Západočeská univerzita v Plzni
Fakulta aplikovaných věd

METODY REDUKCE VELIKOSTI DYNAMICKÝCH SÍTÍ

Ing. Libor Váša

disertační práce
k získání akademického titulu doktor
v oboru Informatika a výpočetní technika

Školitel: Prof. Ing. Václav Skala, CSc.

Katedra: Katedra informatiky a výpočetní techniky

Plzeň 2008

University of West Bohemia
Faculty of Applied Sciences

METHODS FOR SIZE REDUCTION OF DYNAMIC MESHES

Ing. Libor Váša

doctoral thesis
submitted in partial fulfilment of the requirements for a degree of Doctor of Philosophy in Computer Science and Engineering

Supervisor: Prof. Ing. Václav Skala, CSc.

Department: Department of Computer Science and Engineering

Pilsen 2008

Prohlášení

Předkládám k posouzení a obhajobě disertační práci zpracovanou na závěr doktorského studia na Fakultě aplikovaných věd Západočeské univerzity v Plzni. Prohlašuji, že tuto práci jsem vypracoval samostatně, s použitím odborné literatury a dostupných pramenů uvedených v seznamu, jenž je součástí této práce.

V Plzni dne 10. května 2008

Ing. Libor Váša

Abstract

This work focuses on processing of dynamic meshes, i.e. series of triangular meshes of equal connectivity which represent an evolution of a model surface in time. The main goal is to reduce the storage and transmission cost. The work covers the state of the art in the fields of compression and simplification of triangular meshes, and presents several novel methods for dynamic mesh compression and comparison.

By compression we mean methods of encoding the mesh without altering its connectivity. Usually some sort of vertex position prediction is used, combined with delta coding of residuals. PCA (Principal Component Analysis) based compression methods are also discussed.

The simplification methods on the other hand reduce the storage cost of a dynamic mesh by changing its connectivity. There are only a few methods that are directly targeted on dynamic mesh simplification, and therefore we also discuss methods for static mesh simplification.

In the last section of the state of the art we present methods for comparing dynamic meshes that can be used to evaluate the processing methods. We present the currently used metrics and point out their advantages and disadvantages.

In the second part of the thesis, we will present novel methods for dynamic mesh compression, based on PCA. We will first show a combination of PCA and Edgebreaker, which will be improved by using simplification as a part of the algorithm, and further extended by new advanced prediction techniques. We will also show a method for compression of PCA basis, which is useful for achieving low bitrates.

We will show that our final algorithm achieves better results than the state of the art methods using the traditional KG error measure. Finally, we will also derive a new measure focused on the actual perceived quality of the decompressed mesh, which provides high correlation with the results of subjective testing that we have performed.

keywords: animation, compression, triangle mesh, dynamic mesh, simplification, reduction, PCA, error measure, perception, quality

Abstrakt

Tato práce se zabývá zpracováním dynamických sítí. Dynamickou sítí rozumíme sérii trojúhelnikových sítí se stejnou konektivitou, která představuje časový vývoj objektu. Naším hlavním cílem je snížení velikosti datové reprezentace dynamické sítě. Tato práce pokrývá současný stav výzkumu v oblastech komprese a simplifikace dynamických sítí a navrhuje nové algoritmy pro kompresi a porovnávání dynamických sítí.

Kompresí označujeme takové metody uložení dynamické sitě, které nemění její konektivitu. Obvyklým postupem je vytvoření nějakého prediktoru polohy vrcholu, jehož predikce je pak korigována pomocí delta-coding strategie. Práce rovněž popisuje metody založené na PCA (Principal Component Analysis) a waveletové dekompozici.

Narozdíl od kompresí, simplifikace dosahují snížení velikosti dat obecnou změnou konektivity. V literatuře je popsáno jen málo metod pro simplifikaci dynamických sítí, a proto se práce zabývá rovněž metodami pro kompresi statických sítí a možnostmi jejich rozšíření.

V poslední části popisu existujících technik se zabýváme metodami porovnávání dynamických sítí, které je možno použít pro zhodnocení efektivity kompresních a simplifikačních metod. Práce se zabývá současnými metodami pro porovnávání sítí a zmiňuje jejich významné nedostatky.

V druhé části textu zavedeme nové algoritmy pro kompresi dynamických sítí, založené na analýze hlavních komponent. Nejdříve ukážeme kombinaci PCA a algoritmu EdgeBreaker, kterou následně vylepšíme zahrnutím simplifikace do algoritmu a rozšířením o nové pokročilé predikční techniky. Ukážeme také novou metodu komprese báze PCA, která je užitečná pro dosažení nízkého datového toku.

Ukážeme, že náš konečný algoritmus poskytuje lepší výkon než současné metody v měřítku KG error. Nakonec také odvodíme novou míru chyby založenou na skutečné vnímané kvalitě dekomprimované sítě, která vykazuje vysokou korelaci s výsledky realizovaného subjektivního testování.

klíčová slova: animace, komprese, trojúhelníkové sítě, dynamické sítě, simplifikace, zjednodušení, analýza hlavních komponent, měření chyby, vnímání, kvalita

Acknowledgements

I would like to express my gratitude to prof. Václav Skala, the supervisor of this work, who has provided valuable support during my whole doctoral studies. I would also like to thank Aljoscha Smolic and Yucel Yemez for providing valuable resources, to Nikolce Stefanoski and Khaled Mamou for numerous consultations, to my colleagues - Martin Janda, Slavomír Petrík and Petr Vaněček - for numerous hints, comments and Quake battles, and to Vojtěch Hladík for the help with the GPU implementation.

I am dedicating this work to my parents, who have always supported me in both happier and tougher times.

Finally, Gabrielle, my muse and inspiration, is to be thanked for being a great counterpart to my work life.

This work has been supported by the project 3DTV PF6-2003-IST-2, Network of Excellence, No: 511568 and by the project Centre of Computer Graphics, National Network of Fundamental Research Centers, MSMT Czech Rep., No: LC 06008.

Contents

Chapter 1

Introduction

Computer graphics is one of the most constantly developing fields in computer science. While the growth of processing power of common PCs seems to slow down in the last years, computer graphics related hardware is developing still very rapidly, and computer graphics as a supporting science is reaching to problems that seemed impossible to solve just a few years ago.

In the past, computer graphics gave answers to some non-trivial questions related to displaying and processing virtual models. The key issue for most of the computer graphics related problems is the problem of computational complexity. In other words, the results in computer graphics must not only be correct, but they also need to be delivered in a reasonable time.

The problems of computer graphics span from the data acquisition (polygonization and tessellation algorithms) through various processing techniques (simplification, stripification etc.) up to the displaying itself (shading, rendering). Most of the algorithms in each of the areas are designed to provide the best possible quality/performance ratio, so that current limited hardware can process as complex (and therefore realistic) scenes as possible. However, the recent development of accelerating hardware is not making the old problems and their solutions obsolete, it merely allows larger new problems to be solved.

A typical case where recent hardware progresses have allowed broadening of horizons of solvable problems is the current possibility of studying time-variant cases of problems that were so far only studied in their static forms. Current hardware allows real-time acquisition of data on one hand, while emerging display devices allow delivering full 3D impression of a dynamic virtual scene to the user. The applications span from scientific problems, such as dynamic volume data processing, up to pure entertainment, such as 3D television.

While some of the current algorithms used for processing of static scenes can

be used directly for the dynamic case, a large group of existing solutions cannot be used in dynamic case at all, or only at high processing cost. This gives us a wide new area for research, where it seems to be useful to take a look at the old problems from a slightly different point of view. The sophisticated algorithms for static problems give us a good platform for such research, but for many problems a completely new approaches are likely to rise.

Problem of storage requirements is closely related to the general performance criterion. Large and complex meshes allow high precision, but also require high processing power. The problem of transmission of models is also of increasing importance, as today's network connections are still limited in their bandwidth.

The problem of reduction of storage requirements has been addressed by several approaches for the case of static meshes, while in recent years a more complex dynamic case is also being considered, which is also the main topic of this work. We will show the main current approaches used for both static and dynamic case.

1.1 Problem definition

In this work we will focus our effort on a special case of the problem described in the introduction. We will be considering the case of constant connectivity dynamic triangular meshes, and we will investigate possibilities of compression and simplification of such meshes.

A triangular mesh is a spatial dataset, which represents a surface of an object. It consists of geometry and topology. Geometry information is usually represented by a set of three component vectors, where each vector represents one point in 3D space. Topology information is usually represented by a set of index triplets, where each triplet represents indices of three vertices of the mesh topology, which form a triangle.

For some algorithms it is crucial to differentiate between the so-called manifold and nonmanifold meshes. Manifold meshes are defined by several additional conditions for the mesh topology. These are:

- each edge is shared by at most two triangles
- neighborhood of each vertex is topologically equivalent with a disc or a border.

Of importance is also a so-called simple mesh. A simple mesh is a genus 0 manifold mesh, i.e. a closed single component mesh with no holes. Such mesh can be mapped onto a sphere, and it is often used as the simplest basic case on which algorithms are demonstrated.

A dynamic mesh is an ordered set of static meshes, where subsequent meshes represent the development of the mesh over time. In other words, dynamic mesh represents an animation of a 3D surface. In the most usual case, we have one mesh for each frame that is captured, and the time span between the meshes is 1/frame rate. However it is possible to also consider meshes where each frame carries explicit information about the time when it occurs, and where the inter-frame times are not equal.

One important assumption for this work is the one of constant connectivity of the input. We assume that the topology information does not change between subsequent frames. This implies that we have the explicit information about the correspondence of vertices, edges and triangles of subsequent frames. It also implies that the number of vertices and triangles does not change throughout the animation. We can also see the situation as a simple movement of vertex position over time, where each vertex moves between two subsequent frames from it's position in the first frame to it's position in the second frame.

Please note that we are only making this assumption about the input data, while the output data may be of varying connectivity. Also note that we assume that no other information about the viewer is given, i.e. we don't know anything about the relative position of the camera and the object.

There are several approaches to reduction of storage requirements. The most important aspect of reduction is the purpose of the final data. Approaches used for reduction of scientific data follow different criteria than algorithms targeted at human perception. In the first case, the single most important property of reduction algorithm is the reduction/error ratio, where the error is exactly determined by the nature of the data. On the other hand, when the target is a human observer, then the main quality of a reduction algorithm is the reduction/disturbance ratio. In this case, the algorithms may use some special properties of human perception to achieve better reduction rates with equal perceived distortion. In this work we will not be considering reduction for scientific purposes, our only criterion is the visual disturbance.

Our aim is to find and compare algorithms that take a dynamic mesh of constant connectivity as an input, and create a different dynamic mesh, which is visually equivalent to the original one, while its storage requirements are reduced.

The approaches to storage requirements reduction can be divided into two main categories, compressions and simplifications. Compressions are methods that encode the input data in a way that reduces storage requirements, while simplifications modify the input data in order to remove parts that are not necessary for the observer.

Compressions can be applied to geometry, topology or both, and we differ-

entiate between lossy and lossless compressions. Simplifications always involve alterations to both geometry and topology.

1.2 Organization

The rest of this thesis is divided into two main parts. In the first part, we describe existing approaches to the problem outlined above, and in the second we propose new methods for dynamic mesh compression, and to evaluation of the distortion caused by a compression scheme.

In the state of the art part we will first in chapter 2 give brief overview of techniques used for data compression in general. Then we will describe approaches used for compression (chapter 3), simplification (chapter 4) and comparison (chapter 5) of static and dynamic triangular meshes. We will show some drawbacks of the algorithms, some of which will be addressed in the second part.

The second part of this thesis covers our own advances in the field. First, in chapter 6 we will describe the datasets available to us for testing. Then in chapter 7 we will derive two new tools for measuring error, the 4D hausdorff distance and the STED measure. Subsequently, we will derive new compression techniques in chapter 8. We will give experimental results of our approaches in chapter 9, and finally draw conclusions and give ideas for future work in chapter 10.

Chapter 2

General compression techniques

In this short chapter, we will briefly sketch two techniques used generally in data compression - quantization and entropy coding.

2.1 Quantization

Quantization is a general technique used mainly for lossy compression algorithms. It is used to transform a dataset consisting of floating point values into a dataset consisting of integers, that are easy to encode. The idea is quite simple - the dataset is analyzed to find a minimum value and the span of values. Subsequently, each value is represented as a sum of the found minimum and a positive integer number of constant quanta. The quantum size is determined according to the span of the values, and the desired precision, which is usually given by the user in bits.

Formally, having a sequence $v_1 \dots v_n$ of floating point values, the quantizer finds the minimum, maximum and span:

$$m = min(v_1 \dots v_n) \tag{2.1}$$

$$M = max(v_1 \dots v_n) \tag{2.2}$$

$$s = M - m \tag{2.3}$$

and determines the quantization quantum C_q as

$$C_q = \frac{s}{2^Q} \tag{2.4}$$

where Q is a user specified constant which determines the accuracy of the process. Subsequently each value is expressed as

$$v_i = m + kC_q + e \tag{2.5}$$

where k is the integer number of quanta. The residual value e is neglected and the value k is subsequently used as a representation of the original value.

In some applications the span of values is not known, or is irrelevant, and in such cases a different value is used when computing the quantization quantum. Typically, when vertex positions (or vertex position differences) of a mesh are encoded, the length of the body diagonal of the mesh bounding box is used in the denominator of the fraction in expression 2.4.

In the case of dynamic meshes, there are more options, such as largest body diagonal over all the frames, body diagonal of the bounding box of all frames, however we will be simply using the body diagonal of the first frame. This decision is justified by the fact that the final size of the quantum is user controlled anyway, and therefore it is not necessary to use any sophisticated algorithm to influence it.

2.2 Entropy coding

Entropy coding is a general technique used for context free compression of sequences of integer values. It is based on the entropy theorem, which determines a bound on the compression of such sequences[25]. According to the theorem, a sequence $S = v_1 \dots v_n$ of length n cannot be expressed in less than $ent(S).n$ bits, where $ent(S)$ stands for entropy of the sequence. The entropy can be generally expressed as:

$$ent(S) = -\sum_{x \in S} p(x) log_2(x) \tag{2.6}$$

where x are the symbols that appear in the sequence and $p(x)$ is the probability of x appearing in the sequence. When we don't know the probabilities of symbols from the given source, we can estimate them by their relative occurrences.

Entropy coding attempts to reach this limit. The ultimate single code method (i.e. method that assigns a unique binary codeword to each symbol) is the Huffman coding[28], which assigns short codewords to symbols with high probability. The method has problems with sequences of very low entropy (typically under one bit), because the shortest codeword has the length of one bit.

There are also other methods, such as arithmetic coding[45], that can achieve compression rates closer to the theoretical bound. However, such methods are much more complex and still no ultimate method has been found.

In our experiments, we have used Huffman coding as an example, and we will be giving the value of entropy to give an idea about how the data can be possibly further compressed.

Chapter 3

Dynamic mesh compression

Compression in our context means a way of encoding the input dynamic mesh, which does not change the topology of the mesh. There are two main approaches to the problem, each targeted on different redundancy present in the data. In the first case, algorithms try to exploit redundancy in the geometry information. The other approach is to exploit the redundancy in the connectivity encoding.

For the case of geometry compression, algorithms processing dynamic meshes were already proposed. The usual scheme is to encode the first frame as full information, and then to encode only the differences in the positions of the vertices in the next frame. The algorithm usually contains a predictor that estimates the position of the vertex from the positions of the already encoded/decoded vertices, and then encodes the difference between the real position of the vertex and the predicted one. Sophisticated prediction techniques were used, some of which will be mentioned in the following text.

For the case of connectivity compression, only the static case has been investigated so far. This however does not cause any problem, as our task only involves one constant connectivity. The algorithms proposed in the literature usually involve some sophisticated way of encoding a progress of some open border, which traverses through the mesh. The key is usually in encoding the most common cases with the shortest possible bit sequence. The proposed algorithms work with both triangle and tetrahedral meshes, and we will later show how static tetrahedral mesh can be used to represent a dynamic mesh.

There are also hybrid compression techniques, which combine topology and geometry compression into one algorithm. Such techniques can be elegantly extended to cover the case of dynamic meshes.

3.1 Geometry compression of dynamic meshes

3.1.1 Predictor approaches

There are several predictors used for dynamic mesh compression. The main quality of a predictor is its accuracy, because the entropy coding block that follows the estimator works best when the average residual vector is as short as possible.

Some of the predictors take as input only the vertices from the frame that is currently being encoded, which may lead to need of specific ordering of the vertices, denoted as connectivity based encoding, which means that a vertex can be encoded only when at least two of it's neighbors are already encoded. Fortunately, such ordering comes naturally with some of the connectivity encoding schemes that will be described in more detail later. Such predictors only use spatial coherence of the input data.

On the other hand, there are some very simple predictors that only use the vertex positions from previous frame, thus exploiting only temporal coherence of the input. However, the most sophisticated prediction schemes exploit both temporal and spatial coherence of the input data.

3.1.2 Notation

We denote the most frequent values as follows:

$p(v, f)$ - position of a vertex v in time frame f

$pred(v, f)$ - prediction of the position of the vertex v in time frame f

V - total number of vertices in one mesh

T - total number of triangles in one mesh

F - total number of frames

3.1.3 Pure temporal predictors

The simplest way to predict a position of a vertex is to set the prediction into the vertex position from the previous frame:

$$pred_{stat}(v, f) = p(v, f - 1) \tag{3.1}$$

We can also approximate the vertex motion by linear motion (constant velocity), and set the prediction as follows:

$$pred_{lin}(v, f) = p(v, f - 1) + (p(v, f - 1) - p(v, f - 2)) \tag{3.2}$$

Constant acceleration predictor is then a simple extension of the constant velocity predictor, only this time requiring three previous time frames and using quadratic extrapolation

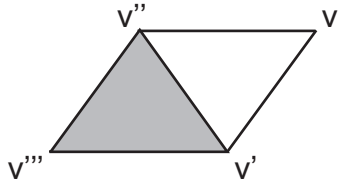

Figure 3.1: Parallelogram predictor

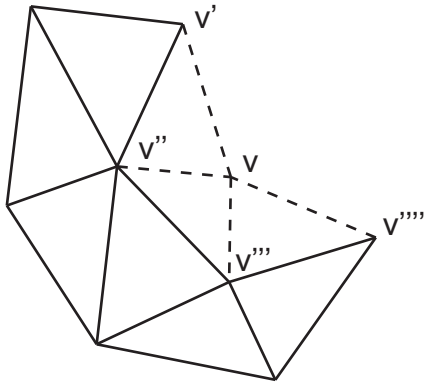

Figure 3.2: Averaging predictor

3.1.4 Pure spatial predictors

Parallelogram predictor ([61], [29]) has been used for compression of static meshes. It assumes that a triangle has been already encoded/decoded, and that the vertex that is currently being encoded is a neighbor of the encoded triangle, sharing one of the edges. The predictor creates a parallelogram from the known triangle by flipping it over the shared edge, and it sets the predicted position to the new vertex of the parallelogram.

$$pred_{paral}(v, f) = p(v', f) + p(v'', f) - p(v''', f) \tag{3.3}$$

Where v', v'' and v''' form the known triangle, and v' and v'' form the shared edge. A slightly different technique is used by the averaging predictor, which uses positions of all vertices encoded and adjacent to the currently encoded vertex. All such positions are averaged in order to obtain the prediction:

$$pred_{avg}(v, f) = [p(v', f) + p(v'', f) + ... + p(v^n, f)]/n \tag{3.4}$$

Where v', v'' ... v^n stand for adjacent vertex positions, n being their number.

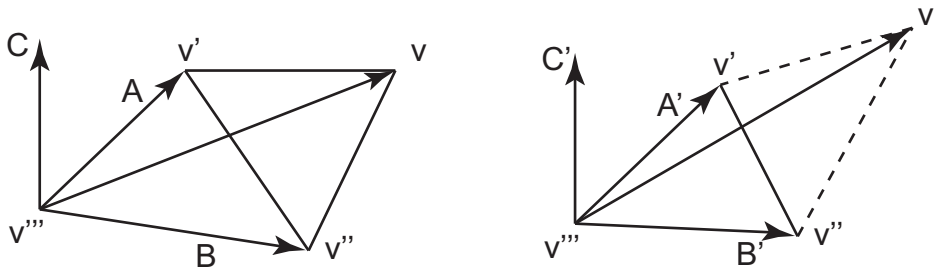

Figure 3.3: Replica predictor

3.1.5 Space-time predictors

Yang et al. [31] have proposed a time-space predictor based on the averaging spatial-only predictor, denoted as motion vector averaging predictor. The idea is that a vertex is expected to move in a way similar to its neighbors. This assumption leads to following predictor formula:

$$pred_{mvavg}(v, f) = pred_{avg}(v, f) - pred_{avg}(v, f-1) + p(v, f-1) \tag{3.5}$$

In [29] Ibarria and Rossignac have proposed two space-time predictors, the ELP (Extended Lorenzo Predictor) and the Replica predictor as a part of their Dynapack algorithm.

The ELP predictor is a perfect predictor for meshes that undergo a translational only movement, i.e. for such meshes it estimates the new position of each vertex exactly. The formulation of the predictor can be rewritten in a way similar to the motion vector averaging predictor:

$$pred_{ELP(v,f)} = pred_{paral}(v, f) - pred_{paral}(v, f-1) + p(v, f-1) \tag{3.6}$$

All the predictors presented so far are denoted as linear predictors, i.e. their formulae can be viewed as linear combinations of neighboring vertices (time or space) with some weights of unit sum. The following predictors are non-linear, i.e. they cannot be expressed as a weighted sum.

First of the non-linear predictors is the Ibarria's Replica predictor. It is a perfect predictor for rigid moving objects, i.e. objects that undergo some combination of translation and rotation. Moreover, the predictor also predicts exact positions of vertices for the case of uniform scaling.

The predictor again needs an adjacent triangle to be already encoded/decoded. The idea is to express the position of the vertex from the previous frame as a linear combination of two of the edge directions of the neighboring triangle, and the orthogonal direction. The combination coefficients are then used in the current frame to predict the position of the vertex. The relation can be written as:

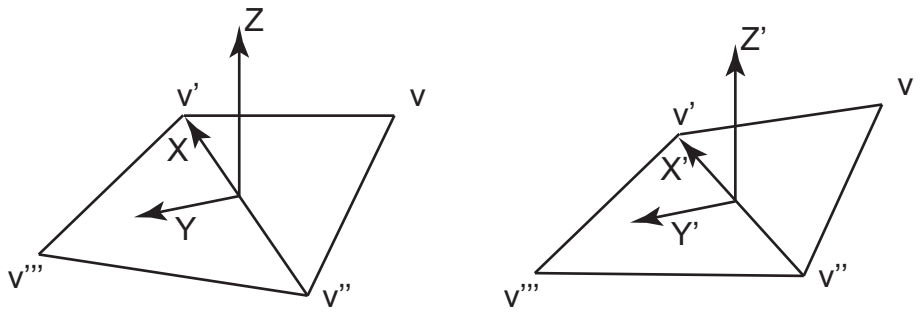

Figure 3.4: Angle preserving predictor

$$A = p(v', f-1) - p(v'', f-1) \tag{3.7}$$
$$B = p(v'', f-1) - p(v''', f-1) \tag{3.8}$$
$$C = \frac{A \times B}{\sqrt{|A \times B|^2}} \tag{3.9}$$
$$p(v, f-1) - p(v''', f-1) = aA + bB + cC \tag{3.10}$$
$$A' = p(v', f) - p(v'', f) \tag{3.11}$$
$$B' = p(v'', f) - p(v''', f) \tag{3.12}$$
$$C' = \frac{A' \times B'}{\sqrt{|A' \times B'|^2}} \tag{3.13}$$
$$pred_{Replica}(v, f) = p(v''', f) + aA' + bB' + cC' \tag{3.14}$$

Note that the normalization used in the computation of C and C' ensures that the predictor works perfectly for uniform scaling. A similar approach is used in another non-linear predictor proposed by Stefanoski and Ostermann [58]. It is a predictor that perfectly preserves the angle between the current and the new triangle. The main difference is in the coordinate system used to express the encoded vertex. The Replica predictor has used two edges of the adjacent triangle and an orthogonal direction, Stefanoski's angle preserving predictor uses the shared edge direction X, a direction Y orthogonal to X that is lying in the plane of the adjacent triangle, and a direction Z which is orthogonal to X and Y. The predictor algorithm is then very similar to the Replica predictor:

$$p(v, f-1) - p(v''', f-1) = xX + yY + zZ \tag{3.15}$$
$$pred_{angle}(v, f) = p(v''', f) + xX' + yY' + zZ' \tag{3.16}$$

3.1.6 Muller octree approach

A slightly modified prediction based algorithm for dynamic mesh compression is presented in the works of Muller and his colleagues [47]. They use the standard

procedure, where the first frame is encoded completely, and for the subsequent frames only the difference from the last frame is being encoded.

The position differences, denoted as "difference vectors" are then clustered using an octree algorithm [75]. The depth of the octree is controlled by the variance of the vectors contained in the cells. Areas of homogeneous motion are contained within large cells, while areas where the difference vectors vary more are further subdivided.

For each cell of the octree is finally selected a substitute vector, which is encoded into the output stream using standard arithmetic coding algorithm CABAC [39]. The final residual vectors are neglected.

The method is inevitably lossy, information is lost when the whole cells of an octree are represented by a single substitute vector, which is moreover quantized. The amount of data loss can be steered by the depth of the constructed octree.

In their latest works [48] the authors have enhanced the algorithm by introducing rate/distortion optimization. For each cell one of following predictors is chosen:

- direct coding - each vector is fully encoded
- trilinear interpolation - eight corner vectors are encoded, the rest is interpolated
- mean replacement - uses the same idea of the original algorithm, i.e. replaces the whole cell content by one substitute vector.

The decision about the predictor is made based upon the rate-distortion ratio for each cell.

A very similar approach, also using octree subdivision, has been proposed by Zhang and Owen[74]. They also use encoding of motion vectors, and utilize trilinear interpolation to predict the actual positions of vertices. The algorithm does not encode the residuals at all, it only transmits the refined octree, and uses reinitialization by inserting a fully encoded an I-frame after several predicted frames.

3.1.7 Stefanoski scalable linear predictive coding

In 2006, Stefanoski et al. [57] have proposed one of the first schemes which combine compression with simplification. Their scheme decomposes the vertices into disjunct sets (layers) by progressive decimation of the original connectivity. The manner of the simplification ensures that during subsequent traversal of the vertices a complete spatial neighborhood of each vertex is available at the decoder. This allows using a spatio-temporal predictor which exploits the whole neighborhood and the previous frame, thus achieving better accuracy of the prediction.

The algorithm starts with the full resolution topology, which is transmitted to the decoder. In the next step, the topology is decimated at both encoder and decoder, yielding a shared hierarchy of levels. The mesh is first divided into disjunct patches (patch = topological neighborhood of a single vertex), using a deterministic, yet not geometry driven (the decoder has no information about the geometry yet), greedy algorithm.

Subsequently, each patch center w_k is removed from the topology and the resulting hole is retriangulated. The retriangulation process is again topology driven, and aims to preserve vertex degree of 6, which is known to be optimal for triangular meshes. The algorithm simply tries all the possible retriangulations of the patches, and chooses the one which minimizes the following expression:

$$Dev(w_k, t) = \frac{1}{|N(w_k)|} \sum_{v \in N(w_k)} |\delta(v, t) - 6| \tag{3.17}$$

where $N(v)$ denotes the topological neighborhood of a vertex v and $\delta(v, t)$ denotes the degree of the vertex v in a candidate triangulation t. The process of patching and decimating is repeated several times, and for the coarsest level the geometry data is transmitted to the decoder using some standard spatio-temporal predictor.

In the next refinement step, a new predictor is used, which employs the linear temporal prediction along with neighborhood average prediction of the motion vector at the given position. The predictor takes the following form:

$$pred_{scalable}(v, f) = p(v, f-1) + \frac{1}{|N(v)|} \sum_{u \in N(v)} \big(p\,(u, f) - p\,(u, f-1)\big) \tag{3.18}$$

Following steps are similar to other predictor based schemes - the residuals are quantized and entropy coded. The scheme provides results which outperform all the previously proposed schemes, usually by 10-20%.

Generally, all the predictor based compression schemes provide an easy to implement and very fast method for dynamic mesh compression. However, the spatio-temporal coherence that is present in the data is only exploited locally, by the predictor inputs. Generally, the fact that even distant parts of the object can behave in a similar way is not exploited by these algorithms. This issue has been addressed by the PCA based compression schemes, which will be described later in this chapter.

3.1.8 Wavelet based compression

A wavelet based approach to dynamic mesh compression has been proposed by Payan in [49]. The method exploits temporal coherence of the data by iteratively

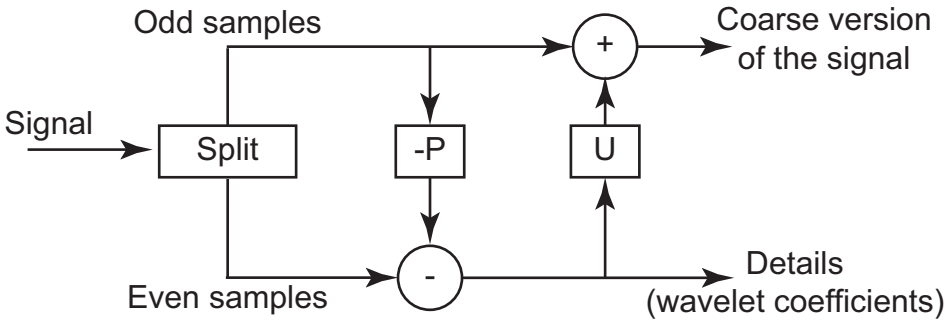

Figure 3.5: Principle of wavelet decomposition

dividing the temporal sequence of vertex positions into high frequency and low frequency parts. The key observation is that for each frequency level a different quantizer can be used, i.e. each frequency can be encoded with different number of bits per sample. The authors propose to search for an optimal set of quantizers that produces the best rate/distortion ratio.

Because the method is aimed to exploit the temporal coherence of the input mesh, the wavelet encoding is applied on the trajectory of each vertex. The trajectory is represented by three sequences of values, each representing the evolution of one component of the position of the vertex in time. Each such sequence is encoded separately.

The wavelet encoding is generally based on two operators: the Prediction operator P, and the Update operator U. The input sequence is first split into two groups of samples: even ones, and odd ones. The Prediction operator is applied to obtain a low frequency subband, and the Update operator is applied to obtain the high frequency subband. The whole process can be subsequently iteratively applied on the low-frequency subband in order to obtain a sequence of various frequency data.

There are many P and U operators proposed in the literature, the authors state that they have obtained best results using the [4,2] operator proposed in [12].

The final step of the algorithm, which actually represents the compression of the data, is the quantization. Each subband is compressed using different bitlength, and the optimal set of bitlengths is found using an iterative process with the constraint that the selected bitrate is matched.

