DEBLOCKING FOR DYNAMIC TRIANGLE MESHES

Jan Rus and Libor Váša
Department of Computer Science and Engineering, Faculty of Applied Sciences
University of West Bohemia, Univerzitní 8, Plzeň, Czech Republic

Keywords: Static Triangle Meshes, Dynamic Triangle Meshes, Data Compression, Coddyac, Clustering, Blocking Artifacts, Deblocking, Visual Quality.

Abstract: Mesh segmentation (clustering) is a useful tool, which improves compression performance. On the other hand, per-partes processing of meshes often leads to new types of artifacts - cracks and shifts on the borders between clusters. These artifacts are detected by both, Human Visual System (HVS) and perceptually-motivated distortion metrics. In this paper, we present a post processing algorithm, which aims at reducing such artifacts without needing any additional data - using only information about the cluster distribution that is already present at the decoder. A rigid transformation, which minimises the border artifacts, is iteratively computed and applied per cluster. Our experiments show that this approach leads to a reduction of distortion, as measured by the STED metric, by up to 18% for low bitrates. We also present visual results confirming that the improvement is well visible.

1 INTRODUCTION

The use of 3D data in the form of triangle meshes is widespread. A great many examples of using this kind of data can be found in engineering applications or in film and gaming industry. These examples include not only static, but also dynamic meshes — 3D animations, which are usually represented by a series of static triangle meshes, one mesh for each frame of the animation. This kind of data representation is usually voluminous and needs to be compressed using a specialised compression algorithm to achieve efficient storage and transmission.

Many of the most efficient algorithms specialised in static or dynamic mesh compression employ vertex clustering or other mesh division technique. Unfortunately, artifacts may appear in the form of cracks or steps between individual clusters of the decompressed triangle mesh (see figure 1). This kind of distortion is concentrated in small areas of the triangle mesh surface, thus it only slightly affects the total distortion caused by compression itself as identified by the usual measures (KG-error, for example), but may negatively affect visual quality of the mesh. Similar artifacts are known in image and video compression, where deblocking filters are used to handle them.

In this paper, we propose a method of post-processing of decompressed triangle meshes, which locally improves the visual quality with focus on compression algorithms including mesh division techniques. We choose clustered Coddyac (Rus and Váša, 2010) as a representative of this type of algorithms in this paper, because it is currently the most efficient compression algorithm for dynamic meshes.

The rest of the paper is organised as follows. Section 1.1 gives a brief overview of existing approaches to dynamic mesh compression with clustering and methods of improving the mesh appearance. Section 2 specifies the scheme of Coddyac compression algorithm and its clustering improvement. A description of the mesh deblocking is presented in section 3 and results of our experiments are reported in section 4. Finally, we give conclusions in section 5.

1.1 Related Work

Many algorithms have been proposed for static and dynamic mesh compression. There are also various algorithms known dividing a given mesh into smaller parts using geometry or connectivity criteria. Meshes can be divided into semantic parts as well. In many cases, efficient compression schemes for static and dynamic mesh compression combine compression algorithms and algorithms of mesh segmentation (clustering) to increase compression ratio.

One such method intended for **static** mesh com-

Rus J. and Váša L..
DEBLOCKING FOR DYNAMIC TRIANGLE MESHES.
DOI: 10.5220/0003829700480057
In *Proceedings of the International Conference on Computer Graphics Theory and Applications* (GRAPP-2012), pages 48-57
ISBN: 978-989-8565-02-0

pression has been proposed by Karni and Gotsman (Karni and Gotsman, 2000). Spectral mesh compression expresses the triangle mesh data as a linear combination of a set of orthogonal basis functions. As in DCT-based image compression, these basis functions are characterised by a frequency. Spectral mesh compression needs only the triangle mesh connectivity to calculate the orthogonal basis, which consists of eigenvectors of Laplacian matrix calculated from vertex adjacency. Computation of the Laplacian eigenvectors is numerically expensive for meshes containing large number of vertices and thus the mesh must usually be partitioned into submeshes and the calculation must be performed separately for each part. The mesh division is performed by MeTiS algorithm (Karypis and Kumar, 1998) to obtain segments with approximately the same number of vertices and minimal number of edges between individual segments.

In contrast to the previous compression method, method described by Lengyel (Lengyel, 1999) is one of the first compression methods intended for **dynamic** meshes. This method initially segments a given dynamic mesh using information about rigid body motions of the mesh. The motion of all vertices of each segment is approximately described by an affine transformation. If one vertex is included in more segments, weighted combination of affine transformations related to these segments is used. Weighting coefficients, transformation parameters and differences (residuals) between original and estimated vertex positions are quantised and encoded. In order to achieve compression efficiency, approximation residuals are minimised using prediction techniques.

