University of West Bohemia
Faculty of Applied Sciences
Department of Computer Science and Engineering

# Diploma Thesis

# Dynamic Mesh Compression With Clustering

Pilsen, 2011 Jan Rus

**Empty list**

## ACKNOWLEDGEMENTS

First of all, I would like to thank Ing. Libor Váša, Ph.D. for providing kind guidance to me throughout the development of this thesis. I would also like to thank my colleagues Ing. Oldřich Petřík and Ing. Jiří Skála who provided me with valuable advices in the area of data clustering and appropriate implementations. Last but not least, I want to thank my parents for their support and help. Without them I would never have come to where I am right now.

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).

I hereby declare that this diploma thesis is completely my own work and that I used only the cited sources.

Pilsen, 16/5/ 2011 ....................................

Jan Rus

## ABSTRACT

The growth of computational power of contemporary hardware causes an expansion of technologies working with 3D-data. Examples of the use of this kind of data can be found in geography or gaming industry. 3D-data may not only be static, but also dynamic.

One way of animated 3D-data representation is expressing them by "dynamic triangle mesh". This kind of data representation is usually voluminous and needs to be compressed for efficient storage and transmission. In this thesis, we are dealing with the influence of vertex clustering on dynamic mesh compression. The mesh is divided into vertex clusters based on the vertex movement similarity and compressed per-partes to achieve higher compression performance. We use Coddyac as a basic compression algorithm and extend it by adding well known clustering algorithms to demonstrate the efficiency of this approach. We also deal with what clustering is the most appropriate for Coddyac and what number of clusters is optimal.

***Keywords:*** 3D dynamic meshes, Data compression, Computer animation, Coddyac, Clustering

## ABSTRAKT

Vlivem neustálého růstu výpočetního výkonu současného hardware se stále více dostávají do popředí technologie pracující s 3D daty. Příklady využití tohoto druhu dat můžeme nalézt například v geografii nebo v herním průmyslu. 3D data nemusí být pouze statická (ve formě modelů), ale rovněž mohou být dynamická (3D animace).

Jedním ze způsobů reprezentace animovaných 3D dat je jejich vyjádření s pomocí „dynamické trojúhelníkové sítě“. Tento druh reprezentace bývá velmi objemný a pro potřeby efektivního skladování a sdílení je třeba data komprimovat. Tato práce je zaměřena na vliv shlukování vrcholů na kompresi dynamických trojúhelníkových sítí. Vrcholy trojúhelníkové sítě jsou rozděleny do shluků na základě podobnosti jejich trajektorií v průběhu animace a každý shluk je pak komprimován samostatně ve snaze dosáhnout lepšího kompresního poměru. Jako základní kompresní algoritmus je použit Coddyac, rozšířený o známé shlukovací algoritmy tak, aby byla demonstrována efektivita tohoto přístupu. V této práci je rovněž řešena otázka nejvhodnějšího shlukování pro Coddyac a optimálního počtu shluků.

***Klíčová slova:*** dynamická trojúhelníková síť, komprese dat, počítačová animace, Coddyac, shlukování

**TABLE OF CONTENTS**


**1 Introduction 3**

1.1 Models and Animations ....3

1.2 Dynamic Triangle Mesh ....4

1.3 Dynamic Mesh Compression ....5

1.4 Aim of the Thesis ....6

1.5 Organisation ....6

1.6 Notation ....7

**2 Related Work 8**

2.1 Quantisation ....8

2.2 Entropy Coding ....9

2.2.1 Huffman Coding ....10

2.2.2 Arithmetic Coding ....11

2.3 Principal Component Analysis ....12

2.4 Static Mesh Compression ....14

2.4.1 EdgeBreaker ....15

2.4.2 Delta Coding and Prediction ....17

2.5 Dynamic Mesh Compression ....19

2.5.1 Dynapack ....20

2.5.2 Shape-space Based Compression ....22

2.5.3 D3DMC ....24

2.5.4 FAMC ....27

2.5.5 Coddyac ....30

2.5.6 COBRA ....31

2.5.7 Vertex Clustering and 3D Mesh Registration ....32

2.6 Error Measurements ....................................................................................................33

2.6.1 Mean Squared Error ....................................................................................33

2.6.2 KG-error ....................................................................................................34

2.6.3 STED ..........................................................................................................34

**3 Clustered Coddyac** **37**

3.1 Clustering in Coddyac ....................................................................................38

3.1.1 K-means ......................................................................................................39

3.1.2 K-means initialization ...............................................................................41

3.1.3 Facility Location .........................................................................................42

3.1.4 Edge-collapse Based Clustering ................................................................43

3.2 Number of Basis Vectors ...............................................................................45

3.3 Compression Scheme ....................................................................................45

3.4 Raised Problems ............................................................................................47

**4 Experimental Results** **52**

4.1 Influence of Clustering on Data-size .............................................................54

4.2 Influence of Clustering Methods ...................................................................57

**5 Conclusion and Future Work** **64**

**Appendix A – Implementation Overview** **69**

# 1 INTRODUCTION

In these days computers are widespread technology bringing information and entertainment to many households, not just scientific institutions and societies. Hence most of the distributed data has audiovisual entertaining character –people mostly share movies, music and computer games.

In the last few years movies, computer games and some industry sectors have one common trend. Latest technologies in these branches of entertainment and industry tend to a massive use of 3D data in the form of 3D models (meshes) and 3D animations (dynamic meshes) and we expect that this trend will continue.

## 1.1 3D Models and Animations

Many different sources of 3D models and animations are known. Static 3D models can be for example created by game developers, graphic designers or architects in 3D modelling software. Real objects can be transformed to their 3D model representation using contact or optical scanners and triangulation techniques. 3D models may be result of computer simulations as well. These 3D models usually consist of a set of triangles defining the surface of the model -3D triangle mesh.

Generally only information about geometry and connectivity of triangles is used to describe a 3D triangle mesh, but normals, texture coordinates or colours can be attached to this mesh description as well. 3D mesh geometry is defined by a set of vertices and each vertex contains a triplet of coordinates. These coordinates are often very precisely expressed, thus geometry constitutes the major part of 3D mesh data.

Naive connectivity describing schemes of triangle meshes contain only indices of triplets of vertices forming the triangles separately for each triangle. However, there are more advanced and data-saving schemes describing which vertices have triangles in common in a mesh and which triangles are neighbouring and what way.3D triangle meshes can be dynamic as well.

## 1.2 Dynamic Triangle Mesh

There are two basic ways how to describe a model animation, or in other words, a dynamic triangle mesh. The first way is to describe the animation by a sequence of meshes where each mesh represents one frame of the animation. These meshes can vary in number of vertices or number of triangles. Disadvantage of this kind of animation description is that it has to contain triangle connectivity description for each frame of the animation even if it does not change.

There is another way how to describe this kind of data. If the connectivity of triangles of animated mesh is constant in the time and just geometry is changing, we can describe the animation by a single mesh, where for every vertex of the mesh there is a vector of values describing its trajectory throughout the animation instead of common XYZ coordinates. Put simply, geometry is stored for each frame, but triangle connectivity is stored just once for the whole animation. Such dynamic mesh representation can be obtained from the one mentioned above by preserving the first mesh of the animation set (for triangle connectivity description) in the first step. In the second step we take coordinates of topologically corresponding vertices from each mesh of the animation set and store them to a vector as a trajectory, figure 1.

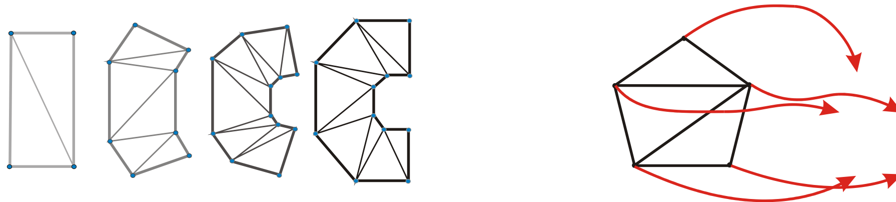


**Fig. 1** Different representations of mesh animation. Series of meshes vs. mesh with constant topology.

The single mesh representation is much more data-saving than the first one because we do not have to store the connectivity of its triangles for each frame of animation. But it is still a relatively large amount of data. The simplest way of connectivity representation is to represent the corners of each triangle of the mesh by indices in the index table related with appropriate vertex coordinates in the vertex table. This two-table representation is used to avoid re-storing of vertex coordinates for each of multiple triangle corners, which have this vertex in common. However, this representation still needs $3T\lceil\log_2 V\rceil$ bits to index vertices of all triangles, where $T$ denotes number of triangles and $V$ denotes number of vertices of the dynamic mesh.

## 1.3 Dynamic Mesh Compression

In general data accuracy and quantity requirements are continually growing and similarly grows the volume of data structures which contain them. Because storage capacities and transmission speeds are limited we need to use compression algorithms to reduce data volume and reduce hardware requirements for storage and distribution of such data.

Unfortunately data of dynamic triangle meshes includes a lot of complex information. Therefore it is very voluminous and needs to be compressed for efficient storage and transmission. ZIP and RAR are popular compression algorithms but they are not primarily intended for dynamic mesh compression. Like video or audio, this kind of data also should be compressed by a specialized compression method to enhance efficiency of transmission and storage.

In the case of dynamic triangle meshes we can use for example PCA-based dynamic mesh compression algorithm called Coddyac [1], which is specialized in this kind of data compression and belongs to the most efficient compression algorithms we know. Coddyac is being developed at University of West Bohemia at Department of Computer Science and Engineering.

Unlike the ZIP compression, Coddyac compression is a lossy compression algorithm, but this may not be an obstacle due to the kind of data. Usual video compression algorithms are also lossy to achieve better compression rates. The task of lossy compression is the reduction of data volume by data encoding method which compresses the data by discarding (losing) some of it. Lossy compression methods are most commonly used to compress multimedia data and this kind of compression uses the imperfection of human perception.

In the context of this thesis it should be mentioned that in usual cases of dynamic mesh compression, topology of meshes is compressed by a lossless compression scheme (an observer is likely to notice missing parts of mesh) and the geometry of meshes is compressed by a lossy compression scheme (not notice less accurate positions of vertices) using quantization, entropy coding or principal component analysis.

## 1.4 Aim of the Thesis

This thesis is focused on improvement of PCA-based dynamic mesh compression algorithms by vertex clustering and it is based on the paper [23] we have presented on the prestigious international Conference on Articulated Motion and Deformable Objects. We choose Coddyac as a representative of this type of compression algorithms.

Principal Component Analysis (PCA) [2] is a very important part of Coddyac algorithm. It is used to simplify the description of vertex trajectories in a compressed dynamic mesh. Efficiency of this simplification, and thus results of the entire compression algorithm, directly depends on mesh movement complexity. More complex movement of vertices in an animation means that less data can be neglected and worse compression ratios are achieved. Therefore reduction of movement complexity seems to be a straightforward way to improve Coddyac compression algorithm and PCA-based dynamic mesh compression algorithms in general. To achieve this kind of complexity reduction we can use clustering of vertex trajectories.

Later in this thesis we deal with improvement of Coddyac (and PCA-based compression algorithms in general) using a clustering of vertex trajectories of the dynamic triangle mesh and with its influence on compression algorithm efficiency. We also deal with what clustering is the most appropriate Coddyac and what number of clusters is optimal.

## 1.5 Organisation

The rest of this thesis is organized as follows: Section 2 gives an overview of some frequently used compression techniques and approaches suitable for static and dynamic mesh compression and background helpful for understanding the rest of the thesis. Coddyac algorithm and some related dynamic mesh compression algorithms are introduced and key features and algorithmic components of the original Coddyac algorithm are described here.

In Section 3 a clustering modification of Coddyac is presented, along with its brief scheme and fundamental problems raised by this modification. One such problem is closely related to topological decomposition of the mesh surface due to vertex clustering and its projection to lower dimension. Possible correction of this problem and description of the tested clustering algorithms is introduced in the same section. Experimental results are presented in Section 4 and the thesis is concluded in Section 5.

## 1.6 Notation

In this thesis we use following notation:

$F$ - number of frames of animation

$V$ - number of vertices in each frame

$T$ - number of triangles of the mesh

$E$ - number of edges of the mesh

$B$ - matrix of original animation, size $3F \times V$

$A$ - average trajectory vector

$S$ - matrix of samples, contains subtraction of $A$ from each column of $B$

$C$ - autocorrelation matrix

$D_i$ - i-th eigenvector, made by eigenvalue decomposition of $C$

$T_i$ - i-th trajectory

$N$ - number of used eigenvectors (components of PCA)

$D$ - basis of the PCA subspace, size $3F \times N$

# 2 RELATED WORK

This thesis is focused on compression of dynamic meshes by specialised compression algorithms. Like most specialised algorithms, dynamic mesh compression algorithms are based on more general compression methods. In the beginning of this section, standard compression methods used for general data compression are described. Then some static mesh compression algorithms are introduced, followed by specialised dynamic mesh compression algorithms related to this thesis. Most of them are lossy. Lossy compression methods cause data corruption, therefore end of this section deals with error measurements and methods of evaluating compression efficiency.

## 2.1 Quantisation

Quantisation is a frequently used method in multimedia compression taking advantage of the imperfection of human perception. It is a process of mapping a continuous set of input values to a smaller discrete set. This method is comparable to rounding values to some unit of precision. Thus it is a lossy data compression method. For example rounding a real number $x$ to the nearest integer value forms a uniform quantizer which can be expressed as:

$$Q(x) = sgn(x) \cdot \left\lfloor \frac{|x|}{q} + \frac{1}{2} \right\rfloor, \tag{2.1}$$

$$R(x) = q \cdot Q(x), \tag{2.2}$$

where the function $sgn()$ is the signum function, $q$ is a quantization step and $\lfloor x \rfloor$ brackets denotes rounding to nearest lower integer. The $R(x)$ function is used to maintain the original range of quantized values.

Geometry data of dynamic meshes are usually represented by vertex coordinates with triplet of 32-bit floats, but we usually do not need such accurate data.

For example 16-bit integers are enough to resolve 15µm details in a model of a human body, thus we can uniformly quantize each of vertex coordinates of original dynamic mesh with 16 bits and cause no visible corruption to geometry data if the mesh size and the size of required details is adequate to the mentioned scale.

Quantisation does not have to be uniform. As indicated by Chow [16], mesh can be separated into several regions with variable detail. These regions can be quantised with different precision, considering curvature and triangle sizes in each of them, to achieve better compression result. Sorkine et al. [15] propose a different approach to geometry quantisation. Geometry of compressed mesh is transformed by applying Laplacian operator associated with the mesh topology first, instead of quantising Carthesian coordinates directly. This approach results into low-frequency errors which are less noticeable by human visual system than the high-frequency errors (human visual system is more sensitive to normal distortion than to tangential distortion).

The quantization is very frequent pre-processing step for entropy encoding, because it can significantly reduce the original range of values and thus the entropy and positively influence the efficiency of entropy encoding.

## 2.2 Entropy Coding

Entropy is a measure of disorder (unpredictability) and is defined by following equation:

$$H(X) = -\sum_{i=1}^{n} p(x_i)\, log_b p(x_i), \tag{2.3}$$

where $X$ denotes discrete random variable with possible values $\{x_1,..., x_n\}$ and $p$ denotes probability mass function of $X$. More ordered values in data-set $X$ leads to lower entropy.

Large data-set of similar or the same elements within a small range of values is easy to compress efficiently by entropy encoding. It is because during entropy encoding the probability of occurrence of encoded symbol is detected and most frequently occurring symbols are then substituted by the shortest code.

Input data for entropy coding is usually quantized to a specified number of bits first. This is often a pre-processing step which positively influences the efficiency of entropy coding by reducing the original range of values. After this step, data is better prepared to be efficiently encoded by some entropy encoding algorithm before storage.

Entropic encoding is quite often used for data compression, because it is able to substitute very long symbol by very short code. Most common entropy coding algorithms are Huffman coding [26] and arithmetic coding [27].

### 2.2.1 Huffman Coding

Huffman coding is in computer science often used for lossless data compression. This coding system uses prefix codes. Prefix code is code system, where neither code word is a prefix of another code word. Prefix codes have variable length based on frequency (probability) of occurrence of symbols in coded data (shortest bit-code for most common symbol). This coding is the best method for fix-code generation (one substituting a code for each symbol of input alphabet). Huffman coding uses binary trees with leaves for each symbol from input alphabet of coded data structured according to the probability of occurrence of these symbols. When the binary tree is built, branches from its root to its leaves are traced to identify substituting prefix binary code for each symbol in the input data. This binary tree can be built by following algorithm:

1) Sort probabilities of symbols in input data.
2) Create a node combining the two smallest probabilities together.
   Sort the remaining probabilities together with created node.
3) Set“0” for a branch of created node, “1” for the rest.
4) Repeat step 2) and 3) until the created node probability is equal to 1.0 (root)

Example binary tree is shown in figure 2. It shows binary tree for input data with alphabet of 4 symbols, where symbol A is the most frequently occurring symbol, and therefore it is substituted with a code of length 1 bit (shortest possible code). Generated prefix binary codes consist of whole bits, thus the length of the substituted bit-code is not precisely adequate to the probability of encoded symbol occurrence in the coded data and the probability is rounded up. Therefore input data are coded with more bits than necessary and compression ratio of Huffman coding is not as effective as it could be. This problem is handled by arithmetic coding.

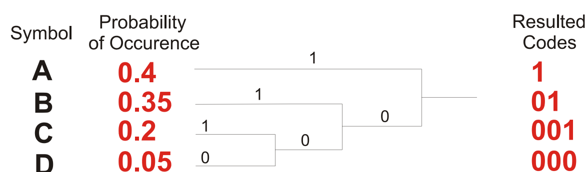



**Fig. 2** Example of binary tree for Huffman coding

### 2.2.2 Arithmetic Coding

Arithmetic coding is able to generate codes with fractions of bits and thus achieve better compression ratios. It is used for lossless data compression similar to Huffman coding. Arithmetic coding differs from Huffman coding in that it does not separate coded data into symbols, substituting each by a bit-code, but it represents the whole coded data-set using a single real number in the range of (0;1].

This interval is iteratively split into sub-intervals by arithmetic coding algorithm, proportionally to the probability of occurrence of each symbol in the coded data-set. When the whole set of symbols is encoded, the resulting interval identifies sequence of symbols that produced it. It is not necessary to store the resulting interval, it is only necessary to store one real number (fraction) lying in the range of this interval. Example of splitting sub-intervals by arithmetic coding is shown in figure 3.

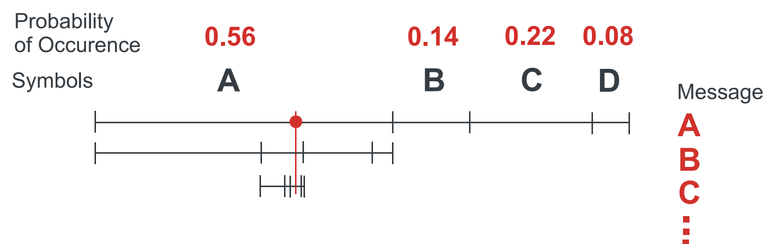



**Fig. 3** Example of splitting sub-intervals by arithmetic coding

Basic algorithm of arithmetic encoding can be improved by changing frequency (probability) tables during the process of data compression. This changing of frequency tables is usually based on symbol sequences occurring during the compression and decompression process. This modification of arithmetic coding can yield 2 or 3 times better compression ratio and it is known as context-adaptive binary arithmetic coding (CABAC) [18].