3.1.9 PCA based compression

A very interesting new approach to dynamic mesh compression has been introduced by Alexa and Muller[2] in 2000. Their approach is completely different from the prediction based schemes presented above, as it uses principal component analysis (PCA) to determine a new basis for the animation. The compression is based on reducing the basis size by omitting the basis elements of low

importance.

The first step of the algorithm is to extract rigid motion from the animation, because rigid motion causes difficulties to the following PCA encoder. Each frame is moved so that its center of mass lies at the origin, and an affine transformation of positive determinant is found which minimizes the squared distance from each vertex position to the corresponding vertex position in the first frame, thus removing any rotation and scaling motion that may be present in the animation. The coefficients of this transformation are then sent with each frame of the animation.

The process then continues with the PCA itself. Each frame of the animation is reordered to form a single column vector of length $3F$ (F being the number of vertices in each frame), i.e. all the X coordinates are stored first, followed by all the Y coordinates and all the Z coordinates. All the column vectors are ordered into a matrix B, which fully describes the animation. This matrix is now viewed as a set of samples in $3F$-dimensional space. The task is to find such orthonormal basis of this $3F$-dimensional space, which is completely uncorrelated, i.e. where information about one component does not provide any information about any other component. The tool for finding such basis is PCA, i.e. finding eigenvectors of the covariance matrix.

The authors propose to use the Singular Value Decomposition (SVD) to find the new basis for the space of frames. The SVD decomposes the matrix to following components:

$$B = \hat{B} \cdot S \cdot V^T \tag{3.19}$$

Where B is the matrix of orthonormal principal component basis vectors, S is a diagonal matrix of importance factors (in fact eigenvalues of $B \cdot B^T$) and V is a matrix of animation representation transforms.

The rest of compression scheme is now straightforward. From the new basis we choose given number of vectors with highest importance factors and compute the representation E_f of each frame in this new reduced basis using the inner product:

$$E_f = B_f^T \cdot (\hat{B}_0, \hat{B}_1, \ldots, \hat{B}_b) \tag{3.20}$$

where b is the selected number of basis vectors. The resulting vector of b components (feature vector) is then sent with each frame. The last part of the encoded representation are the basis vectors themselves.

The decompression is then straightforward too. The feature vectors E_f are multiplied by the basis vectors, and the result is transformed by the affine transform to get the original mesh. If all the basis vectors are used, then the compression scheme is lossless, the reduction can be steered by the number of selected basis vectors b.

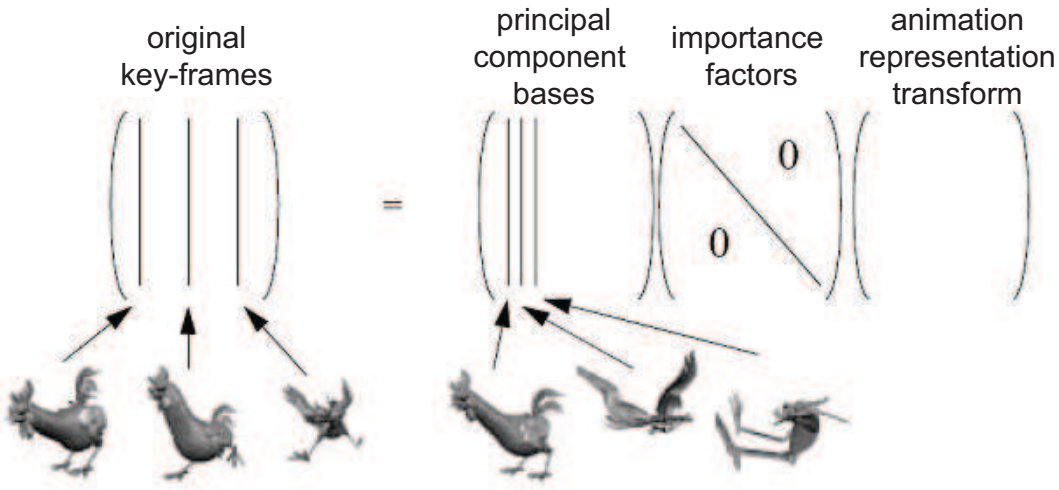

Figure 3.6: The Singular Value Decomposition of a sequence of meshes, taken from [Ale00], modified.

The method provides good results and can be combined with other compression techniques for quantization of the basis vectors and feature vectors, but it is one of the most computationally expensive algorithms, which is caused by the SVD step. However, this step is only performed by the sender, while the receiver only performs simple matrix multiplications, which can be done on desktop machines in real time.

3.1.10 PCA LPC coding

A slight modification of the PCA method has been presented by Karni and Gotsman in [32]. Their method is based on the PCA augmented by Linear Prediction Coding (LPC) of the feature vector components. The LPC method is generally used to encode a sequence of T values by predicting the value as a linear combination of its m predecessors, and then encoding the residuals using arithmetic coding. A single set of weights for the linear combination is used for the whole sequence. The weights are computed using the least squares method from the whole original sequence so that the average squared residual is minimized, i.e. by solving the following overdetermined set of equations:

$$\begin{pmatrix} 1 & x_m & \dots & x_1 \\ 1 & x_{m+1} & \dots & x_2 \\ \vdots & \vdots & \ddots & \vdots \\ 1 & x_{T-1} & \dots & x_{T-m} \end{pmatrix} \begin{pmatrix} a_0 \\ a_1 \\ \vdots \\ a_m \end{pmatrix} = \begin{pmatrix} x_{m+1} \\ x_{m+2} \\ \vdots \\ x_T \end{pmatrix}$$

The sequence of weights $a_0 \dots a_m$ is sent with the sequence of the residual values that are computed as follows:

$$r_i = x_i - (a_0 + \sum_{j=1}^{m} x_{i-j} a_j) \tag{3.21}$$

The LPC coding can be used directly on dynamic meshes when each vertex trajectory is treated as a sequence of values, yielding sort of optimized temporal-only predictor encoding. However, such method would completely omit the strong spatial coherence that is usually present in dynamic meshes. The idea is to apply LPC to the animation at a point when the separate components of the representation are completely uncorrelated, i.e. after PCA.

The algorithm starts with finding the affine transform performed in the same manner as in [2]. Subsequently, PCA is performed on the remaining non-rigid animation, a subset of the basis is selected and the feature vectors are computed. However, the feature vector components are not encoded directly, but using the LPC, each component of the vectors treated as a sequence of length equal to the length of the animation. The decompression is then also performed in a way similar to [2], only the first step being the LPC decoding of the feature vectors.

The authors report reduction of the rate distortion ratio when compared to the PCA-only method and when compared to the DynaPack algorithm for animations of soft body movement. LPC of second or third order has been used.

3.1.11 Clustering based PCA encoding

A further improvement of the PCA based encoding has been proposed in 2005 by Sattler et al. [54]. Their approach is based on reorganizing the data into a set of vertex paths instead of frames. PCA is subsequently applied on these paths, and the results of PCA are used to find a set of clusters of similar temporal behavior. These clusters are then encoded separately using the standard PCA method.

The data reorganization step is generally only a transposition of the B matrix from the original PCA-based encoding scheme. Subsequently, the clusters of locally linear behavior are found using following iterative process:

1. initialize k cluster centers randomly from the data set
2. associate each point to the closest cluster center according to a distance which is evaluated as reconstruction error using center's c most important eigenpaths
3. compute the new centers as mean of the data in each cluster
4. perform PCA again in each cluster

5. iterate the steps 2-4 until the average change in reconstruction error falls bellow some given threshold.

This process creates clusters of vertices, which move almost independently. These clusters are subsequently encoded using the standard PCA-based encoder, i.e. by finding eigenshapes of each cluster and selecting a limited number of these which can be combined to approximate any shape from the input data.

The main drawback we have identified with this method is the iterative refinement initialization and the convergence speed. The random initialization causes the iterative process to converge to radically different solutions each time, and the convergence can take quite long time for moderately complex meshes.

3.1.12 PCA + local coordinate frame approach

An approach which stands between the original eigenshape-based algorithm by Alexa [2] and the CPCA algorithm has been proposed in 2007 by Amjoun and Straßer[5]. The algorithm first performs clustering, and assigns a local coordinate frame (LCF) to each cluster. Subsequently, each vertex is assigned to such cluster centre, where the vertex movement expressed in the LCF is smallest. Finally, within each cluster an eigenshape basis PCA is performed, reducing the dimensionality of the data. Also a bit-allocation process is proposed, which allows assigning different numbers of basis vectors to different clusters.

The algorithm starts with a clustering step. In contrast to [54] a different clustering is used. The k-center approach, proposed in [71], is used to find a given number of cluster centers. A triangle incident with each center is selected, and a local coordinate frame is created at each center in each frame by assigning the first axis to one of the edges of the triangle, second axis lying in the plane of the triangle, orthogonal to the first one, and the last axis orthogonal to the first two computed as a cross product.

In the next step, each vertex v is assigned to one of the centers. Each center c_c is considered as a candidate center, and the vertex position is expressed in the local coordinate frame of the candidate center. The length of the trajectory in the LCF is computed, and the vertex is assigned to the center where the length is shortest.

$$T(v, c_c) = \sum_{i=1}^{f} \|p(v,i)_{LCF(c_c)} - p(v,i-1)_{LCF(c_c)}\|^2 \tag{3.22}$$

For each cluster is then selected an appropriate number of basis components, according to a bit-allocation process, which analyzes the gain of adding a basis vector for each cluster. For details see the original paper.

Finally, each cluster is treated as a sequence of geometries and PCA is applied in the space of shapes. Each frame is expressed in the reduced basis, the coefficients are quantized and encoded using an arithmetic coder.

An interesting feature of this work is that it is to our knowledge the only algorithm which deals also with the encoding of PCA basis. however the solution proposed is trivial. The authors suggest forming the basis into a matrix, which is subsequently uniformly quantized and entropy coded.

3.1.13 Skinning approach to compression

Another clustering based approach has been proposed by Mamou et al.[38] in 2006, however their approach utilizes skinning-like steps instead of PCA. The algorithm segments the mesh according to the properties of motion of the vertices, associates a series of transform matrices to each segment, and finally assigns weights to each vertex, which determine how to blend the transforms to get an accurate prediction of the vertex position.

The algorithm involves a quite large number of least-square optimization steps. First, for each vertex v in each frame f a matrix $M_{v,f}$ is found, which describes the motion of the vertex neighborhood between the first frame and the frame f. The matrix is computed as a least squares-fashion optimization expressed as:

$$M_{v,f} = argmin\Big(\sum_{v_n \in v^*} \|Ap(v_n, 0) - p(v_n, f)\|^2\Big) \tag{3.23}$$

where v^* denotes the third order geometric neighborhood of the vertex v. This process yields a series of $F-1$ matrices for each vertex. The series is reformed into a single column vector, and upon the set of such vectors the k-means clustering is performed, producing a set of clusters.

In the following step, a single series of transformation matrices is found for each cluster C in a similar manner:

$$M_{C,f} = argmin\Big(\sum_{v_n \in C} \|Ap(v_n, 0) - p(v_n, f)\|^2\Big) \tag{3.24}$$

The position of each vertex in the first frame, transformed by the cluster matrix is however still a too coarse estimation of the actual position of the vertex. Therefore a vector of weights is assigned to each vertex, specifying a blend of cluster matrices needed to achieve better precision prediction. The vector of weights is again searched for in a least-squares fashion, and only the cluster into which the vertex belongs and the directly neighboring clusters are considered, thus keeping the length of the vector short.

The vector of weights w is found by solving the following optimization problem:

$$w = argmin_{b \in R^k} \sum_{f=0}^{F} \| \sum_{k=1}^{K} w_k M_{k,f} p(v,0) - p(v,f) \|^2 \tag{3.25}$$

At this point, the encoder sends the clustering information and the series of matrices assigned with each cluster, along with the vertex weights, allowing the decoder to predict the positions as

$$pred_{skinning}(v,f) = \sum_{k=1}^{K} w_k M_k p(v,0) \tag{3.26}$$

The prediction residuals are finally encoded using a technique called temporal DCT, which is basically DCT applied separately on temporal series of the x, y, and z coordinate residuals. The quantized DCT coefficients are finally encoded using arithmetic coding.

3.1.14 FAMC encoder

Frame-based Animated Mesh Compression (FAMC) is currently the most sophisticated algorithm for dynamic mesh compression. It has been adopted by the MPEG consortium as a standard algorithm for compression of animated meshes in a form of the second amendment of the part 16 of the MPEG-4 standard[1]. It combines some of the previously described approaches, it provides useful features and it is to date the most efficient algorithm from the rate-distortion point of view.

The algorithm consists of following steps:

1. skinning-based motion compensation, performed in a way similar to Skinning based compression described in section 3.1.13,
2. transformation of residuals, performed in order to exploit temporal coherence,
3. layered prediction (similar to the scalable approach described in section 3.1.7), which involves simplification, which is used to provide spatial scalability and to remove spatial coherence,
4. entropy coding of the residuals.

We will now describe each step in more detail.

In the first step the animation is processed as described in section 3.1.13. Each vertex is assigned a set of matrices which describe the movement of its neighborhood, and the vectors of matrices are clustered. Finally, each vertex has assigned a vector of weights, which tells how to combine the transforms of the neighboring clusters in order to well predict the vertex position in each frame.

The algorithm at this point encodes the matrix sequences associated with each cluster, and the cluster-assignment weights for each vertex.

The transformation step is new in FAMC, and it is applied on the one-dimensional sequences of prediction residuals assigned with each vertex. The algorithm allows choosing between discrete cosine transform and wavelet transform, and also allows skipping the transform completely. The purpose of the transform step is to exploit the temporal coherence which is present in the residuals.

The layered decomposition is performed by a sequence of edge-collapse operations (see section 4.1.4 for more details), yielding a sequence of simplified versions of the connectivity. Note that the simplification is only driven by topological criteria, because it has to be performed by both encoder and decoder, and the decoder has no information about the geometry at this stage of the algorithm. The residuals are then encoded in a reverse order, which allows using the neighborhood average predictor in order to further exploit spatial coherence of the data.

The layered decomposition allows prediction using spatial neighbors, which works in three modes, I, P and B, which resemble the prediction modes for video encoding: for I frames no neighboring frame is used, while for P frames one and for B frames two neighboring (and already decoded) frames are used to predict the skinning residuals.

Finally, the layered prediction residuals are quantized and encoded in a lossless fashion using the CABAC entropy coder[39].

The FAMC encoder has been shown to outperform all the other compression algorithms in the offered rate/distortion ratio. It provides the possibility to decode only part of the stream, producing a complete (although distorted) version of the geometry (spatial scalability). It also allows decoding of separate frames without the need to decode the whole data stream.

Generally, all the current geometry compression schemes share one serious omission. They all focus on minimizing the difference between the geometrical positions of vertices, while they don't take the topology of the mesh into account. Currently there is no method which would differentiate between the orthogonal and tangential vertex position error.

3.2 Triangular connectivity compression

The usual way to describe connectivity of a triangular mesh is to store a table of index triplets, where each triplet represents one triangle. Although intuitive, this approach has many drawbacks. On one hand, it does not explicitly store adjacency information, i.e. whenever there's a need to obtain neighbors of given

vertex or triangle, it is necessary to search the whole table. Moreover, such representation is very memory expensive, surprisingly even more expensive than the stored geometry information. For simple mesh a so-called Euler equation holds:

$$f + V = E + 2 \tag{3.27}$$

where f represents the number of faces (triangles), V is the number of vertices, and E is the number of edges. From this equation follows that in a mesh there are about twice as many faces than vertices. We can express the space required to store geometry as

$$G = 32 * 3 * V = 96V[bits] \tag{3.28}$$

i.e. each vertex is represented by three 32 bit floating point numbers. The space required to store the connectivity can be expressed as

$$C = 3f\lceil log_2 V\rceil = 3v2\lceil log_2 V\rceil = 6V\lceil log_2 V\rceil[bits] \tag{3.29}$$

From the two equations one can derive that for a mesh of more than $2^{16} = 65536$ vertices the connectivity information requires more space than the geometry. The following algorithms represent the state of the art in the lossless connectivity compression.

In 1962 Tutte [62] has shown that there is a lower bound on the number of bits required to describe a planar triangular mesh. By enumerating all the possible triangulations with a given number of vertices, Tutte has concluded that at least $log_2(256/24) = 3.245$ bits are needed per vertex. Although some algorithms provide lower bitrates for special cases, it is only the last algorithm presented by Alliez and Desburn, which guarantees (under some assumptions) the upper bound to reach this constant.

3.2.1 Topological Surgery

Topological surgery is a static mesh connectivity compression algorithm proposed in 1998 by Taubin and Rossignac [60]. In order to encode a mesh topology, it first cuts the original topology by a vertex spanning tree, which yields a set of triangle stripes (the best way to imagine this procedure is to see it as "peeling" the stripes from the original surface).

The resulting triangle tree is also encoded, along with a "march sequence" which encodes the topology of each triangle strip. The decompression algorithm first recovers the border of the cut mesh from the encoded vertex spanning tree, then it recovers the triangle stripes from the triangle spanning tree and the march sequences, and finally it sews the cuts and thus fully recovers the original topology. We will now describe each of the steps in more detail.

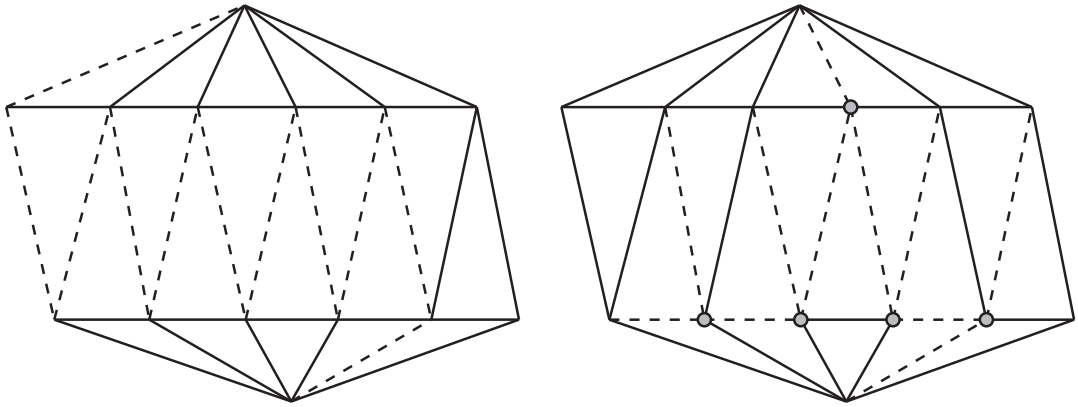

Figure 3.7: Two possible vertex spanning trees of an unwrapped icosahedron. The tree on the left consists of a single branch, while the tree on the right has five branching nodes (white). The cut edges are dashed, all other edges are the march edges.

A vertex spanning tree is a subset of edges of the original mesh, which forms an acyclic connected graph (a tree) which spans all the vertices of the original mesh. We can represent the vertex spanning tree by a tree data structure, i.e. a root node, with branches and leaf nodes.

For every mesh there are many vertex spanning trees, and it is crucial to choose a tree which will have long branches and few branching nodes, because such tree is easily encoded, and it also yields a triangle spanning tree with long branches. By picking a particular vertex spanning tree the edges of the mesh are divided into two groups: the cut edges (i.e. the edges of the vertex spanning tree) and the march edges (all the remaining edges).

The vertex spanning tree is encoded by traversing through it in width first order, where for each branch it is only necessary to encode its length, and two bits, which indicate whether the branch has any siblings and whether the branch ends in a leaf node or a branching node.

A triangle spanning tree is a connected acyclic graph where nodes represent triangles of the original mesh, and edges represent marching edges, which connect the triangles. The triangle spanning tree is fully described by the particular vertex spanning tree, and it can be represented by a binary tree data structure, because every branching node (triangle) can only split into two branches. Therefore, a triangle spanning tree can be encoded by traversing it in breadth first order and encoding each branch by its length and a single bit which tells whether the branch ends in a leaf node or a branching node.

The only remaining piece of information which needs to be encoded in order to fully describe the original connectivity is the internal topology of the triangle branches. The triangle spanning tree contains information about the length of

each branch, and we can imagine reconstructing it like moving an edge which connects the opposite borders of the triangle strip. The border can move either on the left side, or on the right side, and this information is encoded by one bit per triangle in the march table. The branches are encoded in this table in the same order as they appear in the triangle spanning tree.

The problem of choosing the optimal vertex spanning tree is considered to be NP complete. The original paper suggests two possible approaches. First, we can assign some cost to each edge, and construct the tree as a minimum spanning tree using some graph theory algorithm. Good results are obtained when the cost is for example the Euclidean distance from some root vertex. In such case the edges are added to the spanning tree in an order according to their distance, which leads to longer triangle runs.

Even better results are obtained with another sub optimal, but deterministic approach, called layered decomposition. This approach can be best imagined as literally peeling the triangles off the surface, starting at some given point. First, the triangles are divided into layers, according to their topological distance to some original triangle, i.e. all neighbors of the triangle are denoted as first layer, all neighbors of the first layer triangles are denoted as second layer etc.

Subsequently, the layers are converted to stripes, starting at the innermost. The stripes from each layer are constructed in a way that they ideally form a single long stripe. Using this technique, an example mesh of 5138 vertices mesh has been represented by 168 vertex runs.

Using the topological surgery approach, it is necessary to encode at least one bit per triangle for the march table, while the number of bits needed to encode the spanning trees vary according to how many branches are used. Using the layered decomposition the authors claim to achieve 2.16 bits per triangle for the sample mesh of 5138 vertices, while for very regular topology (for example obtained by some subdivision technique) the ratio can drop to as low as 1.3 bits per triangle.

3.2.2 Edgebreaker/cut-border machine

A very elegant and simple way to efficiently encode connectivity of a triangular mesh has been proposed in 1998 independently by Gumhold and Straßer ([27]) and Rossignac([51]). In this text we will describe the Rossignac's Edgebreaker algorithm, which provides some advantages over the slightly earlier Cut-border machine proposed by Gumhold and Straßer.

The basic idea of the algorithm is to encode a traversal through all the triangles of the mesh. For simplicity we describe only the basic version of the algorithm, which works for a simple mesh.

The first triangle is encoded in full, and its edges form the first progressing border. One of the edges of the initial triangle is selected as "open", and it will be used to grow the progressing border. The open edge connects the current

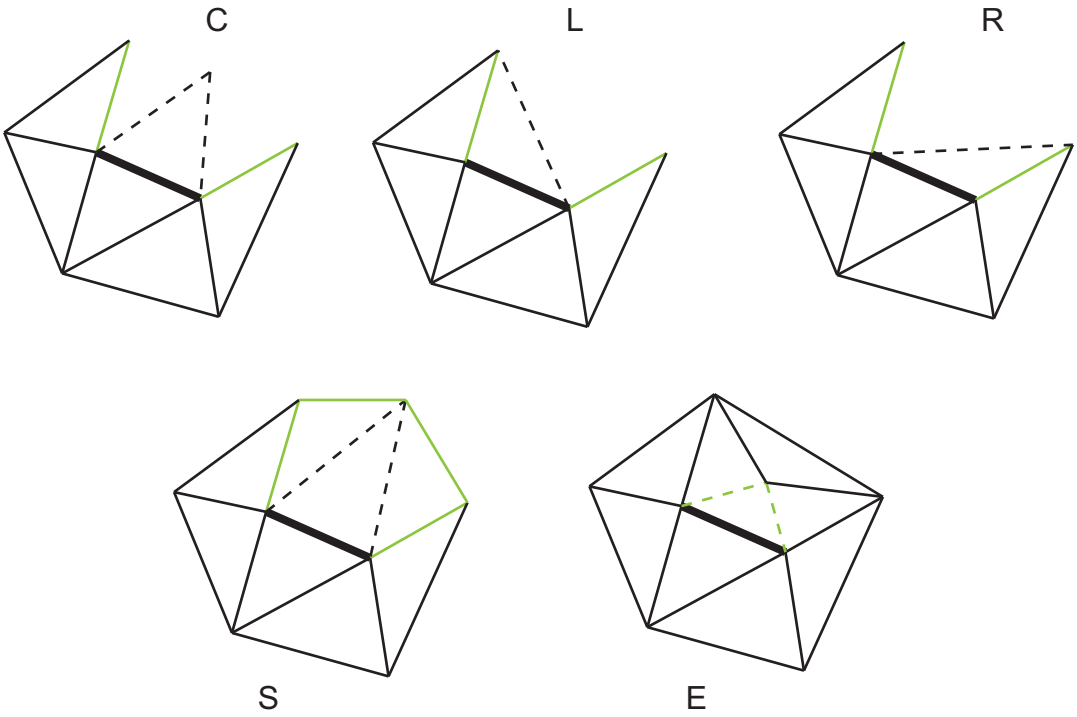

Figure 3.8: Edgebreaker cases. Thick edge represents the gate, green edges belong to the current border.

triangle with exactly one new triangle, which contains the open edge and one more vertex. There are five possible cases for the vertex:

1. it is a new one, which has not been touched by the progressing border yet (case C)
2. it is a vertex that lies on the progressing border immediately to the left of the open edge (case L)
3. it is a vertex that lies on the progressing border immediately to the right of the open edge (case R)
4. it is a vertex that lies on the progressing border both immediately to the left and to the right, i.e. it closes a single triangle hole in the mesh (case E)
5. it is a vertex which lies on the progressing border, but somewhere else than immediately to the left or right (case S)

Each of the cases is encoded by some bit sequence into a so called CLERS string. The new triangle added by the L and R cases has only one edge that lies on the progressing border, and it is selected as the new open edge, and the

algorithm continues. The case C yields two new edges, from which the right one is selected as open edge, and the left one will be later used by some other operation. The S case yields two border edges, and both are processed by a recursive call to Edgebreaker procedure. The E case terminates the Edgebreaker procedure and passes control to either higher recursive call, or terminates the whole algorithm when the whole mesh has been encoded.

When decompressing, the L and R cases do not require any further information, because the third index of the new triangle is the index of the vertex immediately preceding or following the open edge. Neither do the C and E cases require any information, because for the E case the index can be also derived from the progressing border, and the C case implies that the triangle contains new vertex with an unused index.

The main advantage of the Edgebreaker algorithm over the Cut-border machine is that not even the S case requires any identification of the opposite vertex. The idea is that each S case divides the unprocessed portion of the mesh into two parts, each of which will be later closed by an E case. Therefore, S and E cases form a sort of bracket structure, from which one can derive the index of the opposite vertex for each of the S cases.

As we can see, the way the Edgebreaker is processing the mesh is very useful for some of the geometry compression techniques presented earlier, as it encodes the vertices in such order that for each vertex being encoded there is a neighboring triangle available.

Edgebreaker encodes the C case with one bit op-code, while the remaining four cases are represented by a three bit code. For simple meshes there are about twice as many triangles than vertices. The length of the CLERS string is equal to the number of triangles of the mesh, and a new vertex is introduced by a C symbol. From this follows that one half of the symbols in the CLERS string are C symbols. The overall encoded length of the CLERS string can be expressed as follows:

$$CL = \frac{t}{2} bl_C + \frac{t}{2} bl_{LERS} = \frac{t}{2} + 3\frac{t}{2} = 2t \tag{3.30}$$

where CL represents the encoded length of connectivity, bl_C stands for bit length of C code and bl_{LERS} represents the bit length of codes L, E, R and S. In other words, it is guaranteed that for a simple mesh the connectivity is compressed to a maximum of 2 bits per triangle, i.e. 4 bits per vertex.

In practice using arithmetic coding on the CLERS sequence yields about 1.7 bits per triangle, or even less than that for extremely regular connectivity meshes. This property will be later exploited by the SwingWrapper remeshing algorithm described in section 4.1.2

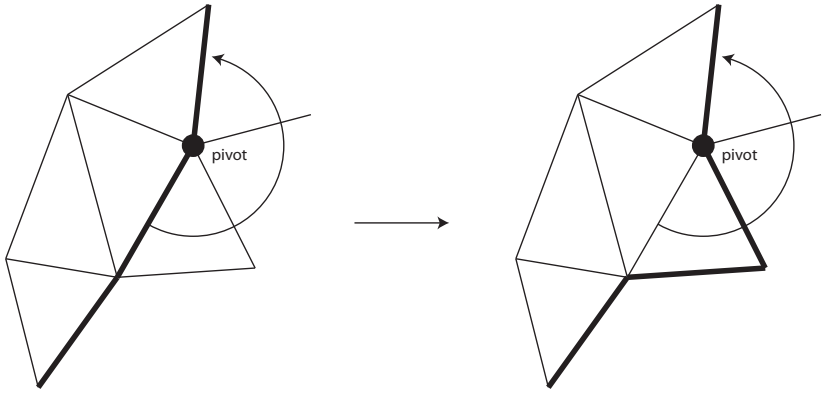

Figure 3.9: The most probable case in valence driven encoding. The pivot vertex is processing its free edges, encountering a free vertex, and adding it to the active edge list. Edges of the active edge list are shown bold.

3.2.3 Delphi geometry based connectivity encoding

A very elegant improvement of the Edgebreaker algorithm has been recently proposed by Coors and Rossignac [16]. The idea is to combine some of the geometry estimators and the Edgebreaker algorithm. The geometry predictor is used to predict the position of the next vertex, and a threshold algorithm is used to deduce a guess about which of the CLERS cases occurs. As both the encoder and decoder use the same estimator, it suffices to send a single confirmation bit instead of one of the CLERS op-codes.

However, the performance of the algorithm depends on the accuracy of the CLERS predictor, which depends on the used geometry prediction. The authors of Delphi have used the parallelogram predictor, and they report that for usual meshes they have achieved guess accuracy over 80%, which lead to compression to about 1.3-1.5 bits per triangle.

3.2.4 Valence-drive connectivity encoding

A slightly different approach to connectivity encoding has been proposed in 1998 by Touma and Gotsman[61]. The key idea behind the algorithm is that vertex valences are the only information needed to traverse through the mesh, and moreover for most meshes the vertex valence shows only low variance around the average of 6, thus have a low entropy.

The algorithm passes through the connectivity by "conquering" vertices, edges and triangles, and outputting the encountered vertex valences into the encoded data stream. It works in following steps:

1. Select a triangle, mark it conquered. Also mark conquered all of its edges and vertices. Output the valences of the vertices.

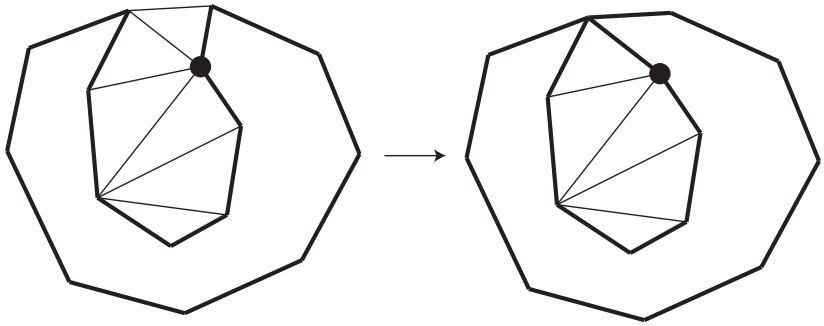

Figure 3.10: The *split* accident: The processing of a pivot vertex has encountered a vertex which is part of the active edge list. The list is split into two (inner and outer), one of which will be processed later.