Differential 3D Mesh Coding scheme (D3DMC) proposed by K. Müller, A.Smolić, M. Kautzner, P. Eisert and T. Wiegand (Müller et al., 2005) is a prediction based approach, which uses an octree data structure for spatial subdivision (clustering) of 3D mesh animation with constant connectivity. D3DMC uses an animation description similar to the usual video compression schemes. Frames of animation are represented as sets of subsequent meshes called Groups of Meshes (GOM) consisting of intra meshes (*I mesh*) and predicted differential meshes (*P mesh*). *I meshes* are compressed as a static meshes using 3DMC and following *P meshes* are coded using octree motion segmentation (spatial clustering) of difference vectors of consecutive frames of the animation. *I meshes* are typically used in the first frame of compressed animation sequence or when the prediction in D3DMC becomes too large.

The octree motion segmentation is also described (Zhang and Owen, 2004) by Zhang and Owen. Octree-based animated geometry compression method only uses two consecutive frames of the animation to generate small set of motion vectors representing the motion of the geometry from the previous frame to the current frame. The octree motion segmentation starts with the minimum bounding box as a topmost cell of the octree structure, which includes all vertices. Eight motion vectors approximating the motion of all vertices enclosed within the octree cell are associated with the cell, one motion vector for each cell corner. If the motion of vertices is not approximated well using motion vertices, the cell is repeatedly split into eight octants until the approximation reaches user defined accuracy. Finally, the motion vectors are uniformly quantised to reduce compressed data entropy and thus enhance the compression ratio of following context-adaptive binary arithmetic coding – CABAC, (Marpe et al., 2003). Only the set of motion vectors and the octree structure are stored.

Another approach described by Sattler et al. (Sattler et al., 2005) is based on clustered principal component analysis (CPCA), analysing trajectories of all vertices throughout the animation time. The division of the mesh depends on similarity of these trajectories and usually leads to meaningful clusters, which are separately encoded by shape-space PCA.

A similar approach to compression of dynamic meshes was proposed by Amjoun and Straßer (Amjoun and Straßer, 2007). Their method uses local principal component analysis (LPCA) of vertex motion. First, the set of vertices for each frame is analysed, clustered and per-partes transformed from world coordinate system into local coordinate system of appropriate cluster. Local coordinate system of each cluster is derived from the plane of its seed triangle and the mesh division depends on the motion of vertices in this system. Finally, each cluster is encoded using shape-space PCA and compressed by arithmetic coding.

The Frame-based Animated Mesh Compression (FAMC) method described by Mamou, Zaharia and Prêteux (Mamou et al., 2008) uses mesh segmentation with respect to motion as well. It is based on a skinning model. Each segment is described by a single affine transformation calculated for each frame of the animation, such that the transformation matrix describes motion of vertices from the first frame of the animation to the desired frame of the animation. The segmentation process is based on hierarchical decimation of the original mesh using connectivity simplifications: two neighbouring vertices are merged into a single vertex if their affine motion and affine motion of all vertices merged previously is similar. The final number of segments corresponds to the number of remaining vertices.

Another approach by Rus and Váša (Rus and Váša, 2010) is based on the Coddyac (Váša and Skala, 2007) compression algorithm with focus on the behaviour of trajectory-space PCA with respect to vertex movement complexity. This algorithm transforms geometry information of a dynamic mesh into the form of a set of vertex trajectories and processes this set by PCA. The PCA is used to reduce the set of vertex trajectories necessary to express the mesh movement. Efficiency of the algorithm directly depends on the mesh movement complexity. Therefore the movement is analysed first and vertices with similar movement are assigned into a common group - cluster. This way the movement complexity is locally reduced and thus the compression algorithm efficiency is improved.

As we already noted, mesh division is often used in static and dynamic mesh compression algorithms to improve their efficiency but it causes artifacts in the form of cracks or steps on the boundaries between individual clusters. These artifacts may be caused by algorithms using octree subdivision as well.

Similar kinds of artifacts are well known in the area of image and video compression. It is usually handled by a post-processing technique known as deblocking. Deblocking filter reduces “blocking-artifacts” by smoothing the sharp edges, often formed between blocks of decompressed image or video frame and thus improves its visual quality. In case of (dynamic) mesh compression, only a negligible part of the mesh data is distorted the way described above, and thus the error identified by usually used error measures, such as KG-error, is only slightly affected by this distortion. We need a different metric to measure the distortion related to artifacts caused by mesh division techniques, which is small relative to the distortion caused by compression itself but significant from the visual quality point of view.

One such error metric for static meshes is MSDM2 (Lavoué, 2011) and its ancestor MSDM (Lavoué et al., 2006), which is inspired by SSIM (Wang et al., 2004) index expressing structural similarity of two images. Both MSDM and MSDM2 exploit differences of local curvature statistics of the mesh surface to assess the visual similarity between 3D meshes (original M_o and distorted M_d mesh). MSDM2 computes distortion maps for multiple scales (radius of local neighbourhoods of vertices) and combines them into a global multi-scale distortion map, which is used for computation of distortion score. The metric is then computed as the average of forward ($M_d \rightarrow M_o$) and backward ($M_o \rightarrow M_d$) global distortion scores.

A suitable error measure for dynamic meshes – STED (Spatio-temporal edge difference) – has been presented in (Váša and Skala, 2011). STED is derived from subjective testing of mesh distortion perception and provides better correlation with human perception of quality loss between original and processed dynamic meshes than any other metric we know of.