## 2.3 Principal Component Analysis

For the purpose of geometry compression many dynamic triangle mesh compression algorithms use the Principal Component Analysis (PCA). PCA is a statistical method used to find the directions of the largest data variance -principal components. These directions are used as axes of a new coordinate system and the original data are transformed into it. PCA method can reduce the dimensionality of a dataset, thus it is quite often used for the purposes of compression of digital images and models or computer face recognition, but it is also used for example to calculate tight bounding boxes [19].

Many dynamic triangle mesh compression algorithms process input data (dynamic meshes) in the form of single connectivity description and a set of vertex trajectories of individual vertices as explained in section 1.2. Trajectory of the i-th vertex is described by a vector $T_i$, consisting of $XYZ$ coordinates of the given vertex for each frame of animation, thus the length of the vector is $3F$. For dense meshes, the trajectory vectors are not spread randomly in the space of dimension $3F$, but they are located in subspace of much lower dimension. This is because vertex trajectories are spatially bound to animated mesh and it is very probable that neighbouring vertices will have similar trajectories. Due to this fact, first step of compression algorithms is finding the subspace and expressing the vertices in this subspace.

To find this subspace we can use the PCA tool of linear algebra. Let $B$ be the matrix of size $3F \times V$ representing the original animation, and the trajectory vector associated with the i-th vertex is stored in the i-th column of this matrix. Then $S$ is matrix of samples obtained by subtracting the average trajectory vector $A$ from each column of matrix $B$. The eigenvalue decomposition of the autocorrelation matrix $C$ of size $3F \times 3F$, computed by matrix multiplication $C = S \cdot S^T$ , is used to obtain a set of eigenvectors $D_i$, $i=1..3F$, and their corresponding eigenvalues.

These eigenvectors are sorted in order of their eigenvalues which specifies their importance and then $N$ first (most important) vectors are selected. $N$ is a user-specified parameter. Basis of the subspace we are looking for is formed by these selected eigenvectors. It is represented by a matrix $D$ of size $3F \times N$ (the i-th column is the i-th eigenvector $D_i$) and each trajectory vector can be expressed as:

$$T_i = A + \sum_{j=i}^{N} c_i^j D_j \tag{2.4}$$

Matrix of subspace basis $D$ is orthonormal. Thus the matrix of combination coefficients $c_i^j$ can be computed as $C = S^T D$. To be able to restore the original mesh we have to store the average trajectory vector $A$, matrix $C$ of size $V \times N$ with combination coefficients $c_i^j$ and subspace basis represented by matrix $D$ of size $3F \times N$. Efficient encoding of the subspace basis (matrix of selected eigenvectors) is described in section 2.5.6.

Basically the principal component analysis is a simple change of basis (figure 4). It does not have any influence on the results of linear operators. This observation can be exploited for prediction of the combination coefficients at the decoder. Prediction as a technique of static mesh compression is described in section 2.4.2.

Put simply, by using PCA on vertex trajectories during compression, we will obtain a new description of trajectories: vectors of PCA coefficients, known as feature vectors. These vectors consist of linear combination coefficients of principal components, which can be ordered by how much they affect the movement of vertices during the whole animation. Theoretically, in the case of randomly generated data, the principal components affect data variance to the same or very similar extent.

However, in the case of real data processed by compression algorithms the first principal component has the strongest influence on data variance and influence of the following components rapidly decreases. Reason for such trajectory transformation is the possibility to ignore less important components of PCA vectors and this way reduce the amount of data describing the original animation.

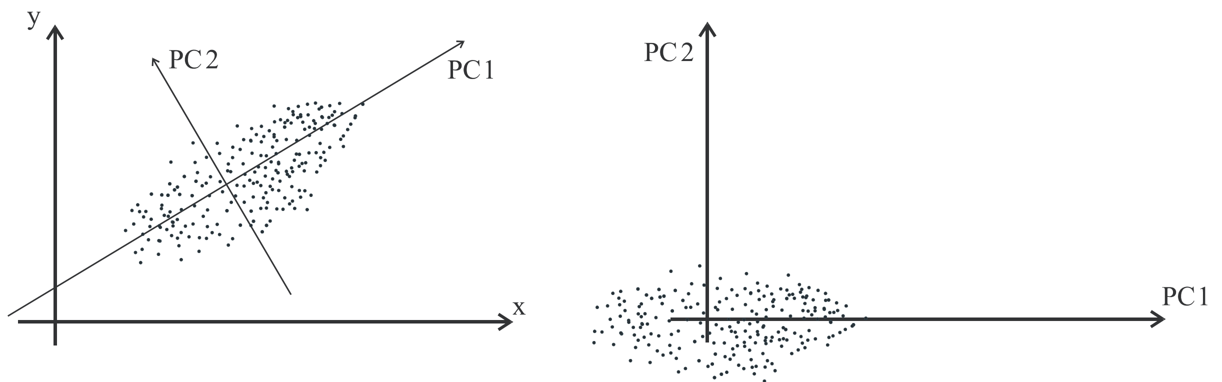



**Fig. 4** PCA change of basis for 2D point cloud

One such real case: tested animation has 200 frames, thus its trajectories have 600 components. Using PCA we detected that first 20 principal components describe mesh animation with error lower than user-specified threshold. User set this threshold to value where he/she was not able to recognise any difference between the original and compressed version of animated mesh. Thus we can neglect 95% of the data without visible loss of movement precision.

This part of compression algorithm is lossy, however, it does not affect the connectivity of the mesh, so for example compressed animation of a running woman may lose information about small movements like shivering of fingers or blinking, but it cannot happen that it would lose any part of her body.

The more complex the movements of an animated mesh are, the less movement can be considered negligible and thus the length of the feature vectors will have to be higher (decreasing compression ratio). Size of the compressed data depends especially on the rate of compression of these vectors, so more principal components decrease the compression ratio. Therefore, one possible way to improve the compression ratio is to reduce the movement complexity, which could be locally achieved by clustering the mesh vertices by similarity of their trajectories.

## 2.4 Static Mesh Compression

Static and dynamic meshes can be stored in files in textual formats (for example WRML) or more effectively in the form of binary bit-streams. What both these formats have in common is that they usually represent triangle meshes using two separate tables. Vertex table contains geometry information (vertices, normals,...) and another table (connectivity table) contains information about connectivity using indices to vertex table for vertices of triangles.

Using vertex table and connectivity table 1:6 compression ratio can be achieved for dense regular meshes without using any other compression method, because each vertex is shared by 6 triangles in average. But there are more sophisticated and more efficient methods of compression of static meshes focused on connectivity compression and geometry compression.

### 2.4.1 EdgeBreaker

EdgeBreaker is used to compress triangle connectivity of the mesh. EdgeBreaker is primarily intended for triangle connectivity compression of static meshes, but if the topology of a chosen dynamic mesh is constant throughout the time of animation(whole animation consists of a series of meshes but can be expressed as only one mesh, with vertices defined by trajectories instead of space coordinates) we can use this compression algorithm as well.

EdgeBreaker uses simple data structure for topology description in pre-processing step. This structure is called CornerTable and consists of two vectors. First vector $V_T$ contains ordered triplets of indices for each triangle and each entry of this triplet indexes vertex coordinates in vertex buffer corresponding with appropriate triangle corner. This vector is $3T$ long. If we have 2 triangles with one common edge, we can create them using 4 vertices in vertex buffer and 6 entries (one for each triangle corner) in the $V_T$ vector of CornerTable. Entries in CornerTable are ordered (0,1,2 for the first triangle, 3,4,5 for the second) thus we can simply determine to which triangle a selected corner belongs by dividing the corner index by 3.

The second CornerTable vector $O_T$ contains indices of opposite corners. If we have 2 triangles with a common edge, opposite corners are those 2 corners, which are not incident with the common edge. Each corner in the second vector has one opposite corner index from the first vector. In the case of missing opposite corner we can use index value of -1 indicating a topological hole in the described mesh.

This topology describing structure can be built with time complexity of O(N) using a hash function. CornerTable structure is depicted in figure 5.

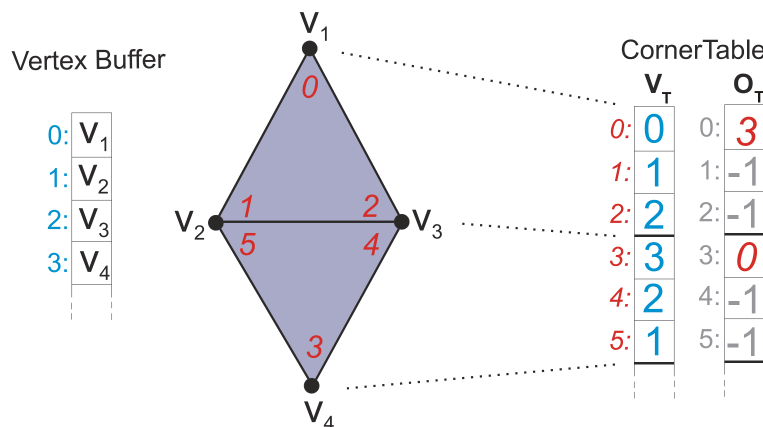



**Fig. 5** CornerTable structure for mesh with 2 triangles.

During the compression itself, EdgeBreaker traverses the triangles of triangle mesh and marks visited triangles and vertices to recognise which part of the mesh is already compressed and which triangle should be compressed next. The EdgeBreaker initial triangle is selected randomly and then one by one neighbouring triangles are visited. EdgeBreaker stores some information about currently visited triangle and then the triangle is left over one of its edges. Thus, the triangles are passed by crossing their common edges. That is why this algorithm is called EdgeBreaker.

During the traversal of the mesh 5 basic situations may arise. These situations are labelled by letters C, L, E, R, and S and depicted in figure 6. Occurring situations are noted into the CLERS string, and they precisely describe the desired mesh connectivity, because we know which triangles are neighbouring and what way. Compressed mesh connectivity is completely described just using the CLERS string. More information about EdgeBreaker compression can be found in [3].

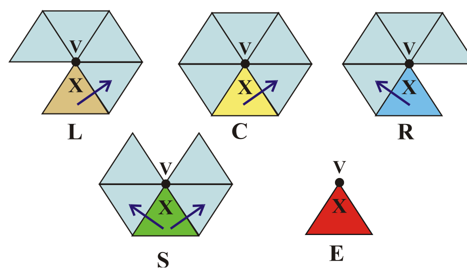



**Fig. 6** Situations coded by EdgeBreaker into CLERS string. Light blue triangles are still not visited, arrows show directions of the next EdgeBreaker step. V is vertex belonging to actually processed opposite corner.

Described method handles only simple meshes, which are topologically identical with sphere. This is inconvenient restriction and thus some modifications were proposed. In final form the EdgeBreaker algorithm is able to handle meshes with topological holes (missing triangles), meshes consisting of more than one topologically closed component (individual objects) and meshes with positive genus value.

The simplest mesh we know is a tetrahedron and more complex mesh with topology identical with sphere can be created from tetrahedron by simple operations. One such operation involves adding one vertex *v* and removing one triangle from mesh surface, which is replaced by 3 new triangles with common vertex *v*.

One operation results in 1 new vertex, 2 new triangles and 3 new edges, see figure 7. Hence we have following equation:

$$V + T - E - 2 = G, \tag{2.5}$$

where $G$ is genus and $V$, $T$ and $E$ are numbers of vertices, triangles and edges. For simple mesh with topology identical with sphere $G$=0. Meshes which contain "handles" have positive genus. For example torus has G=1 and spectacles have G=2.

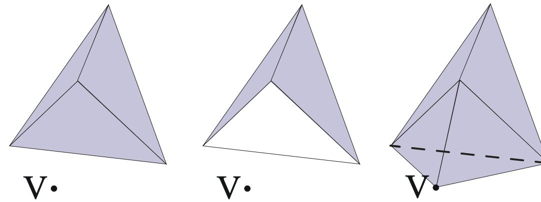



**Fig. 7** Tetrahedron operation for genus value explanation.

For the efficient function of the EdgeBreaker algorithm we need the triangles to be in compact areas - triangles must touch their neighbours by edges, not just by vertices. Otherwise the EdgeBreaker is stopped and started again for still unvisited triangles and this leads to a growth of the amount of data. The worst kind of data for EdgeBreaker compression is a set of separated triangles.

### 2.4.2 Delta Coding and Prediction

The entropy coding, often used and very effective compression technique, is mentioned above. To increase the efficiency of this technique the entropy of coded data should be as low as possible. Due to this claim delta coding and linear prediction methods were proposed.

In dense meshes most edges are short with respect of the model size and distances between vertices are short as well. Traditionally vertex coordinates are related to the origin of the coordinate system and vertex coordinates lie in wide range of values. If the range of values is reduced, vertex coordinates could be compressed more efficiently using quantization and entropy coding.

Delta coding is based on coding differences between values instead of values themselves. This difference is called delta. Order of values in the coded sequence has to be

known because each value is delta coded as a difference between its original value and decompressed value of its predecessor (we obtain one value from the other). If the differences in these pairs of values are small, delta coding greatly reduces data redundancy.

The difference in the pair of equivalent values is equal to 0 and good delta coding should have all deltas minimal (as close to 0 as possible). In combination with quantization delta coding results in data-set with very small range of values. Resulted deltas are ideally close to zero and after quantization gain just few very similar values.

Vertices of static mesh have to be ordered to use delta coding for static mesh compression. To order them the EdgeBreaker (or other similar connectivity compressor) can be used. Connectivity compressor traverses the mesh and creates strip of processed triangles adding one adjacent vertex (triangle) at a time. Each new vertex can be delta coded according to the last coded vertex and resulting delta will be probably close to zero due to topological dependence (pair of vertices belongs to one triangle) of pairs of successive vertices.

This approach can be further improved using prediction of vertex position according to surrounding vertices. A very common prediction method is based on the parallelogram rule [11], see figure 8. The mesh is traversed using connectivity compressor or analyzer and vertices are processed as described above. Coordinates of each new vertex are not delta coded directly, but predicted to lie at the top of projected parallelogram first. This parallelogram is formed by corner vertices of last processed triangle $v_{left}$, $v_{right}$ and $v_{base}$. The X, Y and Z coordinate of the predicted vertex are expressed as:

$$v^X_{predicted} = v^X_{left} + v^X_{right} - v^X_{base} \tag{2.6}$$

$$v^Y_{predicted} = v^Y_{left} + v^Y_{right} - v^Y_{base} \tag{2.7}$$

$$v^Z_{predicted} = v^Z_{left} + v^Z_{right} - v^Z_{base} \tag{2.8}$$

The algorithm traverses the mesh, processing one triangle and related vertex at a time using the prediction equation and transmits differences between original and predicted coordinates of vertices. This difference is called residuum, not delta, but in principle it is delta coding.

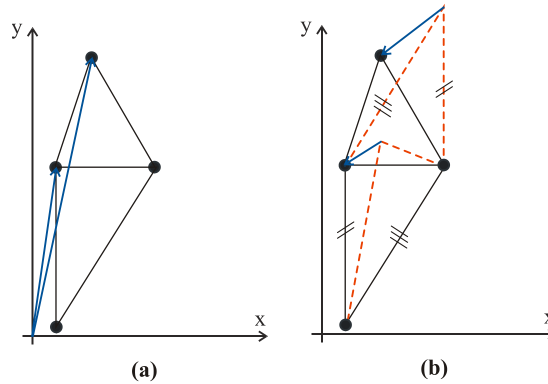



**Fig. 8** Parallelogram prediction b), original stored data a). Transmitted values are represented by blue lines.

Methods were described using topology (connectivity) driven traversing of compressed triangle mesh, but the traversal can be driven by geometry properties of the mesh as well.

## 2.5 Dynamic Mesh Compression

Types of 3D animations can be divided into two groups acording to their connectivity. First group contains animations with constant connectivity; the second contains animations with varying connectivity or varying number of vertices. Animations with constant connectivity can be compressed more efficiently than the others and they are known as dynamic triangle meshes, because they can be represented by single triangle mesh with vertex positions varying in time (vertex trajectories). Because of this animation representation, many methods of dynamic mesh compression use the same or very similar approaches as methods of static mesh compression. Typical compression scheme consists of separate compression of the mesh connectivity and geometry compression, finished by quantization and entropy coding.

Geometry of dynamic mesh is sometimes transformed into the form of vertex trajectory for each vertex of the triangle mesh instead of set of vertex positions for each frame of animation. Geometry is then analysed and compressed in the space of vertex trajectories but there are some algorithms that use for example shape-space PCA instead of trajectory-space PCA during geometry compression.

Most of dynamic mesh compression algorithms are lossy, but do not affect the connectivity of triangles of the compressed mesh. Thus these compression methods only lose geometric precision of animation’s vertex trajectories.

### 2.5.1 Dynapack

Dynapack compression scheme [20] is based on spatiotemporal prediction schemes ELP and Replica. Dynapack uses EdgeBreaker to traverse the compressed mesh (its triangles) and when new (not visited) vertex is reached, its coordinates are predicted by Dynapack and the obtained residuum is encoded using entropy coder. ELP and Replica are extrapolation predictors, because they extrapolate known coordinates of vertex in previous frame using neighbouring vertices to predict its coordinates in currently processed frame. These space-time extrapolating predictors need only two consecutive frames for prediction, but they cannot be used for the first frame of animation and for the first few vertices of each frame, because we need few neighbouring vertices and previous frame for prediction.

In the first frame of the animation space-only prediction, known as parallelogram prediction, is used (described in section 2.4.2). In following frames of the animation the first few vertices are predicted using time-only predictor. This predictor uses vertex coordinates in the last frame as a prediction of vertex coordinates in the currently processed frame.

Extended Lorenzo Predictor (ELP) is space-time predictor perfect for pure translation prediction and for prediction of more general deformations. It is similar to parallelogram prediction. ELP prediction is defined as:

$$v^f = v^f_{left} + v^f_{right} - v^f_{base} + v^{f-1} - v^{f-1}_{left} - v^{f-1}_{right} + v^{f-1}_{base}. \tag{2.9}$$

Put simply, it is parallelogram prediction of the vertex *v* coordinates in the frame *f* corrected using residue between predicted and original vertex coordinates in the frame *f*-1.

Unlike ELP the space-time Replica predictor is capable of predicting rigid motions and uniform scaling transformations. It expresses coordinates of vertex $v^{f-1}$ as in a coordinate system derived from the neighbouring triangle (last reached by EdgeBreaker) first:

$$A = v^{f-1}_{right} - v^{f-1}_{base}, \tag{2.10}$$

$$B = v^{f-1}_{left} - v^{f-1}_{base}, \tag{2.11}$$

$$C = \frac{A \times B}{\sqrt{\|A \times B\|}}, \tag{2.12}$$

$$v^{f-1} = v^{f-1}_{base} + aA + bB + cC, \tag{2.13}$$

where coefficients (new coordinates) $a$, $b$ and $c$ are computed using following equations:

$$D = v^{f-1} - v_{base}^{f-1}, \tag{2.14}$$

$$a = \frac{A \cdot D * B \cdot B - B \cdot D * A \cdot B}{A \cdot A * B \cdot B - A \cdot B * A \cdot B}, \tag{2.15}$$

$$b = \frac{A \cdot D * A \cdot B - B \cdot D * A \cdot A}{A \cdot B * A \cdot B - B \cdot B * A \cdot A}, \tag{2.16}$$

$$c = D \cdot \frac{A \times B}{\sqrt{\|A \times B\|}}. \tag{2.17}$$

Finally, vertex $v^f$ is predicted by replication of this construction on frame $f$:

$$A' = v_{right}^{f} - v_{base}^{f}, \tag{2.18}$$

$$B' = v_{left}^{f} - v_{base}^{f}, \tag{2.19}$$

$$C' = \frac{A' \times B'}{\sqrt{\|A' \times B'\|}}, \tag{2.20}$$

$$v^f = v_{base}^{f} + aA' + bB' + cC'. \tag{2.21}$$

This construction is relative to the neighbouring triangle, and thus it perfectly predicts coordinates of vertices in rigidly transformed parts of the mesh. Meaning of the Replica's coefficients is depicted on the figure 9.