2. Add the three initial vertices into active edge list, AEL. Select one of the vertices to be the active "pivot".

3. Process all the free (not conquered) edges of the pivot in counterclockwise order. The most probable case is that the other vertex incident with the edge is not in the AEL (see figure 3.9); in such case output it's valence to the target data stream and insert it into AEL before the pivot. Other cases which can occur will be discussed later.

4. When all the free edges are processed remove pivot from AEL and mark the following vertex in AEL as pivot. Go to step 3, unless the whole mesh has been processed.

The decoder simply replicates the actions of the encoder and reads the valences of the vertices from the stream, which allows it to determine the number of free edges incident with each vertex.

There are two "accidents" which may happen during the traversal. First, in step 3 a vertex belonging to the AEL may be encountered, separating the closed edge list into two parts (see figure 3.10). Such case is treaded by a special symbol *split*, which tells the decoder to split the AEL into the two parts, followed by an index determining the position of the split. One part of the AEL is then processed while the other part of the AEL is pushed into a stack of lists, from which it will be later popped and processed.

The other accident also appears in step 3, when a vertex belonging to one of the lists currently in the stack is encountered. The case is identified by a *merge* symbol, which tells the decoder to merge the current AEL with a list in the stack. The *merge* symbol is followed by two indices, first identifying which of the lists in the stack should be used for merge, and the second identifying the index of the encountered vertex, thus specifying the merge operation.

The algorithm also treats meshes with boundaries by inserting a *dummy* vertex for each boundary loop. This vertex is connected to every vertex of its boundary loop, thus topologically filling it. Such vertex (usually of high degree) is identified by a symbol *dummy* in the output string, and the decoder removes it after the whole connectivity has been decoded.

Since the most frequent symbol in the output string is the vertex valence, which is expected to be 6, it is possible to efficiently apply entropy coding, reaching to bitrates of 0.2 bpv for very regular meshes, and 2-3.5 bpv for irregular ones. The main drawback of this method, which has been later addressed, is the lack of worst case limit on the bitrate. The reason is that the algorithm transmits indices for the accident operations, and it cannot be predicted how many bits will be needed for encoding them.

The algorithm has been subsequently improved by Alliez and Desburn[3]. They suggest reducing the number of the *split* accidents by replacing the deterministic processing of vertices by an adaptive one. They suggest selecting pivots with low number of free edges, and when such decision is ambiguous, then the neighborhood of the candidate pivot is also taken into account. A weighted average free edge count is computed, giving lower weight to distant vertices. This approach leads to reduction of the number of *split* operations by 70% in comparison to the original algorithm. It also leads to lower number of lists in a list stack, and therefore reduction of the number of the *merge* operations.

The second improvement suggested by [3] attempts to reduce the range of offsets in the *split* operation by renumbering the vertices according to their Euclidean distance to pivot. Such renumbering causes the close vertices (which are likely to be encountered) to posses a low index, thus reducing their entropy, however it requires the decoder to decode the geometry simultaneously with the connectivity.

The final improvement deals with the boundary vertices, which used to be connected to a *dummy* vertex for each boundary loop. These vertices usually had a large degree, which lead to problems with entropy coding. The improvement proposes to join all the *dummy* vertices into a single *ghost* vertex. Such vertex cannot be selected as a pivot, because it will probably be non-manifold, however the traversal can be completed by skipping it.

Using these improvements, and assuming that the number of split operations is negligible, the authors of [3] show that the worst case bitrate is equal to Tutte's constant, i.e. 3,245 bpv.

3.3 Tetrahedral connectivity compression

We will now describe two methods for tetrahedral connectivity compression. The relevance of tetrahedral meshes to dynamic mesh data rate reduction will be justified in the following text.

From Euler equation for triangular meshes follows that the average vertex-triangle order of a low Euler characteristic mesh is 6. Unfortunately, for the case of tetrahedral meshes this does not hold, there are some pathological cases with extreme vertex-tetrahedron order, and even for the meshes usually used in computer graphics or data processing is the vertex-tetrahedron order more variable than in the case of triangular meshes. For example a regular cubic lattice subdivided by the 5 tetrahedra scheme leads to vertex-tetrahedron order 12, while using the 6 tetrahedra scheme produces vertices of average order 14.

The authors of [26] specify the expected relations in a usual tetrahedral mesh as follows:

$$V : E : f : t = 1 : 6.5 : 11 : 5.5 \tag{3.31}$$

where V is the number of vertices, E is the number of edges, f is the number of faces and t is the number of tetrahedra.

3.3.1 Grow&Fold

One of the first efforts in the field of tetrahedral mesh connectivity compression is the Grow&Fold algorithm published by Szymczak and Rossignac [59]. It is based on a combination of ideas of the Edgebreaker and topological surgery algorithms for triangle mesh compression. The algorithm is able to compress the connectivity of a tetrahedral mesh from 128 bits per tetrahedron (four 32-bit indices per tetrahedron) down to a little over 7 bits per tetrahedron.

The first step is similar to building of triangle spanning tree in the topological surgery approach, only this time a tetrahedron spanning tree is constructed. An arbitrary border triangle is selected as a "root door", which leads to the first tetrahedron of the tree. Each tetrahedron adds three new doors, which correspond to the tree new triangles that together with the current door form the given tetrahedron. Each of the doors is then processed in a specific order.

The tetrahedron spanning tree is encoded by three bits per tetrahedron, where the bits declare whether each of the triangles is a "door" to a new branch of the tree or not. The decoder is then able to "grow" the tree by processing the string of encoded tetrahedra. However, this step only reconstructs the connectivity partially, it is necessary to connect the branches of the tree to reconstruct the geometry of the original tetrahedral mesh.

The authors recognize two ways to connect tetrahedra in the tree. First, the "fold" operation connects the tetrahedron to one of its current neighbors, similar

to L or R states in the Edgebreaker algorithm. Each "cut" face, i.e. every face of the grown tetrahedral spanning tree, is assigned a two-bit code, which tells the decoder which one of the edges is the "fold edge", or tells the decoder that no fold operation should be performed upon the given face. There are $2t+1$ external faces of the tetrahedron spanning tree, and therefore there are $4t+2$ bits needed to encode the "folding string", in which the faces are encoded in the same order in which the tetrahedra are processed by the spanning tree construction algorithm.

After the folding of the tree it is still possible that some of the faces that are supposed to be connected are not connected, this case occurs when a face is adjacent to a tetrahedron which is not adjacent in the tetrahedron spanning tree. Such case is solved by the "glue" operation, which is the only one which needs an explicit index, which tells what faces should be glued. It is shown that for reasonable meshes the number of required glue operations is low, and therefore it does not influence the overall performance of the algorithm, which remains at 7 bits per tetrahedron.

3.3.2 Cut-border

The cut-border machine for triangular connectivity compression is in contrast with the Edgebreaker algorithm easily extendible to the case of tetrahedral connectivity compression problem. The extension has been proposed in 1999 by Gumhold and Straßer [26].

The cut-border is a triangle mesh, which divides the tetrahedral mesh into two parts, inner and outer. This border is initialized to a single tetrahedron, and it is grown until it is equal to the border of the whole tetrahedral mesh. The growing of the cut border is performed by processing one triangle of the border at a time, where order of processing of triangles is set by a strategy which will be described later.

There are three basic situations, in which can a cut-border triangle be. It can either be a border triangle, it can be a base of a tetrahedron formed by a new vertex, or it can be a base of a tetrahedron formed by one of the cut-border vertices. Each of these cases is encoded into an output stream, the authors of [26] denote the states as $\triangle$ (close), * (new vertex) and ∞ (connect). Of these only the ∞ needs a parameter, which will tell the decoder which of the cutborder vertices should be used to form a new tetrahedron. The $\triangle$ and * operations do not need any extra information.

In order to improve the compression rate, it is useful to use local indices as parameters of the ∞ operation. Such local indices are created in a way that the nearest vertices have the lowest numbers. One of the edges of the cut-border triangles is selected to be the "zero edge". All the vertices are then searched in the breadth-first order from this edge, until the incident vertex is reached. The order in which it has been found is then encoded as the index for the ∞ operation.

The created sequence of close, new vertex and connect operations is then used by the decompressing algorithm to reconstruct the original connectivity. It is possible that during the decompression the cut-border represents a non-manifold mesh; however the final state of the cut border is guaranteed to be equal to the border of the original tetrahedral mesh.

In order to reduce the number of connect operations with high index, the authors propose to process vertices of the original mesh in a fifo order, while all the cut-border triangles incident with current vertex are processed before the algorithm continues to another vertex. This rule represents one possible strategy for choosing a cut-border triangle.

The choice of zero-edge also influences the number of high index connect operations, the proposed strategy is to set the zero edge for each triangle at the time it is added to the cut border, and set it to the edge that this triangle shares with the triangle which caused its insertion.

The cut-border machine is able to reduce the cost of connectivity compression for tetrahedral mesh from theoretical value

$$C = 4t\lceil log_2 V \rceil \tag{3.32}$$

down to about 11 bits per vertex, i.e. about 2 bits per tetrahedron.

Chapter 4

Dynamic mesh simplification

Triangular mesh simplification is one of the most intensively studied problems of computer graphics. In contrast to compression, simplification always changes the topology of the data, and therefore it almost always changes the shape of the data. From our point of view this represents no problem, as we aim to produce shapes that are visually similar, but not necessarily equal to the input.

There are approaches to various special cases of the problem, however we will only focus on main directions applicable to triangular and tetrahedral meshes. For a deeper review of existing simplification methods see [19, 23, 4].

In the following text we will describe the basic approaches to simplification methods based on vertex removal, edge collapsing, remeshing and geometry images, which is of particular interest for us, because it can be extended to t-variant case in the form of geometry video. We will also give details about tetrahedral mesh simplification methods, because we will show in the last section that a dynamic 3D mesh can be represented by a static 4D tetrahedral mesh.

4.1 Static triangular mesh simplification

4.1.1 Schroeder decimation

The algorithm proposed by Schroeder in [55] works on triangular meshes and is based on vertex removal and retriangulation of the hole. Such approach is known as mesh decimation, and its results strongly depend on the decimation criterion used.

The algorithm works in three steps, which are iteratively repeated until the desired simplification ratio is reached. These steps are

1. characterize the local vertex geometry and topology

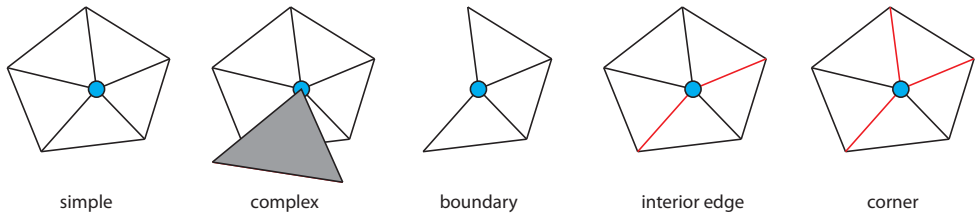

Figure 4.1: Local geometry and topology characteristic for decimation.

2. evaluate the decimation criteria
3. remove vertices and retriangulate the holes

The original algorithm takes into account so-called feature edges, i.e. edges of angle higher than given threshold. Such edges are considered important, and the algorithm tries to preserve them. The feature edges form a geometrical characteristic, which must be evaluated along with the topological characteristic of vertex surroundings. We can distinguish the cases depicted in figure 4.1.

- simple vertex (its neighborhood is topologically equivalent to a disc, it does not incide with any feature edges)
- complex vertex (a non-manifold vertex, cannot occur with our inputs)
- boundary vertex (incides with two border edges)
- interior edge vertex (incides with exactly two feature edges)
- corner vertex (incides with three or more feature edges)

The complex vertices and corner vertices are not considered candidates for decimation. For the simple vertices, a "distance to average plane" criterion is evaluated, while for the interior edge and boundary vertices the "distance to the edge" criterion is used.

The "distance to average plane" criterion is evaluated as follows: an average plane is first constructed from all neighbors of a simple vertex, and subsequently the orthogonal distance of the vertex to this plane is computed. If the distance is lower than some threshold, then the vertex is removed and the hole retriangulated.

The "distance to edge" criterion is evaluated for boundary and interior edge vertices. In the case that the given vertex is removed it is obvious that the edge or border at given location will be replaced by a line segment. The criterion value is the orthogonal distance of the original vertex position to the replacement line segment. If this distance is lower than a given threshold, then the vertex is removed and the hole retriangulated.

The retriangulation is performed using the recursive loop splitting procedure. By removing a vertex a loop of vertices is created which bounds the hole to be

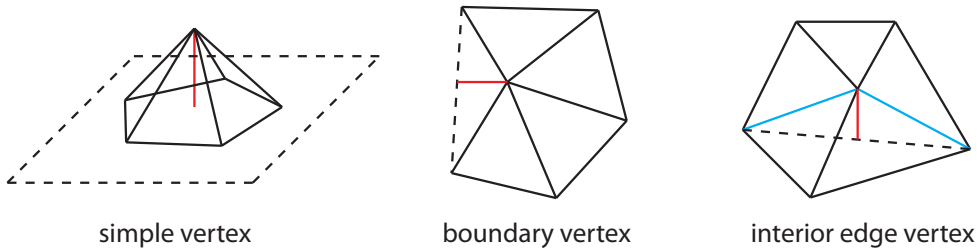

Figure 4.2: Decimation criteria for vertex removal. Red distances represent the criteria value, blue edges are the feature edges.

retriangulated. The algorithm chooses a pair of non-adjacent vertices from the loop, which divides the loop into two parts. Each part is the again divided in the same fashion, until only one triangle remains in each loop. Such triangles are then added to the mesh.

In the case of interior edge vertices and boundary vertices the first splitting edge is always the border edge or the interior edge replacement. However, in all steps of triangulation of holes created by simple vertices, and in later steps of triangulation of holes created by removal of boundary vertices and interior edge vertices, there are usually more possible ways to create the dividing edges. The edges are considered with respect to their "split plane", i.e. a plane that is orthogonal to the average plane of the hole, and which contains the edge. The current boundary loop vertices are evaluated against this plane, and if the plane separates them into two groups consistently with the half loop into which each vertex belongs, then the given edge is accepted as possible split edge. If no acceptable split edge is found, then the vertex is not removed from the mesh.

Still, there may be multiple possible split edges. For each such edge an aspect ratio criterion is evaluated. This criterion is constructed to prefer short edges that well separate the edges of the hole. Let's denote the minimum distance of the loop edges to the split plane d_{min}, and the length of the edge l. The aspect ratio is then simply expressed as

$$C_{aspect} = \frac{d_{min}}{l} \tag{4.1}$$

In order to get best results is the value of the criterion limited by some threshold, and if no edge is found to provide sufficient criterion value, then the vertex is again not removed.

This simplification scheme is very simple and has been extended to cover various special cases and provide better results by using different criteria. Various speedup techniques have been also proposed in order to avoid sorting the candidate vertices in each step of the algorithm [20]. However, the method only uses the vertex positions of the original mesh, even in cases when it could be beneficial to move the vertices to new positions in order to better fit the original shape.

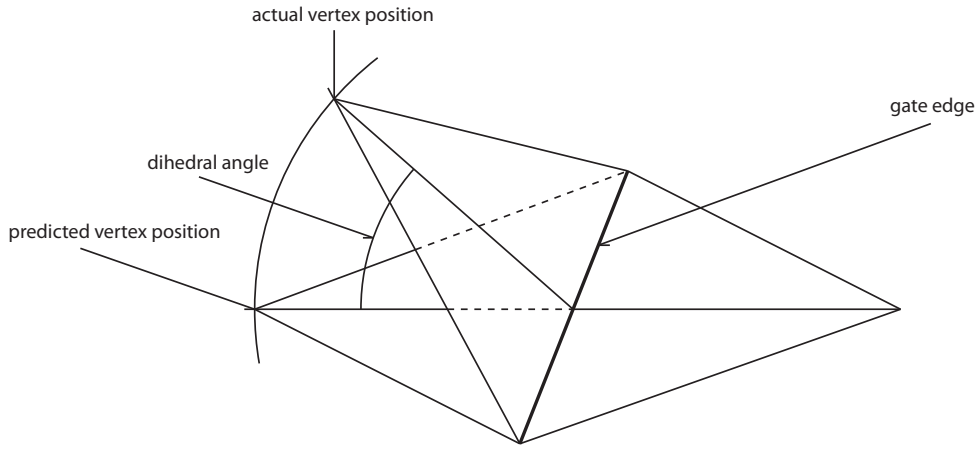

Figure 4.3: Dihedral angle predictor

4.1.2 Attene SwingWrapper

The SwingWrapper algorithm has been proposed by Attene et al. in 2003 [8]. It is basically a remeshing algorithm that works in a way similar to the spinning edge algorithm [13] used for tessellation of implicit surfaces. It completely replaces the original connectivity of the mesh by a new one, which is constructed to be very regular, so that a connectivity compression algorithm such as Edgebreaker can process it very efficiently. Moreover, a new vertex position prediction scheme is suggested, which only uses one parameter to set the position of a new vertex.

The basic idea is to use equilateral triangles of predefined edge length L wherever possible. The algorithm first randomly selects one vertex of the original mesh. The second vertex is also randomly chosen on the intersection of a sphere of radius L centered in the first vertex, and the original mesh. Note that this second vertex and any following vertices are not vertices of the original mesh.

The first and second vertices form an initial edge. Third and fourth vertex, which form an initial pair of triangles, are found as crosspoints of a circle of radius $\frac{\sqrt{3}}{2}L$ centered in the midpoint of the initial edge, which lies on a plane orthogonal to the initial edge, and the original mesh.

The open edges of the initial couple of triangles form four initial gates, from which a combination of remeshing algorithm and Edgebreaker algorithm starts an iterative processing of the mesh. During this process, a new triangular mesh M' is created from the original mesh M.

A midpoint of the gate is selected, a circle of radius $\frac{\sqrt{3}}{2}L$ centered at the midpoint and lying on a plane orthogonal to the gate is constructed, and its intersections with the original mesh are found.

The intersection point which is further from the base triangle of the gate is selected. If it is closer to any existing vertex of M' than one half of the L length, then it is snapped to this vertex by encoding one of the LERS Edgebreaker codes.

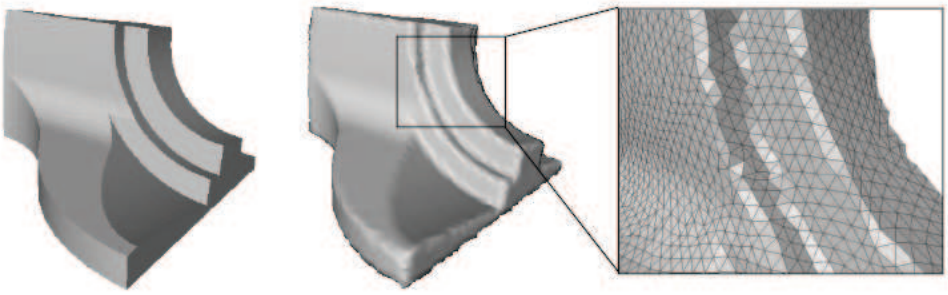

Figure 4.4: Smoothing effect of the SwingWrapper remeshing. Taken form [8].

If the new vertex is not snapped to any existing vertex, then the C case is encoded, along with geometry specification of the new vertex.

The encoding of a new vertex is performed in a way which exploits the regularity of the mesh. The so-called dihedral angle scheme is used to encode the position of the new vertex. The decoder knows that the new vertex lies on a circle centered in the mid-point of the gate, and can guess its position to be on the plane of the gate triangle. The only correcting information is the angle between the guessed position of the new vertex, and its real position, i.e. a single number, which is claimed to be sufficiently quantized to 8 bits, given that both decoder *and encoder* use the quantized positions, so that the error does not propagate and accumulate.

The overall size of the encoded mesh can be easily computed as

$$E = G + C = 8v + 2t = 6t \tag{4.2}$$

where E stands for the encoded length of the mesh, G stands for encoded length of geometry and C stands for the length of encoded connectivity. We can also estimate the dependency of the number of triangles of M' on the selected length L:

$$t = \frac{A}{L^2\sqrt{\frac{3}{4}}} \tag{4.3}$$

where A stands for the area of the original mesh M. Given these relations, we can easily steer the compression to produce compressed representation of desired length. Using some advanced arithmetic coding technique can bring the encoding cost even lower down to about 4 bits per triangle.

There are two main drawbacks of this approach. First, the created mesh is very regular, and it does not use the local properties of the input mesh, i.e. even very flat regions of the original mesh will be sampled with constant density.

The other drawback is that the method performs some smoothing of the original mesh, which is also caused by the fact that it does not adapt to local

changes in the mesh curvature. This feature could be solved by some adaptive technique known from implicit function tessellation, but this would make the encoding more complex and in effect less efficient.

4.1.3 Geometry images

The Geometry images technique has been first proposed by Gu, Gortler and Hoppe in [24]. The method also resamples the original mesh in order to fit a given existing compression technique, only this time the target technique is image compression. The idea is to cut the original mesh, parameterize the boundary onto a square domain, sample the domain and encode the sampled XYZ coordinates using some off the shelf method like JPEG. The steps of the algorithm are quite complex, and therefore we will only mention the general techniques, omitting the implementation details.

The first step of the algorithm is the cutting of the original mesh. The method is supposed to be able to process general genus meshes, and it can be shown that a closed manifold mesh of any genus can be unfolded to a topological equivalent of a disc by a number of cuts. The algorithm proposed by the authors finds such cut by first growing the cut to cover the whole mesh, and then by pruning the unnecessary parts of the cut.

The next step is forming the parameterization of the disc-equivalent onto a square domain. The cut further expanded during this iterative process in order to reduce the geometrical stretch, which is computed using the algorithm described in [53]. First, the disc is parameterized onto square domain using the Floater algorithm [18], and a point of maximum stretch is identified. Subsequently a path from the extreme stretch point to the current border of the disc is found, cut and added to the disc boundary, forming the new cut, which is used as input for the next iteration of the algorithm. The iterative process ends after reaching given number of steps, or when the overall stretch is not reduced by further steps.

Finally, the square domain is regularly sampled, reprojecting the locations from the square image domain back to the geometry and computing the vertex positions. The computed XYZ positions are then encoded in a way similar to the RGB triplet encoding used in JPEG.

The decompression reconstructs the point locations encoded in the geometry image and spans a regular triangular mesh on them. Some problems may arise at the borders of the domain, which require some topological sideband to be also encoded with the mesh.

The compression rate can be steered by choosing different sampling rate of the parametric domain, and by choosing different encoding strategies for the XYZ triplets. The method works very well for simple meshes of regular shape, but it has problems with meshes that are hard to parameterize onto a square domain. The authors suggest to also encode point normals at sampled positions to improve

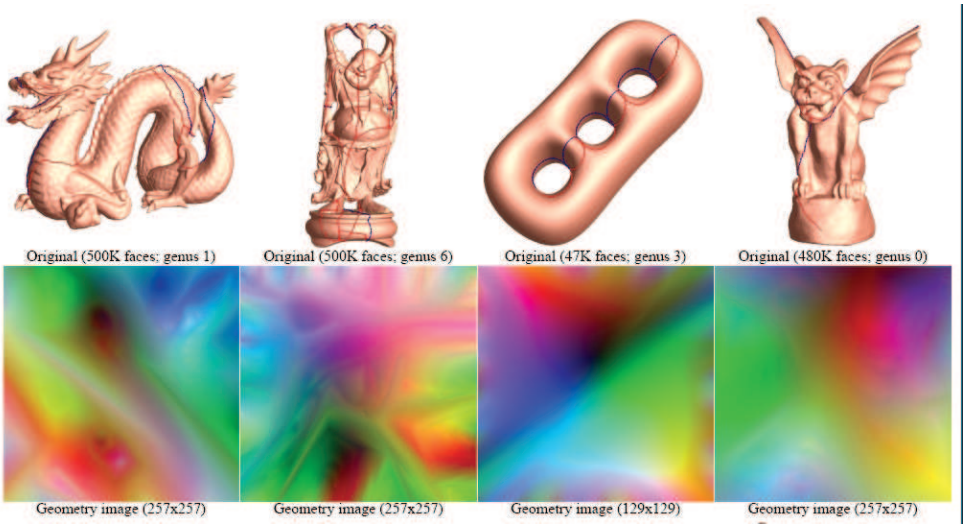

Figure 4.5: Examples of geometry images. The top row shows the cut, the bottom row shows the resulting geometry images. The image has been taken from [24].

the visual effect of the encoding.

4.1.4 Edge collapse

In their work published in 1996 ([50]), Ronfard and Rossignac have presented a new scheme for simplification based on the edge contraction as a basic operation. Along with it, they have presented a cost function based on distance to planes.

The edge collapse is an elementary operation that has been widely accepted for simplification algorithms with many other simplification criteria. The basic idea is that a cost is assigned with each edge, which tells how much distortion would the contraction of the edge cause to the mesh. These costs are used as keys to an optimized priority queue data structure. Subsequently, the top edges with least contraction cost are removed from the queue, and contracted, i.e. the first vertex v_1 of the edge is removed, and in all incident triangles it is replaced by the opposite vertex v_2. The contraction costs of altered edges are updated, and the algorithm continues contracting the next least cost edge, until it reaches the desired simplification ratio.

The main advantage of edge contraction in contrast with vertex decimation is that there is no need for retriangulation. In each edge contraction step exactly two triangles are removed.

After the pioneering work of Ronfard and Rossignac, many methods were proposed to compute the edge contraction costs, and to compute the new position of the vertex. The original algorithm proposes to use a tessellation (topological) criterion, a geometrical criterion and a relaxation process to determine these properties.

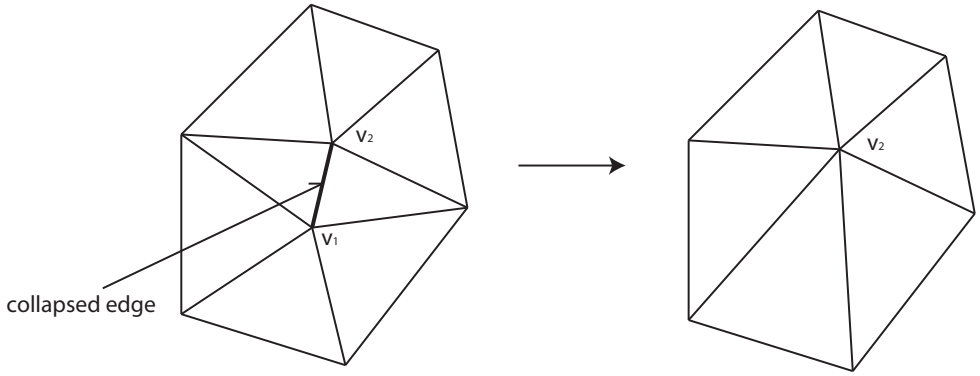

Figure 4.6: Edge collapse operation.

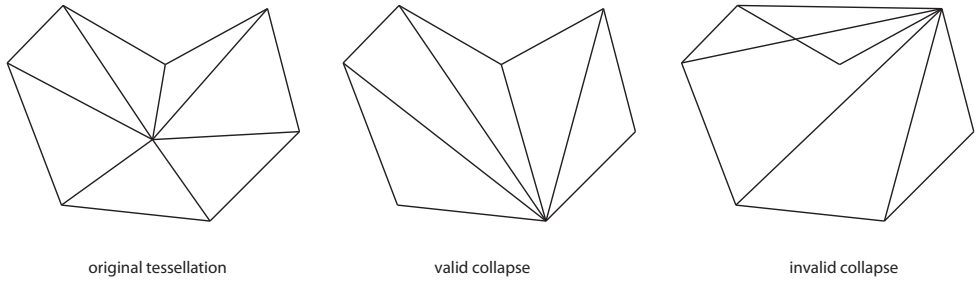

Figure 4.7: Triangle flip.

The tessellation criterion is used to evaluate the edges, preventing triangle flips. A triangle flip occurs when a normal of a triangle flips its direction by 180 degrees. The criterion computes the angle between original and updated normal A_t for all triangles t affected by an edge contraction, and selects the maximum value as the cost:

$$LTE(V_1, V_2) = K \max_{t \in triangles(V_1,V_2)} A_t \tag{4.4}$$

The geometrical criterion is used to evaluate the distortion caused by the edge contraction. For each vertex, a "star" of incident edges is kept. From this star, one can determine a set of planes, which meet at the vertex. Squared distance to any of these planes (represented by implicit function $p.x = 0$) from a point x can be determined as:

$$d^2(x, p) = x.p \tag{4.5}$$

The geometrical criterion value is determined as a maximum of these distances from the new vertex position to all of the planes incident with the original vertices:

$$LGE(V_1, V_2) = \max_{p \in planes(V_1,V_2)} d(V_2, p) \tag{4.6}$$

From the equation follows that the new position of the vertex is one of the positions of the original vertices. Note that the set of planes associated with the

new vertex is the union of the sets of planes of the original vertices, i.e. it is not recomputed from the new tessellation of the neighborhood of the new vertex. The overall cost is then determined as a maximum of LGE and LTE. The authors have also suggested a relaxation process, which will move the replacement vertex to a position which better fits the original local shape. The proposed algorithm is to find a minimum of the sum of distances to the set of incident planes and set the position of the vertex to this minimum:

$$V_2^* = \min_x \sum_{p \in planes(V_2)} d(x, p) \tag{4.7}$$

Note that the set $planes(V_2)$ now already contains the union of the two original sets of planes. The authors discuss the possibility to use this optimized position in the cost computation of edges, but state that it would be inefficient.

4.1.5 Quadric based

The quadric based simplification method proposed by Heckbert and Garland in 1997 ([44]) is based on the pair contraction technique, which is similar to edge contraction, and its main contribution is a novel method of evaluating the pairs for contraction.

The concept of pair contraction is a simple augmentation of the edge contraction described above. The difference is that a pair can be either an edge, or a couple of vertices which is not connected by an edge, but which is very close together. Using the pair contraction instead of edge contraction allows simplification of topology of the mesh, i.e. connecting previously unconnected components of the mesh.