In this paper, we propose a method of improving the mesh appearance similar to mesh fairing (Desbrun et al., 1999). The mesh fairing is method of smoothing the polygon mesh to improve its visual quality by reducing its curvature variation. Both mesh fairing and our approach move vertices of the mesh, while its connectivity stays the same, to change its appearance. In contrast with mesh fairing, our approach – mesh deblocking – is focused on a specific kind of distortion and affects only sharply defined areas of the mesh surface, thus the principle of our approach is closer to the methods of deblocking filters known in image and video compression.

1.2 Notation

In this paper we use the following notation:

F	- number of frames of animation
V	- number of mesh vertices
T_i	- trajectory vector of the i-th vertex, size $3F$
A	- average trajectory vector, size $3F$
B	- matrix of original animation, size $3F \times V$
C	- autocorrelation matrix, size $3F \times 3F$
E_i	- i-th eigenvector of C, size of $3F$
E	- basis of the PCA subspace, size $3F \times N$
N	- number of most important eigenvectors
S	- matrix of samples, computed by subtraction of A from columns of B, size $3F \times V$
β	- set of border triangles of the given cluster
Σ	- set of surrounding triangles of the given cluster belonging to surrounding clusters
d_i	- decompressed position of the i-th vertex
p_i	- predicted position of the i-th vertex
$\tilde{p}_i$	- transformed position of the i-th vertex

In the rest of this paper, the Rus and Váša approach, one of the most efficient dynamic mesh compression methods known at the time, is discussed and used to demonstrate the influence of the mesh deblocking post-process on the dynamic mesh decompression. The STED is used to measure changes in distortion caused by implementing the deblocking post-process.

2 CLUSTERED CODDYAC ALGORITHM OVERVIEW

The Coddyac is a compression algorithm, which is specialised in dynamic mesh compression and contains two well known algorithms: Rossignac's Edgebreaker (Rossignac, 1999) and Principal Component Analysis (PCA). In Coddyac, Edgebreaker is used for triangle mesh connectivity compression and the PCA is used for compression of the mesh geometry. The Coddyac algorithm processes a sequence of meshes with the same connectivity, which is represented as set of vertex trajectories of individual vertices, each described by a vector T of length $3F$. Trajectory vector T_i of the i-th vertex consist of its XYZ coordinates in all F frames. The set of these trajectory vectors is processed by PCA to find the subspace where the trajectories are located, and the trajectories are expressed in this subspace.

The original animated mesh is represented by matrix B of size $3F \times V$ with columns corresponding to vertex trajectories. Next, we compute the matrix of samples S by subtracting the average trajectory vector A from columns of the matrix B. Subsequently, the autocorrelation matrix C of size $3F \times 3F$ is computed as $C = S \cdot S^T$. Dimension of the autocorrelation matrix C is proportional to the number of frames F, thus the computation complexity is independent of the number of mesh vertices V (in the contrast with spectral mesh compression, for example). If the number of animation frames (the dimension of the correlation matrix) is too high, the frame sequence can be trivially cut into several shorter pieces. Next, we use eigenvalue decomposition of the autocorelation matrix C to obtain a set of eigenvectors $E_i, i = 1 \dots 3F$, whose subset forms a basis of the desired subspace. This subset consists of the N most important eigenvectors (according to their respective eigenvalues), where N is a user-specified parameter. The trajectory vector of the i-th vertex is then expressed as:

$$T_i = A + \sum_{j}^{N} c_i^j E_j. \qquad (1)$$

To use this formula, we have to compute the matrix of combination coefficients c_i^j by matrix multiplication $C = S^T \cdot E$. E is a matrix of a selected subset of eigenvectors of size $3F \times N$, where each column corresponds to one of these eigenvectors. The vector of combination coefficients c_i related with the i-th vertex of the dynamic mesh is called a *feature vector*. Now, the mesh can be described by the matrix of the selected subset of eigenvectors E, the matrix of combination coefficients C and the average trajectory vector A. Finally, the COBRA (Váša and Skala, 2009) algorithm is used to compress the PCA basis and thus reduce the size of this data by approximately 90% with respect to a direct encoding.

The PCA step of the compression algorithm can be interpreted as a simple change of basis. This enables us to use linear operators without any influence on their results. This observation is used for the following algorithm efficiency improvement: The mesh surface is progressively traversed by expanding the processed area by one edge-adjacent triangle at a time. All three feature vectors of the vertices of the first processed triangle are stored without any modifications, but each following feature vector corresponding to an added triangle is predicted from already compressed vectors using parallelogram prediction:

$$c^j_{predicted} = c^j_{left} + c^j_{right} - c^j_{base}, j = 1 \dots N, \qquad (2)$$

After the parallelogram prediction the residues between the original and the predicted feature vectors are stored.

This compression scheme compresses vertex trajectories using PCA and therefore the more complicated the mesh movement is, the less information about the mesh movement can be considered negligible and the compression ratio decreases. In other words, the efficiency of the Coddyac compression algorithm directly depends on the mesh movement complexity. By the movement complexity we understand differences between individual vertex trajectories. The vertex trajectories can be complicated, but if they are similar to each other, the movement complexity of the set of these trajectories is low.