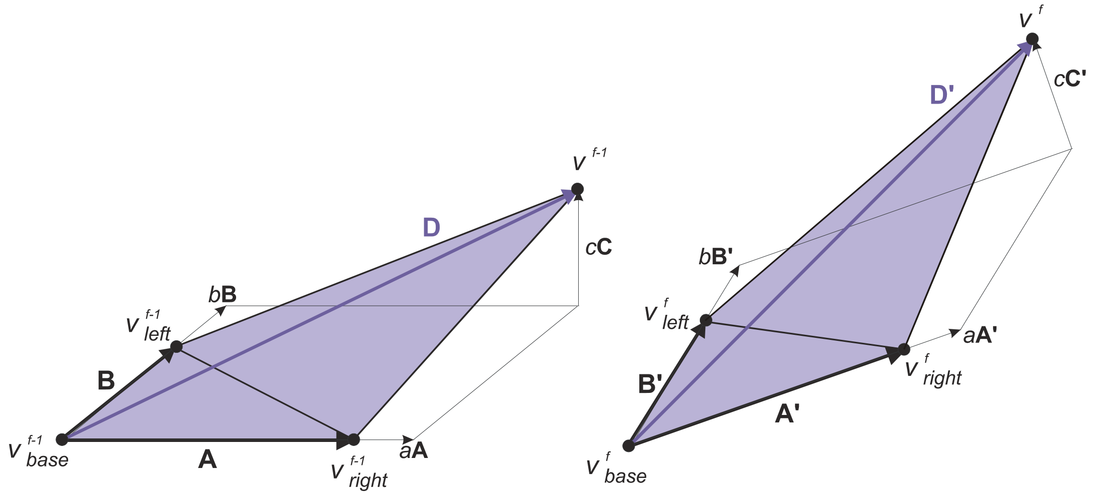



**Fig. 9** Replica predictor. Last frame on the left, currently processed frame on the right.

### 2.5.2 Shape-space Based Compression

Another approach to the compression of dynamic meshes proposed by Alexa and Muller in [13] is based on using shape-space PCA instead of trajectory-space PCA during geometry compression. Shape-spaces describe families of shapes as a linear space.

An isomorphic vertex-edge topology for each frame of animation is expected in this approach as well and only a subset of $F_{key}$ frames (key-frames) of original animation to represent the original animation is used. Key-frames of animation are separately described by the set of shape vectors $B_i$ and it is assumed that all key frames (base shapes) have vectors of the same length. Then the mesh geometry $A(t)$ in the time $t$ of the key-frame animation can be expressed by interpolation between two consecutive key-frames:

$$A(t) = \sum_{i=1}^{F_{key}} a_i(t) \cdot B_i. \tag{2.22}$$

Function $a_i(t)$ provides vector of weights describing the key-frame interpolation in the time $t$:

$$a_i(t) = \left(0, \dots ,0, \frac{t_{i+1} - t}{t_{i+1} - t_i}, \frac{t - t_i}{t_{i+1} - t_i}, 0, \dots ,0\right), \tag{2.23}$$

where $t_i$ is the time stamp of the $i$-th key-frame. However, Alexa's approach separates geometry from animation, thus an alternative animation representation is used:

$$A(t) = \sum_{i=0}^{F_{key}} \widehat{a_\iota}(t) \cdot \widehat{B_\iota}, \tag{2.24}$$

where $\widehat{B_0}$ denotes the average static shape vector and the rest of $\widehat{B_\iota}$ vectors represent linear deviations from the average shape vector. These vectors are ordered with decreasing importance with respect to the animation reconstruction. $B_i$ vectors cannot include transformations such as rotation, and therefore the animation is decomposed into rigid body motion and an elastic part first and transformation matrix $T(t_i)$ from $B_0$ to $B_i$ with the centre of mass shifted to the origin of the coordinate system is computed for each geometry shape (key-frame).

We obtain new animation representation, which decouples animation part and geometry part:

$$A(t) = T^{-1}(t) \cdot \sum_{i=1}^{F_{key}} \hat{a_i}(t) \cdot \hat{B_i}. \qquad (2.25)$$

The geometry is described mainly by $B_0$ and the animation is described by $T_i$ and $a_i$. PCA is used to compute basis vectors $B_i$ and the compression is achieved by using only the first few $\hat{B_i}$ vectors for animation representation see figure 10. The shape-space PCA is used in this dynamic mesh compression approach, thus it is possible to replace original geometry by new geometry while maintaining the original animation of the geometry. For example, if we have 3D animation of running horse, we can represent it the way mentioned above and replace the horse geometry by the elephant geometry to obtain animation of the running elephant.

The disadvantage of this approach is especially the size of the autocorrelation matrix $C$ we are using during PCA to find the shape-space basis. If we use trajectory-space PCA, the autocorrelation matrix $C$ has size $3F\times3F$, where $F$ is number of frames, but if we use shape-space PCA, the autocorrelation matrix $C$ has size $3V\times3V$, where $V$ is number of vertices. Typical compressed animation has hundreds of frames and thousands or tens of thousands of vertices, thus the PCA autocorrelation matrix can be more than two orders of magnitude larger if we use shape-space instead of trajectory space.

These disadvantages are usually handled by using small set of key-frames instead of all frames of compressed animation and by the mesh simplification.

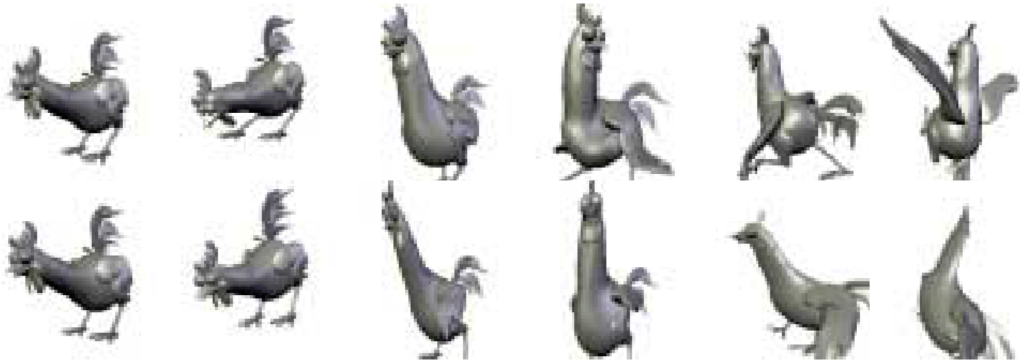


**Fig. 10** Space-shape based compression figure from [13]. Frames of original animation in the first row, animation compressed using only 3 basis vectors in the second row.

### 2.5.3 D3DMC

3D mesh coding (3DMC) is currently part of MPEG-4 Visual standard used for static 3D mesh compression and Muller et al. proposed its extension, Differential 3DMC, in [22]. Differential 3D mesh coding scheme (D3DMC) is a prediction based approach, which uses an octree data structure for spatial subdivision (clustering) of 3D mesh animation with constant connectivity.

D3DMC uses animation description similar to 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 as follows:

1) Previously decoded mesh is subtracted from the current mesh and only difference vectors are further processed
2) Octree motion segmentation (spatial clustering) of difference vectors by Zhang and Owen [21] is performed
3) Resulting information is quantized and encoded using CABAC

I meshes are typically used in the first frame of compressed animation sequence or when the prediction in D3DMC becomes too large. Diagram of the encoder is shown in figure 11.

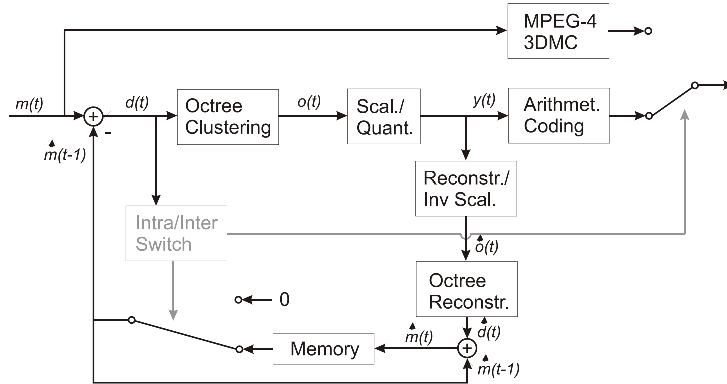



**Fig. 11** Differential 3DMC compression scheme.

The octree motion segmentation is the key point of this compression method. Octree structure is used in this scheme to represent motion of vertices within the space. First, we subtract previously decoded mesh (frame) from the currently processed mesh and obtain difference motion vectors $\Delta v = [v_x, v_y, v_z]$ describing motion between two consecutive frames of 3D animation.

We start the octree motion segmentation with minimum bounding box as a topmost cell of the octree structure, which includes all $V$ vertices within. Eight motion vectors $m_1, \dots, m_8$ 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, then the cell is repeatedly split into eight octants until the approximation reaches user defined accuracy.

Motion of vertices within each cell (difference vectors) is approximated using tri-linear interpolation of the motion vectors. First, the tri-linear ratio $\rho = [\rho_x, \rho_y, \rho_z]$ is computed for each vertex $v$ and corner of the cell with minimum x, y, z $b = [b_x, b_y, b_z]$ as follows:

$$\rho_x = \frac{v_x - b_x}{s} \qquad \rho_y = \frac{v_y - b_y}{s} \qquad \rho_z = \frac{v_z - b_z}{s}, \tag{2.26}$$

where $s$ denotes size of the cell. Then weights of motion vectors $w_1, \dots, w_8$ for the processed vertex $v$ are computed and finally, the vertex motion $\overline{\Delta v}$ is computed using tri-linear interpolation of these motion vectors:

$$w_1 = (1 - \rho_x)\rho_y\rho_z \tag{2.27}$$

$$w_2 = (1 - \rho_x)(1 - \rho_y)\rho_z \tag{2.28}$$

$$w_3 = \rho_x(1 - \rho_y)\rho_z \tag{2.29}$$

$$w_4 = \rho_x\rho_y\rho_z \tag{2.30}$$

$$w_5 = (1 - \rho_x)(1 - \rho_y)(1 - \rho_z) \tag{2.31}$$

$$w_6 = \rho_x(1 - \rho_y)(1 - \rho_z) \tag{2.32}$$

$$w_7 = \rho_x\rho_y(1 - \rho_z) \tag{2.33}$$

$$w_8 = (1 - \rho_x)\rho_y(1 - \rho_z) \tag{2.34}$$

$$\overline{\Delta v} = \sum_{i=1}^{8} w_i m_i. \tag{2.35}$$

Each cell of the octree segmentation structure approximates the motion of vertices within this cell and the computation of the eight motion vectors of the cell is the final step of the animation representation process.

Let $A$ be the matrix of size $3V\times 24$, $V$ is number of vertices within the processed cell and $w_i^j$ is the weight of the i-th motion vector of the j-th vertex and $b$ is vector of length $3V$ in the following form:

$$A = \begin{pmatrix} w_1^1 & 0 & 0 & & w_8^1 & 0 & 0 \\ 0 & w_1^1 & 0 & \cdots & 0 & w_8^1 & 0 \\ 0 & 0 & w_1^1 & & 0 & 0 & w_8^1 \\ w_1^2 & 0 & 0 & & w_8^2 & 0 & 0 \\ 0 & w_1^2 & 0 & \ddots & 0 & w_8^2 & 0 \\ 0 & 0 & w_1^2 & & 0 & 0 & w_8^2 \\ & \vdots & & & & \vdots & \\ w_1^V & 0 & 0 & & w_8^V & 0 & 0 \\ 0 & w_1^V & 0 & \cdots & 0 & w_8^V & 0 \\ 0 & 0 & w_1^V & & 0 & 0 & w_8^V \end{pmatrix} \quad b = \begin{pmatrix} \Delta v_x^1 \\ \Delta v_y^1 \\ \Delta v_z^1 \\ \Delta v_x^2 \\ \Delta v_y^2 \\ \Delta v_z^2 \\ \vdots \\ \Delta v_x^V \\ \Delta v_y^V \\ \Delta v_z^V \end{pmatrix} \tag{2.36}$$

Then we obtain desired set of eight motion vectors, which provide the best estimation of the motion of the vertices in the currently processed cell, by using the least square estimation of $x$ in the equation $Ax = b$, where $x$ is vector of length 24 containing x,y,z triplets for each of the eight motion vectors.

Obtained motion vectors are evaluated according to accuracy of estimation of vertex motion inside the cell. If the motion is not well predicted, then the cell is subdivided into eight octants and the segmentation process is repeated while the estimation error is larger than specified threshold. Finally the motion vectors are uniformly quantized to reduce compressed data entropy and thus enhance the CABAC compression ratio. Only the set of motion vectors and the octree structure are stored.

### 2.5.4 FAMC

Frame-based Animated Mesh Compression (FAMC) method proposed in [5] by Mamou, Zaharia and Prêteux is also segmenting the mesh with respect to motion and it has been adopted by MPEG as a new standard for dynamic mesh compression. FAMC is constructed for compression of the dynamic 3D meshes with constant connectivity and time-varying geometry as compression methods mentioned above. It is based on a skinning model-based motion compensation strategy. Scheme of FAMC compression is depicted in figure 12.

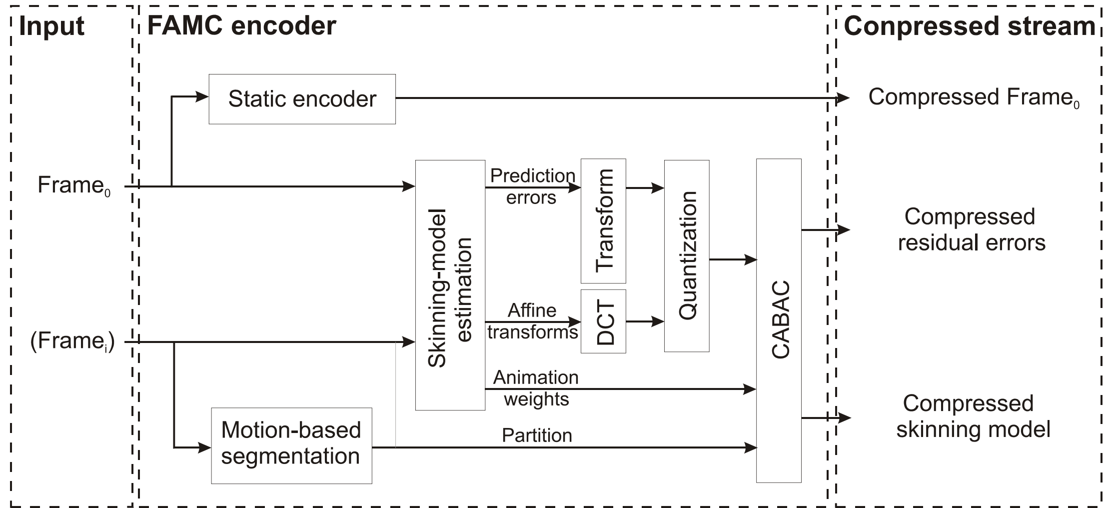



**Fig. 12** Frame-based animated mesh compression scheme.

First, the animated mesh with *F* frames and *V* vertices is segmented into vertex clusters. This segmentation is performed with respect to the motion of mesh vertices, such that motion of each segment can be accurately described by a single affine transformation. This transformation is calculated for each cluster and each frame of the animation and it describes transformation of vertices from the first frame of the animation to the desired frame of the animation. The first frame of the animation is compressed by an arbitrary compression algorithm designed for compression of static meshes.

The segmentation process determines a partition $\Pi = \{\pi_n\}_{n=1,\dots,N}$ of the mesh, defining *N* clusters such that each cluster can be accurately described by a single affine transformation if the partition is optimal.

The optimality criterion is defined in FAMC as the mean square motion compensation error$E(\Pi)$:

$$E(\Pi) = \sum_{f=1}^{F} \sum_{v=1}^{V} \left\| A_f^{n(v)} x_1^v - x_f^v \right\|^2, \tag{2.37}$$

where $x_f^v$ is position of vertex $v$ in the frame *f* described in homogeneous coordinates and $A_f^{n(v)}$ represents the affine transformation from the first frame to the frame *f*. The function $n(v)$ returns the index $n$ of the cluster to which the vertex $v$ belongs. The affine transformation is given by:

$$A_f^{n(v)} = arg \min_A \left( \sum_{p \epsilon \pi_n} \left\| A x_1^p - x_f^p \right\|^2 \right). \tag{2.38}$$

Goal of the algorithm is to determine such a partition $\Pi^*$ of the input dynamic mesh that with a minimal number of clusters the motion compensation error satisfies the bound of user defined error threshold $E(\Pi^*) \leq E_0$ . The FAMC algorithm is based on a hierarchical simplification strategy depending on topological conditions: two neighbouring vertices $v$ and $w$ are merged into single one by using the half-edge collapse operator denoted by $hecol(v, w)$ if their affine motion is similar. This operator removes one of these vertices and connects its incident edges to the remaining vertex. To each remaining $v$ list of ancestor vertices $v^*$ is assigned. In the initial state the list $v^*$ is empty and after each half-edge collapse $hecol(v, w)$ it is updated as follows:

$$v^* \leftarrow v^* \cup w^* \cup \{w\}. \tag{2.39}$$

The decimation process is driven by cost of collapsing edges such that the edge with the minimal cost is collapsed in each decimation step. The cost of the edge collapse is determined by an objective error function $C(v, w)$:

$$C(v, w) = \sum_{f=1}^{F} \left( \sum_{p \epsilon \vartheta(v,w)} \left\| A_f^{v,w} x_1^p - x_f^p \right\|^2 \right), \tag{2.40}$$

where $\vartheta(v, w) = v^* \cup w^* \cup \{v, w\}$, and

$$A_f^{v,w} = arg \min_A \left( \sum_{p \epsilon \vartheta(v,w)} \left\| A x_1^p - x_f^p \right\|^2 \right). \tag{2.41}$$

The clustering process starts with partition consisting of $V$ clusters, one cluster for each vertex, and it iteratively decreases the number of cluster by edge collapses, until the global motion compensation error exceeds the user specified error threshold $E_0$. Modification of this kind of clustering is described in section 3.1.4.

Unfortunately, vertices near the borders of vertex clusters are not described accurately enough and significant distortion on the borders between neighbouring clusters may appear. This behaviour is handled by using the skinning-based motion compensation model, which expresses vertex motion as a weighted linear combination of the motion of cluster the vertex belongs to and motions of its neighbouring clusters. Thus the predicted position $\hat{x}_f^v$ of the vertex $v$ in the frame *f* is obtained by following equation:

$$\hat{x}_f^v = \sum_{n=1}^{N} w_v^n A_f^n x_1^v, \tag{2.42}$$

where $w_v^n$ is the *animation weight*. *Animation weight* is real value coefficient, which is controlling the influence of cluster *n* on the vertex *v*. The vector of all *animation weights* of the clusters influencing the vector *v* is called *animation weight vector* $w^v$ and it is defined as:

$$w^v = \arg \min_{\alpha \epsilon R^N} \sum_{f=1}^{F} \left\| \sum_{n=1}^{N} \alpha_n A_f^n x_1^v - x_f^v \right\|^2, \tag{2.43}$$

This equation is solved only for the cluster which contains the vertex *v* and for the set of neighbouring clusters and all other clusters influences the vertex with $\alpha_n = 0$. Finally, for each frame, the algorithm calculates correction vectors between predicted positions of vertices and their original positions. Thus the compressed animation data contains the first frame mesh with the connectivity description, the clustering information, the set of affine transformation matrices, and the set of weight vectors and correction vectors.