The algorithm works in four steps:

1. select valid pairs for contraction
2. evaluate the pairs (see below)
3. sort the pairs according to their evaluation
4. iteratively contract the pairs and update the mesh

The key step is the evaluation of pairs. First, we will describe the situation at one vertex. A vertex is an intersection of planes, in which lie the incident triangles. If we want express the squared distance of a point x from a plane p, we use a simple dot product:

$$d_p^2(x) = x^T p \tag{4.8}$$

where x is the position represented in homogeneous coordinates and p is the vector of coefficients of the implicit plane equation. If we now want to express

the squared distance from a set of planes incident with some vertex v, we simply add the distances together:

$$d_v^2(x) = \sum_{p \in planes(v)} x^T p \tag{4.9}$$

The sum can be rewritten as follows:

$$\begin{aligned} d_v(x) = & \sum_{p \in planes(v)} (x^T p)(p^T x) & (4.10)\\ = & \sum_{p \in planes(v)} x^(T)(pp^T)x & (4.11)\\ = & x^T(\sum_{p \in planes(v)} pp^T)x & (4.12)\\ = & x^T Q x & (4.13) \end{aligned}$$

From the last expression follows, that computing the sum of squared distances from a set of planes incident with a given vertex can be expressed by a quadratic form. This form can be pre-computed for each vertex of the original mesh. When a pair is considered for contraction, then the quadratic forms of its endpoints are simply added together. The final position after the contraction is found by minimizing the error measure. The minimum is found by finding the zero point of the first derivative of the quadric form, i.e. by solving the following set of equations:

$$\begin{bmatrix} q_{11} & q_{12} & q_{13} & q_{14} \\ q_{21} & q_{22} & q_{23} & q_{24} \\ q_{31} & q_{32} & q_{33} & q_{34} \\ 0 & 0 & 0 & 1 \end{bmatrix} x^* = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 1 \end{bmatrix}$$

where q_{xy} are elements of the summed quadric $Q_{12} = Q_{v1} + Q_{v2}$. Note that the last line of the matrix expresses that we are looking for a solution with homogeneous coordinate equal to 1.

The overall error measure for given pair is expressed as

$$p = x^{*T} Q_{12} x^* \tag{4.14}$$

This measure is then used to sort the pairs, and the pairs of lowest error measure are contracted first.

4.1.6 Locally volume preserving edge collapse

All the edge collapse criteria presented so far were based on the idea that the new cost should be computed with respect to the original mesh. Lindstrom and Turk in their works [36, 37] have dropped this assumption, and their algorithm only uses the current simplified version of the mesh to compute the costs of contraction of each edge. Their criterion is based on local volume preservation, which leads to global volume preservation, which is a problem for some simple vertex placement

schemes, including the original one presented in [50]. We will first describe the algorithm that sets the position of the new vertex after edge collapse, and then we will show a cost function which is closely related to it.

The new vertex position is set by searching for an intersection of three planes, each of which represents some constraint about the position of the vertex. The authors provide equations for several possible constraints, and propose an ordering in which these constraints are evaluated, and their planes constructed. It is possible that some of the constraints produce planes that are almost coplanar, i.e. an underdetermined system which is easily disturbed by rounding errors. Such case is detected by checking the angle between the planes. Cases when the angle is lower than 1 degree are called α incompatible and the next constraint from the priority ordering is selected. The possible constraints are (in order in which they are evaluated):

1. volume preservation
2. boundary preservation
3. volume optimization
4. boundary optimization
5. triangle shape optimization

The volume preservation constraint enforces that the volume is not changed after the edge is collapsed. The space between the original and new tessellation of the neighborhood of the collapsed edge is divided to tetrahedra, each having base in one of the original triangles, and a top in the new vertex. A signed volume of a tetrahedron can be expressed as:

$$V(t) = \frac{1}{6} det \begin{pmatrix} v_x & v_{0x} & v_{1x} & v_{2x} \\ v_y & v_{0y} & v_{1y} & v_{2y} \\ v_z & v_{0z} & v_{1z} & v_{2z} \\ 1 & 1 & 1 & 1 \end{pmatrix}$$

Where v is the location of the top of the tetrahedron, and $v0$, $v1$ and $v2$ are vertices of its base. Note that this volume is negative when the top is located below the base with respect to the normal of the base(i.e. when the volume is reduced) and positive when the top is above the base (i.e. when the volume is added). Therefore, it suffices to sum the volumes up and solve for zero:

$$\sum_{t \in tetrahedra(v_1, v_2)} V(t) = 0 \tag{4.15}$$

Solving this equality constrains the solution v to a plane.

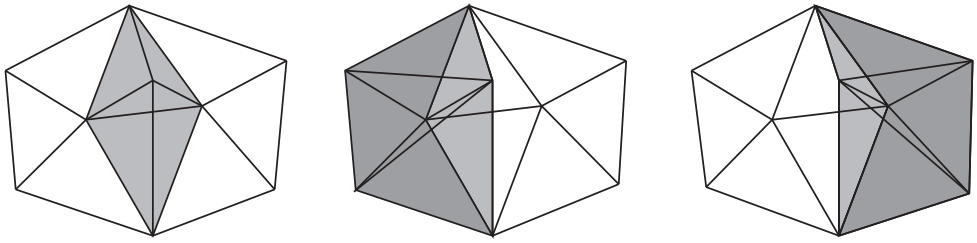

Figure 4.8: Tetrahedra created in the neighborhood of a contracted edge, from left to right two tetrahedra incident with the removed triangles, tetrahedra incident with vertex v_1 and tetrahedra incident with vertex v_2.

The boundary preservation works in a similar manner to the volume preservation, and it is evaluated only for contractions that affect at least one boundary edge. It starts with signed area A defined for each original boundary edge e as

$$A(v,e) = \frac{1}{2}(v \times e_0 + e_0 \times e_1 + e_1 \times v) \tag{4.16}$$

Searching for zero sum of signed areas is however only possible for the case of planar triangles, so the algorithm only minimizes the following expression:

$$\left(\sum_{e \in boundary} \frac{1}{2} (v \times e_0 + e_0 \times e_1 + e_1 \times v) \right)^2 \tag{4.17}$$

The minimization of this term yields two additional planes. Volume optimization is similar to volume preservation, only this time the unsigned version of tetrahedron volumes are minimized, bringing the new surface as close as possible to the original one (the volume "between" the surfaces is minimized). The minimized expression takes following form:

$$\sum_{t \in tetrahedra(v_1,v_2)} (V(t))^2 \tag{4.18}$$

This constraint yields up to three additional planes. The boundary optimization constraint is evaluated in equivalent manner - the minimization expression which produces up to three additional planes has the following form:

$$\sum_{e \in boundary} (A(v,e))^2 \tag{4.19}$$

The triangle shape optimization constraint is only used in the case when the previous constraints have not produced three α -compatible planes. This case usually occurs when the triangles of the original mesh are almost coplanar. In such case, the last constraint simply prefers equilateral triangles to long ones. The minimized expression is:

$$\sum_i (L(v, v_i))^2 \tag{4.20}$$

where L is the distance between two vertices, and v_i are the vertices adjacent to the new vertex v.

The contraction cost that needs to be computed for each edge is finally computed as a weighted sum of volume optimization expression and boundary optimization expression. Equal weight used in the paper for comparative testing against other simplification techniques produced results comparable with the most efficient algorithms.

4.2 Dynamic mesh simplification

4.2.1 Decimation of dynamic meshes

The decimation scheme proposed by Schroeder[55] can be used for the dynamic mesh decimation, given that the decimation order criteria are extended to cover the temporal behavior of the vertices.

One suggestion of such extension has been given by Mathur et al.[41] in 2004. They have proposed two importance measures for vertices, the CURV measure, based on vertex curvature, and the SKEL measure, based on the properties of the skeleton. Using these importance measures, they report improvement of quality of simplified meshes in terms of quadric error as described by Garland [44].

First, the authors define local curvature of vertex i as

$$\kappa_i = \frac{\sum_{l=1}^{n} a_{f_l} \kappa_{f_l}}{\sum_{l=1}^{n} a_{f_l}} \tag{4.21}$$

where n is the number of faces in the one ring neighborhood of vertex i, a_{f_l} is the area of the l^{th} face, and κ_{f_l} is defined as

$$\kappa_{f_l} = \frac{1 - (n_i n_{f_l})}{2} \tag{4.22}$$

where n_i is the unit normal at vertex i and n_{f_l} is the unit normal of the l^{th} face.

The local curvature is defined for each frame f. The average curvature of a vertex over the duration of the animation is defined as

$$\overline{\kappa}_i = \frac{\sum_{f=1}^{F} \kappa_{if}}{F} \tag{4.23}$$

Similarly, we define deviation from the average at frame f as

$$\delta\kappa_{if} = |\overline{\kappa}_i - \kappa_{if}| \tag{4.24}$$

and the average deviation over the duration of the animation is then defined as

$$\delta\overline{\kappa}_i = \frac{\sum_{f=1}^{F} \delta\kappa_{if}}{F} \tag{4.25}$$

Finally, the CURV criterion is defined as

$$CURV(i,\alpha) = \alpha \cdot \delta\overline{\kappa}_i + (1-\alpha)\cdot \overline{\kappa}_i \tag{4.26}$$

This value can now be used as a decimation criterion. The authors also propose to use this measure to determine the accuracy of quantization in the dynapack algorithm[29]. We will not describe the other importance measure, SKEL, because it is based on the knowledge of the mesh skeleton, which we assume not to be known.

We note several drawbacks of this measure. First, the local curvature definition (4.21) uses direct area weighting, which contradicts the ideas presented by Max[42], who suggests using inverse area weighting for computing vertex properties from properties of adjacent faces.

Second, the paper does not give any value for the α constant of (4.26), the authors state that this constant can be used to tweak the measure to obtain desired results.

Finally, the dynapack application is questionable for two reasons. First, there is an overhead associated with using different quantizations, because the decoder cannot evaluate the curvatures and therefore the information about the desired quantization must be transmitted with each vertex. Second, the CURV importance measure behaves basically like a feature detector, however the decision to quantize coarsely the non-feature areas is not supported by any reasoning. We can actually state an opposite assumption, that the eventual artifact introduced in a non-feature area will be more disturbing, because it will be well visible, while in areas of high curvature deviation it will be harder to spot by a human observer.

4.2.2 Geometry video

Geometry videos, proposed by Briceno in 2003 [9], are a straightforward augmentation of the geometry images technique described in [24] to the dynamic case. The authors use a sequence of geometry images to represent a sequence of meshes of constant connectivity, and they suggest encoding such sequence using a video encoding algorithm such as MPEG-4.

The result of the method is again a constant connectivity mesh. The density of the mesh depends on the density of sampling of the parametric domain. The main issue that needs to be solved is how to compute the parameterization so that it releases the stress in all the frames. The authors propose following possibilities:

1. use the "worst frame" to compute the cut
2. use global stress measure to compute the cut

However, the first possibility is only applicable for meshes where a worst frame contains all the high tension areas, which is not always the case (for example morphing objects have high strain at different positions in different frames).

The second approach is capable of producing acceptable results for most cases where geometry images technique works. The authors propose to use average stretch measure normalized to overall stretch in each frame, so that a single frame of high distortion cannot drive the algorithm astray.

The shortest path to current boundary algorithm must also be modified to reflect the changing length of the edges, which is done in a similar manner - average length of edges is used. The iterative cutting algorithm is stopped by the "average average" condition, i.e. when the average distortion computed at vertices from all time steps is in average not decreasing anymore.

4.2.3 Quadric based simplification in higher dimension

The authors of [44] have extended their quadric metric to any dimension in 2005 ([22]). Their approach allows simplification of meshes embedded in any dimension and containing simplices of any dimension. The simplification algorithm remains virtually intact, as it is again based on iterative pair contraction.

The algorithm works as follows:

1. compute fundamental quadrics for all simplices
2. compute aggregate quadrics for vertices
3. evaluate the pairs according to quadrics
4. sort the pairs
5. contract pairs with the lowest evaluation value

In order to derive the generalized quadric based error metric, we will start with generalized Pythagorean theorem. Given a point p in d-dimensional space and some orthonormal basis of the space $B = \{e_i\}_{i=1}^{d}$, we can express the squared distance to this point as:

$$\|x-p\|^2 = \sum_{i=1}^{d} \left((x-p)^T e_i\right)^2 \tag{4.27}$$

$$= \sum_{i=1}^{d} (x-p)^T (e_i e_i^T)(x-p) \tag{4.28}$$

$$= (x-p)^T (\sum_{i=1}^{d} e_i e_i^T)(x-p) \tag{4.29}$$

In order to generalize the notion of distance to a plane which contains a triangle, we must now consider distance to a sub-space which contains a simplex. Each such sub-space is defined by a basis, which can be easily obtained, and can be transformed to an orthonormal basis using the Schmidt orthonormalisation process. This way, we get a set of directions, in which the distance does not contribute to the error metric, and a set of distances which contribute. We can now express such metric as a reduced sum:

$$e(x,p) = (x-p)^T \left(\sum_{i=n+1}^{d} e_i e_i^T \right) (x-p) \tag{4.30}$$

where $e_0 \dots e_n$ is the basis of the sub-space spanned by a given simplex (note that these do not contribute to the error metric) and $e_{n+1} \dots e_d$ represent the additional basis vectors.

As we can see, this expression is not very practical, as it does not use the basis of the subspace, which can be computed easily. However, it can be rewritten in equivalent, but much more convenient form:

$$e(x,p) = (x-p)^T \left(I - \sum_{i=1}^{n} e_i e_i^T \right) (x-p) \tag{4.31}$$

This formula is equal to computing the "full" distance, and then subtracting the distances in those directions which should not affect the error metric. The formula can be rewritten in a form of quadratic expression:

$$Q(x,p) = x^T A x + 2b^T x + c \tag{4.32}$$

where

$$A = I - \sum_{i=1}^{n} e_i e_i^T, b = -Ap, c = p^T A p \tag{4.33}$$

The triplet $Q = (A, b, c)$ will be called fundamental quadric of a point of given tangent hyperplane. Now, we can derive an error metric for a single simplex. For a given simplex σ we can write:

$$Q(x,\sigma) = \int_{p \in \sigma} Q(x,p) \tag{4.34}$$

where the integration is performed over all the points of the simplex. Since all the points p of the simplex have the same tangent plane, we can simplify the expression to

$$Q(x,\sigma) = \omega_\sigma Q(x,p) \tag{4.35}$$

Where p is any point of the simplex, and ω_σ is the hypervolume of the simplex. Again, the expression can be rewritten as a quadratic form, which is described by the (A, b, c) triplet. When considering the error metric for a given vertex, we can simply add the error metrics of incident simplices:

$$Q(x, v) = \sum_{\sigma \in simplices(v)} Q(x, \sigma) \tag{4.36}$$

This measure is called the aggregate quadric, and it is again described by a quadric triplet. Such triplet is associated with each vertex of the original mesh, and it is used for evaluating the candidate pairs. The optimum point location x^* can be computed by solving a linear system of equations, similar to the one used in 4.1.5. The pairs are evaluated by the error metric value in this optimum location, and the contraction starts with pairs of lowest value.

This generalized method is equal to the method described in [44] when applied to the case of triangular meshes in E3, which is now treated as a special case of this general approach. This approach can be readily used for simplification of tetrahedral meshes in 4D, which is of particular interest for us.

4.2.4 TetFusion

TetFusion algorithm proposed by Chopra and Meyer in [14] is a simplification scheme based on a new elementary operation called TetFusion. The algorithm uses geometrical and attribute error measures to steer the iterative process of simplification.

The TetFusion operation replaces a single tetrahedron with a vertex located at the barycentre of the original tetrahedron. The main advantage of this operation over edge collapse is that it removes at least 11 tetrahedra, because by collapsing the tetrahedron into a vertex at least 10 other tetrahedra become degenerated (all face and edge neighbors) and can be removed. The tetrahedra that share one vertex with the prey tetrahedron are stretched by the operation, however, the stretch usually does not flip the tetrahedra.

The algorithm works in following steps:

1. mark all tetrahedra as "unaffected"
2. for each tetrahedron check whether suitable for TetFusion, if yes, then TetFuse and mark affected tetrahedra
3. repeat from 1 until desired reduction ratio is reached

The step 2 of the algorithm involves four tests:

1. scalar attribute error test

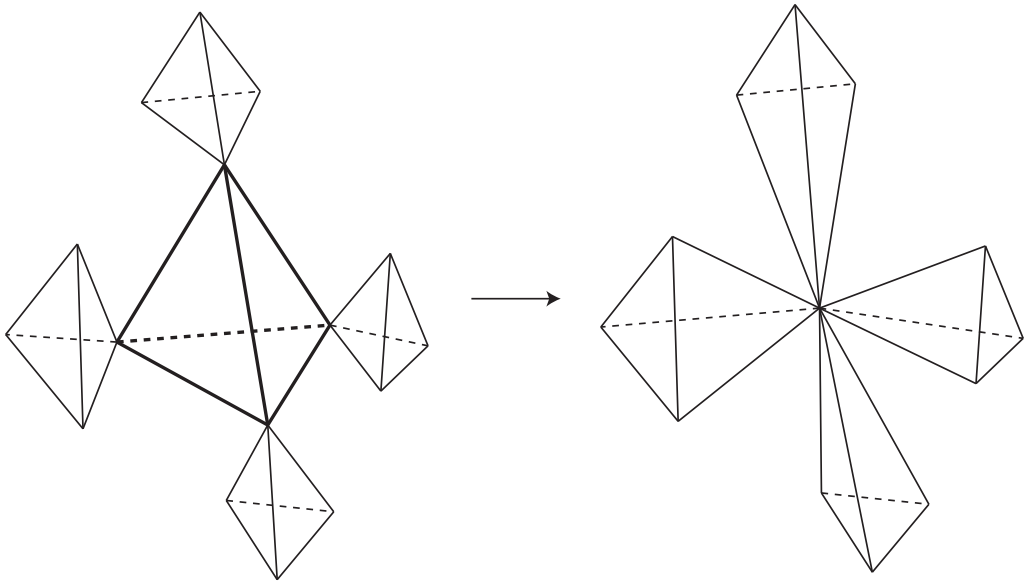

Figure 4.9: The TetFusion operation. The prey tetrahedron is depicted bold, the affected tetrahedra are depicted thin.

2. geometric error test
3. boundary preservation test
4. flipping prevention test

The first test ensures that error of a scalar attribute associated with vertices is not larger than some given threshold. The new scalar value associated with the new vertex is computed using some interpolation technique, and the test passes if the difference from the original values is not bigger than the given threshold for any of the original vertices.

The geometric error test ensures that the geometry of the mesh is not changed more than some given threshold. For each affected tetrahedron (i.e. a tetrahedron sharing exactly one vertex with the prey tetrahedron) a geometric stretch ratio is computed. The geometric stretch ratio is defined as a ratio of original and changed base vector, where base vector is the distance from the base triangle of the affected tetrahedron to its barycentre. If the geometric stretch ratio is larger than a preset threshold, then the tetrahedron is refused, otherwise the test passes.

The boundary preservation test ensures that the boundary of the mesh remains unchanged. The test passes if the prey tetrahedron is not a boundary tetrahedron, and if none of the affected tetrahedra is a boundary tetrahedron.

Finally, the flipping prevention test refuses the rare case when some of the affected tetrahedra flips, i.e. violates the consistency of the mesh. This case only

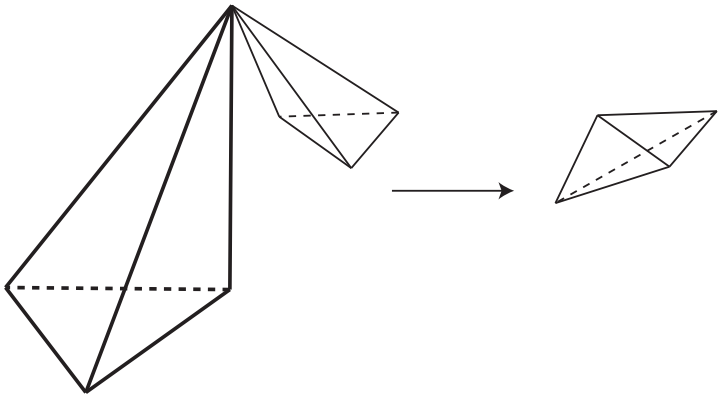

Figure 4.10: Flipping of a tetrahedron.

occurs when the affected vertex moves from one side of the base triangle of the affected tetrahedron to the other side. The test passes if this case does not occur for any of the affected tetrahedra.

The method has been tested for 3D tetrahedral meshes, where it provided good results. However, its application on a 4D tetrahedral mesh has never been tested, and it may require adding some geometry conditions that will penalize simplification of areas of high detail.

4.2.5 Average quadric based simplification

One of the few works targeted specifically at dynamic mesh simplification is the paper by Mohr and Gleicher [46]. Their method simplifies the dynamic mesh and produces again a constant connectivity mesh, i.e. the simplification is performed on the single connectivity, which is used for all the frames.

Their suggestion is to use the equivalent of Garland simplification of a static mesh, with the only difference that the vertex quadric value is evaluated for each frame, and summed to form the collapse cost, i.e. the cost of collapsing an edge $(v1, v2)$ into a final position v is expressed as

$$\sum_{i=1}^{F} v(Q_{v_1,i}Q_{v_2,i})v^T \tag{4.37}$$

where $Q_{v,i}$ stands for the aggregate vertex quadric of vertex v in frame i, and F being the number of frames.

The method generally provides better results than applying quadric based simplification on a single frame, but other possibilities of the global criterion, like using the maximum quadric value, are not discussed.

4.2.6 Quadric based multiresolution mesh

In 2005, Kircher and Garland [34] have proposed an interesting algorithm for creating multiresolution dynamic meshes. They suggest using the quadric error metric (QEM) to produce a hierarchy of simplified versions of the mesh. They first create such hierarchy for the first frame, and subsequently search for so called swaps, which update the structure of the hierarchy to better suit the geometry of subsequent frames.

In the first step, a hierarchy of coarser representations of the first frame is created using the QEM based method described in section 4.1.5. The method is applied iteratively and its results are stored in a tree-like data structure. This structure consists of several levels, each representing a version of the mesh. Each node corresponds to a vertex. Contraction operation used in the QEM based simplification contracts a given number (branching order of the tree) of vertices at given level to form a vertex at a higher level.

The original vertices and their coarser representation are connected by the so-called contraction edges. These edges form a tree structure. Additionally, at each level there are edges which represent the actual topology at the given level. These are actually necessary only for the finest level, but they are useful for updating the mesh during the reclustering of subsequent frames.

The multilevel mesh obtained by simplification of the first frame can be used for any other frame, however it may not be optimal, because the geometry of subsequent frames is different, and therefore the quadrics may also produce a higher error.

The key idea is to use the hierarchy from previous frame to the next frame, and to update it by moving vertices from one cluster to another. It is likely that only small changes are present in the mesh, and therefore only a small number of changes will take place. Therefore, instead of creating the whole structure for each frame from scratch, only a few so-called swaps are performed and encoded with the mesh.

A swap is a primitive operation of reclustering. As a swap we denote moving a vertex v from its current cluster a to a neighboring cluster b. A swap is fully described by the (v, a, b) triplet, and in order to be performed at a given time, it has to be valid and beneficial. A set of such swaps is performed and encoded with each frame of the mesh, which ensures that the hierarchy well follows the geometry throughout the animation.

A swap is valid when following conditions are met:

1. there is a vertex in the cluster b, which shares a topology edge with v (i.e. v lies on the border of a towards b)

2. v is not the only vertex in a

3. v is not a pinch vertex of a, i.e. when v is removed from a, a remains topologically compact.

A swap is beneficial, if the QEM is reduced. The benefit of the swap can be roughly guessed as

$$bf = Q_v(b) - Q_v(a) \tag{4.38}$$

where Q_v stands for the aggregate quadric of vertex v, and a and b are the positions of vertices that correspond to the given two clusters at a coarser level of the mesh. This is a conservative guess, as the positions of a and b may change by the swap and therefore the benefit may be even higher.

However, it is necessary to ensure that not only the level immediately above the current level is positively influenced. Therefore, a generalized multilevel quadric metric is proposed, which sums the quadrics from all the levels with weights ensuring that the contributions from each level are uniform:

$$E = \sum_{i=k+1}^{n} \left(w_i \sum_{e \in M_i E_u} \right) \tag{4.39}$$

where E_u is the quadric error at vertex u of mesh level M_i. The weights w_i are obtained as follows:

$$w_0 = 1 \tag{4.40}$$

$$w_{i+1} = w_i \left(\frac{|M_{i+1}|}{|M_i|} \right)^{\beta} \tag{4.41}$$

where $|M_i|$ stands for the number of vertices at level i, and β is an empirically determined constant that compensates for the quadric value growth. The value of the constant has been determined to be 1.9127.

When the multilevel structure is created, it can be cut on any level, and only the original topology of that level, and its updates are then transmitted as the simplified version of the animation. This approach avoids the need of sending the complete topology with each frame, while it still allows altering the topology according to the changes in geometry.

We have identified two main drawbacks of this method. First, the method only uses temporally local information to determine the simplification. This may lead to a situation, when two subsequent frames are simplified in a radically different manner, and although both were originally similar and both were simplified locally optimally, they become very different. This behavior may occur repeatedly, resulting in some visible "flickering" artifacts.

The other drawback is that the number of vertices in each level of the representation remains constant throughout the span of the animation. The validity checks don't allow disappearing of clusters, and there is no mechanism that would add new clusters when necessary.

Chapter 5

Evaluation tools

There are two main approaches to evaluating the quality of data reduction. The first, based on the concept of Peak Signal To Noise Ratio (PSNR) is used mainly in the field of compressions, where each vertex has its counterpart in the compressed representation of the mesh. This approach cannot be used in the case when the connectivity of the mesh is altered. In such case it is usual to evaluate some approximation of the Hausdorff distance between the meshes.

5.1 PSNR

PSNR is a very simple measure of error mainly used to evaluate the accuracy of geometry predictors. Its definition varies slightly with the used approaches, but usually takes following form:

$$PSNR = 20log\left(\frac{1}{V}\sum_{i=1}^{V}\left(\frac{x_i - x_i^{encoded}}{\Delta x}\right)^2\right) \tag{5.1}$$

Where V is the number of vertices predicted, x the actual position of each vertex, $x^{encoded}$ is the encoded position of the vertex, and Δx is the span of values of the position, or the length of the body diagonal of the model.

The measure is easy to implement and fast to evaluate, but it has many drawbacks. First, it cannot be used when the connectivity of a mesh is changed. Second, it does not take into account that some movement causes larger disturbance in the mesh (movement orthogonal to the surface normal) while other may cause almost no disturbance at all (tangential movement).

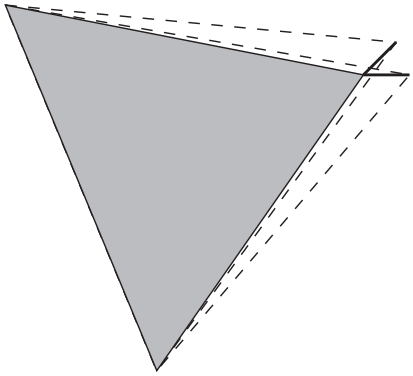

Figure 5.1: Problem of rotation invariance in the DA-Error measure. The original triangle (gray) is distorted in two different but visually equal ways. The distortion vectors (bold) are of equal length, however the error measure will evaluate the horizontal shift as more acceptable (error $\cong 1$) and the diagonal as less acceptable (error $\cong \sqrt{2}$). Rotating the whole scene by 45 degrees will however swap the preference.

5.2 KG-Error

A slightly different approach to measuring error has been taken by Karni and Gotsman[32]. Their approach processes the axes separately, and takes a different approach to normalization. However, the basic properties of MSE measure are preserved, and we are mentioning this approach only because it is being used by some recent papers ([58, 57]) as an error measure. The measure is expressed in percent as follows:

$$KG_{error} = 100 * \frac{\|A - A'\|}{\|A - E(A)\|} \tag{5.2}$$

In the expression the matrix A is a $F \times 3V$ matrix, which represents the original animation sequence. In the matrix, each row represents temporal development of one coordinate of one vertex. Similarly, the matrix A' represents the distorted (encoded/decoded) version of the animation. The matrix E(A) in the denominator of the expression is an $F \times 3V$ matrix, which is computed from the A matrix by replacing values at each column by the mean value of the given coordinate in the given column.

This metric shares the main disadvantages of the PSNR error, it is not sensitive to the type of error introduced (noisy vs. smooth) nor does it detect temporal artifacts, such as shaking.

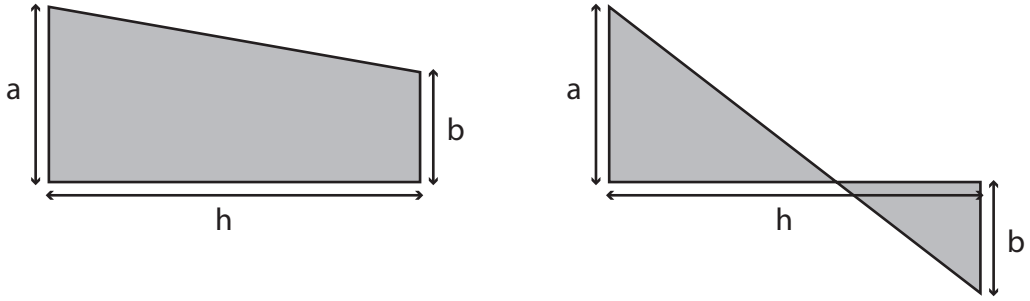

Figure 5.2: Ribbon area computation - straight and twisted trapezoid.

5.3 DA error (Ribbon measure)

Another error measure[48, 30] based on local vertex to vertex movement is being used by the MPEG consortium. In this measure the coordinate axes are again treated separately, however this time a temporal development is also taken into account. The movement of the original vertex and the distorted version of the vertex between two frames is taken into account in a form of a ribbon, whose area is computed. As only one coordinate is taken into account every time, this becomes a quite easy computation in 2D, we only have to distinguish between the twisted and non-twisted case.

For the non twisted case, the area of the ribbon can be computed as an area of a trapezoid (see figure 5.2):

$$D = \frac{1}{2}(a+b) * h \tag{5.3}$$

A slightly more complex version is required in the case of twisted ribbon:

$$D = \frac{a^2+b^2}{2(a+b)} * h \tag{5.4}$$

The algorithm is also scale invariant by incorporating a constant W equal to the largest span of coordinate values over the length of the animation:

$$M_X = \max_{v=1..V}(\max_{f=1..F}(x_{v,f})), m_X = \min_{v=1..V}(\min_{f=1..F}(x_{v,f})) \tag{5.5}$$

$$M_Y = \max_{v=1..V}(\max_{f=1..F}(y_{v,f})), m_Y = \min_{v=1..V}(\min_{f=1..F}(y_{v,f})) \tag{5.6}$$

$$M_Z = \max_{v=1..V}(\max_{f=1..F}(z_{v,f})), m_Z = \min_{v=1..V}(\min_{f=1..F}(z_{v,f})) \tag{5.7}$$

$$W = max(|M_X - m_X|, |M_Y - m_Y|, |M_Z - m_Z|) \tag{5.8}$$

We can express the error in a given coordinate axis X over all frames as:

$$D_A(X) = \sum_{f=1..F}\sum_{v=1..V} D(X) \tag{5.9}$$

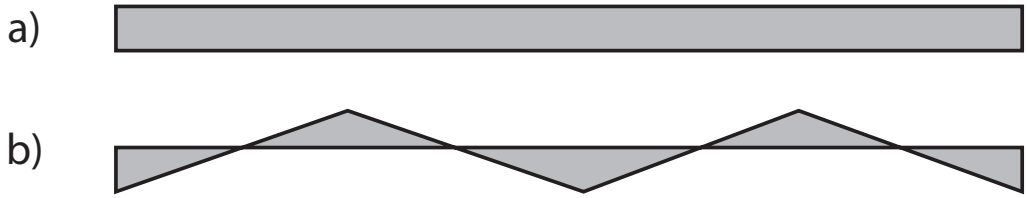

Figure 5.3: Temporal behavior of static and oscillating error, expressed by ribbon measure.