The clustering improvement of the basic Coddyac algorithm lies in clustering the mesh vertices depending on mutual similarity of their trajectories, which leads to local reduction of the movement complexity (complexity in one cluster is lower than in the whole mesh). Lower movement complexity enables shortening of feature vectors while maintaining the same error and thus improves the compression ratio.

Experiments made in this area have shown that the k-means (Kanungo et al., 2002) clustering of vertex trajectories with L_1 distance norm is the most suitable for Coddyac and it can improve compression bitrate by 37%-46% without changing the error caused by compression.

In the preceding text, N denotes a user-specified number of basis vectors, but when clustering is a part of the compression algorithm, then it is necessary to change the approach to setting the number of basis vectors. Assuming that each vertex cluster has different movement complexity, we also have to set a different number of basis vectors to maintain the same accuracy of movement of individual parts of the compressed dynamic mesh. To do so, we use a single scalar value instead of the set of N-values, which represents a tolerable PCA-introduced error specified by the user. The modified Coddyac algorithm calculates the number of basis vectors automatically for each vertex cluster. The average amount of PCA-introduced error using N basis vectors is expressed as:

$$\Delta_{PCA}^{N} = \frac{1}{V}\sum_{j=1}^{V}\frac{1}{3F}\sum_{i=1}^{3F}\frac{|T_j^i - \overline{T_j^i}|}{l}, \qquad (3)$$

where l is the average edge length of the animation, T_j^i is the i-th component of the j-th original trajectory and $\overline{T_j^i}$ is the i-th component of the j-th trajectory reconstructed using N basis vectors. In order to select the number of basis vectors for a specific cluster, we select the smallest possible N for which this average PCA-introduced error falls below a user-specified value.

3 MESH DEBLOCKING

As mentioned in the introduction, some distortion of a decompressed mesh may appear when using lossy compression algorithms with clustering. A significant part of this distortion is usually caused by the data quantisation step, which is essential for efficient data storage.

There are two basic ways in which this factor may affect the mesh surface. First, all mesh vertices are globally affected by noise-like distortion, inflicted by quantisation of vertex coordinates. In this paper, we are interested in the second kind of distortion, which affects the mesh surface only locally and may be caused for example by quantisation of PCA basis vectors or quantisation of local coordinate systems of vertex clusters. It is possible that artifacts in the form of cracks or steps between individual vertex clusters (parts of the mesh) are produced by mesh compression.

Using the exact knowledge of mesh division used during the phase of mesh decompression, we are able to locate borders of individual vertex clusters and thus the areas of the mesh surfaces where the described artifacts may appear. By mesh deblocking we want to suppress or even remove these artifacts by moving vertices of triangles to a more suitable position with respect to the problematic boundary of appropriate clusters. The goal of this process is not to reduce the distance between original and decompressed positions of vertices, but to increase the visual quality of

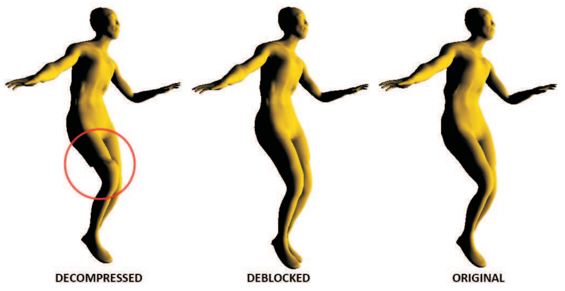

Figure 1: Comparison of decompressed, deblocked and original frame of the dance animation. 13.6% improvement of the STED error is achieved by deblocking.

the decompressed mesh. The result of such operation can be seen in figure 1.

We avoid the deformation of the processed cluster such as mesh smoothing, and thus bringing additional error to inner vertices of this cluster (not only on the borders). We believe that a rigid transformation, which affects all cluster vertices regularly (in the same way), is a better idea. This transformation should meet two requirements:

- Minimal dihedral angle between pairs of triangles on cluster boundaries is required to minimise the step-like artifacts.
- Reasonable length of edges between clusters.

We can use a combination of the parallelogram prediction with the rigid transformation to fulfil the specified requirements and fit the cluster boundaries better to soften the transitions between them.

3.1 Algorithm Description

The mesh deblocking algorithm starts with finding edges, that connect borders of different clusters. Triangles containing these edges form a set of border triangles (figure 2), defining areas of mesh distortion described earlier and potentially forming step-like artifacts between individual clusters.

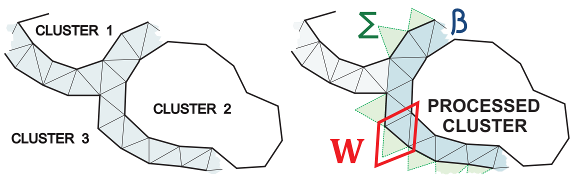

Figure 2: Left: clusters surrounded by (darkened) border triangles. Right: green Σ triangles and blue β triangles surrounding processed cluster forming wedges. One of the wedges W is highlighted in red.