### 2.5.5 Coddyac

The Coddyac is a compression algorithm, which is specialized in dynamic triangle mesh compression and uses dynamic meshes with constant triangle connectivity as input data.

Coddyac contains two well-known algorithms, Rossignac's EdgeBreaker and principal component analysis, as described in sections 2.4.1 and 2.3. In Coddyac, EdgeBreaker is used for compression of triangle connectivity (topology) of the mesh. As mentioned above, EdgeBreaker is primarily intended for triangle connectivity compression of static meshes, but if the triangle connectivity of chosen dynamic mesh is consistent throughout the time of animation we can use this compression algorithm as well.

Geometry of the compressed dynamic mesh can be compressed at the same time as triangle connectivity. Geometry information of dynamic mesh is transformed into the form of a set of vertex trajectories and processed by principal component analysis first, as described in section 2.3.The size of compressed data depends especially on the rate of compression of these trajectories. The PCA basis matrix *E* and the means vector *A* are stored and vertex trajectories are replaced by feature vectors in the structure of dynamic triangle mesh. Finally, combination coefficients in feature vectors are quantized.

During triangle connectivity compression all three feature vectors *c* of EdgeBreaker's initial triangle (one for each triangle corner) are stored without any further modifications. The mesh is then traversed by EdgeBreaker and each new feature vector is predicted from already compressed vectors using parallelogram prediction:

$$c^j_{predicted} = c^j_{left} + c^j_{right} - c^j_{base}, \qquad j = 0,1,\dots,N-1\,, \tag{2.44}$$

where *N* is user defined number of basis vectors, equal to the length of compressed vectors. After parallelogram prediction the residue vector between original and predicted feature vector is stored.

At the end of compression algorithm resulting CLERS string and residue vectors are encoded using arithmetic coder. Finally, matrix of PCA basis is compressed by specialised algorithm COBRA [8].

### 2.5.6 COBRA

Compression of the Basis for PCA Represented Animations (COBRA) is highly efficient extension of dynamic mesh compression techniques based on PCA. While volume of connectivity data is usually negligible, volume of PCA basis is large, when we use PCA-based dynamic mesh compression algorithm. COBRA can reduce the size of the PCA basis by 90% with respect to direct encoding and thus achieve approximately 25% increase of performance of the compression algorithm without any significant loss of accuracy.

Standard way of PCA basis compression is a simple quantization, but COBRA is based on non-uniform quantization and uses non-least-squares optimal linear prediction to increase the compression ratio. Non-uniform quantization was already mentioned in section 2.1. Key observation for COBRA is, that basis vectors have character of trajectories. Each basis vector can be interpreted as a trajectory of a moving point, which moves smoothly. Therefore COBRA predicts these basis vectors using linear movement predictors such as:

$$pred_1(v_f) = v_{f-1} + (v_{f-1} - v_{f-2}) = 2v_{f-1} - v_{f-2}, \quad (2.45)$$

where $v_f$ is vertex position in frame $f$. There are two more possible predictors using the speed estimation $s$, the acceleration $a$, and the change in acceleration $c$:

$$s = v_{f-1} - v_{f-2}, \quad (2.46)$$

$$\mathrm{a} = (\mathrm{v}_{\mathrm{f}-1} - \mathrm{v}_{\mathrm{f}-2}) - (\mathrm{v}_{\mathrm{f}-2} - \mathrm{v}_{\mathrm{f}-3}) \quad (2.47)$$

$$= v_{f-3} - 2v_{f-2} + v_{f-1}, \quad (2.48)$$

$$c = \left((v_{f-1} - v_{f-2}) - (v_{f-2} - v_{f-3})\right) \quad (2.49)$$

$$- \left((v_{f-2} - v_{f-3}) - (v_{f-3} - v_{f-4})\right); \quad (2.50)$$

$$pred_2(v_f) = v_{f-1} + s + a; \quad (2.51)$$

$$pred_3(v_f) = v_{f-1} + s + a + c; \quad (2.52)$$

Predicted basis vectors are compared with the original basis vectors and obtained residues are quantized and encoded by the entropy coder. COBRA uses non-uniform quantization in this step. Each vertex trajectory of the compressed dynamic mesh is expressed by linear combination of basis vectors depending on combination coefficients in the

appropriate feature vector. The size of the coefficients varies very significantly, but this variance can be predicted well. Size of the coefficients corresponds with its order in the feature vector (first coefficient is the biggest and the appropriate basis vector is the most important), which is caused by the nature of PCA. This behaviour is used by non-uniform quantization. COBRA uses finer quantization for the more important basis vectors. For more details see the original source [8].

### 2.5.7 Vertex clustering and 3D mesh registration

It should be mentioned that there is a paper dealing with the topic similar to the topic of this thesis. Various techniques of vertex clustering are described and compared in the Impact of vertex clustering on registration-based 3D dynamic mesh coding [33] by Ramanathan, Kassim and Tan. Three different vertex clustering techniques are considered: multilevel k-way graph partitioning [30] for topology-based clustering, Lloyd's k-means clustering [31] for geometry-based clustering and spectral clustering [32] for semantic mesh decomposition and it is derived from the experiments that the compression obtained through the semantic clustering achieves the best compression ratio.

New Iterative Closest Point (ICP) [34] based 3D dynamic geometry compression scheme is presented in this paper. Whole mesh is segmented according to the mesh motion by this compression algorithm and for each segment of the current mesh the appropriate segment in the temporal reference is detected using ICP. Finally, the motion of all vertices in each cluster of the mesh is described by affine transformations and the set of correction vectors. While the mesh is segmented and the affine transformations are calculated, reconstruction error for each couple of clusters is determined. Subsequently the segmentation algorithm groups the vertices into three sets according to their error.

The first set of vertices contains clusters of vertices, which can be expressed accurately by affine transformations. The second set of vertices consists of clusters of vertices, which can be accurately described only if we use both affine transformations and correction vectors and the third set of vertices is consisting of vertices, which cannot be expressed efficiently using affine transformations and the DPCM-based techniques are used to encode them.

Vertices of the mesh are divided this way to achieve better compression ratios due to the fact, that vertices in the first two sets (70% of all vertices) can be described using a few

affine transformations and correction vectors. However, come clusters can be reconstructed with very high reconstruction error. These clusters are identified during the compression process and reclustered to decrease the reconstruction error and the ICP registration is performed again for these clusters. Finally, conclusion of this paper is that it is possible to achieve about 10% better compression ratio over the Lloyd's k-means and k-way clustering if the spectral mesh decomposition is used.

## 2.6 Error Measurements

Many different algorithms for dynamic mesh compression are known. To compare efficiency of these compression algorithms RD curves are mostly used. RD is abbreviated form of rate-distortion. RD curve is a function, typically expressed as a graph, showing the relationship between the bit-rate and distortion. Distortion is the amount of damage of animated mesh caused by compression algorithm (difference between original and decompressed data) and bit-rate indicates the amount of compressed data.

In the case of dynamic mesh bit-rate is measured in *bpfv* –bits per vertex and frame. *Bpfv* shows how many bits in average do we need to represent one vertex in one frame. Present-day compression algorithms are able to compress dynamic meshes with *bpfv* lower than 1 bit, while distortion remains almost undetectable. Distortion of decompressed mesh is measured with respect to its original for each frame of animation sometimes. It is measured separately for each frame of the animation or summed or averaged for the whole animation.

### 2.6.1 Mean Squared Error

Mean squared error (MSE) is widely used method of error measurement, not only for dynamic mesh decompression distortion. MSE is averaged sum of squared distances or deviations between original values (vertices) $v_i$ and their corresponding distorted versions$\bar{v}_i$:

$$e = \frac{1}{V}\sum_{i=1}^{V} \|v_i - \bar{v}_i\|^2. \tag{2.53}$$

However, there are more sophisticated error measures.

### 2.6.2 KG-error

To compare errors resulting from compression of 3D animated meshes KG-error measure is often used. Karni and Gotsman presented this measure in [7] and defined it as:

$$e = 100 \cdot \frac{\|B - \tilde{B}\|}{\|B - R(B)\|}, \tag{2.54}$$

Where $B$ is matrix containing original dynamic mesh description and $\tilde{B}$ is a matrix containing description of the dynamic mesh after the compression and decompression steps(distorted animation).$B$ and $\tilde{B}$are matrices of size$3V \times F$. Each column $col_i$ of matrix representing the dynamic mesh contains positions of all vertices of the dynamic mesh in $i$-th frame of the animation and each element of this column contains one coordinate of one vertex of dynamic mesh.

*R(B)* is average matrix which contains average spatial value of vertices for each frame of animation. In this formula Frobenius norm is used denoted by$\|x\|$. Resulting error value is expressed in percents and it is invariant to uniform scaling.

### 2.6.3 STED

Spatiotemporal edge difference (STED) is a novel (2011) error measure derived from results of subjective testing of mesh distortion perception presented in [17]. It provides better correlation with human perception of quality loss between original and processed (compressed) dynamic meshes than any other. STED combines measurement of spatial and temporal deviation of edge lengths caused by processing the mesh. This measure is focused on local changes of error, therefore distortion is evaluated only for close neighbourhood of each vertex and these values are summed to obtain the overall error.

Relative edge difference of spatial edge $e_{ij}$ connecting $i$-th and $j$-th vertex is defined as:

$$ed(e_{ij}, f) = \frac{el(e_{ij}, f) - \overline{el}(e_{ij}, f)}{el(e_{ij}, f)}, \tag{2.55}$$

where *el* function provides length of edge $e_{ij}$ in the frame $f$ and overline denotes the distorted version of this length. Relative edge length difference is used to increase sensitivity to distortion in densely sampled areas of the mesh.

For each vertex of the original and distorted mesh local standard deviation of $ed$ is computed. It is computed for given vertex and a set of its topological neighbours (connected by edges) in user specified topological distance d (vertex is maximally d edges distant from the given vertex). Set of edges incident with given $i$-th vertex and any of its neighbours is denoted $NE(i, d)$. Edges in this set may significantly vary in length; therefore we further use weighted average instead of arithmetic average of this set of edges:

$$avged(i,f,d) = \frac{\sum_{e \in NE(i,d)} ed(e,f) el(e,f)}{\sum_{e \in NE(i,d)} el(e,f)}. \tag{2.56}$$

Local deviation around i-th vertex in frame f and for distance d is then given by:

$$dev(i,f,d) = \sqrt{\frac{\sum_{e \in NE(i,d)} \left( (ed(e,f) - avged(i,f,d))^2 \, el(e,f) \right)}{\sum_{e \in NE(i,d)} el(e,f)}}. \tag{2.57}$$

The scale-independent spatial part of STED error for given distance (radius) $d$ is defined as average of this local deviation over all vertices and frames of animation:

$$STED_s(d) = \frac{1}{VF} \sum_{i=1}^{V} \sum_{j=1}^{F} dev(i,d,f). \tag{2.58}$$

In temporal error computation the average speed of the i-th vertex in a frame f and temporal window of width $w$ around this frame is calculated first. To do so, we have to know temporal edge length $tel$ ( ${}_{x}v_{i}^{f}$ denotes coordinate $x$ of $i$-th vertex in the frame f):

$$ld = max_{i=1 \dots V, j=1 \dots V} \left( \left\| v_i^1 - v_j^1 \right\| \right), \tag{2.59}$$

$$dx(i,f) = \frac{{}_{x}v_i^{f+1} - {}_{x}v_i^{f}}{ld}, \tag{2.60}$$

$$dy(i,f) = \frac{{}_{y}v_i^{f+1} - {}_{y}v_i^{f}}{ld}, \tag{2.61}$$

$$dz(i,f) = \frac{{}_{z}v_i^{f+1} - {}_{z}v_i^{f}}{ld}, \tag{2.62}$$

$$tel(\mathrm{i}, \mathrm{f}, dt) = \sqrt{dx(i,f)^2 + dy(i,f)^2 + dz(i,f)^2 + dt^2}. \tag{2.63}$$

The value *dt* (distance in time) is the temporal distance between consecutive frames and it is used to handle possible infinity values caused by processing static vertices. Using relative temporal edge length instead of absolute length the metric sensitivity in areas of a very slow motion increases. Now, we can define the average spatiotemporal speed of the $i$-th vertex in the $f$-th frame using temporal window $w$ as:

$$s(i,f,w,dt) = \frac{\sum_{lf=\max(1,f-w)}^{\min(F,f+w)} tel(i,lf,dt)}{\min(F,f+w) - \max(1,f-w)}. \tag{2.64}$$

Next, the relative temporal edge difference can be defined:

$$ted(i,f,w,dt) = \frac{\left\| tel(i,f,dt) - \overline{tel}(i,f,dt) \right\|}{s(i,f,w,dt)}. \tag{2.65}$$

By putting previous formulas together we can construct formula of the overall temporal error averaged over all vertices and frames:

$$STED_t(w,dt) = \frac{1}{V(F-1)} \sum_{i=1}^{V} \sum_{j=1}^{F-1} ted(i,f,w,dt), \tag{2.66}$$

where $F$-1 is used because of *tel* function, which is not defined in the last frame of animation.

To obtain overall spatiotemporal error, formulas of STED*s* and STED*t* are combined into the form of hypotenuse of weighted spatial and temporal error:

$$STED(d,w,dt,c) = \sqrt{STED_s(d)^2 + c^2 \cdot STED_t(w,dt)^2}. \tag{2.67}$$

Parameter $c$ is a weighting coefficient.

# 3 CLUSTERED CODDYAC

As a basic compression algorithm Coddyac is chosen due to its high efficiency. Coddyac is PCA based dynamic mesh compression algorithm, thus it compresses vertex trajectories using principal component analysis. The more complex the set of trajectories is, the less information about this set can be considered negligible and the compression ratio decreases. Because geometry takes the major part of the dynamic mesh, size of the final compressed data depends especially on the rate of compression of the feature vectors (contain PCA combination coefficients). Length of these vectors is equal to number of selected principal components (basis vectors), so less principal components means shorter vectors and thus better compression ratio.

As noted, the efficiency of the Coddyac compression algorithm directly depends on the complexity of movements of the animated mesh. By movement complexity of a set of vertices we mean differences of their trajectories. The trajectories can be very complex, but they have to be similar to each other to decrease movement complexity of the set of vertices. It follows that if different parts of the mesh move in a relatively simple manner, but differently, the global movement of the mesh will be complex. Therefore, after application of PCA, all trajectories are described by vectors longer than necessary –we are using combination of more principal components in feature vectors than we need.

For example to describe the movement of the torso of the chicken in figure 13 on the left, principal components describing movement of his wings are also used, even though this movement almost does not affect the torso.

**Fig. 13** Chicken dynamic mesh movement simplification. Three important movements for each vertex of chicken mesh on the left. On the right clustered chicken mesh with only one important movement for torso and each wing.

To reduce the length of the feature vectors, we must select those vertices of the mesh, whose trajectories are similar each other and include them in a common group – a cluster. This way the movement complexity in individual clusters is reduced and so is the necessary number of principal components. That leads to shorter PCA vectors and better compression ratio.

Simply put, one possible way to improve the compression ratio of Coddyac and PCA based dynamic mesh compression algorithms in general is to reduce the movement complexity of dynamic mesh, which could be locally achieved by clustering the mesh vertices by similarity of their trajectories.

## 3.1 Clustering in Coddyac

Many algorithms have been proposed for dynamic mesh compression. There are also known various algorithms of mesh division into smaller parts (clusters) depending on the topology or geometry criteria. Our method combines the compression algorithm and division of the mesh to increase compression ratio.

Clustering algorithms divide the given set of data into subsets according to specified criteria, so that data in the same cluster are similar in some sense. Two basic types of clustering are hierarchical clustering and partitional clustering. Hierarchical clustering builds progressively clusters using already specified clusters and final hierarchy may be represented in a tree structure. Partitional clustering builds all clusters by dividing given data set at once.

In the context of dynamic mesh compression three ways of mesh division can be considered: geometry-driven division, topology-driven (connectivity-driven) division and division into logical parts. Division into logical parts (for example, finding the limbs, head and torso on a human model) can be efficiently used for compression of meshes with rigid motions or setting different level of quantisation for specific areas of mesh (face more accurately quantized than chest). Topology-driven division scheme is used if vertex-edge incidence or triangle connectivity is found to be good criteria for mesh division. Most frequently used way of division is geometry-driven division, which divides the mesh on the basis of geometrical properties such as vertex positions or trajectories.

In this thesis we want to divide meshes according to their movement complexity and similarity. Movement is a geometrical property of dynamic mesh, thus niether strictly

topology-driven nor logical division is not considered further. For example front legs of a running horse model could be divided into separate parts using dividing into logical parts, but they will probably move in a very similar way, thus it is better to join them into one cluster using geometry-driven approach according to PCA compression efficiency.

Geometry-driven division of dynamic meshes can be performed by clustering of vertices using their trajectories. We have tested several methods of vertex clustering, whose functions and modifications are described below.

### 3.1.1 K-means

K-means [4] is one of the partitional clustering algorithms. The clusters are iteratively refined according to the specified distribution criteria. Each k-means cluster is represented by its centre and the data points are usually assigned to clusters based on their distance to the centres of clusters. Cluster centres are usually calculated as an average of all values in the cluster, so the centres do not have to correspond with data-points in the input data-set.

Algorithm consists of the following steps:

Initialization: Choose the $k$ elements of the data as initial centres of clusters for the next step

1) Assign each point to nearest centre
2) Calculate new centre of each cluster as average of all elements in the cluster
3) Classify the data into $k$ clusters specified by their distance from the centres of previous clusters
4) If the contents of clusters changed, go to step 2)

Algorithm consists of two phases, which are repeated. These are classification stage, when the data are divided into individual clusters, and the phase of learning, where we control reassignment of data into individual clusters and calculate their new centres. The shortest Euclidean distance between the centres of cluster and the selected element from the specified set of data is most often chosen as a distribution criterion. Cluster centres are then calculated as the arithmetic average of all values within the cluster. The initial values of centres of clusters are randomly selected elements from the specified set of data.

To run the K-means algorithm we need to know in advance how many classes (clusters) will be used, i.e. the value $k$. The $k$ value varies according to the purpose of using k-means algorithm or input data. In the case of vertex trajectory clustering the $k$ value varies in dependence on the input data.

The calculation of distance of trajectory vectors can be modified in clustering algorithms and in addition to Euclidean distance ($L_2$norm) we have experimented with calculation of the distance using any norm $L_p$ by the following formula:

$$L_p = \left( \sum_i |x_i|^p \right)^{\frac{1}{p}}, \quad (3.1)$$

where $x_i$ denotes difference between $i$-th components of two vectors. Furthermore, we have experimented with clustering of trajectories, which have been adjusted, as depicted in figure 14.

First, trajectories shifted into the origin of the coordinate system using origin of trajectory and average trajectory position were tested. Origin of trajectory is position of the corresponding vertex in the space of the first frame of animation and average trajectory position is calculated as the arithmetic average of corresponding vertex positions in all frames of the animation.