The errors for the Y and Z axes are computed in equivalent way. The final error is then expressed as a sum of the axis errors, normalized by the W constant and the overall duration of the sequence:

$$D_A = \frac{D_A(X) + D_A(Y) + D_A(Z)}{3W(t_F - t_0)} \tag{5.10}$$

One advantage of the D_A error is that it can deal with sequences of irregular framerate, i.e. sequences where the temporal distance between subsequent frames is not known. Therefore it can be applied on animations as well as sets of keyframes, which are not equidistant.

The MPEG group also suggests using the "peak measure" E_P, which is computed as a maximum coordinate difference over the whole sequence:

$$P_X = \max_{f=1..F, v=1..V} |x_{f,v} - x'_{f,v}| \tag{5.11}$$

$$P_Y = \max_{f=1..F, v=1..V} |y_{f,v} - y'_{f,v}| \tag{5.12}$$

$$P_Z = \max_{f=1..F, v=1..V} |z_{f,v} - z'_{f,v}| \tag{5.13}$$

$$E_P = \frac{1}{W} \max(P_X, P_Y, P_Z) \tag{5.14}$$

However, we believe that the match of both these measures with subjective difference perception is highly questionable. The D_A measure quite counterintuitively favors oscillating around the original value to constant distance (see figure 5.3). Neither of the methods is rotation invariant, as can be seen from figure 5.1 and the peak measure also depends on the scale of the data.

5.4 Mesh, METRO

One of the alternatives to PSNR is the Hausdorff distance. It is defined as a symmetric maximum minimum distance between points of meshes M and M'. It is usually not evaluated exactly, only some approximate value is found using some sampling algorithm ([15, 7]).

The Hausdorff distance definition starts with the one-way minimum distance from a point p to a mesh M:

$$d(p, M) = \min(\|x - p\|_{x \in M}) \tag{5.15}$$

where $\|.\|$ represents the L2 norm. Note that not only vertices of M are used as x in the previous equation, but also all the internal points of edges and faces of M. Now, we can define the one-way (forward) distance from mesh M to a mesh M' as

$$d_f(M, M') = \max_{p \in M} \left(d\left(p, M'\right) \right) \tag{5.16}$$

Again, all internal points of M are used as p in the previous equation. Finally, the symmetric distance is defined as follows:

$$d_s(M, M') = \max(d_f(M, M'), d_f(M', M)) = d_s(M', M) \tag{5.17}$$

This measure is usually not evaluated exactly, but some sampling technique is used. The algorithms described in [15] and [7] sample both meshes uniformly, and then compute the maximum minimum distance between a set of points and a set of triangles using some spatial subdivision technique.

The Metro tool additionally provides the possibility to evaluate the RMS (Root Mean Square) value. This value takes into account the minimum distance to the other surface from each sample point used in the Hausdorff distance computation. Each distance is squared, the squares are averaged, and the square root of the average is given.

Both Hausdorff distance and RMS measure can be applied "frame by frame" to a dynamic mesh, but such approach cannot take into account the temporal properties of the dynamic mesh, i.e. the fact that some directions of the shift cause more disturbance because the mesh is evolving in orthogonal direction. In section 7.4 we will test these measures, using the average Hausdorff distance over the length of the animation, and the average RMS distance over the length of the animation.

5.5 Triangle difference

The Triangle Difference (TD error) has been proposed by Zadražil in [73]. The idea is to work with triangles instead of points, and evaluate the difference of a local, relative property of the triangle (it's area) rather than working with absolute properties such as 3D coordinates of points.

The TD error measure is only applicable when measuring difference between surfaces of known point to point correspondence, i.e. on compressed meshes. The

algorithm exploits the knowledge of this correspondence, which also implies the correspondence of triangles. The overall error is expressed as

$$TD_{error} = \frac{\sum_{i=1..T} \left(A(t_i^{original}) - A(t_i^{distorted}) \right)^2}{TFd} \tag{5.18}$$

where $A(t)$ stands for area of triangle t, T stands for the total number of triangles and d is the length of the main diagonal of the first frame of the original animation.

Zadražil not only gives this new measure, but also shows its usability by evaluating a correlation of this measure to the results of a set of subjective tests. The correlation is compared with the results of other error measures (ribbon, KG error, PSNR) and it is concluded that the TD error shows highest correlation.

The author explains the better correlation by the fact that TD error is the first measure that takes more than one vertex into account, and thus a smooth distortion is preferred when compared with a random distortion of the same amount. In other words, the algorithm will find no error when the three vertices of a triangle are shifted in the same direction by the same amount, however if the directions will be different, then the error will be large.

There are still some drawbacks of this method. First, it is unable to detect global transformations such as translation and rotation of the whole object. Second, the method can be lead astray by some special shape changes of the triangles, which result in a zero change in triangle area, however this can be considered a singular case unlikely to appear in reality. Finally, the method does not incorporate a temporal error in any way.

5.6 Error measures summary

We have shown that there is a quite large amount of error measures used to evaluate the performance of dynamic mesh compression algorithms. Still, we can easily identify serious problems with all the measures, which raises a question about the value of results of all performance tests performed to date.

We have identified following objectives for ideal error measure construction:

1. **Shift invariance**, i.e. when both the original and the distorted animations are translated by the same amount, then the distortion should be evaluated as identical as in the non translated case. This invariance is required because we have no knowledge about the position of the viewer, i.e. the probability of viewing the sequence from a shifted position is equal to viewing from any other possible position.

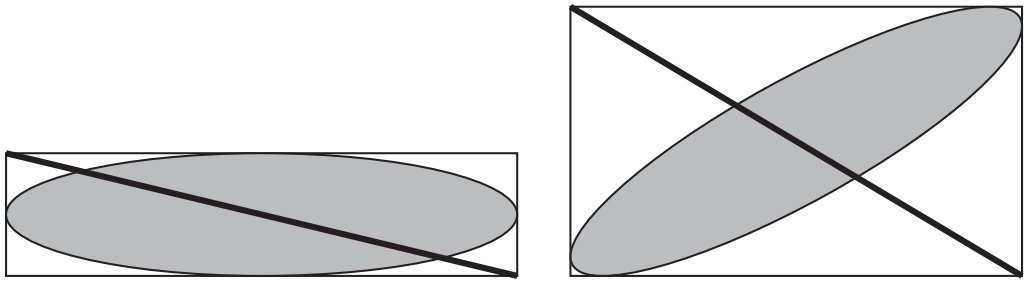

Figure 5.4: Difference between the length of main diagonal of a bounding box. The object (gray) is rotated, which results in different normalization constant (length of the bounding box diagonal).

2. **Rotation invariance**, i.e. when both the original and the distorted animations are rotated by the same constant angle, then the distortion should be evaluated as identical as in the non rotated case.

3. **Scale invariance**, i.e. when both the original and the distorted animations are scaled by the same amount, then the distortion should be evaluated as identical as in the non scaled case.

4. **Animation duration invariance**, i.e. when the distortion has constant character, then the error value should not increase with increasing number of frames.

5. **Detection of spatial artifacts**

6. **Detection of temporal artifacts**

7. **Overall correlation with subjective quality evaluation**

Most of the methods described satisfy the shift invariance and the scale invariance condition, however the rotation invariance is not satisfied by the methods that process the coordinates separately (Ribbon measure) and also by the methods that use first frame bounding box main diagonal for normalization (triangle difference), because the length of the diagonal may change with rotation of the object (see figure 5.4).

None of the methods attempts to address even the most obvious spatial artifacts (random shift of vertices versus smooth shift of the same amount). The temporal artifacts, such as frame to frame shaking of the shapes, are not only undetected (most of the papers about dynamic mesh compression evaluate error in a frame-by-frame fashion, and present the dependency of the error on frame number), but even favored by some measurement methods (Ribbon measure).

The overall correlation with subjective quality evaluation seems to be a crucial property of any error measure, and still the only work which actually involves

subjective testing has been performed only recently by Zadražil under our guidance.

In chapter 7 we will describe new tools for evaluating the difference between two dynamic meshes, which overcome some of the drawbacks of the methods presented here.

Chapter 6

Testing data

In the following chapters we will propose several methods for dynamic mesh compression and simplification. We have tested these methods on multiple datasets of different nature, length and complexity. In this chapter, we will give some details about the datasets used for the testing. We will give a brief description of each dataset, a table of basic parameters and a wireframe rendered image for each dataset.

6.1 Chicken sequence

This dataset has became a de-facto standard in dynamic mesh compression testing algorithms. It is an artificial sequence with low polygon count and very irregular tessellation. The chicken character was created by Andrew Glassner, Tom McClure, Scott Benza and Mark Van Langeveld. This short sequence of connectivity and vertex position data is distributed solely for the purpose of comparison of geometry compression techniques.

	Vertices	Triangles	Frames	Length [s]	Original size [MB]
Chicken	3030	5664	400	16	9,25
Dance	7061	14118	201	8,04	10,83
Dolphin	6179	12278	101	4,04	4,76
Cowheavy	2904	5804	204	8,16	4,52
Human jump	15830	31660	222	8,88	26,81
Cloth	9987	19494	200	8	15,24
Walk	35626	67571	187	7,48	50,83
Snake	9179	18354	134	5,36	9,38

Table 6.1: Basic parameters of testing sequences.

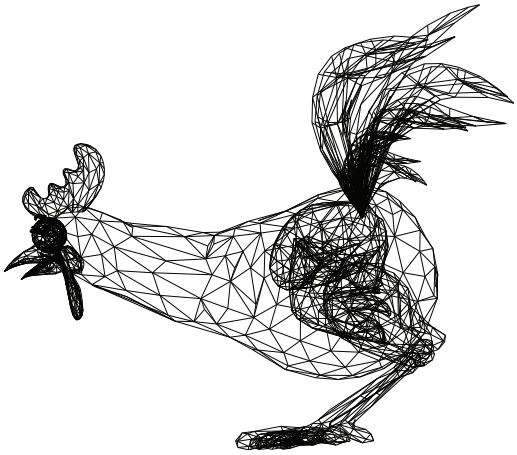

Figure 6.1: Chicken sequence example.

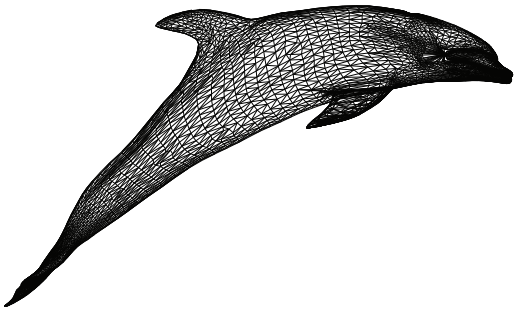

Figure 6.2: Dolphin sequence example.

6.2 Dolphin

This dataset has been created from a static mesh, and its movement is very regularly sinusoidal. Most of the triangles are very sharp, but overall the tessellation is uniform.

6.3 Cow

This dataset has also been created from a static mesh, this time using a simulation of physical behavior of a flexible material. Therefore the movement is not regular, however it is very realistic. The mesh has low polygon count and irregular tessellation.

6.4 Dance

The dance dataset has been created using a bone system. The movement is quite smooth, and the tessellation is quite regular, however its density is much higher

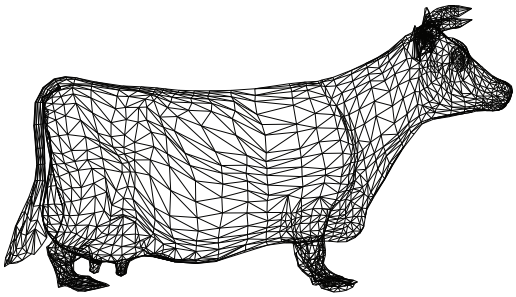

Figure 6.3: Cow sequence example.

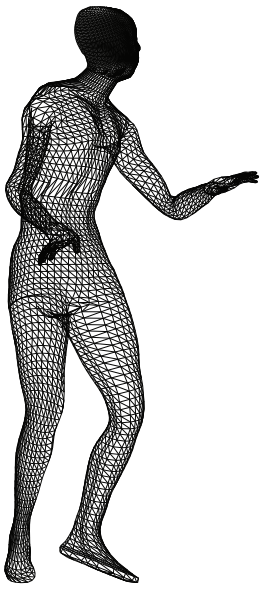

Figure 6.4: Dance sequence example.

in the head area than in the rest of the body.

6.5 Human jump (human)

The human jump sequence has been obtained by reconstruction from real wold data[52, 6]. The character of the movement is very natural, however the mesh, although quite dense, is of very little detail. The tessellation is almost perfectly regular.

6.6 Falling cloth

The Falling cloth sequence has been created using 3D Studio Max (www.autodesk.com) to simulate a collision between a falling ball, a piece of cloth, a rigid torus and

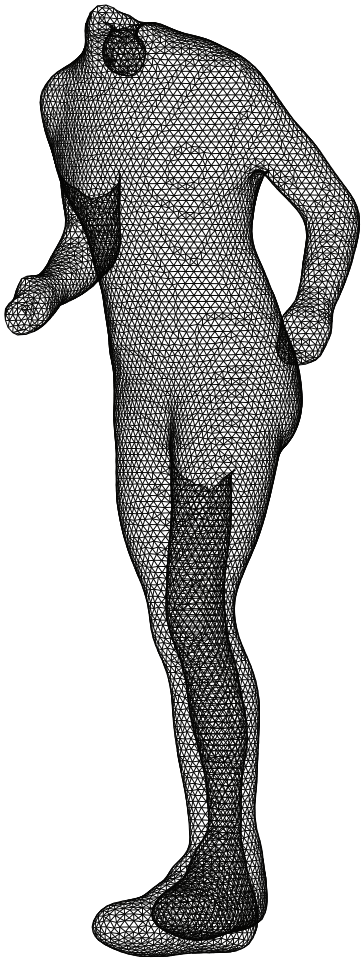

Figure 6.5: Human jump sequence example.

a rigid desk. There are two interesting features of this animation: first, there are parts that are completely rigid, i.e. the torus and the bottom. Second, this animation is not created using a bone system, and therefore representing it traditionally with skinning techniques is not straightforward (although with some difficulty possible, for details see [33]).

6.7 Walk

The Walk sequence is an artificial sequence created using the software Poser (http://graphics.smithmicro.com/go/poser). It is a highly detailed sequence of a walking person. With its vertex count of 35626, it is the highest detail sequence used in our experiments. Some compression methods, such as eigenshape based PCA, are having difficulties processing this sequence, due to the fact that a matrix of size $V \times V$ is required during the processing.

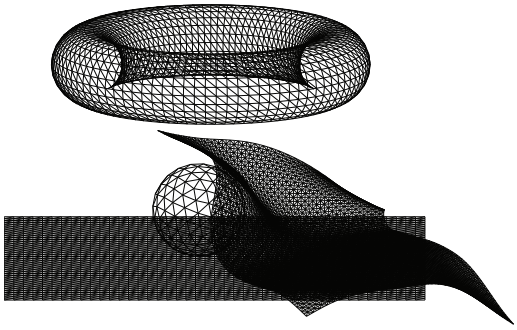

Figure 6.6: Falling cloth example.

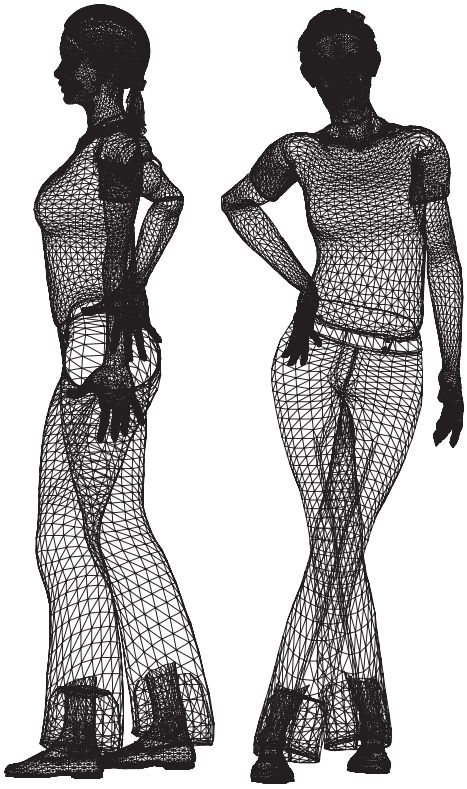

Figure 6.7: Walk sequence example.

6.8 Snake

The Snake sequence is a very common dataset used to test dynamic mesh compression algorithms. It is a relatively low resolution sequence, and the subsequent frames usually show quite significant difference, i.e. the movement captured by the sequence is very fast.

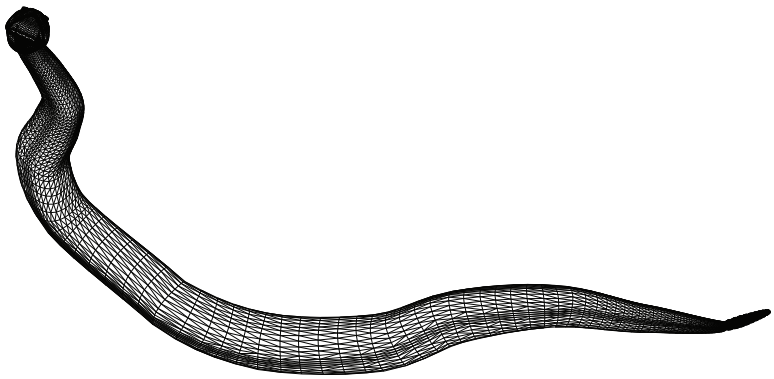

Figure 6.8: Snake sequence example.

Chapter 7

Proposed error measures

Inspired by the obvious insufficiency of the current error measures described in chapter 5 we have constructed two error measures targeted specifically on the case of dynamic meshes. The first method generalizes the concept of Hausdorff distance and allows comparison of dynamic meshes of different connectivity (i.e. applicable for the simplification algorithms). The second algorithm has been constructed to identify some of the spatial and temporal artifacts which are disturbing for human vision.

7.1 4D tetrahedral mesh representation

In our work [69] we have proposed to use a different representation of a dynamic triangular mesh of constant connectivity. We have suggested representing the entire animation by a single static tetrahedral mesh in a four-dimensional space, where the fourth dimension represents time.

The conversion to such representation is performed by converting each triangle in two subsequent time steps into three tetrahedra that partially cover a space-time prism formed by the triangle. Note that the sides of this prism are non-planar, and therefore cannot be represented exactly. However, if we insert a diagonal into each side of such prism, and preserve the diagonal direction when processing the neighboring prisms, then we obtain a consistent mesh without holes. In order to keep the diagonal direction, we have proposed using following scheme for each triangle prism:

1. create a tetrahedron from the three vertices of the base of the prism, and a vertex from the top with the largest index

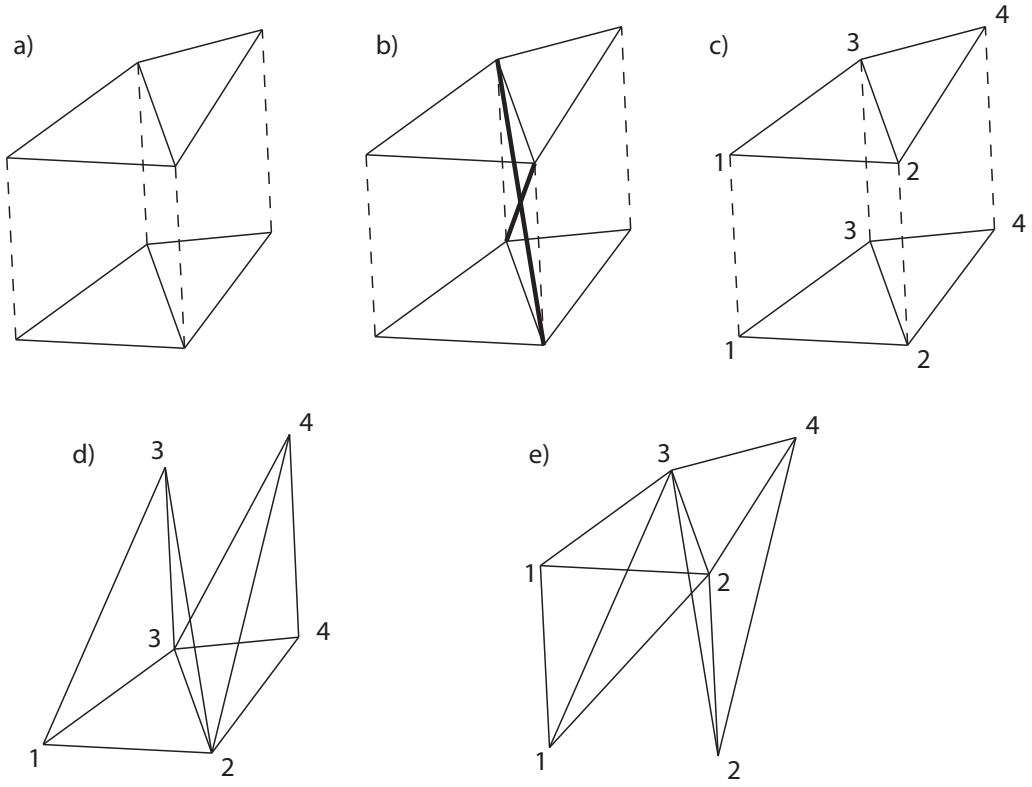

Figure 7.1: a: Two moving triangles as two 4D prisms, bottom of each prism is the triangle in time t, top of each prism is the triangle in time $t + 1$. b: Two possible diagonals of a common side. c: vertex indices. d: bottom tetrahedra. e: top tetrahedra. Note that the selected diagonal is consistent.

2. create a tetrahedron from the three vertices of the top of the prism, and a vertex from the bottom with the smallest index

3. create a tetrahedron from the two vertices of the bottom of the prism with the two larger indices, and two vertices from the top with the two smaller indices

Using this procedure we get one consistent 4D mesh. The last question that needs to be addressed is what units should be used for the spatial and temporal dimensions of the 4D space. In our approach, we have followed the idea that an equal shift in each direction should cause an equal disturbance in the mesh. In order to set the units, we have first decided to use the universal relative spatial units, and then to find a constant α, which will relate the absolute time unit of one second to the relative spatial unit. We are aware that this constant may depend on the dynamic properties of the data, on the size of the viewing screen, but no apriori information about the viewing angle is usually given, and so we must use a general solution.

The relative spatial unit is defined as a distance in a model divided by the body diagonal of the model. By representing all the models in such units, we can expect similar spatial behavior and distribution in all cases. In order to relate this unit to the absolute temporal units by relating constant α we will have to perform subjective testing, but for the time of being we can do following considerations:

1. time span of 1/100s is almost unrecognizable for a human observer, while spatial shifts of 10% is on the limit of acceptability, therefore we expect α to be smaller than $0.1/0.01 = 10$

2. time spans of units of seconds are on the limit of acceptability, while spatial shift of 0.1% is almost unrecognizable, therefore we expect α to be larger than $0.001/1 = 0.001$

Saying that, we can guess the value of the relating coefficient to be about 0.1, i.e. time span of 100ms is equal to spatial shift of 1%.

We can also see the problem of relating time and space as a problem of finding the expected speed of the movement in the animation. We have measured the relative speed of vertices in a number of dynamic meshes which contained human-observable animations, and we have found out that the local speed is usually about 0.3, i.e. 30% per second, which supports our original guess and gives a better estimate of the relating constant (for this constant a 100ms difference is equal to 3% shift in space).

We can now process the mesh using the tetrahedral mesh processing techniques described above, because we can compute distances in the 4D space. We can also create the separate frames by slicing this tetrahedral mesh by a $t = const.$ plane.

Note that this representation actually raises the size of the mesh. The connectivity can no longer be represented by a single frame triangular connectivity, which can be encoded with $2t$ bits using Edgebreaker. The connectivity of the resulting tetrahedral mesh can be compressed using the cut-border algorithm to 2 bits per tetrahedron, but the number of tetrahedra is equal to $3T(F-1)$, where T is the number of triangles and F is the number of frames used. One of our proposed future research topics is to determine whether this overhead can or cannot be justified by the benefits of this representation of the mesh.

7.1.1 4D metric

We have generalized the idea of computing Hausdorff distance as a measure of difference of two static triangular meshes to the case of dynamic meshes in [69]. The generalization is quite straightforward using the dynamic mesh representation described in previous section.

First, both compared meshes (i.e. the original and the compressed or simplified version) are converted to the representation by a static tetrahedral mesh in 4D. Subsequently, both meshes can be uniformly sampled, from each point a closest point on the other mesh is found using a series of optimized point to tetrahedron distance test. The process is repeated for the backward distance, yielding a guess of distance between the dynamic meshes.

The main drawback of this approach is its computational cost. A naive implementation of the algorithm would find a distance from every test point of the first mesh to every tetrahedron of the other mesh, which would lead to a computational complexity class of $O((VF)^2)$, where V stands for the number of vertices and F stands for the number of frames. Such computation becomes unmanageable even for common complexity animations. Therefore, a number of speedup techniques must be employed. Our implementation uses the following:

- spatial subdivision scheme, which allows that only a limited number of tetrahedra need to be processed when searching for the closest one. We are using uniform four-dimensional grid of cells which covers the original data set in space and time

- in preprocessing stage accurate tests are used to determine which cells incide with tetrahedra. The usual assumption that all the cells which incide with axis aligned bounding box of the tetrahedron also incide with the tetrahedron becomes too inefficient in the 4D case. We are using normal driven approach, which decides about the incidence according to the positions of each cell's vertices with respect to the tangent hyperplane of the tetrahedron. Only those cells, which have vertices above and below the hyperplane, can be incident with the tetrahedron. This approach leads to significant reduction of the number of tetrahedra in each cell, yielding an overall speedup of about 30%

- a highly optimized point-to-tetrahedron distance evaluation routine is used to further speed the process up. We're first evaluating distance to the edges of the tetrahedron. These distances are only relevant when the tested point orthogonally projects itself onto the given edge, which is in practice a rare case. It can be also shown that a point can be orthogonally projected onto a face (of the tetrahedron) only if it projects at least onto two of its edges. We save the information about the result of projection onto edges, and in most cases we can skip the distance to a face computation completely.

Our implementation of the Hausdorff distance can also be used to map the found minimal distance to vertex colors, thus showing the distribution of the error, which may be useful when considering various simplification criteria.

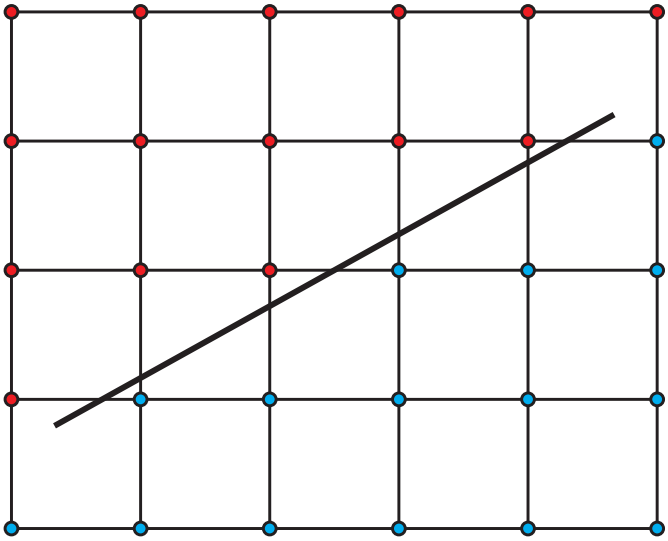

Figure 7.2: Grid point evaluation. The actual algorithm works in 4D. The grid cells that have all corners of equal evaluation cannot be intersected by the simplex.

We have tested the method on the available data, comparing our results to applying Hausdorff distance in a frame-by-frame fashion. When the distances to the other mesh are shown as vertex colors, then our distance algorithm produces results that are visibly different from the case when only two corresponding frames are compared using the standard Hausdorff distance metric.

In our work [21] we have also suggested using this metric to compare and classify animations for artificial intelligence applications. The idea is that a robot can acquire a surface representation of a process that takes place in its vicinity. Subsequently, it can be compared to a set of known "actions", classified as one of them, and the robot can decide based on the classification. The advantage of such approach is that the tessellation of the sample acquired by the robot and the one of the database item can be very different, and still a correct classification can be obtained. Also, the robot is given better information, because the animation in the database may contain also some future development of the process.

We have tested this idea on a dataset of a human jump, which contains two non-identical sequences of a human figure jumping. The mesh contains about 15000 triangles, and we have evaluated the distances from one jump sequence to the other (50 frames) and a sequence of human jump to a different part of the dataset, where the figure has been walking. The distribution of the sampled distances is shown in the figure 7.3. When summed up, it can be used to conclude that the distance from one jump sequence to the other is significantly lower than the distance from jump sequence to a walk sequence.

Despite the optimizations, it is currently only possible to compare the meshes offline, due to computational expenses.

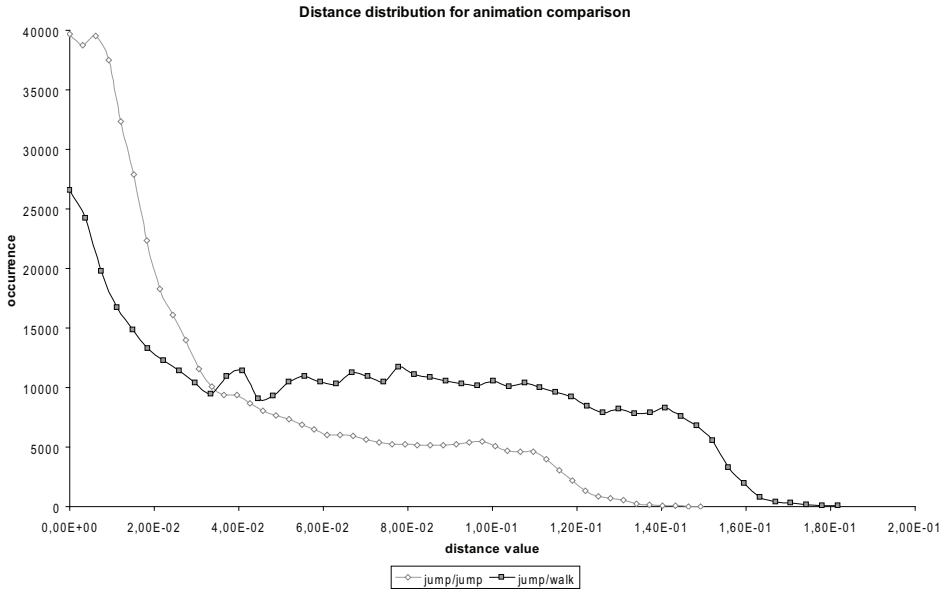

Figure 7.3: Animation comparison results.

