieved so far, as shown in Figure 12. Further improvements are still possible, due to the high entropy characterizing the coefficient matrices. We therefore developed two variants of this technique, which involve a *clustering* step.

In order to reduce the entropy of the coefficient matrix in vertex space, spatial predictors similar to the ones described in Section 4.1 can be used. Instead of predicting vertex coordinates, they are used to predict the coefficients characterizing a vertex from the coefficients that characterize its neighbours. In this simple way, the entropy of the coefficients is drastically reduced. Since the spatial correlation between edges is weaker, as already pointed out, the same principle cannot be applied in edge space. However, the

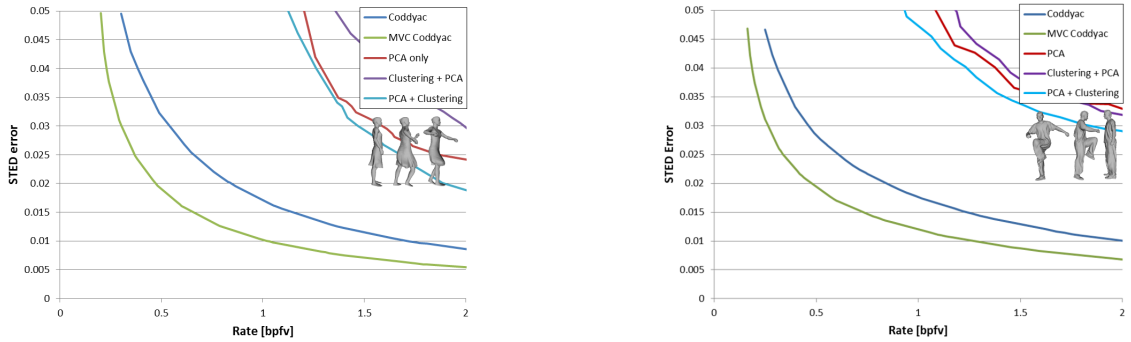

Figure 12: Rate-distortion plots for PCA-based compression in vertex space [31, 30] and in edge space for the Samba data set (left) and the March2 data set (right).

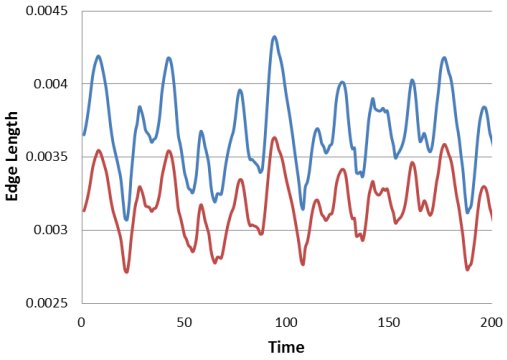

Figure 13: Two edges from the Dance data set with similar trajectories. Notice that the actual length values are different, but the behaviour of the edges is the same. Therefore, their relative trajectory are similar and they will are described by similar PCA coefficients.

experiments show that there are subsets of edges, not necessarily close in the spatial sense, which have very similar trajectories and are characterized by similar coefficient vectors (see Figure 13).

This similarity can be exploited in order to reduce the amount of coefficients by identifying a number of *coefficient clusters* and using the average trajectory of each cluster to predict the actual trajectories of its edges. In order to perform the clustering, we express each edge in terms of *relative* length and angles, that is, instead of considering the actual edge coordinates, we normalize both lengths and angles, dividing them by their average value. The average lengths and angles for each edge are encoded with high precision. This representation improves the detection of edges that are characterized by similar trajectories and reduces the range of values of the associated coefficients, thus possibly reducing the entropy of the edge data.

We consider each row-vector of C_l as a point in $\mathbb{R}^h_l$, and the actual clustering is then carried out using a standard k-means algorithm. The number of clusters is a parameter of the algorithm. At the end of the clustering step, an integer label index $I_i \in [1,k]$ is assigned to each row-vector C^l_i. We replace row C^l_i in C_l with the vector $C^l_i - H_{I_i}$, where H_{I_i} is the centroid (average vector) of the cluster I_i. In this way, we reduce the entropy of C_l, since edges with similar trajectories are expected to be close to their cluster centroid. The same process is repeated on the C_θ matrix, with a possibly different number of clusters.