Second, trajectories were rigidly transformed to the average trajectory. Average trajectory is calculated as the arithmetic average of position of all vertices for each trajectory component. Trajectories were adjusted this way to maximally increase influence of their shapes during clustering step of compression algorithm, but neglecting of this information led to unpredictable (and negative) results.

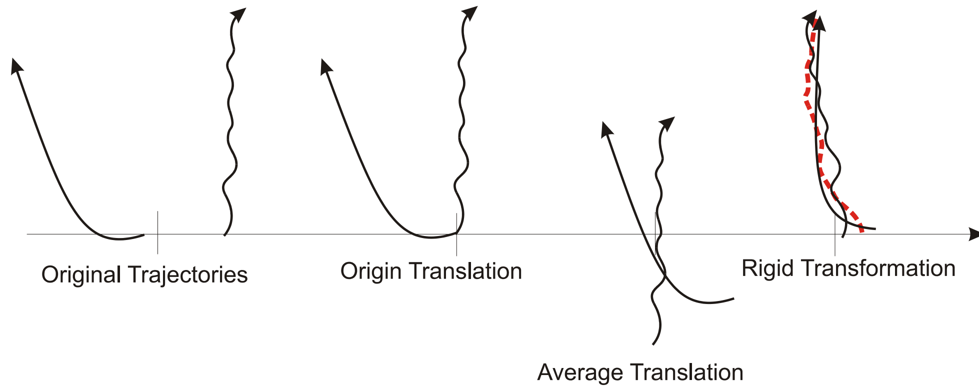



**Fig. 14** Tested trajectory adjustments, red dotted shape represents average trajectory for rigid transformation.

We tried to neglect trajectory distribution using these adjustments to strengthen influence of trajectory shape and thus construct clusters with trajectories as similar as possible. Unfortunately the additional adjustment of trajectories suppresses original distribution and direction of trajectories and this leads to more distorted results of clustering.

Due to our observations and close relation between clustering and topology compression, original trajectories without any adjustment are used in the rest of this thesis.

### 3.1.2 K-means initialization

K-means provide different clustering with randomly chosen initial clusters and better results of k-means clustering are obtained only when initial clustering is close to the final solution. Thus the initial selection of cluster centres is extremely important. From experimental observations it is known, that if random initial clustering is used, some patterns (trajectories in the case of this thesis) have the same cluster membership for each run of the k-means algorithm. This observation was exploited by Khan and Ahmad in [28]. They proposed a novel cluster centres initialization algorithm CCIA for k-means clustering.

In CCIA, clusters are computed for each individual attribute of the input set of patterns in the first step of the k-means initialization. Initial centres are chosen with maximum distance between them and they are chosen from the input data-set from which outliers are removed. It is assumed, that each pattern attribute is normally distributed in the space of patterns. For specified number of clusters $K$ the normal distribution curve is split into $K$ intervals such that we obtain $K$ areas with the same size and the midpoint of this interval is set as a cluster centre. This is performed to eliminate the outliers and to keep maximum distance between cluster centres. Thus each value of this attribute is associated with one of the $K$ clusters.

Then k-means is used on the input data-set to obtain cluster label for each pattern and this process is repeated for each attribute. Vector of this labels assigned to single pattern is called pattern string. At the end of this process we obtained $K'$ distinct pattern strings representing $K'$ clusters. Next, we calculate centres of these clusters and if $K'$=$K$, we use this centers as initialization centers for k-mens. If $K' > K$, we merge similar clusters together until $K'$=$K$. The merging of clusters is provided by density-based multiscale data condensation method (DBMSDC) proposed by Mitra et al. [29].

We are using much simpler cluster initialization in this thesis. We are clustering vertices using their movement to separate vertices with different trajectories, thus we are using initial cluster centres with maximum distance while these centres (trajectories) describing animated mesh movement with maximum accuracy. To find such trajectories we transform vertex trajectories using PCA as indicated in section 2.3. Obtained vectors of transformed trajectories (patterns) include combination coefficients (attributes) of principal components, which are ordered according to their influence on the animated mesh motion. We find attribute related with the most important principal component for each pattern and pattern with largest absolute value of this attribute is set as an initial center of the cluster. This process is repeated for the first $K$ most important principal components.

### 3.1.3 Facility Location

Facility location [6] algorithm is similar to the k-means algorithm mentioned above. The main difference is that the centres of clusters (called facilities) are always chosen from the set of input data. The algorithm tries to find a placement and number of facilities, to which it connects the other elements of the initial set. Each connection of element to facility is evaluated by implementation specific cost. For example Euclidean distance between element and facility is a common choice of evaluation. Algorithm selects such locations of facilities to make the price of the connection of all elements minimal (for example sum of Euclidean distances of the elements from their respective cluster centres). One cluster contains all elements connected to a common facility and the facility itself, thus the number of resulting clusters depends on the number of facilities.

Such an algorithm would open a facility in each element of the input data-set and make that way the total cost equal to 0 (each element will be facility). That is why facility location uses the so called facility cost. Facility cost is the opening price for the facility.

Each facility needs to pay a constant opening price, which is one of the inputs of the algorithm and reduces the number of facilities. The aim of the algorithm is to find a balance between the number of clusters and their sizes. Number of created clusters is thus dependent only on the specified facility cost and the input data-set. Like the k-means algorithm, Facility location is also iterative. Clusters are created and removed while there is a better overall price for the allocation of clusters.

Such an overall cost $P$ is calculated as:

$$P = \sum_{i=1}^{N_F} \left( C_F + \sum_{j=1}^{N_i} f_c(i,j) \right), \quad (3.2)$$

where $N_F$ is number of facilities, $N_i$ is number of elements connected to $i$-th facility and $C_F$ is constant opening price for each facility. Cost function $f_c(i, j)$ calculates the cost of connection between $i$-th facility and $j$-th element of clustered data-set.

In our case there is not a big difference between k-means and facility location. If we use facility location algorithm with some facility cost and use the number of clusters as input $k$ in k-means, then the location and size of clusters should be the same or very similar for both algorithms. Basically k-means is a variation of facility location algorithm, where we are finding minimal sum of costs of $k$ facilities instead of minimal number of facilities. In this case facility cost is not set as an input of clustering algorithm.

### 3.1.4 Edge-collapse Based Clustering

Unlike previous methods, which were only dependent on the geometry of the animation, this method is also influenced by the connectivity of the animated mesh using hierarchical decimation strategy. The clustering algorithm uses a priority queue, from which an edge of the mesh is selected that has the best (lowest) evaluation. Evaluation of edge may for example correspond to its length, or similarity of trajectories of vertices that are connected by the edge.

Lowest cost edge is picked from the queue and collapsed into one vertex. Edges adjacent to this point are re-evaluated according to the changes caused by the collapse. This process is repeated until the desired number of remaining vertices is reached. The final number of clusters corresponds to the number of vertices resulting from collapsing the edges. Each final vertex has a tree of its collapsed ancestors and clusters are defined by leaves of this tree. This clustering method in combination with FAMC cost function is used by Mamou’s skinning method.

In original Mamou’s FAMC clustering algorithm original mesh is simplified while global mean square motion compensation error remains lower than the predefined threshold.

Cost of collapsing the edge is defined in section 2.5.4 and exact description of this algorithm can be found in [5].

This algorithm was also tested in combination with modified feature vectors. The similarity of trajectories is no longer assessed on the basis of Euclidean distance of vectors, but by the number of important components of the vectors, which they have in common. The components which are zero after quantization may be neglected by the compression algorithm. Therefore the similarity of trajectories is assessed by the number of zeros in specific common components. In this case we do not want to cluster trajectories which are as similar as possible, but those that allow us to maximize what can be neglected, see figure 15. Trajectories were modified as follows:

**0-trajectories:** A threshold is given. Values in feature vectors, which are smaller than the threshold are rewritten to 0, the others are rewritten to 1. Parts of vectors with 0 are those parts which we want to discard. These vectors, that have the most of common 0, are therefore joined to the common cluster.

Cost function $f_c(a,b)$ for thresholded vectors $a$ and $b$ is defined as:

$$f_c(a,b) = \sum_{i=1}^{3F} (1 - a_i)(1 - b_i) \tag{3.5}$$

Clusters should therefore be designed so that it can be neglected as much of the vectors as possible, achieving a reduction of the resulting data stream.

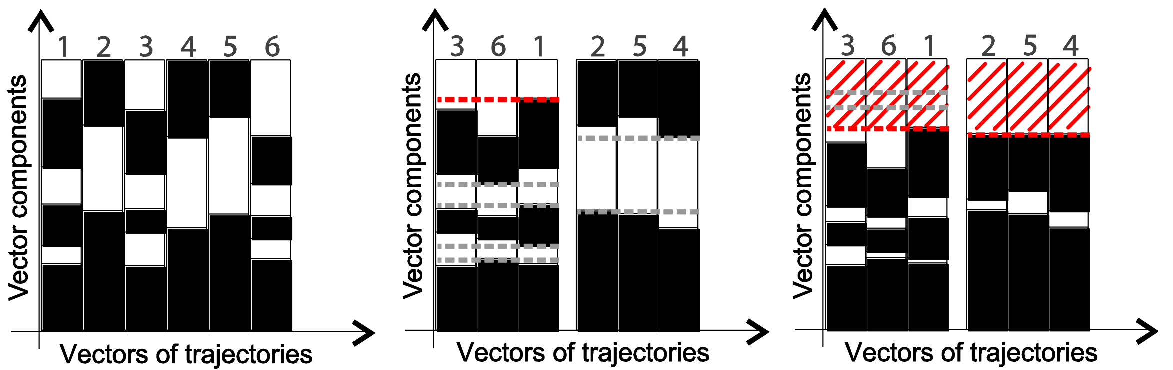



**Fig. 15** Non-zero components of vectors are black, zero components are white. Modified vectors (left) are clustered by their common zero-components (middle). Zero-components common for all cluster (white between grey dotted lines) are moved to the end of the vectors. Sequences of zeros on the end of these vectors common for one cluster can be neglected (hatched area).

## 3.2 Number of Basis Vectors

In Coddyac, setting the number of basis vectors was used to influence the quality of the output. Number of basis vectors was a user specified integer $N$ and its higher or lower values led to increase and decrease of compressed mesh size and caused adequate error. When we use clustering as a way to improve efficiency of Coddyac compression, we have to change the approach to setting the number of basis vectors.

The aim of clustering in the context of dynamic mesh compression is to identify areas of mesh where the vertices are moving with low entropy (similarly), join them into the one common cluster and use it to express their movement by a lower number of basis vectors, while maintaining the same accuracy. The entropy of vertex movement (similarity of vertex trajectories) might vary significantly between clusters; therefore different number of basis vectors for each cluster should be selected.

In our modification of Coddyac a single scalar value of tolerable PCA-introduced error has to be specified by the user, and the algorithm calculates the appropriate number of basis vectors automatically for each cluster.

We can express the average amount of PCA-introduced error using $N$ basis vectors as:

$$\Delta_{PCA}^{N} = \frac{1}{V}\sum_{j=1}^{V}\frac{1}{3F}\sum_{i=1}^{3F}\frac{\left|T_j^i - \bar{T}_j^i\right|}{l}, \tag{3.6}$$

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 $\bar{T}_j^i$ 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.

## 3.3 Compression Scheme

Before the compression algorithm starts, the input set of meshes representing frames of animation is transformed into the form of single mesh and set of trajectories (one trajectory for each vertex). Then the single mesh is clustered according to vertex trajectories and cluster index is assigned to each vertex.

In our scheme, the full mesh (figure 16a) connectivity is compressed first and it is stored in a file together with the indices of clusters (figure 16b) for each vertex. As the number of clusters is relatively small (small variance of values), and their indices are often repeated, the set of indices can be efficiently compressed for example by arithmetic encoding. Connectivity is compressed by EdgeBreaker.

Before the next phase we remove those triangles, whose vertices belong to more than one cluster (figure 16c), and so the dynamic mesh is topologically and geometrically divided into smaller components (figure 16d) corresponding to chosen clustering of vertices. Each cluster is represented by a stand-alone object now.

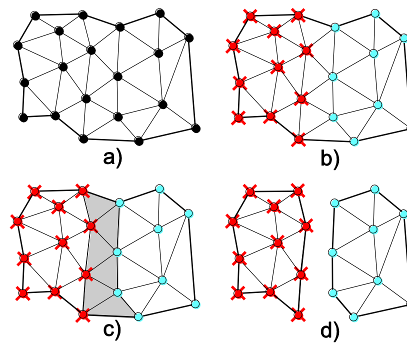



**Fig. 16** Topology of full mesh (a) is clustered (b) and stored. Triangles between clusters (c) are removed, and the mesh is divided into components (d).

The second phase is used only for compression of geometry, not connectivity. Compressed mesh is processed by EdgeBreaker algorithm again, but it is executed for each component individually and resulting connectivity description is not stored. Geometry of the components is compressed by PCA and COBRA separately by the Coddyac algorithm. Every time a new vertex is reached by EdgeBreaker, this event is handled by geometry compressor. Geometry compressor predicts trajectory of this vertex, residuum between predicted and original trajectory is quantised and encoded using arithmetic coder. Simple scheme of this compression is depicted in figure 17.

Clustering compression results in one set of cluster indices for whole mesh, one CLERS string with connectivity description of whole mesh and number of sets of compressed geometry data (basis, means vector and feature vectors), one set for each cluster.

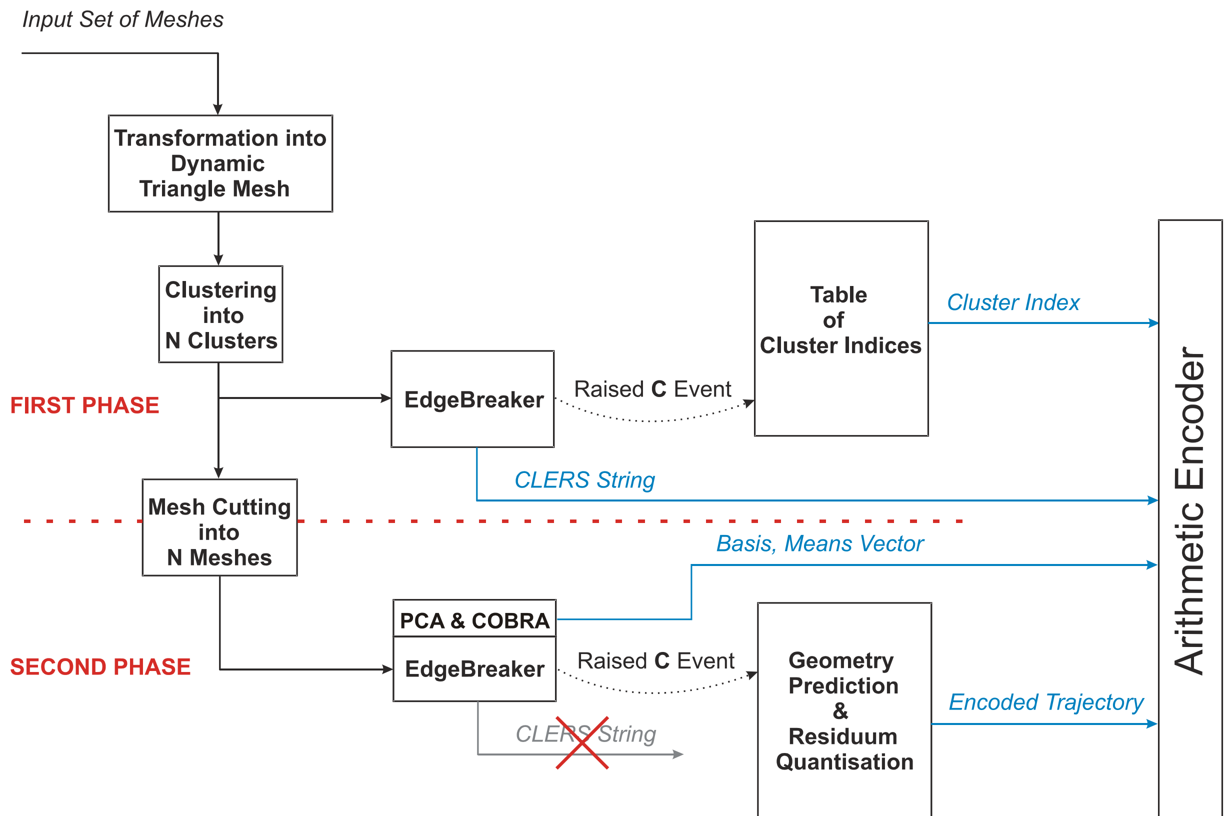



**Fig. 17** Simple scheme of clustered Coddyac compression.

## 3.4 Raised Problems

After applying a clustering algorithm on the vertices and PCA on the input data-set we obtain an index for each vertex of the triangle mesh (and the corresponding trajectory) that determines to which cluster the vertex belongs. For reasons of topology compression, as indicated in section 2.5.5 (for the description of connectivity the EdgeBreaker algorithm is used), it is necessary that the clusters are topologically compact. This means that the vertices belonging to one cluster are not topologically separated by vertices of another cluster. Each cluster should consists of triangles which are touching their neighbours by edges, not just by corners, to enable EdgeBreaker to traverse the cluster topology by crossing edges of neighbouring triangles.

It is actually a projection from an n-dimensional space, in which the algorithm performs the clustering of the vertices of the mesh, on to the 2-dimensional space of the surface of an animated mesh. Unfortunately, after such projection individual clusters may overlap (figure 18). If we use simplification clustering, as in edge-collapse based clustering mentioned above,

where clustering is processed by mesh simplification (edge collapses) directly on the surface of the mesh, we obtain clusters without overlapping areas. But if we use clustering method without dependency on the mesh connectivity, it is necessary to correct the clusters on the surface of the animated mesh to create areas of triangles as connectivity-compact as possible.

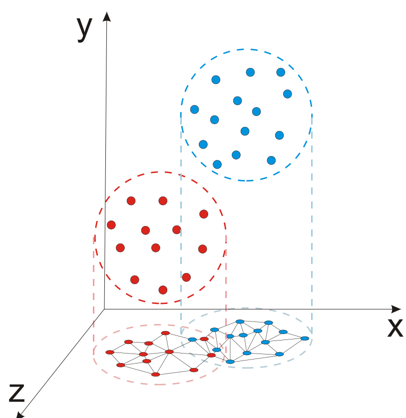



**Fig. 18** Clusters separated in for example 3D space (xyz) can overlap after projection onto 2D surface (xz). This situation can be hard to solve for EdgeBreaker compression.

This inconvenient characteristic of projection of high-dimensional clustering into lower dimensions may cause malfunction of EdgeBreaker and has to be solved. EdgeBreaker algorithm is built to traverse mesh by crossing its edges. It should behave the same way while traversing smaller separate parts of the mesh defined by clustering of mesh vertices. Therefore our compression algorithm has to take into account the connectivity of the vertices in common cluster by employing a correction algorithm.

Due to this observation the smallest possible cluster of vertices or its separate part has to contain at least 3 vertices. If there are only one or two separate vertices, it is not possible to accept them by EdgeBreaker as individual cluster. Therefore when this situation occurs these vertices have to be reassigned into the surrounding cluster. If the cluster or cluster part contains 3 or more vertices it is acceptable by our compression algorithm, but if the number of vertices is small (for example 10 vertices), it will negatively affect final compression ratio because each such cluster needs its own initialisation data, as described further in this section.

There are two ways to resolve the situation of small cluster, see figure 19. The first option is to connect the remote cluster parts by a "bridge", the second option is to "drown" the remote cluster part. Both of these options lead to a situation when some vertices of a cluster are reassigned to a different cluster. This creates an error in the original assignment of vertices into clusters and leads to a reduction in the efficiency of compression algorithms.