Next, the best candidate cluster for transformation (deblocking) is selected. We choose such cluster, which can bring the most significant improvement in the mesh visual quality using a proper rigid transformation. Estimation of this improvement is based on the width of the surrounding strip of corresponding border triangles β (the distance of the transformed cluster from the incident clusters).

Estimation of the artifact size reduction is calculated using an arithmetic average of the lengths of the edges of the incident border triangles, which are connected to adjacent clusters. Candidates for the deblocking transformation are iteratively selected and transformed while the estimated reduction of artifact size is decreasing. In other words, when the difference between two consecutive estimations is lower than a specified threshold (we choose 0.1% in our experiments), the iterative algorithm stops. Calculation of an estimation of the artifact size reduction will be described later in this section.

Once the candidate cluster for the transformation is selected, the transformation algorithm finds all surrounding triangles Σ from adjacent clusters, which have one edge in common with any triangle in β. These Σ triangles are used to form a set of triangle wedges W consisting of one Σ triangle and one β triangle sharing a common edge, as depicted in figure 2.

In other words, each wedge includes an Σ triangle lying out of the processed candidate cluster, which can be used for a parallelogram prediction of the position of the tip of the β triangle lying on the boundary of the processed candidate cluster (see figure 3) and thus wedges can be used for an estimation of position of the vertices on the boundary of the processed cluster after the desired deblocking transformation.

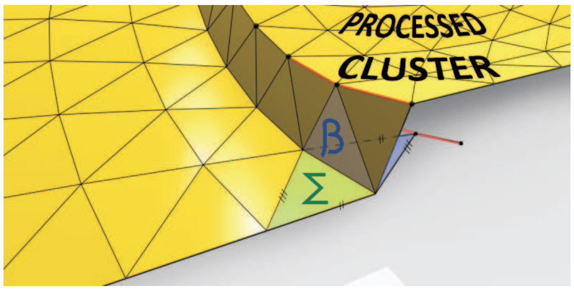

Figure 3: Green Σ triangle used for parallelogram prediction of boundary vertex position of the processed cluster after deblocking transformation.

Now, the position of the boundary vertices of the processed cluster can be estimated using the parallelogram prediction from clusters surrounding the processed candidate cluster.

Parallelogram prediction is used to guarantee a meaningful length of edges connecting the boundary of the processed cluster with the rest of the mesh. It also minimises the dihedral angles between the triangles of the processed cluster and the surrounding clusters.

A difference vector between the decompressed position $d_i \subset D$ and the predicted position $p_i \subset P$ of the i-th vertex belonging to the boundary of the processed cluster is denoted as force $f_i \subset F$ and determines the direction and the distance in which this vertex has to be moved to fulfil the transformation requirements. Each boundary vertex can be affected by a number of forces and thus the total force vector affecting this vertex is calculated as:

$$f_i = \frac{1}{M_i} \sum_{j=1}^{M_i} (p_i^j - d_i^j), \tag{4}$$

where M_i denotes the number of forces affecting the i-th border vertex.

Using this approach, two sets of vertices are obtained. A set of boundary vertices with the original (decompressed) position D and a set of boundary vertices with the desired position P, $p_i = d_i + f_i$, both of the size of M elements. In the following step of deblocking algorithm a rigid transformation of D to P is calculated. We do not use vertices with no force assigned to them in the following calculations, therefore P, D and F have the same number of elements.

To find the appropriate transformation, we use SVD-based method of fitting of two 3D point sets described in (Arun et al., 1987) obtaining matrix of rotation R and vector of translation t.

In some special cases, the calculation of the matrix of rotation R may fail and thus R does not describe the rotation we are looking for. In such cases, only the translation vector t is calculated using the difference between the centroids of the given sets. The calculation of the difference between the centroids of D and P an thus the calculation of the vector of translation as well is equivalent to the calculation of the average force vector $\overline{f}$:

$$\begin{aligned} t &= \frac{1}{M}\sum_{i=1}^{M} p_i - \frac{1}{M}\sum_{i=1}^{M} d_i \\ &= \frac{1}{M}\sum_{i=1}^{M} (d_i + f_i) - \frac{1}{M}\sum_{i=1}^{M} d_i \\ &= \frac{1}{M}\sum_{i=1}^{M} f_i = \overline{f} \end{aligned} \tag{5}$$

Finally, all vertices d_i of the processed cluster are transformed using the rigid transformation obtained