This method introduces a small overhead to the data size, since we have to properly encode the cluster centroids and the label characterizing each edge, therefore adding $2E + (k_l \times h_l) + (k_\theta \times h_\theta)$ values to the size of the problem. However, the compression rate is improved, even if the entropy reduction is not as significant as expected. As a consequence there is 5–10% size reduction with respect to PCA only (see Figure 12).

The clustering technique may be alternatively employed as a preprocessing step, identifying the edges that are characterized by similar trajectories by performing the clustering directly on the S_l and S_θ matrices, possibly after trajectory normalization and before applying the PCA. This approach is more expensive than the previous one, since the space $\mathbb{R}^N$ of the trajectories is much larger than the space of coefficients ($\mathbb{R}^h_l$ or $\mathbb{R}^h_\theta$) and the computation of the distances between vectors is computationally more expensive. After the

preprocessing, each edge is characterized by its cluster label, and each cluster consists of edges with similar trajectories, expressed in terms of edge coordinates. We then apply PCA separately on each cluster. In fact, since the variance of the trajectories is supposedly small, it should be possible to efficiently decompose the edges with few basis and shorter coefficient vectors. We can also employ a different number of basis vectors for each cluster, which allows us to further reduce the overall size. The overhead of the preprocessing clustering step consists of a set of PCA basis vectors for each cluster plus one label for each edge, for a total amount of $E + k_l \times h_l + k_\theta \times h_\theta$ values. Note that applying clustering after PCA results in a significant improvement compared to the standard PCA-based technique, while applying clustering on trajectories as a preprocessing step usually introduces an overhead that may cause, at least for small compression rates, a worse performance than the standard PCA (see Figure 12). In general, we are still not able to achieve a result better than [31], but the improvements with respect to the methods presented in Sections 4.1 and 4.3 are remarkable.

5 Discussion

Despite the premises, our findings regarding redundancy in the edge data are disappointing. As a consequence, none of the proposed approaches was able to significantly improve the state of the art. In fact, none of the presented techniques was able to outperform the commonly used techniques for compression of both static and dynamic geometry, and in most cases the performance is significantly worse than the state-of-the-art approaches. Therefore, the question is: *what did not work out?*

The answer is neither trivial nor straightforward. There are several problems in performing geometry compression in the space of edge coordinates; some of them were known from the beginning, while others emerged during our work. Undoubtedly, working in edge space has the disadvantage of working in a space whose size is twice as big than the standard vertex space. In the context of a single static mesh, it is not possible to select less than $3V$ data to uniquely identify the geometry of a triangular mesh. However, when dealing with dynamic geometry, it is possible to encode less than $3VN$ data to recover the whole sequence, as demonstrated in [33]. Our hope was to find redundancy in the data and to further reduce the amount of data needed to reconstruct the sequence.

However, this redundancy in edge space is not as strong as we expected, and it did not impact the final results as we had hoped. One reason is the difficulty to exploit spatial redundancy in edge space, which is one of the fundamental aspects of all compression technique. In vertex space, the spatial coherence may be exploited in simple and linear ways, using geometric predictors or PCA. In edge space, spatial coherence appears to be harder to find. It does not mean that there is no spatial coherence between edges. In fact, it means that the spatial relationship between edges is not trivial and, most likely, this relationship is not linear. For example, we noticed that the accuracy of a spatial predictor of the edge length benefits from adding weights depending on the angle between edge vectors. Particularly, parallel edge vectors are characterized by a similar behaviour and may be used to predict the behaviour of another parallel edge. However, this relationship is also not linear, since it involves the angle between vectors, and cannot be easily plugged into an edge predictor, since it requires a knowledge of the geometry that we usually do not have. Including more information into our predictors (for example, interior angles) requires encoding more geometric data, thus resulting in higher bit rates.