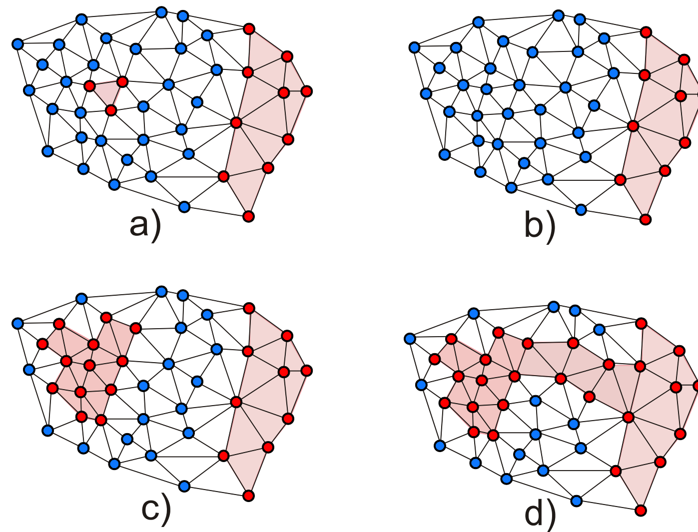



**Fig.19** Correction of overlapped clusters. Small cluster part a) could be drown b). Large cluster part c) is better to connect to the rest of cluster by bridge d).

If there is a separate part of a cluster on the surface of the mesh and it is sufficiently small, it is possible "drown" it. This means that all vertices of the small separate part of the cluster are connected to the cluster, which is adjacent or surrounding it. The greater the drowned part of this cluster is, the greater the error of its "drowning" arises. If the cluster is large enough, it is better to build a "bridge" between the two specific parts of the cluster by reassigning vertices between them. The farther away these parts are, the larger number of vertices has to be reassigned in building a "bridge", and the greater error occurs. Building a "bridge" raises a number of inconveniences: way to do so, how long it will be and what if the number of vertices for building the bridge is greater than the number of vertices of the distant part of the cluster and several situations which are difficult to solve. One such situation is presented in figure 20.

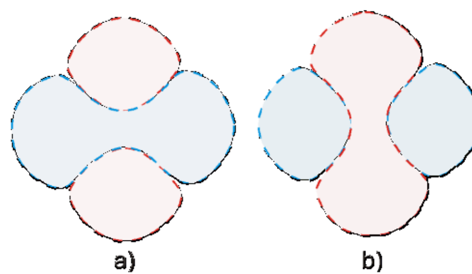



**Fig.20** Bridges between clusters. a) creation of blue bridge splits red cluster into two parts, b) creation of red bridge splits blue cluster into two parts and we need both bridges to solve this situation correctly.

To avoid these problems, we have chosen a possibility to build a "virtual" bridge, see figure 21. Such a bridge no longer consists of the triangles of the animated mesh. We can imagine that it is built above the surface of the mesh. Such a bridge only carries information about the triangle from one part of cluster in which it starts and the second triangle, where it should end, and what is the relative position of these triangles.

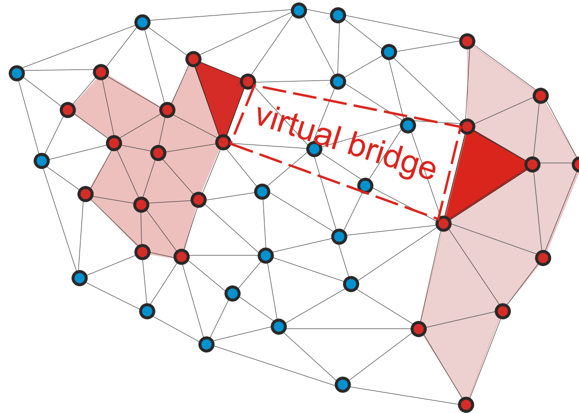



**Fig.21 V**irtual bridge built over the blue cluster.

Virtual bridge doesn't need any additional data structure or to store any special information into data stream of compressed mesh. When separate part of compressed cluster is reached, instead of starting new EdgeBreaker traversal with establishing new component, storing 3 vertex trajectories of the initial triangle, only usual parallelogram prediction is performed. This prediction is based on the shape and position of the last processed and compressed triangle of the distant cluster part. This prediction is inaccurate and the position of the first vertex of actually processed part of cluster has to be fixed using residuum vector with large values but we need only two residuum vectors instead of three vectors usually stored during EdgeBreaker initialisation and no changes in vertex clustering are needed. One more inaccurate prediction for vertices of the first triangle is performed by prediction scheme, but all following predictions are as accurate as in the rest of the mesh.

With this design we not only avoid the difficulties in the construction of "bridge", but also prevent the emergence of any errors that may occur by design.

Another problem raised by Coddyac modification using clustering is the need of proper selection of the number of clusters. The smaller clusters we choose to cover the surface of the mesh, the better complexity reduction can be achieved. However each cluster requires initialization data, which negatively affect the final compression ratio. Therefore, we need to

find the optimal number of clusters, enough to ensure that movements of vertices included in them is as ordered as possible.

Geometry of the clusters is compressed by PCA separately by the Coddyac algorithm. Each cluster has its own set of feature vectors, basis and means vector, which PCA needs to decompress the original data and thus with the increasing number of clusters, the volume of data for the feature vectors decreases, but the volume of data for basis and means vectors increases.

This geometric data constitutes the largest portion of compressed data, thus the size of the stored data file with compressed dynamic mesh strongly depends on the sum of volume of feature vectors data and volume of basis and means data. While changing the number of clusters one of these volumes increases and another decreases, and we can expect that there is a minimum in the sum of this data. Hence we can quickly estimate optimal number of clusters using descent method on this sum function. This behaviour was verified during experiments and it is presented in following section.

# 4 EXPERIMENTAL RESULTS

Our compression method uses similar procedures as the methods mentioned in Related Work section, but achieves better compression ratios. Now we will show how significant improvements in compression ratio can be achieved through clustering, but unlike the above-mentioned papers, we are dealing with an additional statement: how clustering affects the Coddyac algorithm and what clustering is the most appropriate for this algorithm. We also examine the optimal number of clusters and what is the impact of the number of clusters on the compressed data.

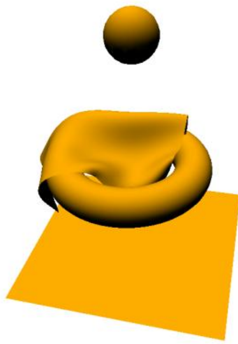
In our experiments following dynamic triangle meshes were used. Dynamic meshes are originally stored in textual WRML format in the form of sequences of static meshes, one mesh for each frame of animation. These meshes contain only information about vertex positions for individual frames and connectivity of their triangles (no normal or texturing coordinates). This data-set contains not only artificial animations of single objects, but scanned real data (jump animation) and animation of multiple separated objects (chicken and cloth animation) as well.

**Dance**

| | |
|---|---|
| Number of vertices: | 7061 |
| Number of frames: | 201 |
| Raw size: | 40.8MB |
| Coddyac compressed size: | 121.3kB (69.3kB clustered) |
| Property: | standard skinned mesh animation |

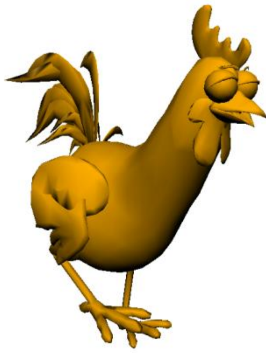
**Humanoid**

| | |
|---|---|
| Number of vertices: | 7646 |
| Number of frames: | 154 |
| Raw size: | 77.4MB |
| Coddyac compressed size: | 93.4kB (50.3kB clustered) |
| Property: | strongly rigid motions in animation |



**Cloth**

| | |
|---|---|
| Number of vertices: | 9987 |
| Number of frames: | 200 |
| Raw size: | 60.8MB |
| Coddyac compressed size: | 214.6kB (146.3kB clustered) |
| | |
| Property: | set of objects, some of them are static |



**Chicken**

| | |
|---|---|
| Number of vertices: | 3034 |
| Number of frames: | 400 |
| Raw size: | 30.3MB |
| Coddyac compressed size: | 192.6kB (121.5kB clustered) |
| | |
| Property: | set of objects, often tested data-set |

**Jump**

| | |
|---|---|
| Number of vertices: | 15830 |
| Number of frames: | 222 |
| Raw size: | 90.4MB |
| Coddyac compressed size: | 815.1kB (514.8kB clustered) |
| | |
| Property: | scanned real data |

### 4.1 Influence of Clustering on Data-size

Our clustering modification of Coddyac algorithm is based on the hypothesis that with increasing number of clusters the local entropy of vertex trajectories is decreasing and we need lower number of principal components to express their motion. In other words, with increasing number of clusters the volume of data needed for encoding the feature vectors for each trajectory decreases, but the volume of cluster-initialisation data (basis, means, unpredicted trajectories, etc.) increases. We assume that there is a minimum in the sum of this data and it is possible to use it for quick estimation of optimal number of clusters. The algorithms mentioned above and their modifications have been implemented and subsequently we have measured the impact of vertex clustering on the selected dynamic meshes.

As you can see in the figures 22 and 23, our assumption was verified during the experiments. There is a significant minimum of compressed data file size function for both Dance and Humanoid animation data-sets, and almost linear growth of PCA basis and means data for all clusters included in PCA-stream was noticed. Volume of feature vectors data representing vertex trajectories is highlighted using red colour.

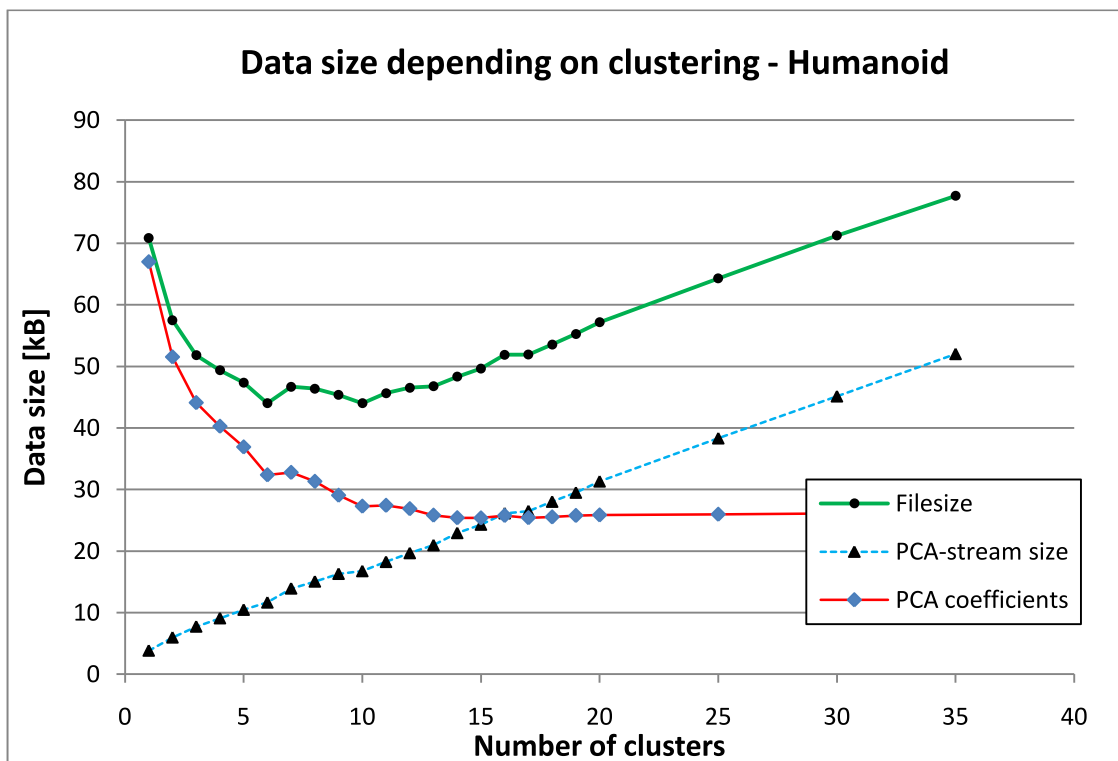



**Fig. 22** Data size comparison of whole compressed file and included PCA-stream.

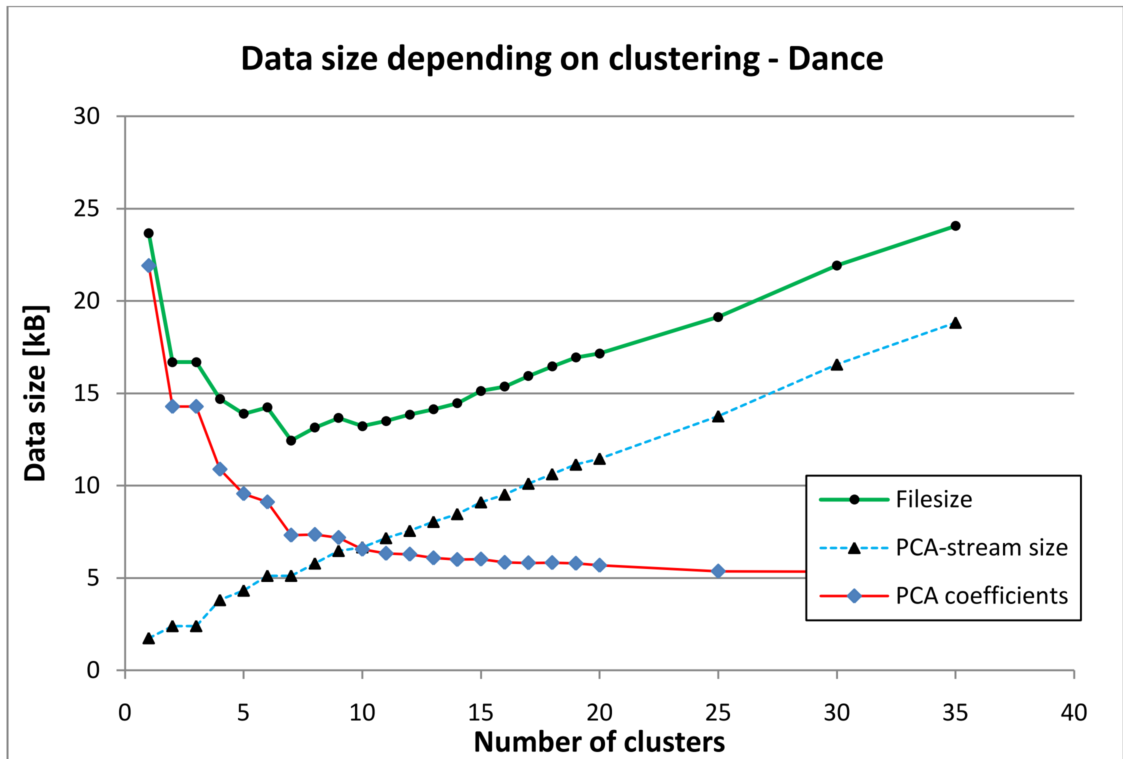



**Fig. 23** Data size comparison of whole compressed file and included PCA-stream.

It can be seen that volume of this data is decreasing until number of about 20 clusters is reached and then it stops. This behaviour is caused by the bottom limit of the number of used principal components.

We need at least 3 principal components to place trajectories correctly into the space, and thus roughly preserve the original shape and volume of compressed dynamic mesh. The entropy of the vertex trajectories usually decreases too much by using more than 20 separate clusters and therefore trajectories in common clusters are very similar each other. Therefore we need only 3 or 4 principal components to describe their motion with sufficient precision (no visible distortion of decompressed dynamic mesh). We are not able to decrease the length of feature vectors by clustering when this limit is reached, and volume of this data becomes constant for the whole animation.

It seems that the optimal number of clusters for these data-sets is in the range of 5-10 clusters according to minimum of file size function. This assumption was proved to be correct by following experiments, when RD-curves for different number of clusters were obtained.

The figure 24 shows the RD curves for clustered Coddyac and their comparison with not clustered Coddyac (only 1 cluster used). In this experiment we used k-means clustering algorithm because the same clustering algorithm was used in the previous experiment. For this experiment the number of 8 clusters provided the best result.

**Fig. 24** Comparison of different number of clusters. Red line –not clustered, blue line –the best result

| Humanoid | | | |
|---|---|---|---|
| Number of clusters | KG-error | Number of used basis vectors | |
| | | Total | Average |
| 1 | 0,8 | 26 | 26,0 |
| | 0,4 | 29 | 29,0 |
| | 0,1 | 33 | 33,0 |
| 3 | 0,8 | 13, 20, 9 | 13,9 |
| | 0,4 | 14, 21, 9 | 14,5 |
| | 0,1 | 17, 23, 11 | 16,7 |
| 8 | 0,8 | 7, 11, 3, 4, 13, 6, 8, 6 | 7,3 |
| | 0,4 | 8, 13, 3, 5, 16, 7, 8, 6 | 8,1 |
| | 0,1 | 8, 15, 7, 5, 18, 14, 11, 8 | 10,6 |
| 50 | 0,8 | 3, 3, 3, 3, 3, 5, 5, 3, 4, 3, 3, 3, 3, 3, 3, 3, 3, 3, 5, 3, 3, 3, 3, 3,.. | 3,3 |
| | 0,4 | 3, 3, 3, 3, 3, 5, 7, 4, 5, 6, 3, 3, 3, 3, 3, 7, 6, 3, 4, 6, 6, 5, 3, 3, 3,.. | 4,3 |
| | 0,1 | 3, 3, 3, 3, 3, 5, 7, 5, 5, 8, 3, 3, 3, 3, 3, 9, 7, 3, 5, 6, 6, 5, 3, 3, 4,.. | 4,7 |

**Fig. 25** Comparison of different number of clusters with the decreasing average number of basis vectors.

The table in figure 25 shows how rapidly the clustering decreases the number of basis vectors needed to maintain the same error when the number of clusters is increasing.

## 4.2 Influence of Clustering Methods

In the case of k-means clustering, we have further experimented with different calculations of distances in the clustered data-set and we discovered that the usually used Euclidean distance calculation ($L_2$ norm) can be in average slightly outperformed by $L_1$ norm distance calculation, especially in the ranges of the KG-error below the value 0.2 (figure 26).