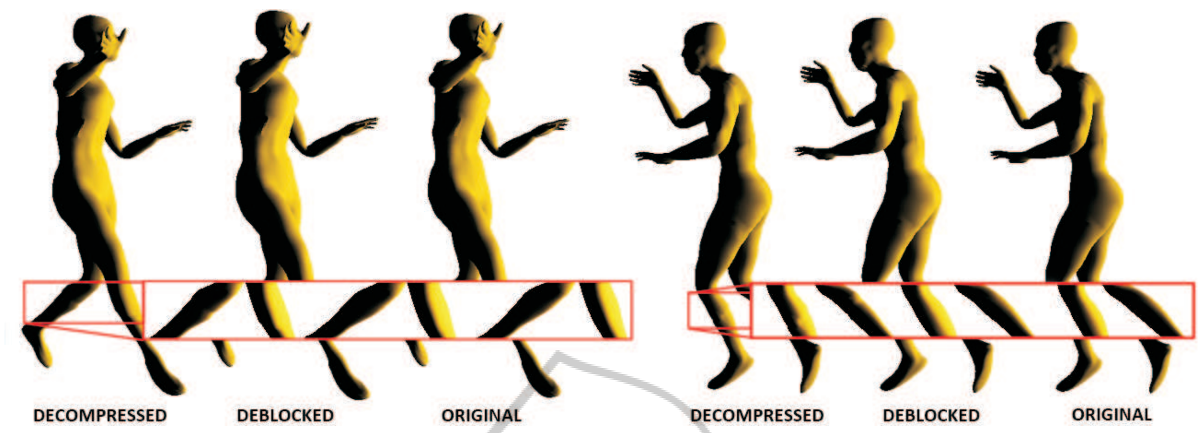

Figure 4: Frames 21 and 45 of the dance animation decompressed, deblocked and compared with the original animation. Despite the fact that the animation was compressed quite accurately with about 0.45 bpfv and STED 0.015, the mesh deblocking brought almost 10% improvement of the STED error of this data.

by the application of SVD on the sets of border vertices O and P. In other words, the whole cluster is rotated and translated in order to minimise the size of the artifacts on its boundary. The transformed vertex position $\tilde{p}_i \subset \tilde{P}$ close to the predicted vertex position p_i of the i-th vertex of the processed cluster is then calculated as follows:

$$\tilde{p}_i = R \cdot d_i + t \tag{6}$$

If the calculation of the rotation matrix R fails, the whole processed cluster is only translated in the direction of average force vector $\overline{f}$. This cluster is discarded from the list of possible candidates for the next deblocking transformation iteration, because it could be evaluated as the best candidate for transformation again without any chance to further improve the visual quality of the mesh.

Earlier in the text, it was mentioned that the change of the artifact size related to a given cluster has to be estimated to identify which cluster should be transformed to gain probably the most significant improvement of the visual quality of the mesh in the current iteration. The estimation algorithm emulates the transformation algorithm until it reaches the phase of application of the rigid transformation on all cluster vertices. Next, the average length l_{avg} of force vectors assigned to boundary vertices of the tested cluster is computed:

$$l_{avg} = \frac{1}{M}\sum_{i=1}^{M} |f_i| \tag{7}$$

The value of l_{avg} is used as an approximation of the visual quality improvement acquired by the application of deblocking transformation on this cluster. This average length value is calculated for each cluster. The candidate cluster with the largest l_{avg} is chosen as the best candidate for the deblocking transformation.

The whole deblocking algorithm is briefly described in the following pseudocode:

```
repeat
{
  select the best candidate cluster for deblocking
     -parallelogram prediction of border vertices
     -computation of forces in border vertices
     -calculation of average force length l_avg
     -selection of the best candidate

  if(best_l_avg > threshold * last_best_l_avg) break
  else store best_l_avg as last_best_l_avg

  calculate transformation for selected cluster
     -parallelogram prediction of border vertices
     -computation of forces in border vertices
     -SVD to obtain rotation R and translation t
     -if SVD failed, calculate translation t

  if(SVD failed) translate cluster using t
  else rotate and translate cluster using R and t
}

store deblocked mesh
```

4 EXPERIMENTAL RESULTS

The approach of the mesh visual quality improvement described in this paper has been implemented and subsequently, we have measured the impact of the mesh deblocking on a set of selected dynamic meshes. During our experiments, we were more interested in the visual quality of the compressed meshes than in the error measured by commonly used error metrics,

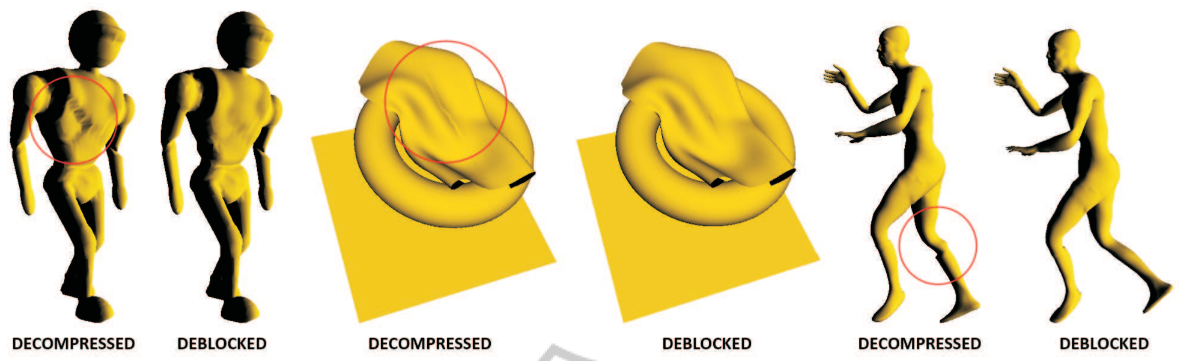

Figure 5: Comparison of meshes decompressed with deblocking and without deblocking.