We experienced several problems in terms of temporal coherence of edges, which eventually made it impossible to outperform the existing approaches. As mentioned earlier, an edge may be stretched or bent, and it is reasonable to assume that the deformation affects not only one edge, but a subset of them. However, not all the edges of the subset are affected in the same way; some edges are distorted more while others are only slightly modified, depending on their position and orientation. Also, in most cases, such as for articulated shapes, the bending of certain edges results in the bending of possibly distant edges, due to the structure of the object, or the bending energy can be distributed across various edges in a non-uniform fashion. This relationship can help to exploit the redundancy in the behaviour of the edges, but it is not trivial to model and requires to introduce some additional knowledge about the problem, making it harder to reach a competitive rate-distortion tradeoff.

Moreover, other problems arise from our framework. Unexpectedly, we found that parts of the mesh which are supposedly not affected by any stretching or bending deformation may exhibit some variation in the edge data. This is particularly true for data acquired by scanning a real, moving object (for example, the *Jump* data set) which are usually affected by noise. That means that the assumption of rigidity of a large part of the shape, that should result in reliable temporal predictors and smooth edge trajectories, cannot always

be guaranteed. A strictly related problem is that the MSGI projection operator, which we employ to recover spatial coordinates from edge data, is quite sensitive to noise and requires the edge data to be compressed with high precision in order to get the shape without introducing too much distortion in the process. As far as we know, the only alternative to MSGI is the DSO algorithm [6], which appears to be slightly more robust but does not significantly impact the rate-distortion plots. Moreover, DSO is more expensive than MSGI in terms of computational and memory complexity, with the result that the compression technique will not be of any practical use.

Another potential problem of our framework is the fact that we mainly focus on adapting compression techniques that were designed for working in vertex space. Geometric vertex data have high entropy values, and even a simple technique such as parallelogram prediction may significantly reduce this value, leading to an outstanding size reduction. The same holds for Laplacian compression and PCA, which efficiently exploit spatial and temporal coherence. The entropy reduction is given by the fact that these techniques allow to pass from the common $\mathbb{R}^3$ space to a local frame, where the span of vertex coordinates is smaller and lots of coordinates are almost equal to zero. The same transformation, however, does not work so well for the edge data, since lengths and angles are not affected by the change from global to local frames, and the entropy is not significantly reduced. This also explains why the PCA approach is the one that achieves the best results in our case: the entropy is not significantly reduced, but the amount of encoded data is much smaller than encoding the whole sequence using spatio-temporal predictors. Unfortunately, we cannot further reduce the amount of data without affecting the quality of the mesh sequence.

5.1 Future work

After having summarized the problems which arose in our experiments, the question becomes: *can we make things work out?*

It is not trivial to effectively tackle the issues mentioned above. In general, the problem of efficiently compressing static and dynamic geometry represented in edge shape space is affected by issues that are strictly related to the non-linear nature of this representation. As pointed out, to improve edge predictors we could develop new, non-linear predictors based on additional geometric information, unlike most of the vertex predictors which require only connectivity information. Adding geometry information has several drawbacks, such as increasing the size of the problem and adding more redundancy. If, for example, we decide to encode geometric information about interior angles of the mesh, the size of the problem passes from $6V$ to $10V$ for each mesh. Besides, since the interior angles and edges are strictly correlated, interior angle compression will suffer the same problems as edge compression.

On a practical note, most of the techniques working in vertex space are easy to implement and achieve a huge size reduction at low cost. A more complex predictor would probably not be of practical use, due to its expensiveness. The complexity of the process will also depend on the complexity of the projection operator needed to recover the geometry of the mesh in terms of vertex coordinates, which is another bottleneck in the compression pipeline. Due to the intrinsic nature of the problem and of the involved data, it appears definitely hard to improve the current techniques in terms of both robustness and efficiency.

Acknowledgements

This work was supported by the SNF under project number 200020-146764. The authors thank Libor Váša for his valuable help.