7.2 Error vectors measure

The key idea behind the Error Vectors (EV error) measure is that the difference we want to evaluate is not the absolute difference between vertex positions, but rather local variance of these differences. If the local variance of differences is low, then the perceived error should be lower than in a case when the local variance of differences is high.

The method makes use of vertex to vertex correspondence, and therefore is only applicable in cases when connectivity is unchanged from original to compressed version. The method works in two steps:

1. evaluate the difference at each vertex. The difference is evaluated in every frame, and is preserved as a 3D error vector, i.e. both its direction and amount will be known in the next step

2. traverse the mesh and evaluate the local variance of the error vectors.

During the second step a local average of the error vectors is found for each vertex location. The average is computed within a topological neighborhood of a vertex, and in order to incorporate the temporal error also error vectors from given number of previous and following frames are incorporated into the average. The contribution of a vertex to the average error is then computed as an absolute value of the difference between the error vector associated with the vertex, and

the average error vector computed across its neighborhood. More formally, we can define an error vector associated with vertex v in frame f as:

$$ev(v,f) = x_{v,f}^{original} - x_{v,f}^{distorted} \tag{7.1}$$

Subsequently, a local average around vertex v in frame f is computed as:

$$la(v,f) = \frac{\sum_{lf=max(1,f-w)}^{min(F,f+w)} \left(\sum_{i \in neighbors(v)} ev(i,lf) \right)}{\|neighbors(v)\| * (min(F,f+w) - max(1,f-w))} \tag{7.2}$$

where w is the size of a temporal window. Finally, the error is computed as:

$$EV_{error} = \frac{1}{VF} \sum_{v=1}^{V} \sum_{f=1}^{F} \|ev(v,f) - la(v,f)\| \tag{7.3}$$

This method has been tested in the work of Zadražil[73], who has performed subjective testing and evaluated correlation between the subjective results and various error measures. The overall result of his experiments is that both EV error measure and TD error measure show in many cases higher correlation with the subjective evaluation than the previously used methods such as KG error or Ribbon error. The correlation coefficients found by [73] are summarized in table 7.1.

model	prism	dance	chicken
MSE	0.61	0.46	0.72
Hausdorff distance	0.70	0.35	0.72
KG	0.73	0.36	0.71
EV	-0.02	0.53	0.09
WEV	-0.02	0.54	0.82
TD	-0.01	0.60	0.59
Ribbon	0.75	0.37	0.72

Table 7.1: Correlation of objective evaluation methods and the mean opinion score in experiments by Zadražil[73].

7.3 Spatio-temporal edge difference

Our ultimate measure is the spatio-temporal edge difference[68] (STED). This method achieves to our knowledge the best correlation with subjective difference evaluation. It is based on ideas of TD error and EV error, however it has been tweaked in order to maximize the correlation with real comparison results. The key ideas are the following:

- It is possible to eliminate some of the singular cases present in the TD measure by using edge as the elementary item upon which the measure is performed. This choice allows us to measure a property independent of absolute position (edge length), while there does not exist a possibility that the endpoints of the edge are moved relative to each other, and the property remains unchanged (this can easily happen in the case of triangle area)
- Areas of the mesh that are more densely sampled are likely to contain fine geometric detail, and thus are more sensitive to distortion. This fact can be exploited by using relative edge difference rather than its absolute length.
- Temporal artifacts and distortions can be included in the same framework by considering virtual edges connecting temporally subsequent positions of a vertex.
- We can use the idea of significance of local changes in error rather than absolute value of the error. In order to do so, we express a standard deviation of the edge difference around each vertex, and add these values to obtain the overall error.

7.3.1 Spatial error

Formally the error can be derived as follows: first, we denote length of an edge connecting vertices v_i and v_j (i.e. of the edge e_{ij}) in frame f as follows:

$$el(i,j,f) = el(e_{ij},f) = \|v_{i,f} - v_{j,f}\| \tag{7.4}$$

Note that the *el* property applies separately on original and distorted meshes. Now, we can define relative edge difference as a property of a spatial edge connecting vertices v_i and v_j (i.e. of the edge e_{ij}) in frame f as follows:

$$ed(i,j,f) = ed(e_{ij},f) = \|\frac{el(e_{ij},f)^{original} - el(e_{ij},f)^{distorted}}{el(e_{ij},f)^{original}}\| \tag{7.5}$$

Subsequently we assign to each vertex the local standard deviation across the edges of given topological distance from the vertex. The user specifies a topological distance d. For each vertex v a set of vertices of topological distance lower or equal to d is found and denoted $NV(v,d)$. Finally, a set $NE(v)$ of edges incident with any of the vertices of each $NV(v,d)$ is found (for illustration see figure 7.4).

Now, we will compute the average relative edge difference around the vertex. Due to the fact that the surroundings of a vertex may contain edges of very varying length, we compute a weighted average, where the weight of an edge is determined by its original length:

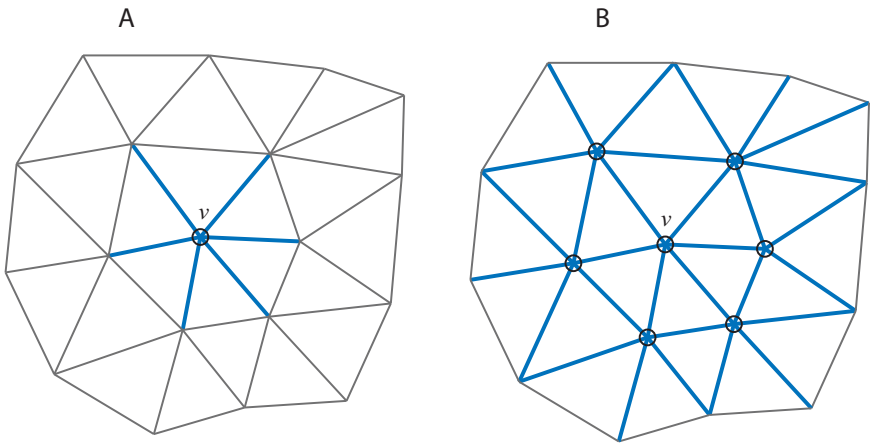

Figure 7.4: Neighborhoods of vertex v. The black circles denote items of $NV(v)$, the thick blue edges denote items of $NE(v)$. The A case shows the situation when allowed spatial distance is 0, the B case shows the situation when allowed spatial distance is 1.

$$avged(v,f) = \frac{\sum_{e \in NE(v)} ed(e,f) el(e,f)^{original}}{\sum_{e \in NE(v)} el(e,f)^{original}} \tag{7.6}$$

Now, we can express the local deviation around a vertex v in frame f. Note that we are again using weighting by edge length:

$$dev(v,f) = \sigma(ed(NE(v),f)) = \sqrt{\frac{\sum_{e \in NE(v)} \left((ed(e,f) - avged(v,f))^2 \, el(e,f)^{original} \right)}{\sum_{e \in NE(v)} el(e,f)^{original}}} \tag{7.7}$$

Finally, we have to average the value over all the vertices and all the frames. Note that the value of (7.5) has a character of ratio, i.e. it is scale independent, and therefore also the values of (7.7) and (7.8) are scale independent.

$$STED_{spatial} = \frac{\sum_{v} \sum_{f} dev(v,f)}{VF} \tag{7.8}$$

7.3.2 Temporal error

The reasoning behind the derivation of temporal error is coherent with the derivation of the spatial error, only this time we will consider virtual edges connecting vertices in subsequent frames. The idea of computing relative edge length, which has been used in the spatial error case to increase sensitivity in high precision

areas, will be used again, this time to increase sensitivity in areas of very slow motion.

This feature has been introduced based on the behavior of the falling cloth sequence, where some compression methods introduce errors to the static parts of the scene, i.e. the static torus starts to move slightly. This is very noticeable, and therefore such artifact should be detected by an error measure.

However, we cannot use the exact equivalent of (7.5). The reason is that in some animations, such as the dance, it is possible that some vertices become almost static for short periods of time (usually the legs of the dancer), however the time period over which the movement becomes static is too short for human observers to start detecting disturbing artifacts.

In order to evaluate the steadiness of the movement of a vertex v in time step f, we compute its average speed within a temporal window of radius w around the frame f. First, we define temporal edge length as follows:

$$ld = \max_{u \in V, v \in V} \|u - v\| \tag{7.9}$$

$$tel(v, f) = \sqrt{(\frac{v_x^f - v_x^{f-1}}{ld})^2 + (\frac{v_y^f - v_y^{f-1}}{ld})^2 + (\frac{v_z^f - v_z^{f-1}}{ld})^2 + fd^2} \tag{7.10}$$

Note that this value is not defined in frame 0. The fd (frame distance) term is used to determine the temporal distance between subsequent frames of the animation. It's main purpose in the computation is to avoid infinity or near to infinity result for the case of completely static vertices.

Also note that the character of (7.10), i.e. the presence of square root and a constant fd, introduces a need for an early spatial normalization. However, we want to avoid rotation dependence, and thus we cannot use the main diagonal of the first frame bounding box. Instead, we use the distance ld of the two most distant points in the first frame of the original sequence. This way, we achieve a more robust relative distance, which is rotation invariant. The computation of ld in (7.9) is not necessarily quadratic and can be sped up to almost linear (for details see [56]).

The average spatio-temporal speed of a vertex v around frame f is defined as.

$$s(v, f) = \frac{\sum_{lf=max(f-w,1)}^{min(f+w,F)} tel(v, lf)^{orig}}{min(f + w, F) - max(f - w, 1)} \tag{7.11}$$

Note that due to using the term fd in (7.10) this value never becomes zero. Now we can define the relative temporal edge difference as follows:

$$rted(v,f) = \frac{\|tel(v,f)^{original} - tel(v,f)^{distorted}\|}{s(v,f)} \tag{7.12}$$

Finally, the overall temporal error is defined as average over all the vertices and all the frames:

$$STED_{temporal} = \frac{\sum_f \sum_v rted(v,f)}{V(F-1)} \tag{7.13}$$

7.3.3 Overall error and its parameters

We define the overall error as a hypotenuse of weighted spatial and temporal error:

$$STED_{error} = \sqrt{STED_{spatial}^2 + (c.STED_{temporal})^2} \tag{7.14}$$

There are several constants that we have used so far in the definition without discussion of their value. These are:

- topological distance d used to compute the vertex neighborhood $NV(v,d)$,
- temporal distance value between subsequent frames fd used in (7.10),
- temporal window size w used in (7.11),
- the relating constant c used in (7.14).

The actual values of these constants have been determined using results of subjective testing, achieving the best possible correlation of (7.14) with the subjective testing results.

7.4 Performed subjective testing

We have performed series of subjective tests in order to approve/disapprove the relation of various measures with subjective evaluation of distortion amount. Our testing is based on the MUSHRA[40] technique, which allows quickly obtaining a large corpus of data by allowing the users to freely compare multiple stimuli.

We have prepared multiple distorted versions of some of the available datasets, using various distortions. The distortions were introduced by:

- gaussian noise added to vertex positions (noises with various deviations have been used), random values for each vertex and each frame,
- gaussian noise added to vertex positions, random values for each vertex,

- gaussian noise added to vertex positions, random values for each frame (i.e. each frame has been shifted by a random amount),
- value $A\sin(\omega x)$ added to each coordinate of each vertex, i.e. a smooth distortion of amplitude A and frequency ω,
- value $A\sin(\omega f)$ added to each coordinate of each vertex, i.e. a smooth temporal shifting of the whole mesh,
- result of compression scheme which will be presented in section 8.1 using a coarse quantization,
- result of compression scheme which will be presented in section 8.1 using a low number of basis trajectories.

This way we have achieved distortions of varying nature. We have prepared a set of 9 distorted versions of the datasets chicken, dance and cloth, giving together 30 datasets (including the originals).

In each test, a group of subjective evaluators has been shown a set of 10 versions of a dataset, including the original, which has been labeled as such. The evaluation took place in a computer laboratory with a projection screen and nine computers. The projection screen has been playing the original version of the animation, while each computer has played back one of the nine distorted versions. The evaluators have been briefly introduced to the problem of dynamic mesh compression, and then they have been asked to perform an evaluation with following instructions (see the original instruction sheet in appendix A, in Czech language only):

1. Have a look at all the animations, ideally in their full length. Focus on possible artifacts and differences with respect to the original.
2. Find an animation with worst degradation and assign it a mark 10.
3. Have a look again at all the animations, and assign marks 0-10 according to how acceptable is the distortion. Try to keep the marks proportional, i.e. double the mark value means two times less acceptable distortion.
4. Give a mark 0 only to such dataset where you cannot find any difference with respect to the original
5. Do not consult the marks with each other, nor show each other the nature or location of problematic parts. Try to work on your own.

We have had overall cca 100 voluntary subjective evaluators, students of third and fourth year of computer science. No subject has been through more than two tests. There were four females, the rest of subjects were males.

dataset	number of tests	value	#1	#2	#3	#4	#5	#6	#7	#8	#9
chicken	43	mean	7,77	9,84	1,63	7,70	2,16	6,30	6,64	0,95	1,79
		deviation	1,73	0,43	1,63	1,88	2,09	2,19	1,97	1,17	1,67
dance	43	mean	8,93	1,95	3,58	6,74	9,56	1,88	2,02	7,95	6,35
		deviation	0,96	1,76	1,80	1,57	0,80	1,88	1,65	1,60	1,89
cloth	37	mean	9,24	7,86	2,22	1,32	2,73	8,78	6,00	2,46	2,32
		deviation	1,19	1,34	1,77	1,23	1,59	1,47	1,49	1,94	1,63

Table 7.2: Results of subjective testing.

The results of the testing are shown in table 7.2, we are only giving the mean marks (from now on will be denoted MOS, Mean Opinion Score) and the standard deviation of the marks. The actual datasets used as #1 - #9 can be found on the accompanying DVD along with details of the introduced distortion, however such details are irrelevant for following considerations, because we are trying to reach a measure that does not rely on the knowledge of the character of the error.

Error measures evaluation

For every distorted animation we can evaluate any of the error metrics presented so far. Subsequently, we can observe the match of the computed error values, and the results of subjective testing. Note that we don't want to achieve matching values, as we don't know anything about the magnitude of the error. What we do want to achieve is the correlation between the computed values and the subjective testing results.

As a measure of correlation we use the Pearson correlation coefficient, which is defined for two variables X and Y as:

$$\rho_{X,Y} = \frac{E[(X - E(X))(Y - E(Y))]}{\sigma_X \sigma_Y} \tag{7.15}$$

The coefficient takes values from the interval $< -1, 1 >$. If $\rho_{X,Y} = 1$, then there is a linear dependence between the variables, coefficient value of -1 shows an inverse linear relation, and zero value shows that there is no relation between the values of X and Y.

The correlation coefficient can be estimated from a limited sample of the values by following equation:

$$\rho_{X,Y} = \frac{\sum(x - \overline{x})(y - \overline{y})}{\sqrt{\sum(x - \overline{x})^2 \sum(y - \overline{y})^2}} \tag{7.16}$$

where $\overline{x}$ denotes average value. This way, we have evaluated the correlation of existing metrics with the subjective testing. The results are summarized in table 7.3, the figure 7.5 shows an example of correlation of the KG error with the results of subjective testing.

The results have confirmed the conclusions drawn by Zadražil. All the existing error measures provide only limited, or even negative correlation with the results of a subjective testing which involves multiple kinds of distortion. The error vectors approach provides some improvement, but the correlation coefficient values are still relatively small. The best results so far are given by the TD error measure, which consistently correlates with the MOS with coefficient higher than 0.6, in the dance sequence case it reaches up to 0.89. Note that we have used different datasets and different kinds of distortion than in the original experiment by Zadražil.

STED parameters

In this section we will estimate the parameter values for the STED measure. The main objective is to achieve as high correlation with MOS as possible.

First, we will only consider the spatial part of the STED value, i.e. result of (7.8). The equation requires setting only a single parameter, the width of a topological neighborhood over which the deviation (7.7) is computed. For the experiment, we have considered only the spatial error, and we have achieved best results with topological width 1 for the neighboring vertices, i.e. edge neighborhood of shape equivalent to the one depicted in figure 7.4B. Figure 7.6 shows the measured dependence of the correlation coefficient with the topological neighborhood width, table 7.4 gives the measured correlation values for neighborhood of width 1.

Now, we will focus on setting the parameter values for the temporal error. We have to set three parameters - the window size for computation of vertex speed, the temporal distance parameter used in (7.10) and the overall weights relating spatial and temporal error in (7.14).

Through a series of experiments we have found values for these constants so that the overall error expressed by (7.14) correlates to the subjective testing results as closely as possible. We are showing graphs that describe the development of the correlations around the values that we are using.

The best results have been obtained for speed window of width $w = 5$ and temporal distance coefficient $dt = 0.0003$. The weighting coefficient is easiest to optimize, and the highest correlation has been obtained for value $c = 9,144 \times 10^{-5}$. Using these constants, we can evaluate the error values shown in table 7.5 along with the correlations with MOS.

STED conclusions

Using the proposed STED measure, we are able to robustly determine an error introduced by a compression. We can detect certain kinds of spatial and temporal artifacts, namely the difference between regular and random vertex shifts.

Chicken sequence measure correlation

	#1	#2	#3	#4	#5	#6	#7	#8	#9	**correlation**
MOS	7,767	9,837	1,628	7,698	2,163	6,302	6,640	0,953	1,791	
KG error	2,188	1,863	1,856	1,850	2,946	1,795	0,931	3,950	1,832	**-0,531**
Ribbon mean $x10^4$	3,384	2,563	2,188	3,127	3,540	1,666	1,281	7,559	3,447	**-0,491**
Ribbon peak $x10^2$	0,449	0,208	0,909	0,157	1,113	0,498	0,100	0,119	0,056	**-0,326**
Hausdorff $x10^2$	1,036	0,939	1,354	0,957	2,246	0,673	0,482	0,879	0,606	**-0,324**
RMS $x10^3$	1,697	1,571	1,893	1,561	3,105	0,909	0,848	4,150	2,132	**-0,688**
TD error $x10^6$	1,116	0,840	0,114	0,831	0,266	0,927	0,202	0,000	0,280	**0,814**
WEV 1 $x10^3$	0,490	3,441	0,110	0,000	0,141	2,231	1,719	0,000	0,103	**0,685**
WEV 3 $x10^3$	1,558	4,396	0,325	0,006	0,436	2,851	2,195	0,000	0,299	**0,730**
WEV 5 $x10^3$	2,145	4,521	0,424	0,012	0,585	2,927	2,258	0,000	0,405	**0,746**

Dance sequence measure correlation

	#1	#2	#3	#4	#5	#6	#7	#8	#9	**correlation**
MOS	8,930	1,953	3,581	6,744	9,558	1,884	2,023	7,953	6,349	
KG error	0,481	6,529	0,225	0,482	0,614	0,495	2,726	0,457	0,282	**-0,539**
Ribbon mean $x10^3$	0,528	5,861	0,201	0,430	0,644	0,444	2,544	0,460	0,285	**-0,529**
Ribbon peak $x10^2$	0,277	3,117	0,159	0,334	0,611	0,671	5,234	0,378	0,201	**-0,599**
Hausdorff $x10^2$	0,280	1,561	0,134	0,282	0,402	0,417	2,900	0,378	0,224	**-0,565**
RMS $x10^3$	0,505	8,556	0,243	0,509	0,707	0,713	4,119	0,570	0,353	**-0,570**
TD error $x10^{12}$	1,382	0,000	0,300	1,381	1,783	0,063	0,733	1,899	0,734	**0,894**
WEV 1 $x10^4$	0,000	0,000	4,185	8,965	2,124	0,197	0,499	0,000	0,000	**0,134**
WEV 3 $x10^3$	0,003	0,000	0,529	1,134	0,726	0,057	0,176	0,003	0,002	**0,262**
WEV 5 $x10^3$	0,006	0,000	0,539	1,154	0,920	0,066	0,218	0,006	0,003	**0,302**

Falling cloth sequence correlation

	#1	#2	#3	#4	#5	#6	#7	#8	#9	**correlation**
MOS	9,243	7,865	2,216	1,324	2,730	8,784	6,000	2,459	2,324	
KG error	0,339	0,258	0,215	0,226	2,022	0,209	0,214	0,652	0,221	**-0,271**
Ribbon mean $x10^4$	1,886	1,337	1,056	0,884	8,188	1,017	0,612	2,505	1,493	**-0,242**
Ribbon peak $x10^2$	0,155	0,287	0,061	0,164	3,309	0,087	0,414	1,167	0,024	**-0,288**
Hausdorff $x10^2$	0,235	0,215	0,045	0,154	2,026	0,110	0,197	0,744	0,045	**-0,259**
RMS $x10^3$	0,379	0,285	0,255	0,191	2,195	0,197	0,131	0,753	0,271	**-0,280**
TD error $x10^3$	2,876	0,292	0,000	0,090	0,333	0,691	0,458	0,110	0,000	**0,687**
WEV 1 $x10^4$	0,000	0,136	0,000	0,012	0,049	1,861	1,127	0,044	0,000	**0,528**
WEV 3 $x10^4$	0,010	0,444	0,000	0,032	0,161	2,354	1,436	0,125	0,000	**0,562**
WEV 5 $x10^4$	0,020	0,609	0,000	0,040	0,218	2,396	1,470	0,156	0,000	**0,581**

Table 7.3: Correlations between objective and subjective error measures.

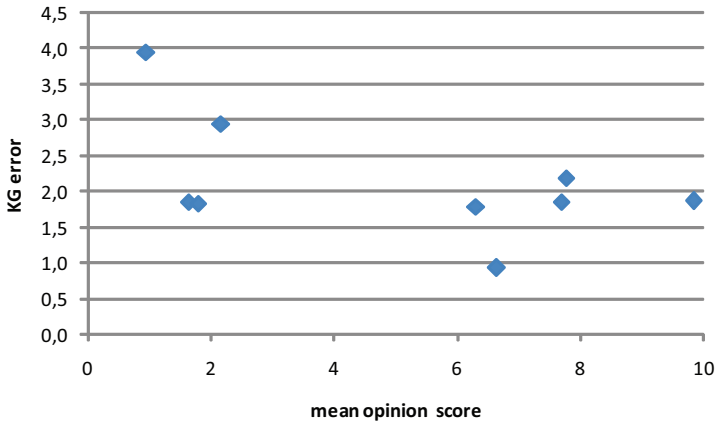

Figure 7.5: Correlation of the KG error with the results of subjective testing.

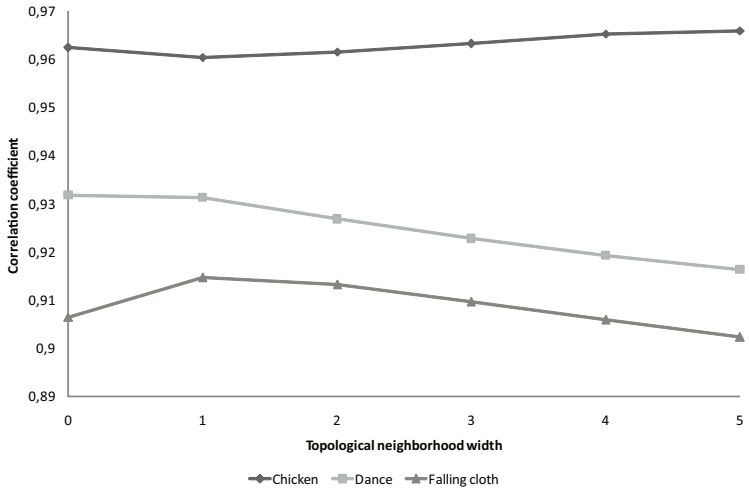

Figure 7.6: Dependence of the correlation coefficient and the topological neighborhood width.

Quite surprisingly, our experiments showed that the influence of temporal error is relatively small, the contribution of the temporal error term in (7.14) is about 12% for the chicken sequence, and less than 2% for the dance and cloth sequences. Nevertheless, the inclusion of the temporal term improves the correlation and may be important in cases when very little spatial error is introduced into the animation.

The method has some drawbacks as well. In some cases, such as rigid translation, rotation and scale of the animation, it is possible that the measure does not report any difference. We consider such eventuality highly unlikely and easy to detect and compensate. It is possible that in some cases of extremely smooth

	#1	#2	#3	#4	#5	#6	#7	#8	#9	correlation
chicken (x10^3)	4,475	5,065	0,638	5,020	0,905	3,003	2,515	0,000	1,242	**0,960**
dance (x10^3)	1,065	0,000	0,501	1,063	1,158	0,128	0,409	0,925	0,575	**0,931**
falling cloth (x10^3)	0,576	0,253	0,000	0,033	0,170	0,383	0,218	0,083	0,000	**0,915**

Table 7.4: Correlation of the spatial STED measure and subjective error measures.

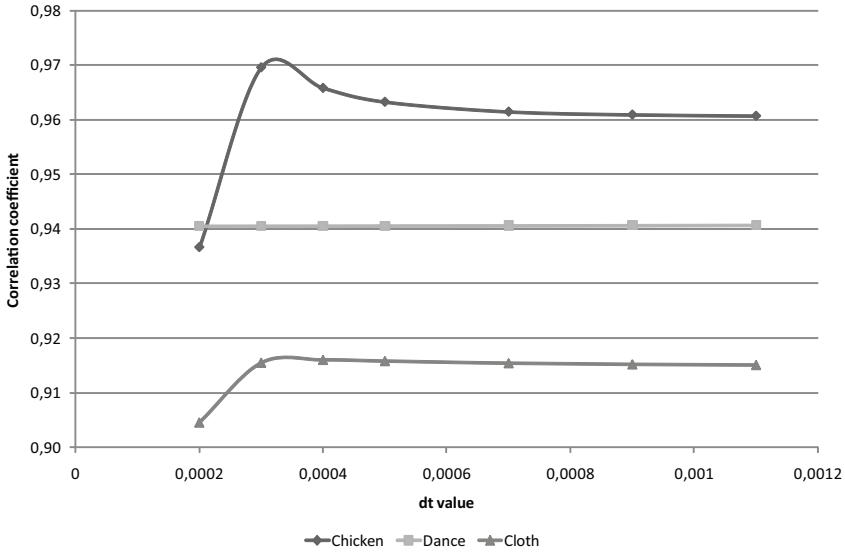

Figure 7.7: Dependence of the correlation coefficient and the temporal distance constant.

deformation, such as global taper or shear of the animation, the error measure might be smaller than expected.

Also, we have identified a class of artifacts, which will not be detected by the measure at all. One can easily imagine that a deformation depicted by figure 7.10 is well visible, while all the edge lengths remain unchanged. This is not only a problem of meshes with border and one edge around which the mesh is rotated. Situation depicted by figure 7.11 is also not detected by the measure. Such a thing may occur in practice, however a larger deformation that leaves edge lengths unchanged is again unlikely.

Despite the drawbacks, during the testing we have received measures that correlate with the subjective testing with coefficient constantly higher than 0.9, and in the case of chicken sequence we have reached to a value of 0.97. Such high correlations are not achievable with any existing error measure technique. Moreover, the STED measure is fast to evaluate, especially compared to Hausdorff distance based measures such as the one used by the Metro tool.

Currently the measure only works for compression evaluation, i.e. for cases

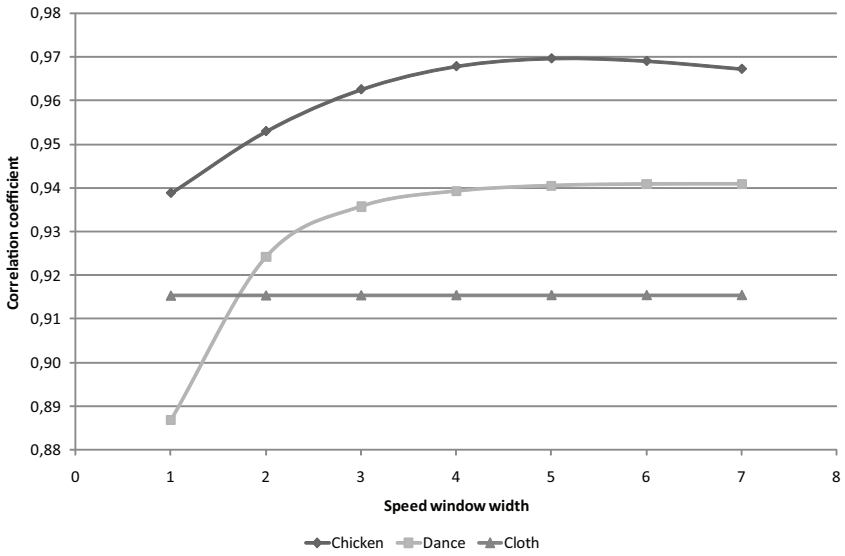

Figure 7.8: Dependence of the correlation coefficient and the speed window width.

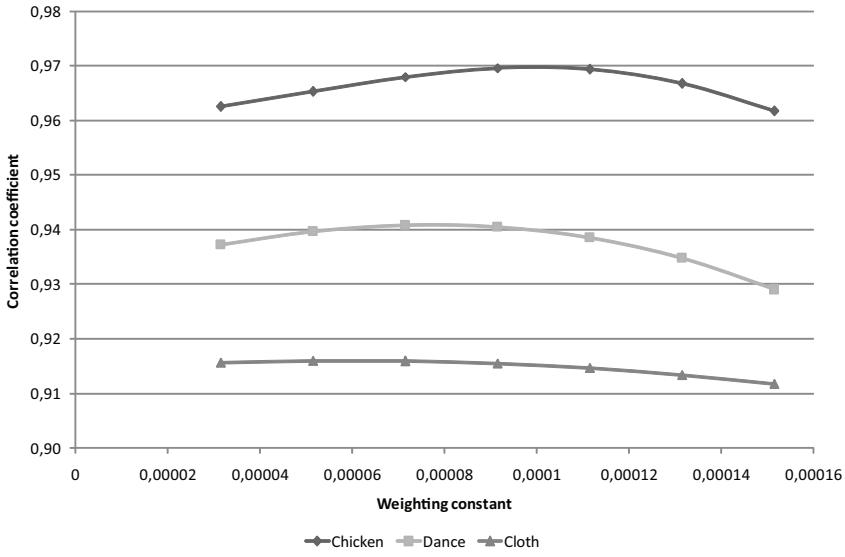

Figure 7.9: Dependence of the correlation coefficient and the weighting coefficient.

when original and distorted sequences share the topology. There is a possibility to extend the measure also to the case of mesh sequences of unequal topology. This could be done by fine resampling of both the meshes, which will convert them into a shared topology, using some strain minimization criterion. There are some techniques used for this purpose, such as the Face2Face([10, 11]), wavelet decomposition([72]) or remeshing([35]), however we have not performed any experiments with such approach so far.