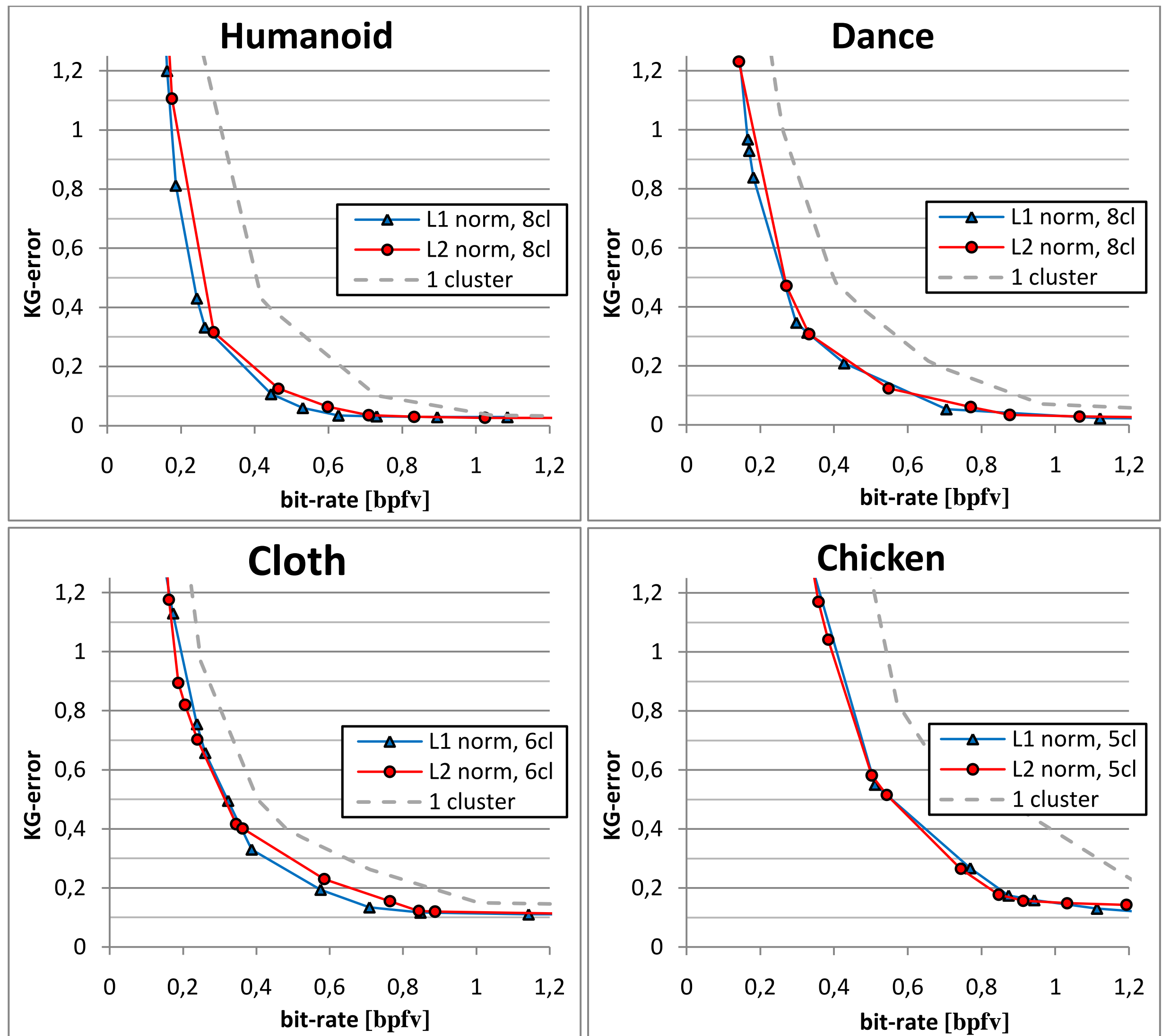



**Fig. 26** Comparison of different distance norms for k-means clustering.

To ensure that the tested clustering methods work well and the number of clusters is optimal, we were testing random clustering as well. The random vertex clustering algorithm creates (user specified) *k* clusters using randomly selected *k* initial cluster vertices. Area of each cluster is then iteratively expanded by appending all topologically neighbouring vertices while some unassigned vertices remain. Experimental results are presented in the figure 27.

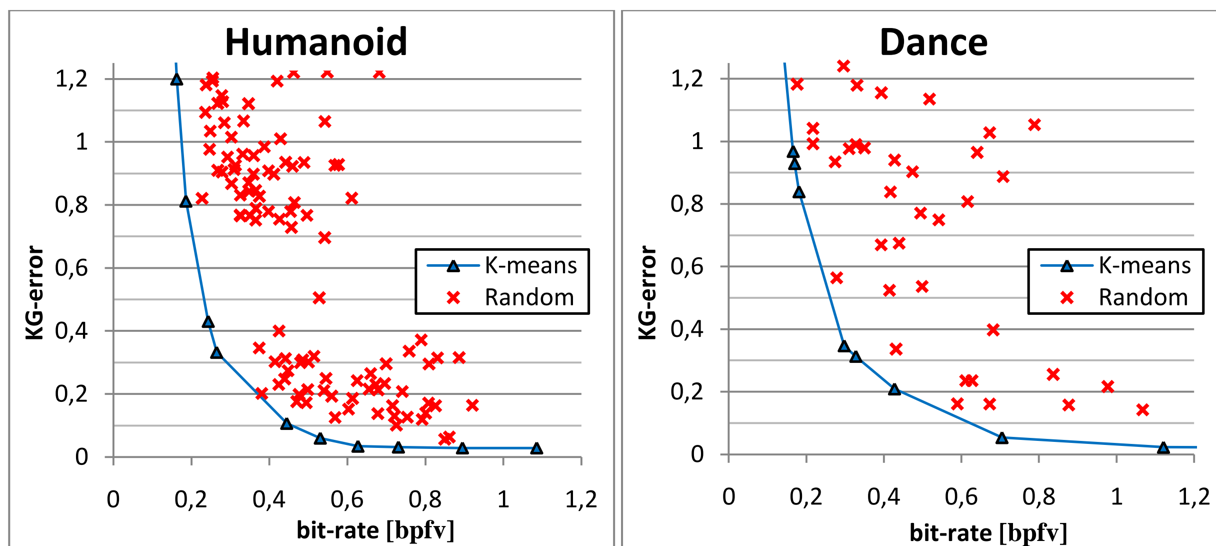



**Fig. 27** Comparison of random clustering and k-means clustering.

As indicated in section 3.1.4, we were testing edge-collapse based clustering as well. First, we have experimented with the original FAMC cost function, but computation of this cost function is too time consuming, therefore we tried to use another cost function. This function is described in the same section and it uses thresholded feature vectors.

The modified clustering is exploiting the original geometry-topological approach, but the cost function is focused on the parts of feature vectors, which could be neglected by final arithmetic coding –zero or almost zero coefficients of feature vectors. This vector value thresholding causes significant loss of information about exact vertex trajectories, but we expected that this loss of information could be compensated by the nature of the edge-collapse based clustering algorithm, which considers topological distance between vertices (using edge collapsing). Vertices, which are close to each other on the surface of the animated mesh, have usually very similar trajectories.

However, compression results of this clustering modification are not better than the original FAMC cost function and the original FAMC cost function provides in average the same or worse compression results than the strictly geometry-driven k-means clustering. Only the dance animation was compressed with slightly better compression ratio, while maintaining the same error. Results of experiments with edge-collapse based clustering are depicted in the figures 28, 29 and 30.

Finally, we were testing the facility-location clustering algorithm, which is similar to the k-means, but neither of these clustering approaches outperformed the k-means (figure 31).

**Fig. 28** Compression results using the edge-collapse based clustering with original cot function

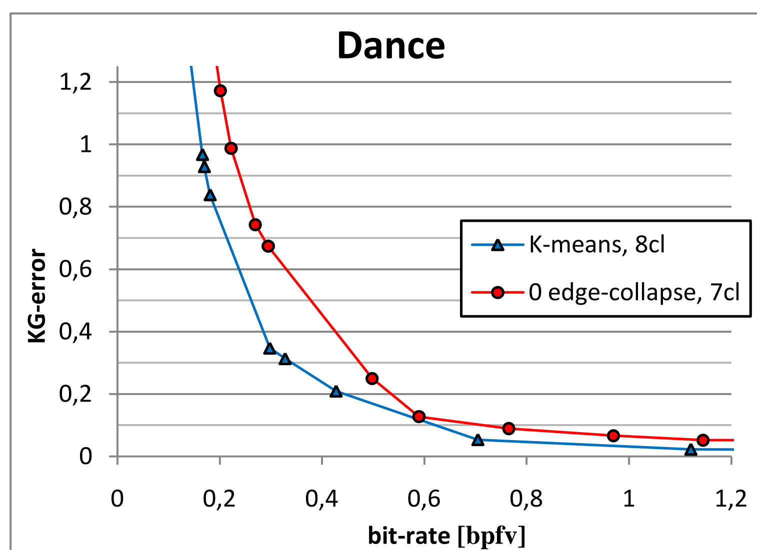



**Fig. 29** Compression results using the edge-collapse based clustering with modified cost function

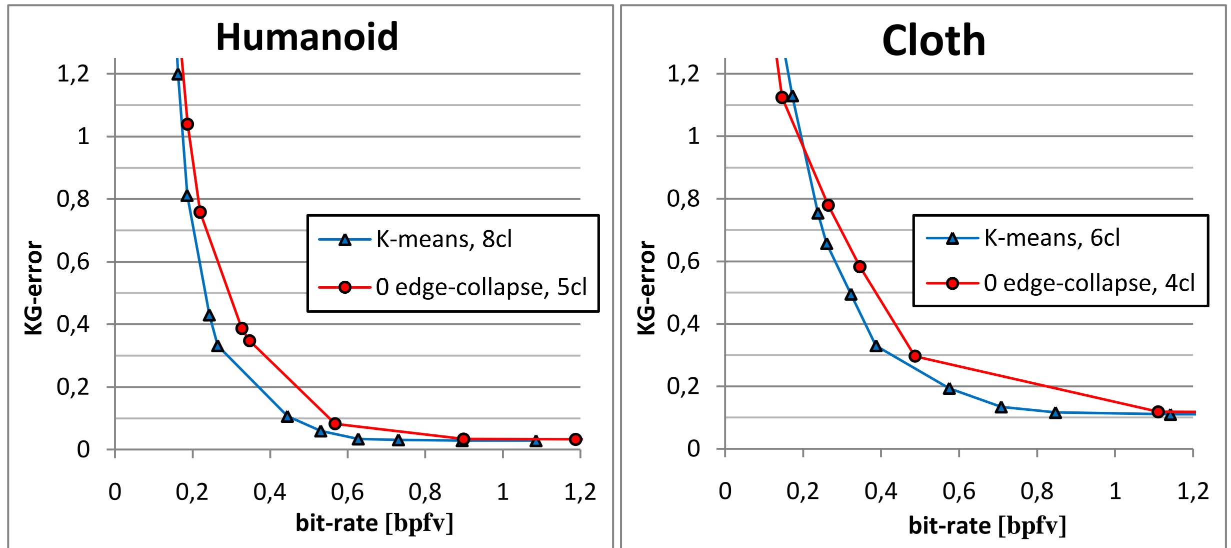



**Fig. 30** Compression results using the edge-collapse based clustering with modified cost function

Dance
KG-error
K-means, 8cl
Facility location, 8cl
bit-rate [bpfv]

Humanoid
KG-error
K-means, 8cl
Facility location, 9cl
bit-rate [bpfv]

Chicken
KG-error
K-means, 5cl
Facility location, 6cl
bit-rate [bpfv]

Cloth
KG-error
K-means, 6cl
Facility location, 8cl
bit-rate [bpfv]

**Fig. 31** Compression results using the facility location clustering

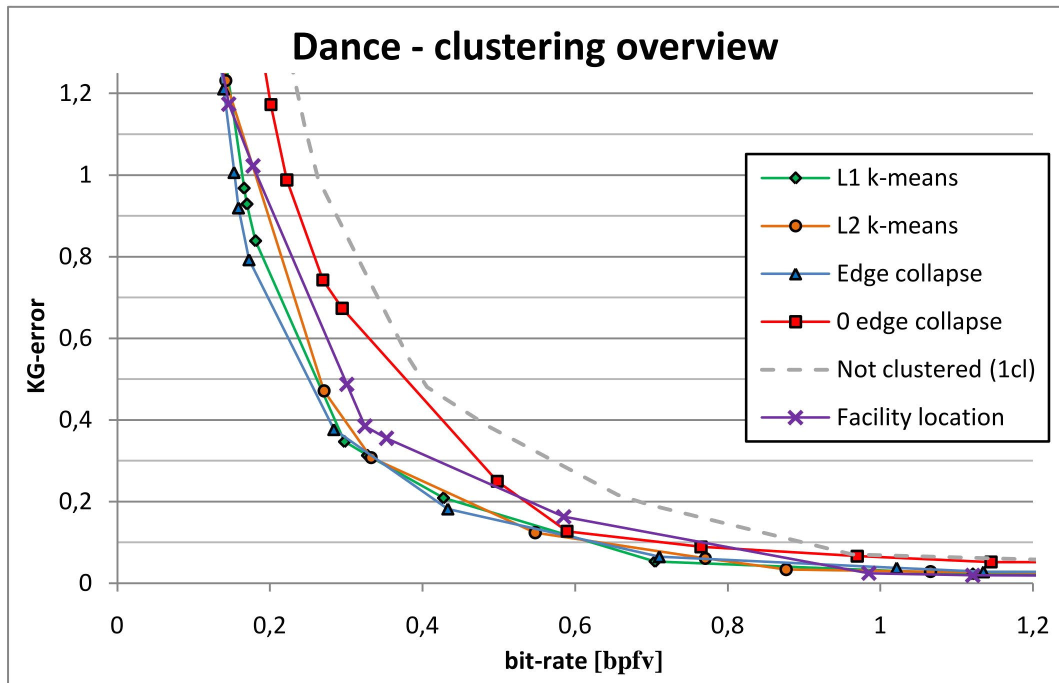



**Fig. 32** Comparison of all tested clustering methods

Due to our experiments we recommend the k-means clustering method using the $L_1$ norm distance calculation for PCA-based dynamic mesh compression, because it leads to the best compression results in the range of 33%-49% better bit-rate while maintaining the same KG-error.

The figure 32 shows that each method has brought some improvement of compression ratio. In the figure 32, comparing different methods of clustering, we can see that the largest improvement was brought by the Edge-collapse based clustering method with original FAMC cost function, which is described in section 3.1.4 but in general, k-means provides the same or better compression results and it is at least an order of magnitude faster. Unlike the original version of Coddyac, the algorithm improved by clustering can achieve significantly lower bit rate while maintaining the same error.

The best compression results were achieved using small number of clusters (5-8), except the jump animation, where 25 clusters had to be used. All animations, except the jump animation, are artificially created by artists or scientists but the jump animation was made by scanning real data and thus it contains information about vertex trajectories with much higher entropy (and noise) than the others.

Implementations of the FAMC and the facility-location algorithms were kindly provided by Ing. Oldřich Petřík and Ing. Jiří Skála.

When comparing errors resulting from compression of 3D animated meshes we used both KG-error measure and STED measure, in order to obtain more detailed information about caused distortion. Both of these metrics are described in section 2.6. Results measured by STED error are slightly different from the previously mentioned results, because STED error better correlates with the human perception of the distortion caused by clustered Coddyac compression.

Clustered Coddyac compression causes, that visible cracks between neighbouring clusters may appear. These cracks are almost ignored by KG-error measurement, but the STED notices them. With increasing number of clusters increases the length of the borders between clusters in general and thus the amount of cracks and the error caused by compression algorithm increases as well. You can see the different behaviour of measured error in the figure 33 and the compression results obtained using the STED instead of KG-error in the figure 34.

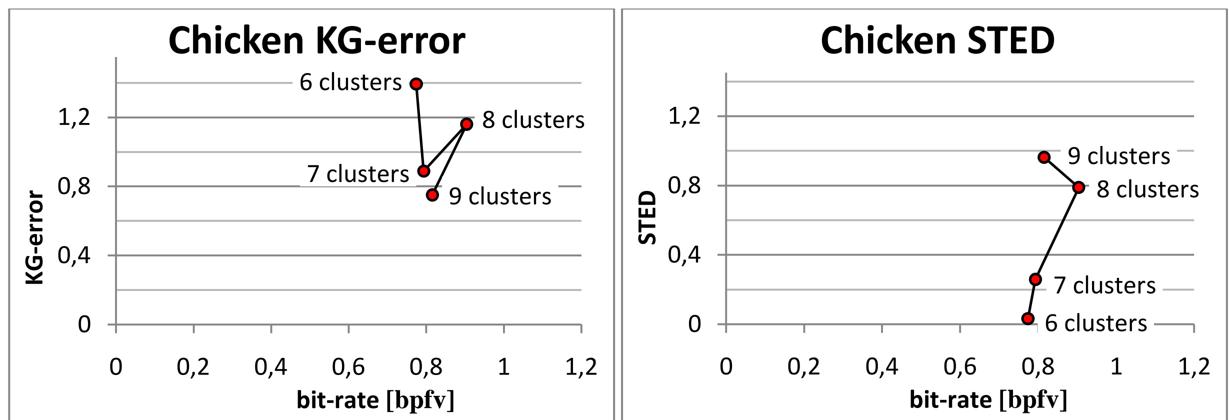



**Fig. 33** Different behaviour of measured error (fixed compression configuration, varying number of clusters).

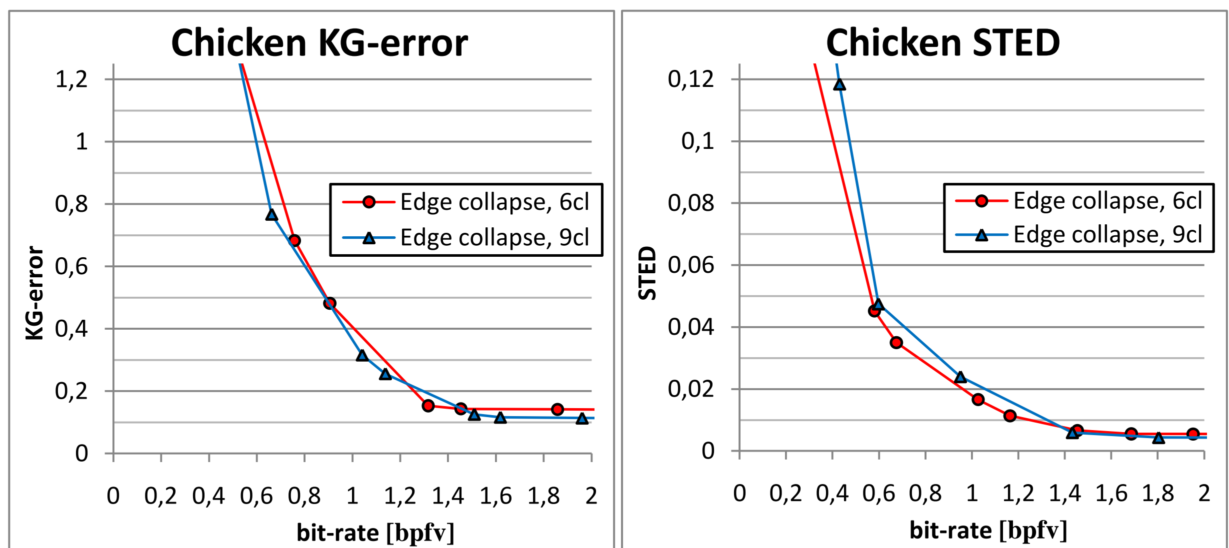



**Fig. 34** Different compression results obtained using the STED and the KG-error.

More compression results obtained using the STED metric are depicted in figure 35.