References

[1] M. Alexa and W. Müller. Representing animations by principal components. *Comput. Graph. Forum*, 19(3):411–418, 2000.

[2] P. Alliez and M. Desbrun. Valence-driven connectivity encoding for 3D meshes. *Comput. Graph. Forum*, 20(3):480–489, 2001.

[3] R. Amjoun and W. Straßer. Efficient compression of 3D dynamic mesh sequences. *Journal of the WSCG*, 15(1–3):99–106, 2007.

[4] I. Baran, D. Vlasic, E. Grinspun, and J. Popović. Semantic deformation transfer. *ACM Trans. Graph.*, 28(3):36:1–36:6, 2009.

[5] M. Bourges-Sévenier and E. S. Jang. An introduction to the MPEG-4 animation framework eXtension. *IEEE Trans. Circuits Syst. Video Technol.*, 14(7):928–936, 2004.

[6] S. Fröhlich and M. Botsch. Example-driven deformations based on discrete shells. *Comput. Graph. Forum*, 30(8):2246–2257, 2011.

[7] E. Grinspun, A. N. Hirani, M. Desbrun, and P. Schröder. Discrete shells. In *Proc. of the ACM SIGGRAPH/Eurographics Symposium on Computer Animation*, SCA '03, pages 62–67, 2003.

[8] L. He and S. Schaefer. Mesh denoising via L_0 minimization. *ACM Trans. Graph.*, 32(4):Article 64, 8 pages, 2013.

[9] L. Ibarria, P. Lindstrom, J. Rossignac, and A. Szymczak. Out-of-core compression and decompression of large n-dimensional scalar fields. *Comput. Graph. Forum*, 22(3):343–348, 2003.

[10] E. S. Jang, J. D. K. Kim, S. Y. Jung, M.-J. Han, S. O. Woo, and S.-J. Lee. Interpolator data compression for MPEG-4 animation. *IEEE Trans. Circuits Syst. Video Technol.*, 14(7):989–1008, 2004.

[11] Z. Karni and C. Gotsman. Compression of soft-body animation sequences. *Comput. Graph.*, 28(1):25–34, 2004.

[12] M. Kilian, N. J. Mitra, and H. Pottmann. Geometric modeling in shape space. *ACM Trans. Graph.*, 26(3):Article 64, 8 pages, 2007.

[13] S. Kircher and M. Garland. Free-form motion processing. *ACM Trans. Graph.*, 27(2):Article 12, 13 pages, 2008.

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

[15] Y. Lipman, O. Sorkine, D. Levin, and D. Cohen-Or. Linear rotation-invariant coordinates for meshes. *ACM Trans. Graph.*, 24(3):479–487, 2005.

[16] K. Mamou, T. Zaharia, F. Preteux, N. Stefanoski, and J. Ostermann. Frame-based compression of animated meshes in MPEG-4. In *Proceedings of the 2008 IEEE International Conference on Multimedia and Expo*, pages 1121–1124, 2008.

[17] S. Marras, T. J. Cashman, and K. Hormann. Efficient interpolation of articulated shapes using mixed shape spaces. *Comput. Graph. Forum*, 32(8):258–270, 2013.

[18] K. Müller, A. Smolic, M. Kautzner, P. Eisert, and T. Wiegand. Predictive compression of dynamic 3D meshes. In *Proceedings of the 2005 IEEE International Conference on Image Processing*, volume 1, pages 621–624, 2005.

[19] K. Müller, A. Smolic, M. Kautzner, P. Eisert, and T. Wiegand. Rate-distortion-optimized predictive compression of dynamic 3D mesh sequences. *IEEE Int. Conf. Image Process.*, 21(9):812–828, 2006.

[20] J. Peng, C.-S. Kim, and C.-C. Jay Kuo. Technologies for 3D mesh compression: A survey. *J. Vis. Comm. Im. Repr.*, 16(6):688–733, 2005.

[21] J. Peng and C.-C. J. Kuo. Geometry-guided progressive lossless 3D mesh coding with octree (OT) decomposition. *ACM Trans. Graph.*, 24(3):609–616, 2005.