	#1	#2	#3	#4	#5	#6	#7	#8	#9	correlation
chicken (x10^3)	4,475	5,568	0,638	5,020	0,905	3,731	2,578	0,129	1,242	**0,970**
dance (x10^3)	1,065	0,241	0,501	1,063	1,158	0,128	0,409	0,925	0,575	**0,941**
falling cloth (x10^3)	0,576	0,253	0,060	0,033	0,170	0,387	0,218	0,083	0,003	**0,915**

Table 7.5: Correlation of the overall STED measure and subjective error measures.

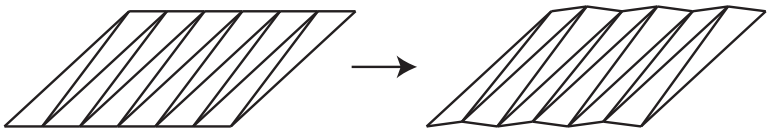

Figure 7.10: The "newspaper" artifact, undetected by STED measure.

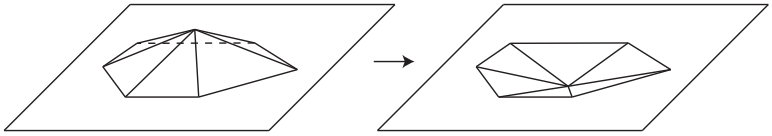

Figure 7.11: The "hill/valley" artifact, undetected by STED measure.

For the experiments in following sections we will use two measures. First, the existing KG error, which will be used to compare our methods with results of competing algorithms, because current literature usually uses this measure to evaluate error. We will also provide the STED values, which will be used to give a better idea about the real life performance of our algorithms, and also to tweak the algorithm parameters to obtain best rate/distortion ratio.

Chapter 8

Proposed compression methods

In this chapter we will describe several methods for dynamic mesh compression that we have published recently. We will start with Connectivity Driven Dynamic Mesh Compression algorithm (Coddyac) [65], which combines Edgebreaker value prediction with PCA, then we will describe a combination of PCA with connectivity driven simplification[67], and then we will present two new predictors that can be used for more accurate estimation of PCA coefficients describing an animation[64].

Finally, we will show a new basis compression scheme[66], which can be combined with any of the PCA based compression algorithms, allowing an even better rate/distortion ratio. The algorithm is based on a geometry-aware prediction and non uniform quantization which regularly distributes the error, thus avoiding unnecessary oversampling of the basis.

We will not give any experimental results in this chapter, all the measurements will be summarized in chapter 9, using KG error measure, and the STED error measure.

8.1 Connectivity driven dynamic mesh compression (Coddyac)

The original idea behind Coddyac is to combine the most powerful tools for exploiting spatial and temporal coherence of the data. Table 8.1 classifies the current methods according to the methods used for spatial and temporal coherence exploitation.

The table shows, that although the existing approaches used efficient approaches for both temporal coherence exploitation (PCA) and spatial coherence exploitation (local prediction), no algorithm has combined these into one method.

Method	Temporal coherence	Spatial coherence
Dynapack	Local predictor	Local predictor
Frame space PCA (Alexa)	none	PCA
PCA+LPC (Karni)	LPC	PCA
Clustering PCA (Sattler)	PCA	Clustering
Octree based (Mueller)	none	octree
Coddyac	PCA	Local predictor

Table 8.1: Comparison of existing methods according to used method for exploiting spatial and temporal coherence.

The core of Coddyac is the usage of principal component analysis to describe the temporal behavior of the animation. Subsequently, a modified parallelogram predictor is used to encode the PCA coefficients while exploiting the fact that neighboring vertices are likely to have similar temporal development.

The Coddyac algorithm consists of following steps:

1. represent trajectory of each vertex by a vector of length $3F$
2. perform PCA over the space of trajectories, obtain a basis
3. encode the basis (send it to the decoder)
4. express each trajectory as a linear combination of the new basis
5. assign a vector of PCA coefficients (feature vector) with each vertex
6. process the topology by the Edgebreaker machine, predict the assigned vectors using parallelogram rule
7. encode the prediction residuals (send them to the decoder)

The decoding procedure is quite simple, the decoder first extracts the used basis of the space of trajectories, and then for each vertex computes its trajectory by computing a weighted sum of the basis vectors.

8.1.1 PCA in Coddyac

For Coddyac we have chosen a temporal PCA in the space of trajectories. This decision has been motivated by following arguments:

- PCA in the space of shapes is not efficient and requires rigid motion compensation.
- PCA in the space of trajectories is much faster, involving only $O(VF^2)$ operations, while PCA in the space of shapes is $O(FV^2)$. We expect that $V >> F$, because in the future the meshes are likely to became more

order	eigenvalue	percent
1	722337,8	68,89%
2	167162,6	15,94%
3	87062,9	8,30%
4	30829,5	2,94%
5	16503,6	1,57%
6	6998,5	0,67%
7	3412,9	0,33%
8	3136,0	0,30%
9	2516,4	0,24%
10	1751,8	0,17%
11	1234,8	0,12%
12	884,5	0,08%
13	770,2	0,07%
14	574,2	0,05%
15	497,9	0,05%
16	407,5	0,04%
17	395,2	0,04%
18	331,2	0,03%
19	277,4	0,03%
20	242,6	0,02%
remaining 1180	1222,3(sum)	0,12%
	1048549,7	**100%**

Table 8.2: Eigenvalues associated with 20 most important eigentrajectories of the chicken sequence.

detailed, however the length of a scene is dictated by the rules of film editing. Moreover, it is quite easy to cut a long sequence into shorter paths, while cutting a too detailed mesh into parts is much more difficult.

- Using PCA in the space of trajectories is much more memory efficient.
- Trajectory space PCA representation is suitable for direct and memory efficient displaying using modern GPUs with programmable vertex processing pipeline.

In our experiments, it was usually possible to reduce the number of used basis vectors to about one tenth without any significant loss of quality. Table 8.2 shows the eigenvalues of the 20 most significant eigentrajectories in the chicken sequence. The table shows that 99,88% of the variance of the data is contained in the first 20 eigenvectors out of total 1200.

8.1.2 PCA basis encoding

As we will see later, the PCA coefficients can be encoded very efficiently, and therefore the basis encoding becomes a non negligible issue. For high fidelity

compression, we're usually having approximately 60 eigentrajectories, each of length $3F$. We also have the means of each coordinate in each frame, which can be treated as a first eigentrajectory with default coefficient 1,0.

Direct encoding of such amount of data may take more bytes than the size of encoded coefficients, and therefore we have proposed a non-trivial algorithm for basis encoding, which will be described in section 8.4.

8.1.3 PCA coefficient prediction

At this point of algorithm, each vertex has a vector of PCA coefficients assigned, representing the trajectory of the vertex. This representation well exploits the temporal coherence of the data - the components of the vectors are uncorrelated. However, they still contain a lot of spatial coherence - vectors that are assigned to neighboring vertices are likely to be similar.

The current methods have used Clustering to exploit this fact, however, using the cluster center as a prediction implies that the size of residual in such case depends on the radius of the cluster. On the other hand, in Coddyac we're using the vertices in the immediate topological neighborhood, and therefore the size of the residual depends on the length of mesh edge, which is much smaller.

For the prediction, we have used the parallelogram rule described in section 3.1.4. The only difference is that we're generalizing it from prediction of XYZ coordinates (vectors of length 3) to a prediction of PCA coefficient vectors c_i^j of arbitrary length. The generalized rule is expressed as follows:

$$c_{i,pred} = c_{i,left} + c_{i,right} - c_{i,opposite} \tag{8.1}$$

where c_i is the i-th component of the coefficient vector c. This prediction formula is applied on every component of the PCA coefficient vector separately, giving a prediction of the coefficient vector at a vertex encountered by the C operation during Edgebreaker (see section 3.2.2) processing of the mesh.

Finally, the prediction residuals are quantized and encoded using entropy coding. We're using Huffamn[28] coding with different context (codetable) for each component of the eigenvectors.

The decoding procedure is a straightforward inverse of the encoding.The decoder performs the same prediction as the encoder, receives a correction and by adding the prediction and correction, it obtains a decompressed version of the PCA coefficients. These coefficients are subsequently turned into trajectories by multiplying the basis vectors (matrix $\mathbf{B}$) by the coefficient vector and adding the means (vector $\mathbf{m}$):

$$\mathbf{t} = \mathbf{B}.\mathbf{c} + \mathbf{m} \tag{8.2}$$

The overall algorithm is controlled by two parameters: The size of the PCA basis (number of used eigentrajectories) and the quantization constant used to determine the quantization quantum. Both of these values influence the amount of error, however the character of the introduced error is different.

8.2 Combined compression and simplification

The main idea behind the combined compression and simplification algorithm is replacing the extrapolation of Coddyac by interpolation. Interpolation generally provides a more robust and accurate prediction, thus reducing the residual entropy. However, in order to be able to perform this replacement, we have to prepare the mesh into a state, where a complete geometry of each vertex neighborhood is known to both encoder and decoder during each prediction.

This state is achieved by topology-guided decimation, which is simultaneously performed by both encoder and decoder. A series of decimation steps is performed, producing a coarse version of the mesh connectivity. The geometry of this coarse version is transmitted using an extrapolating predictor, i.e. using Coddyac. Subsequently, vertices which have been removed during decimation are returned into the mesh, using interpolating prediction.

The proposed scheme consists of following steps:

1. compute PCA of the vertex trajectories
2. transmit the connectivity of the mesh
3. decimate the mesh (at both encoder and decoder)
4. transmit the PCA coefficients for non-decimated vertices, using parallelogram predictor
5. transmit the PCA coefficient for decimated vertices, using neighborhood average predictor

Our suggestion is to use neighborhood average (NA) predictor, i.e. predicting a value (PCA coefficient) as a mean of the value in the topological neighborhood. This predictor usually provides a very robust estimation. After the step 2 the encoder sorts all the vertices according to the accuracy of their prediction by the NA predictor. This ordered set is then used as a priority queue for vertex removal.

The removal of vertices is performed simultaneously at both encoder and decoder. Note that at this point the decoder has no information about the geometry at all, and therefore the retriangulation of the hole must be performed solely according to connectivity criteria. The only information the decoder receives is the

index of the vertex to be removed, however this leaves some control to the encoder - it knows how the decoder will retriangulate, and based on this knowledge it can avoid removing such vertices where the retriangulation would introduce geometrical problems - details on this will be given later.

The strategy of the encoder is following:

1. set all vertices unlocked, evaluate the decimation costs
2. pick the best predicted vertex v from the head of the queue
3. if v is locked then remove v from queue and go to 2
4. simulate the retriangulation after removal of v, if it violates geometrical criteria, then remove v from the queue and go to 2
5. lock the neighbors of v
6. mark the vertex v to be removed by the decoder at the current level of decimation
7. go to 2 if there are some vertices left
8. go to 1 if further simplification is required

The simplification process is repeated several times in order to create multiple simplification levels of the mesh. After each step all the vertices are unlocked (step 1), and the prediction accuracy (i.e. simplification cost) is reevaluated at each vertex, again allowing all the vertices to be removed in the following simplification step.

After this process, the encoder and decoder share a simplified connectivity, and every removed vertex can be predicted from its neighbors (however, the decoder has still no information about the geometry of any vertex).

Note that there is some overhead associated with the decimation. However, we don't need to send the exact order of vertices to be removed (which would take $V.log_2V$ bits, V being the number of vertices), the only information the decoder needs is the level of decimation at which each vertex should be removed. To transmit this information, we only need $V.log_2s$ bits, where s is the number of simplification levels, or even less than that when entropy coding is used (simplification levels are not distributed uniformly, most vertices will be removed at first level, less at second etc.).

At this point, the encoder and decoder share a series of increasingly simplified versions of the mesh, where each finer one can be obtained by inverse vertex removal, i.e. we know where each vertex should be placed to reach a finer version of the topology. In our experiments, we have usually driven the algorithm to reduce the number of vertices to less than one fifth of their original number.

Now we use the Coddyac algorithm to transmit the PCA coefficients of the coarse mesh. The encoder traverses the triangles and predicts the coefficients using the parallelogram rule, and sends the residuals to the decoder, until the whole coarse mesh is transmitted.

At this point, the decoder can start playing the animation, while it is still receiving refinement information. The parallelogram predictor is now replaced by the NA predictor, and the decoder continuously refines the mesh by reversing the previous decimation. It is guaranteed that for each vertex the decoder has all the neighbors available, and therefore it can compute the NA prediction. Again, the encoder only sends the quantized residuals.

8.2.1 Algorithm details

There are two gaps in the description that are to be filled - the conditions for selecting the vertices for decimation, and the criteria for retriangulation performed in the decoder. We are suggesting simple approaches for both tasks.

If we were creating only one decimation level, then we generally would not care about the geometrical properties of the created holes, however when multiple simplification levels are being created it is advisable to preserve a reasonable quality mesh. On the other hand, the encoder cannot control the way in which the hole is retriangulated by the decoder, because such controlling would require additional information to be sent.

The retriangulation process must be driven by connectivity only, as this is the only information available at the decoder. Although more advanced methods exist for this task, we have chosen a simple "ear cutting" algorithm[43] for triangulation of a simple hole. The decoder however has no information about the convexity/concavity of the hole border, and therefore it simply considers any vertex to be a candidate ear tip.

The only criterion the decoder uses for selecting an ear tip is the regularity of degrees of the vertices. It is known that the expected degree of a vertex is six, and good shaped meshes have very regular distribution of vertex degrees. Cutting an ear does not increase the ear tip degree, while it increases the degree of the two vertices next to the tip by one. Therefore, we select the candidate tip where the following expression is maximal:

$$\delta(tip) - \delta(tip_{left}) - \delta(tip_{right}) \tag{8.3}$$

where $\delta(x)$ stands for the degree of vertex x. The tip is cut, and the remaining hole is again searched for the best candidate tip, until it is fully retriangulated. Ambiguous situations are solved by selecting a candidate tip vertex with the lowest index, so that both the encoder and the decoder have the same retriangulation.

There is however no guarantee that the retriangulation is geometrically correct. Therefore, we perform the retriangulation at the encoder as well and check the geometrical correctness. If it is not preserved, then the vertex is not decimated. We only evaluate the criteria for the first frame, as it is likely that if the geometry is not compromised in the first frame, then it will not be changed in the following frames either. Note here that the simplification is only a tool for compression, and we do not aim for perfect simplified version of the model, nor is geometry preservation of essential importance for us.

First, we check for normal flips. We compute an average normal of the hole by averaging the normals of all removed triangles. If any of the normals of the new triangles is oriented opposite to the original normal (we use scalar multiplication), then the decimation is not performed.

Second, we check for "sharp" triangles. We check all the corner angles of the new triangles, and if any angle is lower than some threshold (0.05 rad in our experiments) then the decimation of the vertex is also cancelled.

Finally, we do not consider decimation of all the vertices. We only use the first 80% of the priority queue, the rest is considered to be too badly predicted by the NA predictor.

8.3 Progressive predictors

In this section we will show how updating the predictor during the transmission of the coordinates of a single vertex can improve the precision of the prediction. The nature of PCA removes global correlation between the coordinates, and thus we cannot directly determine anything about a subsequent coordinate based on the value of the previous one. However, when the neighborhood is known, we can make some assumptions about the relations between the neighborhood and the decoded vertex, allowing a more efficient compression.

Now, we will make the crucial observation, upon which our predictors are based. The situation depicted in Figure 8.1 is quite common in static mesh decimation and coding. The vertex v has been removed in the decimation step, and in the reconstruction its position is unknown. The task is to predict the position as precisely as possible, so that the residual will be very small, and thus the overall entropy of the residuals will be low.

If we consider only the X coordinate of v, we probably cannot do a much better prediction than just averaging the X coordinates of the neighborhood, which are known to the decoder. After having done so, the encoder sends the residual, which corrects the initial guess. The following task is to predict the Y coordinate.

If the model had not been preprocessed, we could have guessed something about Y from X, however, this cannot be done for our case of data previously

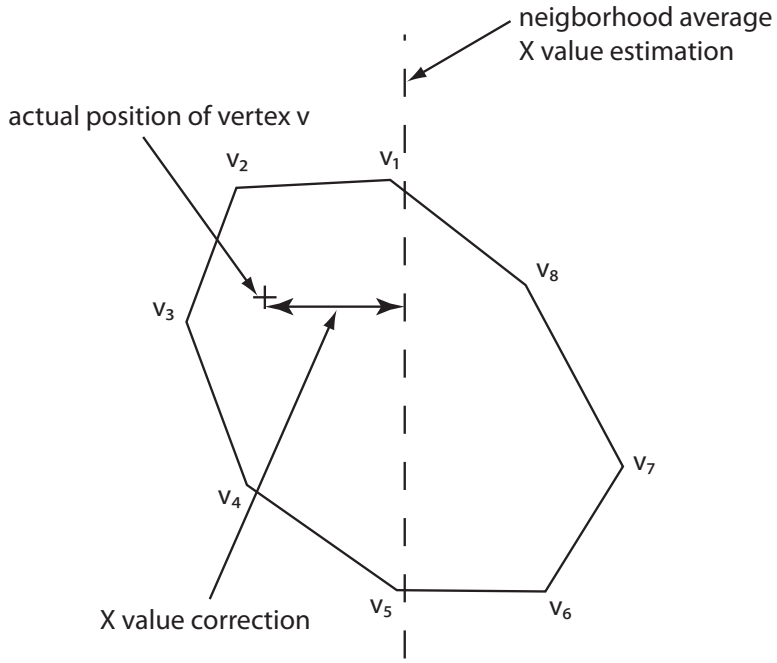

Figure 8.1: Inverse decimation step - a vertex is being added to a neighborhood.

decorrelated by PCA. On the other hand, if we consider the neighborhood as well, we can extract some information from the correction of the X value. In the situation in Figure 8.2, it clearly makes sense to derive the guess of the Y value from the Y values of vertices v_2, v_3 and v_4, rather than from the other vertices.

The generalization of this idea to a longer vector, which describes the trajectory of a vertex, is the key to the prediction algorithms we will now present. The most direct approach to forming such vector is to concatenate the XYZ coordinates of subsequent positions of each vertex into a vector of length $3F$, F being the number of frames of the sequence. However, such vector shows a strong correlation between the coordinates, which can be removed to improve the compression performance. Therefore, it is better to first decorrelate the data by PCA and apply the prediction directly onto the PCA coefficients.

After the decorrelation step, a decimation of the original mesh is performed in the way described in section 8.2, and therefore the whole neighborhood of a vertex is known to the decoder every time a prediction is performed.

8.3.1 Least squares prediction (LSP)

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

The task is to insert a vertex x. We denote the number of its neighbors N and the number of components of the assigned vectors C. The coefficient vector

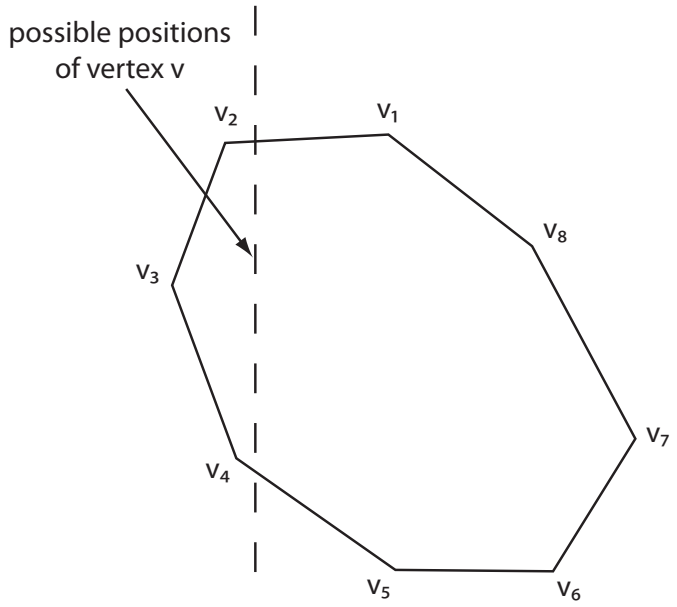

Figure 8.2: Coordinate prediction after the correction of the X value has been transmitted.

assigned to the i-th neighbor will be denoted v^i, it's j-th component will be denoted v^i_j. These vectors are known to both encoder and decoder. The vector assigned to the added vertex will be denoted v, it's j-th component v_j. This vector is only known to the encoder, the decoder needs to compute its prediction, denoted $pred(v)$.

We assume that $C > N$, because for regular meshes N is usually about 6, while C needs to be about 30-60, depending on the length and character of the animation.

For the first N components of the vector v we have to use neighborhood average prediction in the following form:

$$pred(v_j) = \frac{1}{N} \sum_{i=1}^{N} v^i_j \tag{8.4}$$

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

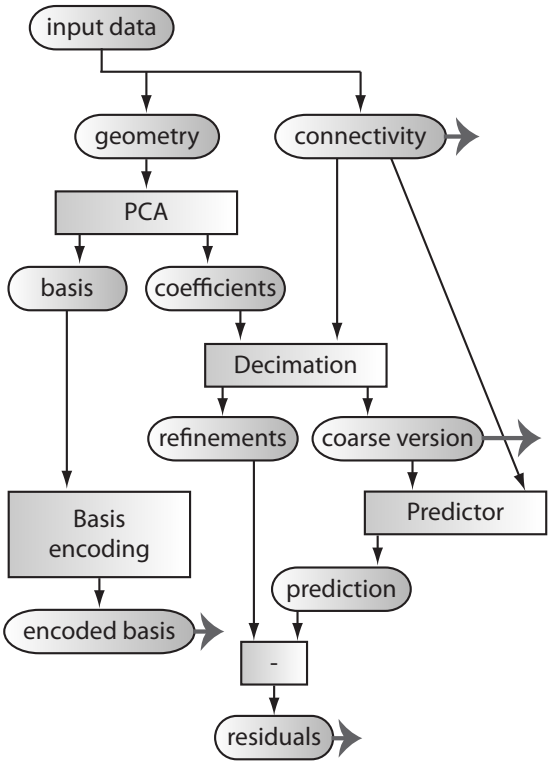

Figure 8.3: Block scheme of the encoder. The thick arrows denote data that are transmitted to the decoder.

$$\begin{aligned} w_1 v_1^1 + w_2 v_1^2 + \cdots + w_N v_1^N &= v_1 \\ w_1 v_2^1 + w_2 v_2^2 + \cdots + w_N v_2^N &= v_2 \\ &\cdots \\ w_1 v_N^1 + w_2 v_N^2 + \cdots + w_N v_N^N &= v_N \end{aligned} \tag{8.5}$$

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

$$pred(v_{N+1}) = \sum_{i=1..N} w_i v_{N+1}^i \tag{8.6}$$

Again, this prediction is computed at both encoder and decoder, and the encoder sends a correction, which makes the actual value of v_{N+1} known to the decoder. Thus, the set of equations (8.5) can be extended by one row. This makes the set overdetermined, however it can be still solved using the least squares

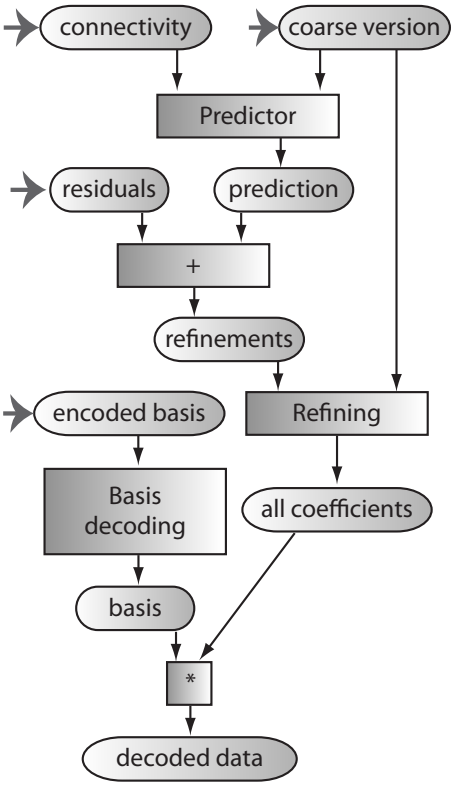

Figure 8.4: Block scheme of the decoder. The thick arrows denote data that are received from the encoder.

minimization. Such solution will yield an even more precise estimate of the weights w_i, which will be then used to predict v_{N+2}. In this manner, with each correction sent to the decoder, a new set of weights is found and used to predict the subsequent component, until the whole vector has been transmitted.

There are two additional circumstances that require treatment:

1. The coordinates are quantized, and therefore it might happen that two or more neighbors have the same values assigned. This may lead to underdetermination of the equation set (8.5). We solve such case by simply removing the neighbors with equal values assigned from the computation. Note that this state is likely to be corrected after more components are transmitted, as it is unlikely that two vertices shared the same position.

2. The values transmitted are the PCA coefficients, and thus their magnitude is approximately sorted from large to small. The quantized residuals of the large values contribute a big part to the entropy, and thus we should focus on precise prediction of these. Therefore, we transmit the coefficient vectors in the reverse order, first transmitting the small coefficients, which allows

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

8.3.2 RBF based predictor (RBFP)

The second predictor we are presenting is based on Radial Basis Function (RBF) interpolation[17, 63]. This approach becomes natural when the prediction is interpreted as a general interpolation.

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

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

$$f(x) = \sum_{i=1..N} \lambda_i \phi(\|x - x_i\|) + a \tag{8.7}$$

where x_i are the locations of N known points, $\|.\|$ denotes euclidean norm, $\phi(r)$ is some function (the function we have used will be discussed later). The values λ_i are unknown, and are determined by the given values at the known points. The equation (8.7) should have the correct value y_i at the given points, thus we get the following set of linear equations:

$$\begin{aligned}
\lambda_1\phi(\|x_1 - x_1\|) + \cdots + \lambda_N\phi(\|x_1 - x_N\|) + a &= y_1 \\
\lambda_1\phi(\|x_2 - x_1\|) + \cdots + \lambda_N\phi(\|x_2 - x_N\|) + a &= y_2 \\
&\cdots \\
\lambda_1\phi(\|x_N - x_1\|) + \cdots + \lambda_N\phi(\|x_N - x_N\|) + a &= y_N
\end{aligned} \tag{8.8}$$

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

$$\lambda_1 + \lambda_2 + \cdots + \lambda_N = 0 \tag{8.9}$$

For exact derivation of this additional row see [17]. This gives us a determined set of linear equations (8.10), which has a symmetric matrix.

$$\begin{pmatrix} \phi(\|x_1 - x_1\|) & \phi(\|x_1 - x_2\|) & \dots & \phi(\|x_1 - x_N\|) & 1 \\ \phi(\|x_2 - x_1\|) & \phi(\|x_2 - x_2\|) & \dots & \phi(\|x_2 - x_N\|) & 1 \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ \phi(\|x_N - x_1\|) & \phi(\|x_N - x_2\|) & \dots & \phi(\|x_N - x_N\|) & 1 \\ 1 & 1 & \dots & 1 & 0 \end{pmatrix} \begin{pmatrix} \lambda_1 \\ \lambda_2 \\ \vdots \\ \lambda_N \\ a \end{pmatrix} = \begin{pmatrix} y_1 \\ y_2 \\ \vdots \\ y_N \\ 0 \end{pmatrix} \tag{8.10}$$

Now that we have described how RBF interpolation works, we can apply it in a quite straightforward way on our case. The first component of the vector assigned to a vertex is estimated from its neighbors by simply averaging them, i.e. using the formula (8.4). The encoder simulates this prediction and sends a correction, and thus the first component is now known at the decoder as well.

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

Subsequent steps are straightforward repetition of the last step. The known components of the transmitted vector are treated as spatial coordinates, and the first unknown component is treated as a value, which is being interpolated using RBF. The dimension of the interpolation space increases, however the size of the set of linear equations remains unchanged.

One question that remains to be answered is the choice of the function $\phi(r)$. Literature about RBF does not offer any ultimate function that will provide the best results in any situation, generally any function can be used, and the efficiency of the functions is difficult to predict.

We have performed experiments with various functions suggested by the RBF literature, such as the thin plate spline (TPS) of form $\phi(r) = r^2 log(r)$ or compactly supported functions such as $\phi(r) = e^{-cr^2}$, however the most accurate predictions have been obtained by using power function $\phi(r) = r^\beta$, where the value of β is little less than 2, usually about $1.7 - 1.9$.

We don't have a derivation of this value, and it varies slightly depending on the dataset used, however it can be seen that using $\phi(r) = r^1 = r$ is not a good choice, because this function has a non-zero derivative for zero argument, i.e. the radial function is not smooth. A simple function with zero derivative at zero is $\phi(r) = r^2$, unfortunately this function cannot be used, because it makes the set of equations (8.10) degenerate. Therefore, a function $\phi(r) = r^{1.8}$ is a compromise, which produces a solvable set of equations, while the function has a zero derivative at zero, and is also reasonably smooth around zero (in contrast with functions such as $\phi(r) = r^{1.1}$, which also have zero derivative at zero, but

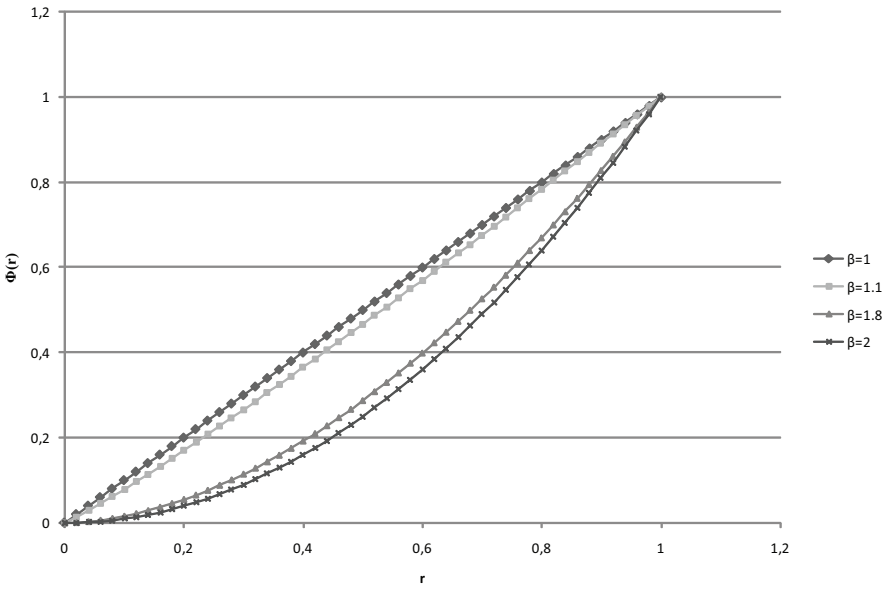

Figure 8.5: Examples of basis functions of form $\phi(r) = r^\beta$. For $\beta = 1$ and $\beta = 1.1$ we get sharp development around zero, while $\beta = 2$ yields a degenerate set of equations. $\beta = 1.8$ is a compromise.

in fact are almost sharp, see figure 8.5).