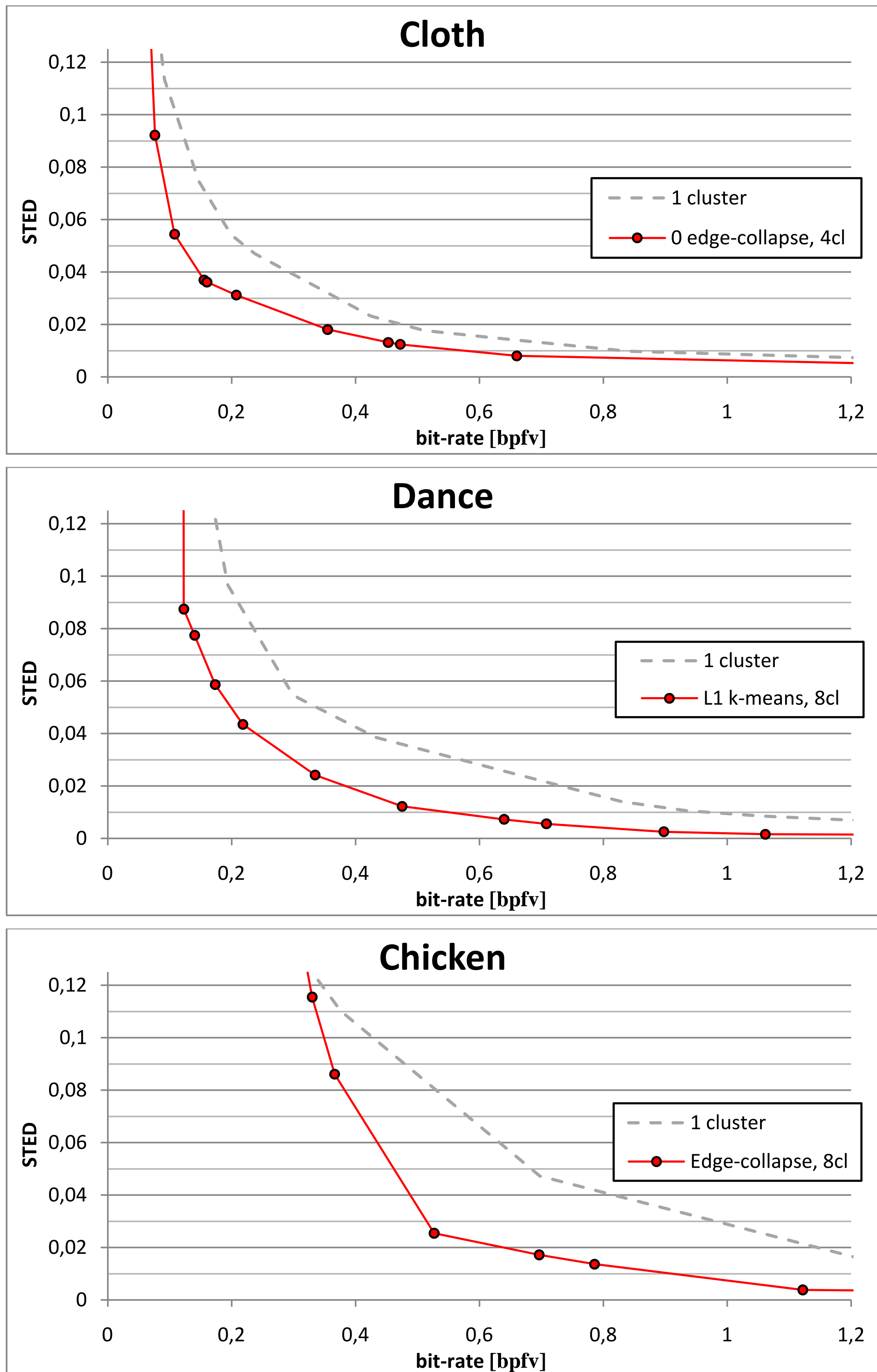



**Fig. 35** Compression results obtained using the STED metric.

# 5 CONCLUSION AND FUTURE WORK

In this thesis we have examined the influence of vertex clustering on the Coddyac algorithm. The improvement lies in the suitable division of compressed mesh into smaller parts, which are then compressed separately. Purpose of this decomposition is to reduce the complexity of movement in animation, which leads to a better compression ratio. The selection of the appropriate number of clusters depends on many aspects hence we use the “try and error” method. But as can be seen in figures 22 and 23, by using the descent method we can quickly find a minimum of the dependence of compressed animation size on the number of clusters and then find the best configuration in the sense of RD curves. This optimal number of clusters is not dependent on the structure of animated mesh neither the complexity of vertex trajectories, but especially on the differences between trajectories of individual vertices and for most tested meshes we have found that values of 5-10 clusters provide good results.

We have tested several methods for clustering of dynamic meshes. Of these methods the best results are in average provided using the k-means clustering algorithm. It should be noted that the improvement compared to the edge-collapse based clustering with original cost function is negligible (+/- 5%), but k-means is at least an order of magnitude faster. The figure 32 shows that the use of clustering in the original compression algorithm is able to improve the performance of the algorithm in terms of errors and data rate reduction. The bit rate can be reduced by 33%-49% compared to the Coddyac output while maintaining the same KG-error.

Theoretically including clustering algorithm can improve the compression ratio without substantial increase in time complexity, because clustering algorithms are usually not too time-consuming. But since we mostly used the k-means clustering algorithm, which works with trajectories processed by PCA, for example to analyze the movement of 3D-mesh, the resulting time required for compression can rise to twice the original time complexity of the Coddyac. However, the use of clustering has almost no effect on the decompression time, which is more important for a practical application of the algorithm.

In the future we would like to explore the possibility of avoiding the use of PCA in the stage of clustering, automatize the selection of the number of clusters and find more efficient clustering algorithms. Currently, our work is focused on clustering, where each vertex belongs to only one cluster, but we would also like to try to soften the transitions between clusters as for example in [14], to enhance the impression of animations compressed with coarse

quantization and to comply with the requirements of the STED error metric. For this future modification of our compression algorithm, we consider using some of the ideas of the extension of MPEG-4 standard FAMC recently proposed by Petřík and Váša in [25].

Another possible future improvement is clustering by using the time-line, i.e. not only examining the changes in position of vertices in time, but also the changes of movement complexity of individual trajectories depending on time. This could lead to more clusters but shorter feature vectors. We would also like to use automatic rate-distortion optimisation to find the optimal cluster-varying setting for the compression algorithm [24].

Results of this thesis were presented in [23] and awarded as the Best Suitable Commercial Application on the $6^{th}$ international Conference on Articulated Motion and Deformable Objects in 2010.

# REFERENCES

1. Váša, L., Skala, V.: CoDDyAC: Connectivity Driven Dynamic Mesh Compression. Proceedings of 3DTV Conference 2007 (2007)
2. Smith, L. I.: A tutorial on Principal Component Analysis. (2002)
3. Rossignac, J.: EdgeBreaker: Connectivity compression for triangle meshes. IEEE Transactions on Visualization and Computer Graphics, Vol. 5, No. 1, January – March (1999)
4. Kanungo, T., Mount, D. M., Netanyahu, N. S., Piatko, C. D., Silverman, R., Wu, A. Y.: An Efficient k-means Clustering Algorithm: Analysis and Implementation. IEEE Transactions on pattern analysis and machine intelligence, Vol. 24, No. 7,July 2002 (2002)
5. Mamou, K., Zaharia, T., Prêteux, F.: FAMC: The MPEG-4 Standard for Animated Mesh Compression. ARTEMIS Department, Institut TELECOM
6. Charikar, M., Khuller, S., Mount, D. M., Narasimhan, G.: Algorithms for facility location problem with outliers. In Symposium on Discrete Algorithms, 642-651(2001)
7. Karni, Z., Gotsman, C.: Compression of soft-body animation sequences. In Elsevier Computer & Graphics, Vol. 28, pages 25-34 (2004)
8. Váša, L., Skala, V.: COBRA: Compression of the Basis for PCA Represented Animations. Computer Graphics Forum, Vol 28, pages 1529-1540 (2009)
9. Amjoun, R., Straßer, W.: Efficient Compression of 3D Dynamic Mesh Sequences. Journal of the WSCG (2007)
10. Sattler, M., Sarlette, R., Klein, R.: Simple and Efficient Compression of Animation Sequences. Proceedings of the 2005ACM SIGGRAPH/Eurographics symposium on Computer animation (SCA 2005), pages 209-217 (2005)
11. Gotsman, C., Touma, C.: Triangle Mesh Compression. Graphics Interface, pages26-34 (1998)
12. Skála, J., Kolingerová, I.: Clustering Geometric Data Streams. SIGRAD, pages17-23 (2007)

13. Alexa, M., Müller, W.: Representing Animations by Principal Components. Computer Graphics Forum 19(3): (2000)

14. Kavan, L., Sloan, P. P., O'Sullivan, C.: Fast and Efficient Skinning for Animated Meshes. Computer Graphics Forum 29(2), pages 327-336 (2010)

15. Sorkine, O., Cohen-Or, D., Toledo, S.: High-Pass Quantization for Mesh Encoding. Symposium on Geometry Processing, 42–51 (2003)

16. Chow, M.: Optimized Geometry Compression for Real-Time Rendering. Proc. IEEE Visualization, 347–354 (1997)

17. Váša, L., Skala, V.: A Perception Correlated Comparison Method for Dynamic Meshes. IEEE Transactions on Visualisation and Computer Graphics, Vol. 17(2), (2010)

18. Marpe, D., Schwarz, H., Wiegand, T.: Context-Based Adaptive Binary Arithmetic Coding in the H.264 / AVC Video Compression Standard. IEEE Transactions on Circuits and Systems for Video Technology, Vol. 13, No. 7, 620-636 (2003)

19. Dimitrov, D., Holst, M., Kriegel, K.: Experimental Study of Bounding Box Algorithms. Proceedings of International Conference on Computer Graphics Theory and Applications - GRAPP, 15–22, Funchal, Portugal (2008).

20. Rossignac, J., Ibaria, L.: Dynapack: Space-Time compression of the 3D animations of triangle meshes with fixed connectivity. Symposium of Computer Animation 03 (2007)

21. Zhang, J., Owen, C. B.: Octree-based animated geometry compression. DCC ´04: Proceedings of the Conference on Data Compression (Washington, DC, USA), IEEE Computer Society, p. 508 (2004)

22. International Organisation for Standardization, ISO/IEC 14496 Part 2: Visual. International Organisation for Standardisation (2001)

23. Rus, J., Váša, L.: Analysing the Influence of Vertex Clustering on PCA-Based Dynamic Mesh Compression. Articulated Motion and Deformable Objects, Heidelberg: Springer, p. 55-66 (2010)

24. Petřík, O., Váša, L.: Finding Optimal Parameter Configuration for a Dynamic Triangle Mesh Compression. Articulated Motion and Deformable Objects, Heidelberg: Springer, p. 31-42 (2010)

25. Petřík, O., Váša, L.: Improvements of MPEG-4 Standard FAMC for Efficient 3D Animation Compression. Proceedings of 3DTV Conference 2011 (2011)
26. Huffman, D. A.: A Method for the Construction of Minimum-Redundancy Codes. Proceedings of the I.R.E, p. 1098-1102 (1952)
27. Rissanen, J. J., Langdon, G. G.: Arithmetic Coding. IBM Journal of Research and Development 23 (2), p. 149-162 (1979)
28. Khan, S. S., Ahmad, A.: Cluster center initialization algorithm for k-means clustering. Pattern Recognition Letters - PRL 25 (11), p. 1293-1302 (2004)
29. Mitra, P., Murthy, C. A.: Density based multiscale data condensation. IEEE Transactions Pattern Analysis and Machine Intelligence 24 (6), p. 734-747 (2002)
30. Hendrickson, B., Leland, R. W.: A multi-level algorithm for partitioning graphs. Proceedings of the 1995 ACM/IEEE conference on Supercomputing, p. 28 (1995)
31. Lloyd, S. P.: Least squares quantization in pcm. IEEE Transactions on Information Theory 18 (2), p. 129-137 (1982)
32. Liu, R., Zheng, H.: Segmentation of 3D meshes through spectral clustering. Proceedings of Pacific Graphics, p. 298-305 (2001)
33. Ramanathan, S., Kassim, A. A., Tan, T.: Impact of vertex clustering on registration-based 3D dynamic mesh coding. Image and Vision Compution 26 (7), p. 1012-1026 (2008)
34. Besl, P., McKay, N.: A method of registration of 3D shapes. IEEE Transactions on Pattern Analysis and Machine Intelligence 14 (2), p. 239-256 (1992)

# APPENDIX A - Implementation Overview

Experimental software provided with this thesis was created in the form of modules for the Modular Visualization Environment 2 (MVE-2) platform. The set of modules was written in C# using Microsoft Visual Studio 2010.

MVE-2 project is based on .NET technology and it is developed at the University of West Bohemia. It is a data-flow based modular environment primarily intended for researchers and students as data visualization and processing tool (figure 36). MVE-2 experiments are defined by maps of modules, which are linked together representing subroutines of the experiment and appropriate data-flow. Each module has individual settings, input ports and output ports (output port of one module can be connected with input port of another module to transmit desired data between them). Maps of experiments are executed using MVE-2 MapEditor which is able to collect and backup results of experiments as well.

Mve-2 contains default library of various modules for data loading, saving, processing and visualization, but it is possible to create and plug in new modules using inheritance mechanism (each module is inherited from the Module API class of the MVE-2). You can learn more about MVE-2 from mve.kiv.zcu.cz.

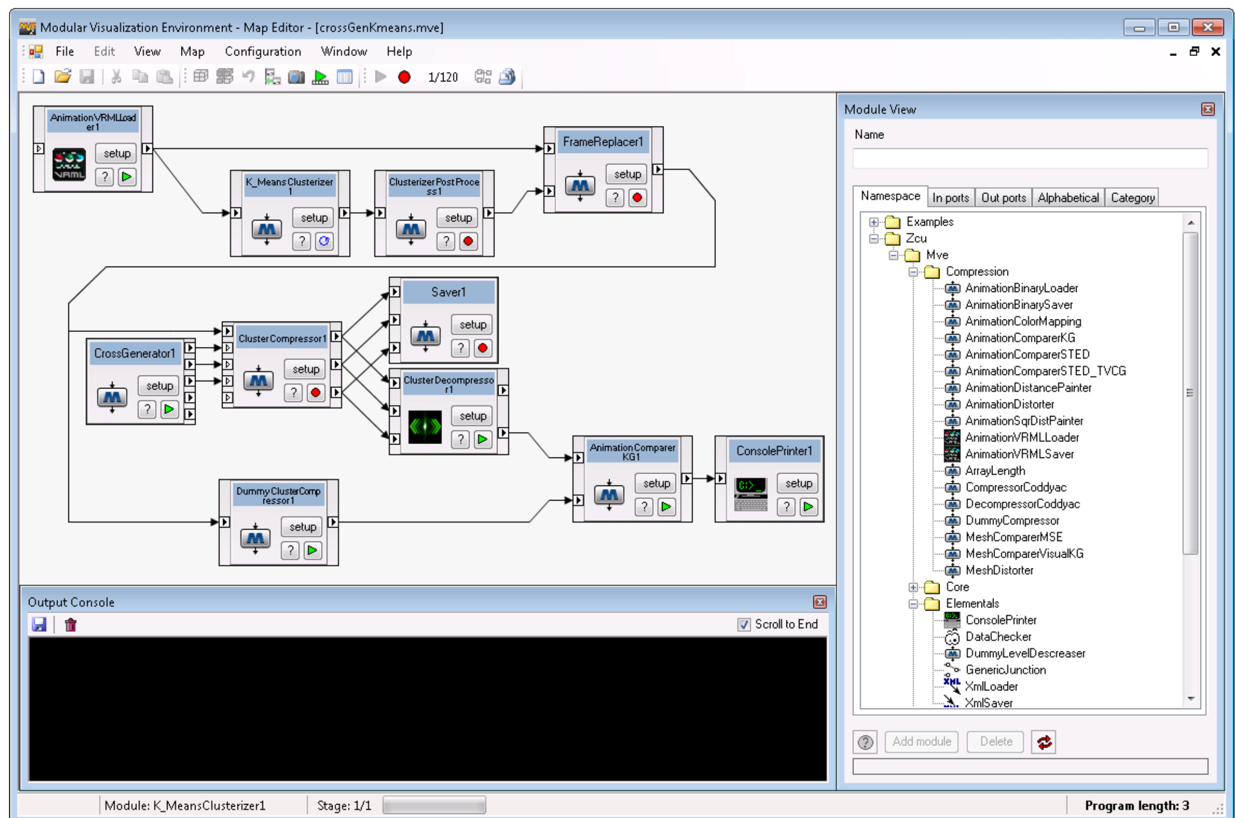



**Fig. 36** Modular Visualization Environment 2

Crucial part of the experimental software consists of the ClusterCompressor and ClusterDecompressor modules. These modules implement the clustered Coddyac compression (and decompression) scheme described in section 3.3. Expected data type of the ClusterCompressor input and the ClusterDecompressor output is UniformDataArray. It is standard MVE-2 data type which is (in the case of this thesis) used for 3D animation representation, where each component of the array contains one frame of the compressed animation in the form of triangle mesh (TriangleMesh data type). ClusterCompresor generates result of the compression divided into three individual blocks of data, one block for each output port of the module. These output ports provide Topology Description and Geometry Description using UniformDataArray data type and stream with compressed PCA basis using MveMemoryStream data type. ClusterDecompressor module expects the same kind of data as an input. Using the ClusterCompressor settings, user can set the quantization coefficients VectorQuantization and BasisQuantization and maximum trajectory error TrajectoryError, which may be caused by the clustered Coddyac compression.

Next very important module is ClusterizerPostProcess used to fix clustering of vertices of the input mesh, if clusters are not topologically compact. This module has only one parameter (SmallComponentSize), which specifies, what is the maximum number of vertices the separated cluster part may consist of in order to drown it.

The DummyClusterCompressor is useful when we want to compute deviations between the decompressed dynamic mesh and the original dynamic mesh. Order of vertices is changed during the clustered Coddyac compression and the DummyClusterCompressor changes the order of vertices of the original dynamic mesh without any other modification of geometry. This module has no settings.

Outputs of ClusterDecompressor and DummyClusterCompressor can be compared later for example by AnimationComparerKG or AnimationComparerSTED, which are available as a part of Dynamic Mesh Compression Toolkit for MVE-2. This toolkit can be downloaded from above mentioned web pages.

Next, we need to measure compression bit-rate to complete the information about the efficiency of the clustered Coddyac compression and this information is provided by Saver module. This module has three input ports equal to ClusterCompressor output ports and only one setting defining the saved file name.

Finally, two clustering modules were created and three modules were provided by Ing. Oldřich Petřík (FAMCClusterizer), Ing. Jiří Skála (MveClusterer) and Ing. Libor Váša Ph.D. (RandomClusterizer).

The FAMCClusterizer is part of reimplementation of the FAMC compression scheme and it expects UniformDataArray with the original dynamic mesh as an input, similarly to other clustering modules. Settings of this module enable determining the clustering StopCondition precisely specified by ClusterCout or MaximumError constants and the method for connecting individual components of the dynamic mesh (ConnectMethod).

RandomClusterizer setting enables to set the desired number of clusters as well. This module performs clustering with random positions and sizes of clusters.

MveClusterer is implementation of the facility location clustering method. Unlike the other clustering modules, this module processes data defined using TriangleMesh data type with vector of data (VectorND) assigned to each vertex of the mesh. Due to this fact, BypassToMveClusterer and BypassFromMveClusterer modules were implemented to change the data structure and format from/to usually used UniformDataArray.

FAMCZeroClusterizer is a modification of the FAMCClusterizer and it differs from the original module in the way it computes the edge-collapse cost function, but it has the same settings, input ports and output ports as the FAMCClusterizer. The cost function is described in section 3.1.4.

The last created module is K_MeansClusterizer, which performs k-means clustering. This clustering module provides setting of the NumberOfClusters and the PercentageAccuracy specifying how accurately the clustering has to be performed (uses PCA). It contains boolean TrajectoryShiftToCenter setting as well, but it is usually set to false, because the trajectory shifting into the origin of the coordinate system has negative effect on the compression results. Finally, the DistanceNorm setting is provided by the K_MenasClusterizer module. This setting enables to set any value $n$ to enable the $L_n$ distance norm calculation during k-means clustering.

Source codes of these modules are thoroughly commented and sample MVE-2 maps are attached to provide complete picture of the function of these modules.