[22] J. Rossignac, A. Safonova, and A. Szymczak. 3D compression made simple: Edgebreaker on a corner-table. In *Proceedings of Shape Modeling International*, SMI '01, pages 278–283, 2001.

[23] M. Sattler, R. Sarlette, and R. Klein. Simple and efficient compression of animation sequences. In *Proceedings of the 2005 ACM SIGGRAPH/Eurographics Symposium on Computer Animation*, SCA '05, pages 209–217, 2005.

[24] O. Sorkine, D. Cohen-Or, and S. Toledo. High-pass quantization for mesh encoding. In *Proceedings of the 2003 Eurographics/ACM SIGGRAPH Symposium on Geometry Processing*, SGP '03, pages 42–51, 2003.

[25] N. Stefanoski, X. Liu, P. Klie, and J. Ostermann. Scalable linear predictive coding of time-consistent 3D mesh sequences. In *Proceedings of 3DTV 2007*, pages 1–4, 2007.

[26] N. Stefanoski and J. Ostermann. Connectivity-guided predictive compression of dynamic 3D meshes. In *Proceedings of IEEE International Conference on Image Processing*, pages 2973–2976, 2006.

[27] R. W. Sumner and J. Popović. Deformation transfer for triangle meshes. *ACM Trans. Graph.*, 23(3):399–405, 2004.

[28] C. Touma and C. Gotsman. Triangle mesh compression. In *Proceedings of Graphics Interface*, pages 26–34, 1998.

[29] L. Váša and G. Brunnett. Exploiting connectivity to improve the tangential part of geometry prediction. *IEEE Trans. Vis. Comput. Graph.*, 19(9):1467–1475, 2013.

[30] L. Váša, S. Marras, K. Hormann, and G. Brunnett. Compressing dynamic meshes with geometric Laplacians. *Comput. Graph. Forum*, 33(2):145–154, 2014.

[31] L. Váša and O. Petrik. Optimising perceived distortion in lossy encoding of dynamic meshes. *Comput. Graph. Forum*, 30(5):1439–1449, 2011.

[32] L. Váša and J. Rus. Dihedral angle mesh error: A fast perception correlated distortion measure for fixed connectivity triangle meshes. *Comput. Graph. Forum*, 31(5):1715–1724, 2012.

[33] L. Váša and V. Skala. CODDYAC: Connectivity driven dynamic mesh compression. In *Proceedings of 3DTV*, pages 1–4, 2007.

[34] L. Váša and V. Skala. COBRA: Compression of the basis for PCA represented animations. *Comput. Graph. Forum*, 28(6):1529–1540, 2009.

[35] L. Váša and V. Skala. Combined compression and simplification of dynamic 3D meshes. *Comput. Animat. Virtual Worlds*, 20(4):447–456, 2009.

[36] L. Váša and V. Skala. Geometry-driven local neighbourhood based predictors for dynamic mesh compression. *Comput. Graph. Forum*, 29(6):1921–1933, 2010.

[37] L. Váša and V. Skala. A perception correlated comparison method for dynamic meshes. *IEEE Trans. Vis. Comput. Graph.*, 17(2):220 –230, 2011.

[38] D. Vlasic, I. Baran, W. Matusik, and J. Popović. Articulated mesh animation from multi-view silhouettes. *ACM Trans. Graph.*, 27(3):Article 97, 9 pages, 2008.

[39] T. Winkler, J. Drieseberg, M. Alexa, and K. Hormann. Multi-scale geometry interpolation. *Comput. Graph. Forum*, 29(2):309–318, 2010.

[40] J. Zhang and C. B. Owen. Octree-based animated geometry compression. In *Proceedings of the 2004 Data Compression Conference*, pages 508–517, 2004.

[41] J. Zhang and C. B. Owen. Hybrid coding for animated polygonal meshes: combining delta and octree. In *Proceedings of the 2005 International Conference on Information Technology: Coding and Computing*, volume 1, pages 68–73, 2005.