Table 1: Comparison of results of mesh deblocking.

Animation	Bitrate [bpfv]	Original Error	Deblocked Error	Improvement
Chicken	0.1509	0.1311	0.1162	11.32%
	0.4467	0.0383	0.0381	0.58%
	0.5062	0.0298	0.0307	-3.19%
Cloth	0.1610	0.0620	0.0507	18.22%
	0.3319	0.0184	0.0177	3.92%
	0.6934	0.0105	0.0103	2.06%
Dance	0.2090	0.0680	0.0606	10.80%
	0.2835	0.0448	0.0421	6.09%
	0.4300	0.0191	0.0184	3.64%
Humanoid	0.1469	0.1030	0.0986	4.26%
	0.3051	0.0268	0.0258	3.92%
	0.3914	0.0127	0.0135	-6.32%
Jump	0.1576	0.1204	0.1124	6.65%
	0.3308	0.0737	0.0716	2.86%
	0.9749	0.0292	0.0292	-0.01%

such as KG-error. Therefore, STED has been chosen to measure the impact of the mesh deblocking of the decompressed meshes.

Spatio-temporal edge difference (STED) is a novel method of measuring error caused by processing of dynamic mesh. It has been derived from results of a subjective testing of mesh distortion perception published in (Váša and Skala, 2011). STED combines the measurements of spatial and temporal deviation of edge lengths caused by processing the mesh and it is focused on local changes of error. The distortion is evaluated only for a close neighbourhood of each vertex and these values are summed to obtain the overall error. STED error is defined as a weighted combination of the spatial error $STED_s$ and the temporal error $STED_t$:

$$STED(d,w,dt,c) = \\ = \sqrt{STED_s(d)^2 + c^2 \cdot STED_t(w,dt)^2}, \quad (8)$$

where c is a weighting coefficient, dt is the temporal distance between consecutive frames, d denotes topological distance (a vertex is maximally d edges distant from the given vertex) and w represents the width of the temporal window.

We tested the mesh deblocking algorithm on several 3D dynamic meshes, such as Dance, Chicken and Jump. These meshes contain only information about vertex positions for individual frames and connectivity of their triangles. The experimental data-set consists of artificial animations of single objects, multiple separated objects and scanned real data. We have experimented with different compression settings as well. Unlike the original version, the decompressed meshes processed by mesh deblocking can achieve lower STED error while maintaining the same bitrate. The improvement of the STED error depends on how precisely the mesh is compressed and thus on the compression bitrate. A lower bitrate means more distortion that can be suppressed by the mesh deblocking. On the other hand, a high bitrate can lead to a mesh so close to the original that the mesh deblocking can bring additional error, because it is only estimating the correct shape of the mesh depending on the close surroundings of the step-like artifacts. This case of negative influence of the mesh deblocking can be handled by additional information appended to the compressed mesh, defining whether it is advantageous to adjust the mesh by the deblocking algorithm.

Even though the modification of the mesh by deblocking affects only a close neighbourhood of the triangles forming the step-like artifacts between individual clusters, it can lead to a significant improvement of STED error and the visual quality of the mesh. STED error can be decreased by up to 18% using mesh deblocking, which can lead to a significant improvement of visual quality of the mesh as well. It may happen, that the dynamic mesh is compressed so precisely that visible cracks or step-like artifacts caused by clustering appear only in a small subset of the animation frames (figure 4). Suppressing these artifacts may lead to a small improvement of STED error, while significantly improving the visual quality of the small subset of the frames of animation.

The results of the mesh deblocking for several different dynamic meshes are shown in figure 5. The results of the selected subset of experiments are given in table 1. Only results of optimal configurations are presented in the table. Rate-distortion optimal configurations (number of basis vectors, quantisation of basis vectors and quantisation of feature vectors) were determined for the clustered Coddyac compression of dynamic meshes with the deblocking post-process and without it and compared to each other. There are sub-optimal configurations with even larger improvement (20%–35%) of the STED error in comparison with the original (not deblocked) meshes, but it is difficult to determine the contribution of such compression configurations which can not be compared with optimal results derived from the rate-distortion curve.

Note that mesh deblocking becomes inefficient for compression configurations with bitrate close to 0.5 bpfv (bits per frame and vertex) or higher. The reason for this behaviour is that the clustered Coddyac compression algorithm is able to compress dynamic meshes with almost no distortion using such a bitrate. Therefore, there is a low probability of the occurrence of the step-like artifacts between individual clusters we are dealing with in this paper. For example, the dance animation in figure 4 is compressed with a bitrate of 0.46 bpfv and contains only 3 or 4 frames that are visibly distorted by the clustered Coddyac compression. Two of these distorted frames are shown in figure 4. Similarly the dance animation in figure 1 is compressed with a bitrate of 0.19 bpfv and contains visible step-like artifacts, but these artifacts are successfully suppressed by the mesh deblocking.

5 CONCLUSIONS

We have presented a mesh deblocking algorithm targeted on the reduction of artifacts in clustering-based mesh compression schemes. We have demonstrated its efficiency on the example of clustered Coddyac algorithm and STED error measure for dynamic meshes, where we achieve a reduction of the compression-introduced error by up to 18% without the need for any additional data. Our result is confirmed in the presented images, where the artifact reduction is clearly visible. A similar implementation may be used for post processing of static meshes, which is also confirmed by our preliminary experiments in this application.

Apart from a thorough testing of the deblocking process for static meshes, in the future, we also intend to improve the processing speed of our algorithm to allow the real-time processing of very large meshes. We also intend to perform experiments with more complex deblocking approaches using a transformation that is smoothly changing across each cluster rather than a rigid one, as in this work. We also plan to use similar techniques to reduce temporal errors (frame shaking) that may arise in dynamic mesh processing.

ACKNOWLEDGEMENTS

This work has been supported by the Ministry of Education, Youth and Sports of the Czech Republic under the research program LC-06008 (Center for Computer Graphics).

REFERENCES

Amjoun, R. and Straßer, W. (2007). Efficient compression of 3d dynamic mesh sequences. *Journal of the WSCG*.

Arun, K. S., Huang, T. S., and Blostein, S. D. (1987). Least-squares fitting of two 3-d point sets. *IEEE Trans. Pattern Anal. Mach. Intell.*, 9:698–700.

Desbrun, M., Meyer, M., Schröder, P., and Barr, A. H. (1999). Implicit fairing of irregular meshes using diffusion and curvature flow. In *Proceedings of the 26th annual conference on Computer graphics and interactive techniques*, SIGGRAPH '99, pages 317–324, New York, NY, USA. ACM Press/Addison-Wesley Publishing Co.

Kanungo, T., Mount, D. M., Netanyahu, N. S., Piatko, C. D., Silverman, R., Wu, A. Y., Member, S., and

Member, S. (2002). An efficient k-means clustering algorithm: Analysis and implementation. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 24:881–892.

Karni, Z. and Gotsman, C. (2000). Spectral compression of mesh geometry. In *SIGGRAPH*, pages 279–286.

Karypis, G. and Kumar, V. (1998). Metis: A software package for partitioning unstructured graphs, partitioning meshes, and computing fill-reducing orderings of sparse matrices version 4.0. In *University of Minnesota, Department of Comp. Sci. and Eng., Army HPC Research Center, Minneapolis*.

Lavoué, G. (2011). A Multiscale Metric for 3D Mesh Visual Quality Assessment. *Computer Graphics Forum (Proceedings of Eurographics Symposium on Geometry Processing 2011)*, 30.

Lavoué, G., Drelie Gelasca, E., Dupont, F., Baskurt, A., and Ebrahimi, T. (2006). Perceptually driven 3D distance metrics with application to watermarking. In *SPIE Applications of Digital Image Processing XXIX*.

Lengyel, J. E. (1999). Compression of time-dependent geometry. In *In I3D 99: Proceedings of the 1999 symposium on Interactive 3D graphics*, pages 89–95. ACM.

Mamou, K., Zaharia, T. B., and Prêteux, F. J. (2008). Famc: The mpeg-4 standard for animated mesh compression. In *ICIP*, pages 2676–2679.

Marpe, D., Schwarz, H., Blttermann, G., Heising, G., and Wieg, T. (2003). Context-based adaptive binary arithmetic coding in the h.264/avc video compression standard. *IEEE Transactions on Circuits and Systems for Video Technology*, 13:620–636.

Müller, K., Smolic, A., Kautzner, M., Eisert, P., and Wiegand, T. (2005). Predictive compression of dynamic 3d meshes. In *ICIP (1)*, pages 621–624.

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

Rus, J. and Váša, L. (2010). Analysing the influence of vertex clustering on pca-based dynamic mesh compression. In *Articulated Motion and Deformable Objects*, pages 55–66, Heidelberg. Springer.

Sattler, M., Sarlette, R., and Klein, R. (2005). Simple and efficient compression of animation sequences. In *SCA '05: Proceedings of the 2005 ACM SIGGRAPH/Eurographics symposium on Computer animation*, pages 209–217. ACM Press.

Váša, L. and Skala, V. (2007). Coddyac: connectivity driven dynamic mesh compression. In *3DTV-CON 2007*, pages 1–4, Piscataway. IEEE.

Váša, L. and Skala, V. (2009). Cobra: Compression of the basis for pca represented animations. *Computer Graphics forum*, 28(6):1529–1540.

Váša, L. and Skala, V. (2011). A perception correlated comparison method for dynamic meshes. *IEEE Transactions on Visualization and Computer Graphics*, 17(2):220–230.

Wang, Z., Bovik, A. C., Sheikh, H. R., and Simoncelli, E. P. (2004). Image quality assessment: From error visibility to structural similarity. *IEEE TRANSACTIONS ON IMAGE PROCESSING*, 13(4):600–612.

Zhang, J. and Owen, C. B. (2004). Octree-based animated geometry compression. In *Data Compression Conference'04*, pages 508–520.