Also note that the prediction algorithm is used not only in the reconstruction step, but also in the simplification step, where the prediction error is used as decimation cost. This leads to a slightly different decimation strategy. In the case of neighborhood average, the algorithm removed vertices that were close to the center of their neighborhood, while the progressive predictors prefer vertices that lie in the plane of their neighborhood, even when they are slightly off-center. In other words, the simplification is now more precise in removing vertices that carry little geometrical information.

8.4 Encoding of basis for PCA represented animations

Our basis compression algorithm is build on two blocks - prediction and non-uniform quantization. Although both techniques are widely used in compression tasks, and their employment may seem obvious, there remain some questions that need to be answered in order to achieve efficient compression. Namely, we must select a proper prediction scheme, which will provide as low entropy residuals as possible, and then we must quantize the residuals accurately so that all the values are transmitted neither too coarsely, nor too finely.

8.5 Prediction

The key observation for following derivations is that the basis vectors, which need to be encoded, still retain the character of trajectories. In other words, if one interprets one basis vector as a trajectory of a moving point, then the point moves smoothly. Figure 8.6 shows the first basis vector of the chicken sequence interpreted as three trajectories, one for each coordinate.

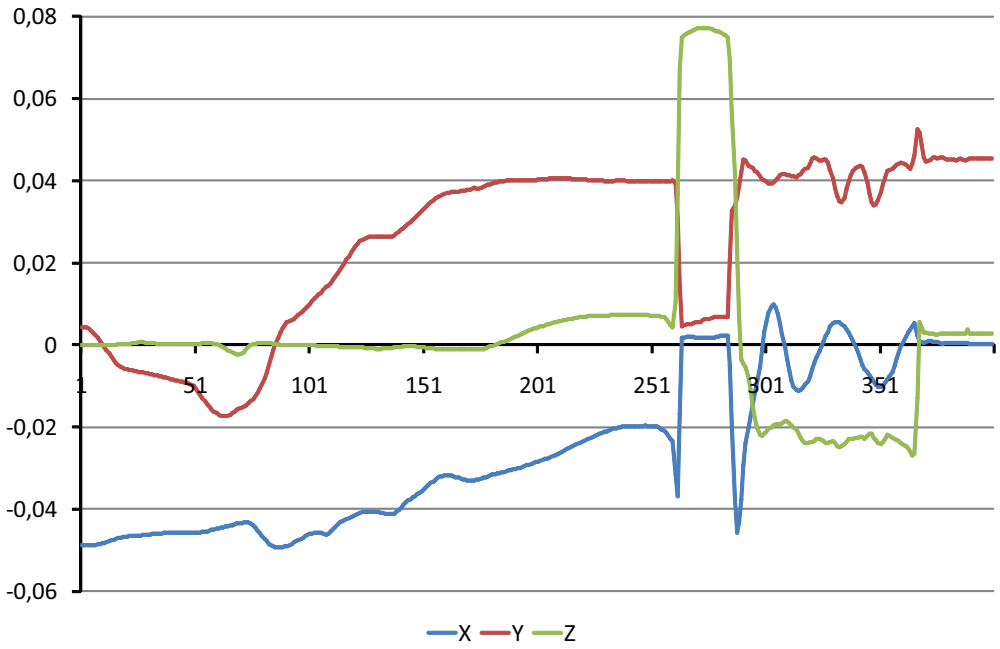

Figure 8.6: First basis vector of the chicken sequence. The sudden jump in frames 250-300 is the chicken popping eyes, the subsequent sinusoidal development of the X axis is the flapping of wings.

This observation has been made previously by Karni and Gotsman in [32], who noticed this behavior of PCA coefficients of subsequent frames (note that they have used an eigenshape based PCA). Their suggestion was to apply linear predictive coding, LPC, to predict and encode the values. The LPC concept is based on predicting a given value in a series as a linear combination of a given number of previous values. The same set of combination coefficients is used for the whole sequence (or for multiple sequences of the same behavior) and their values are found in a least squares optimization process applied by the encoder on the whole sequence. For more details see the original source[32].

We have first followed this suggestion for the basis as well, however we have found that for the purposes of efficient encoding it is sub-optimal. Imagine a situation depicted in figure 8.7. In this simplified scenario, we are given a value v_{f-1} (preceding value, one of the XYZ coordinates) and we want to predict the

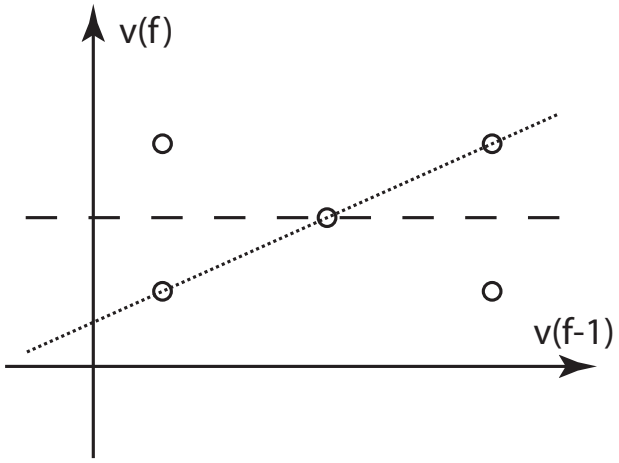

Figure 8.7: The two linear predictors for a given data set (each data point is represented by a circle).

value v_f by a linear formula

$$v_f = kv_{f-1} + q \tag{8.11}$$

The LPC algorithm suggests to apply least squares optimization to find the values k and q, in our case we will get the dashed line. We can now perform quantization, which will result only in three possible values: 0 for the exactly predicted point, $+\alpha$ for the points above the prediction, and $-\alpha$ for points below it. The vector of residuals is:

$$\mathbf{r} = [-\alpha, +\alpha, 0, -\alpha, +\alpha] \tag{8.12}$$

Thus the probabilities are

$$p(-\alpha) = \frac{2}{5}, p(+\alpha) = \frac{2}{5}, p(0) = \frac{1}{5} \tag{8.13}$$

The entropy is computed as follows:

$$\begin{aligned} E &= -\sum p \log_2(p) \\ &= -(\frac{2}{5}\log_2(\frac{2}{5}) + \frac{2}{5}\log_2(\frac{2}{5}) + \frac{1}{5}\log_2(\frac{1}{5})) \\ &= 1.522[b] \end{aligned} \tag{8.14}$$

However, if we construct a different linear predictor, such as the one drawn in the figure as a dotted line, we get following residuals, probabilities and entropy:

$$\mathbf{r} = [0, +\alpha, 0, -\alpha, 0] \tag{8.15}$$

$$p(-\alpha) = \frac{1}{5}, p(+\alpha) = \frac{1}{5}, p(0) = \frac{3}{5} \tag{8.16}$$

$$\begin{aligned} E &= -\sum p \log_2(p) \\ &= -(\frac{3}{5}\log_2(\frac{3}{5}) + \frac{1}{5}\log_2(\frac{1}{5}) + \frac{1}{5}\log_2(\frac{1}{5})) \\ &= 1.379[b] \end{aligned} \tag{8.17}$$

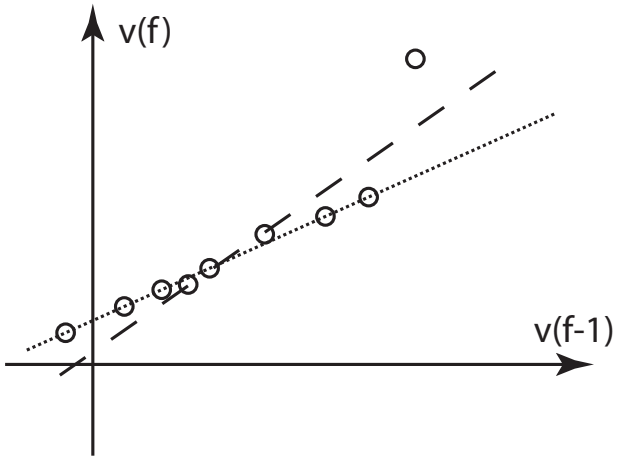

Figure 8.8: Least squares solution is lead away from a good predictor of most of the data by an outlier.

This rather artificial example shows that there are cases, when least squares solution leads to a sub-optimal prediction. Indeed, it is a generally known problem[70] of the least squares optimization that the solution can be lead astray by outliers, because the squared difference of these has a large influence on the overall solution. Figure 8.8 shows a more realistic case, when most of the data points are linearly dependent and can be accurately predicted by the dotted line, however the least squares solution will be twisted by the outlier point, which will cause a significant increase of residual entropy.

Figure 8.9 shows a real world example. The dataset is the PCA basis of the first 100 frames of the chicken sequence. The samples show the dependency of a basis value (v_f axis) on its predecessor (v_{f-1} value) where appropriate predecessor is available. A least squares fitting of such data produced the depicted line, which can be used to predict v_f from v_{f-1}. However, if we consider the real situation, we can use simple delta coding expressed as:

$$pred_1(v_f) = v_{f-1} \tag{8.18}$$

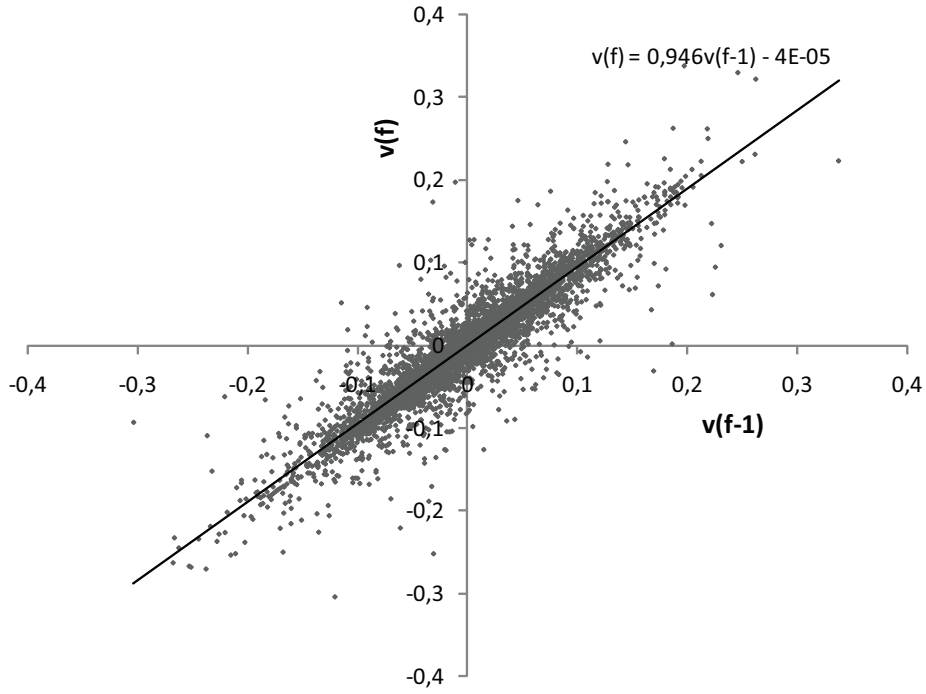

Figure 8.9: Real data prediction.

Such coding is least-squares sub-optimal, however it delivers residuals with smaller entropy.

In a similar way, we can fit the triplets (v_{f-2}, v_{f-1}, v_f), where v_{f-2} is a value from frame $f-2$, v_{f-1} is the value from frame $f-1$ and v_f is a value in frame f which we're trying to predict. We can either use least squares fitting, or linear movement prediction in a form

$$pred_2(v_f) = v_{f-1} + (v_{f-1} - v_{f-2}) = 2v_{f-1} - v_{f-2} \tag{8.19}$$

The last two predictors that we have experimented with use three and four preceding values to predict the current value v_f, estimating the speed s, the acceleration a, and the change in acceleration c to obtain prediction as follows:

$$
\begin{aligned}
s &= v_{f-1} - v_{f-2} \\
a &= (v_{f-1} - v_{f-2}) - (v_{f-2} - v_{f-3}) \\
&= v_{f-3} - 2v_{f-2} + v_{f-1} \\
c &= ((v_{f-1} - v_{f-2}) - (v_{f-2} - v_{f-3})) - \\
&\quad - ((v_{f-2} - v_{f-3}) - (v_{f-3} - v_{f-4})) \qquad (8.20) \\
pred_3(v_f) &= v_{f-1} + s + a = 3v_{f-1} - 3v_{f-2} + v_{f-3} \qquad (8.21) \\
pred_4(v_f) &= v_{f-1} + s + a + c \\
&= 4v_{f-1} - 6v_{f-2} + 4v_{f-3} - v_{f-4} \qquad (8.22)
\end{aligned}
$$

The overall prediction algorithm must also prevent error accumulation by using quantized values in the encoder. The overall scheme is summarized in algorithm 1.

input: basis vector B_i
input: quantization constant Q_i (its computation will be described in the following section)
input: order of prediction o, i.e. the number of preceding values needed by the predictor
for $coord \leftarrow 0$ **to** 2 **do**
  for $j \leftarrow (1 + coord * F)$ **to** $(o + coord * F)$ **do**
    $q \leftarrow round(B_{i,j}/Q_i)$;
    send q to entropy coder;
    $B_{i,j} \leftarrow q * Q_i$;
  end
  for $j \leftarrow (o + 1 + coord * F)$ **to** $(F + coord * F)$ **do**
    $pred \leftarrow predictor(B_{i,j-1}, B_{i,j-2}, ..., B_{i,j-o})$;
    $residual \leftarrow B_{i,j} - pred$;
    $q \leftarrow round(residual/Q_i)$;
    send q to entropy coder;
    $B_{i,j} \leftarrow pred + q * Q_i$;
  end
end

Algorithm 1: Basis compression.

8.5.1 Quantization

The final step in encoding the values is quantization. The predictor produces a floating point value, which is divided by a quantization constant, the result is truncated and passed to an entropy coder for encoding.

However, we have found out that careful treatment of basis quantization may lead to further improvement of compression ratio. Recall the decompression equation 8.2. It can be expanded to following form:

$$\mathbf{t} = B_1.c_1 + B_2.c_2 + \cdots + B_{N_B}.c_{N_B} + \mathbf{m} \tag{8.23}$$

where B_i represents the i-th row of the matrix B, i.e. a basis vector. The error introduced by the quantization of the PCA coefficients c_i is equal for each term of equation 8.23, as all the coefficients are quantized with equal quantization constant.

We can also see that the error is generally additive, and therefore we want it to be equal for every term. However, the size of the coefficients c_i varies significantly. Fortunately this variance can be well predicted - the first coefficient is usually much larger than the second, which is larger than the third etc., which is a behavior caused by the nature of PCA.

Thus, if we had used an uniform quantization, we can expect the error of half the quantization constant, which will be multiplied by a very large constant in the first term. Such behavior is undesirable, and thus we must use finer quantization for the more important basis vectors, while the less important ones can be quantized more coarsely.

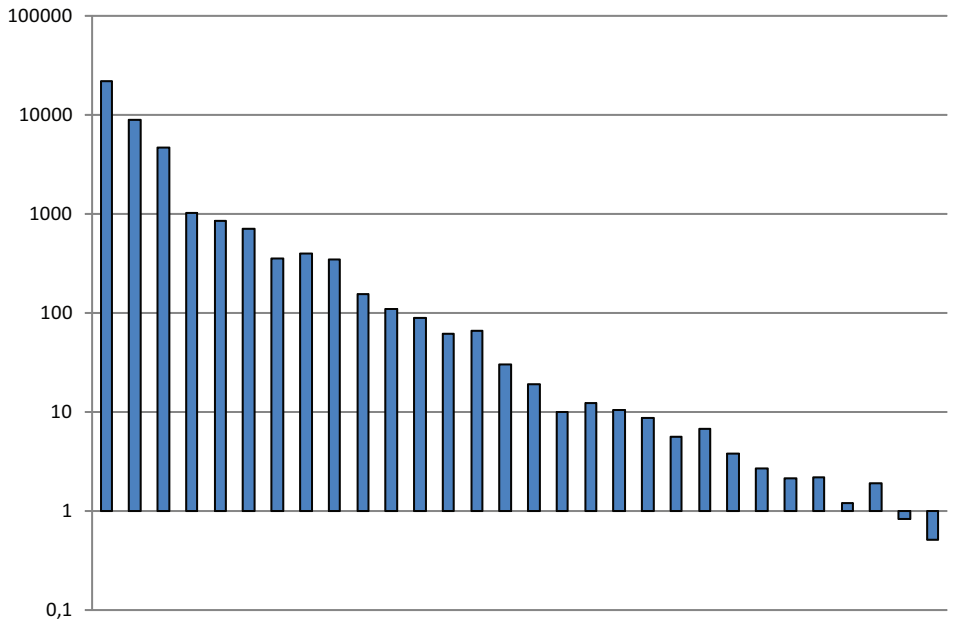

Figure 8.10: Coefficient sums.

When given a quantization constant Q from user, we can split this constant to the given number of terms of equation 8.23, however we should follow the

magnitudes of corresponding coefficients c_i. To do this, we can first compute the average

$$a_i = \frac{1}{V} \sum_{v=1..V} \|c_i^v\| \tag{8.24}$$

of absolute values of each coefficient over all vertices $v_{1..V}$. This gives an approximation of how many times will an error be repeated in the final decompressed sequence. Figure 8.10 shows the summed absolute values of the PCA coefficients in the chicken sequence, note that the scale of the figure is logarithmic.

If we want to keep the error of the terms in equation 8.23 equal, then we must use quantization constants inversely proportional to the averages of coefficients. Thus, the quantization constant for i-th basis vector should be computed as:

$$Q_i = \frac{Q}{N_B + 1} a_i^{-1} \tag{8.25}$$

In this equation i is the order of the basis vector and Q is the quantization constant, which must be divided into $N_B + 1$ parts, since the equation 8.23 has $N_B + 1$ terms.

We could use an exact value for each coefficient, however that would require transmitting the quantization constant with each basis vector. To avoid this, we have decided to use a power function approximation to compute the quantization constants for each basis vector at both encoder and decoder. We perform a least squares minimization to obtain the constants k and q in the equation 8.26.

$$ki^{\,q} = Q_i \tag{8.26}$$

Since the quantization of the first basis vector is of biggest importance, we subsequently shift the approximation curve by a constant a so that the first quantization constant perfectly fits the user specified intended error:

$$k.1^q + a = k + a = Q_1 \tag{8.27}$$

$$a = Q_1 - k \tag{8.28}$$

This way, we only need to transmit the constants k, q and a, and we get a series of increasing quantization constants, that well fits the distribution of absolute values of PCA coefficients, and for the first coefficient gives exactly the user specified error amount.

This derivation can be directly applied to compression of the means vector, which is transmitted along with the basis. Its components can be predicted using any of the equations 8.18-8.22 and the residuals should be quantized using quantization constant Q_m:

$$Q_m = \frac{Q}{N_B + 1} \tag{8.29}$$

The key issue arising with the presented prediction algorithms is the decompression speed. The decompression consists of coefficient restoration, from which then the original trajectories are restored. Our optimized version of the coefficient restoration algorithm, which builds the RBF matrix incrementally is currently capable of achieving about real-time decompression for moderately complex mesh sequences. The chicken run sequence (cca 16 second animation) can be decompressed in about 5 seconds on a 3,2GHz Pentium D PC, however more detailed meshes take longer to decompress. On the other hand, the algorithm still leaves quite large space for optimization, namely the solution of the least squares equation set is likely to change only slightly with added components, and therefore the solution from previous step can be used in a following step to speed the computation up.

Chapter 9

Experimental results

In this chapter we will show results of experiments with an implementation of algorithms described in chapter 8. We are showing rate/distortion curves for each algorithm and each dataset. The rate has been measured by the bit per frame and vertex value, expressed from the overall encoded bitlength *length* as follows:

$$bpfv = \frac{length}{FV} \tag{9.1}$$

As we have stated in chapter 7, we will provide two distortion measures for each dataset, the KG error, which is used to compare the results with competing algorithms, and the STED error, which more closely approximates the perceived quality.

9.1 Coddyac results and STED considerations

The first series of figures 9.1-9.16 shows the results of the Coddyac algorithm described in section 8.1. We're showing results for all the test sequences mentioned in chapter 6, and we are showing results for varying numbers of basis vectors and varying quantization constants. Each curve in a graph represents a series of experiments with constant number of basis vectors (given by the legend) and a varying quantization constant, spanning from 7 (the leftmost point, i.e. the lowest data rate, coarsest quantization) to 13 (the rightmost point, i.e. the highest data rate, finest quantization).

The first observation that can be made based on the results is that the used measures provide significantly different results. The basic development of the curves remains unchanged, i.e. finer quantization leads to lower error in both cases. However, the similarity fails when different numbers of basis trajectories is used. In the KG error measure curves, the error decreases with the number

configuration <basis>-<quantization>	20-13	25-8	40-8	20-9	10-8	30-13	15-13	35-9	10-12
MOS	1,524	9,238	9,619	6,000	7,000	1,619	1,952	6,571	1,238
standard deviation	1,167	0,889	0,590	1,581	1,732	1,532	1,532	1,805	1,300

Table 9.1: Results of the subjective testing with the human jump sequence.

of basis trajectories, which seems to be intuitive. In contrast to that, in the STED measures this relation is more complex. For fine quantization the relation holds, however for coarse quantizations the relation is reversed, i.e. more basis vectors introduce more error. This can be explained by the fact that STED mainly focuses on local error, which exhibits an additive behavior, and therefore increases when more basis vectors with error are added together.

Second conclusion that can be made is that the selection of compression parameters (number of basis vectors, quantization constant) is strongly dependent on the measure used. Generally, the KG measure prefers higher number of basis vectors, while the quantization may remain coarse, while the STED measure usually drops significantly with finer quantization, and the effect of adding basis vectors is less significant.

For example in the snake sequence, according to the KG measure, the optimal result at bitrate of 0.5 bpfv is obtained by using 15 or more basis vectors. According to the KG experiment, it is optimal to increase the number of basis vectors to 20 at the bitrate of approximately 0.7 bpfv, and then again to 25 when allowed bitrate reaches 0.93 bpfv. In contrast to that, according to the STED measure experiment, 10 basis vectors suffice up to a bitrate of 0.95 bpfv, and the additional bits should be invested in finer quantization. This is a significant difference that greatly influences the decision about the optimal configuration of a compression scheme.

The behavior of the STED measure can be considered counterintuitive, and therefore we have prepared a series of distorted versions of the human jump sequence, for which the STED and KG measures are inconsistent in decision about which animation is more distorted. We have performed a blind subjective test to determine whether the actual observations match the results of STED or KG error.

Table 9.1 summarizes the findings from 21 subjective evaluators. The column headers give the compression parameters used in the format <number of basis vectors>-<quantization constant>. We have intentionally selected such parameters so that we can test hypotheses following from figures 9.5 and 9.6. The findings are:

1. According to figure 9.6 (STED), the settings 10-8 should provide result with smaller error than settings 40-8. Table 9.1 confirms this with values 9,62 for 40-8 and 7,00 for 10-8.

2. At data rate cca 0.75 bpfv, according to figure 9.5(KG), the optimal configuration of the coder is 35-9, while according to figure 9.6 (STED) it should be 15-13. The test showed that result of configuration 15-13 achieved MOS 1,95, while the configuration 35-9 performed significantly worse, reaching 6,57.

3. Similarly, at bitrate cca 0.5 bpfv, according to figure 9.5 (KG), the optimal configuration of the coder is 25-8, while according to figure 9.6 (STED) it should be 10-12. The test showed that result of configuration 10-12 achieved MOS 1,23, while the configuration 25-8 performed significantly worse, reaching 9,23.

4. According to figure 9.5 (KG), no significant improvement is achieved when improving quantization from configuration 20-9 to 20-13, which contradicts figure 9.6 (STED), which shows that such change leads to drop of error to about one half. The subjective results confirm the second conclusion by MOS 6,00 for configuration 20-9 versus 1,52 for configuration 20-13.

5. Similarly, according to figure 9.6 (STED), no significant improvement is achieved when adding basis trajectories from configuration 15-13 to 30-13, which contradicts figure 9.5 (KG), which shows that such change leads to drop of error to about one half. The subjective results confirm the first conclusion by MOS 1,95 for configuration 15-13 versus 1,62 for configuration 30-13.

Generally, all the tested assumptions following from STED measures have been confirmed by the experiment. There is however one unsolved contradiction following from the test. The configuration 15-13 has reached MOS 1,95, which is worse than the result of configuration 10-12, which has scored 1,23. This relation should be other way round, because 15-13 has both finer quantization and more basis vectors. We believe that this is caused by the fact that even at such low rates, the visual error is so small, that the observers could not distinguish between the two versions, and the difference is caused by random factors. This explanation is supported by the standard deviation values, which are of similar magnitude as the MOS values themselves.

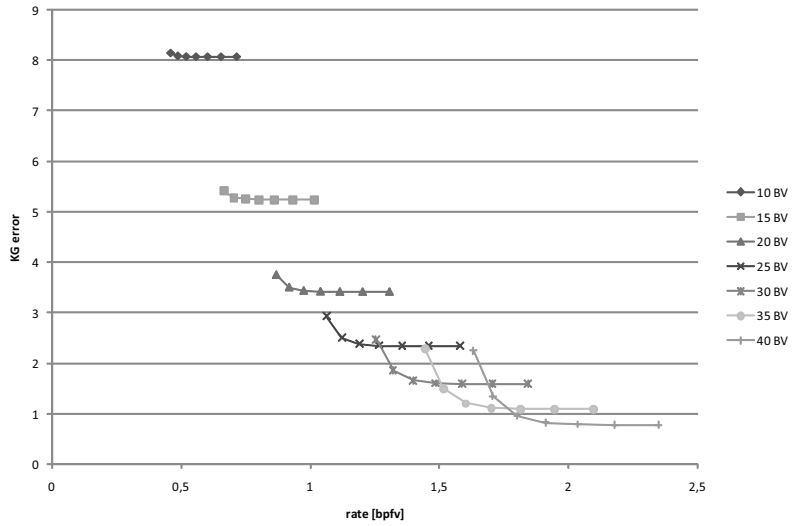

Figure 9.1: Rate/distortion curves for the **chicken** sequence using the Coddyac algorithm.

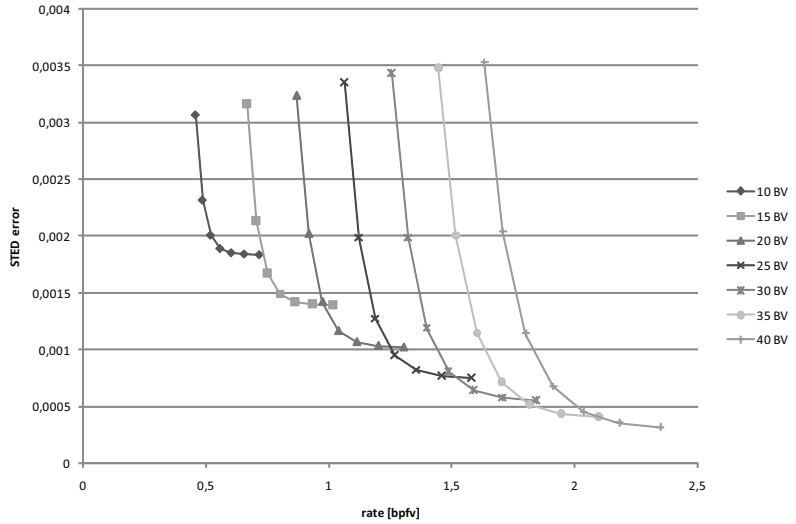

Figure 9.2: Rate/distortion curves for the **chicken** sequence using the Coddyac algorithm.

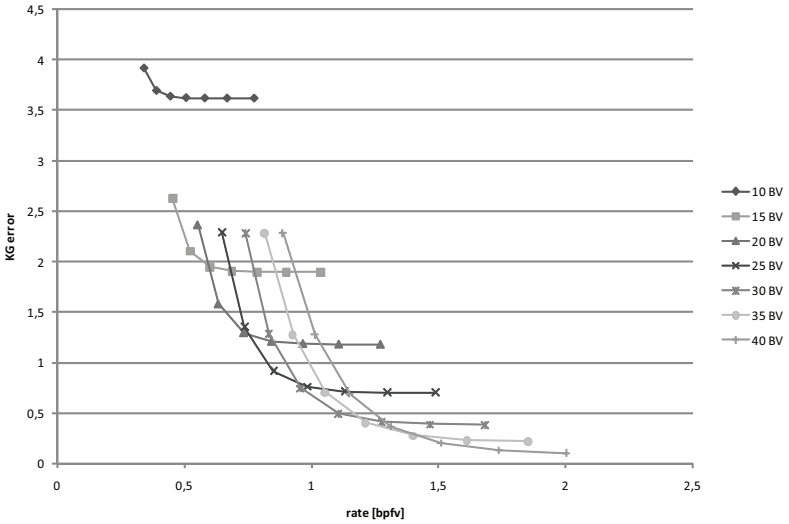

Figure 9.3: Rate/distortion curves for the **dance** sequence using the Coddyac algorithm.

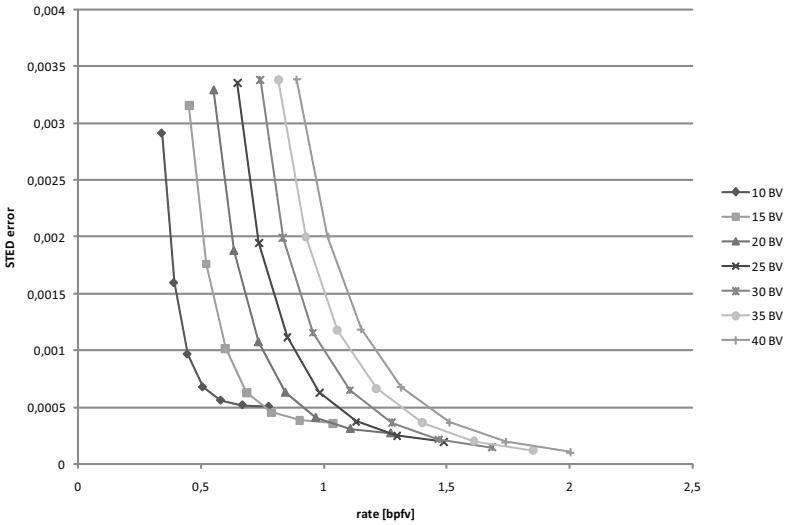

Figure 9.4: Rate/distortion curves for the **dance** sequence using the Coddyac algorithm.

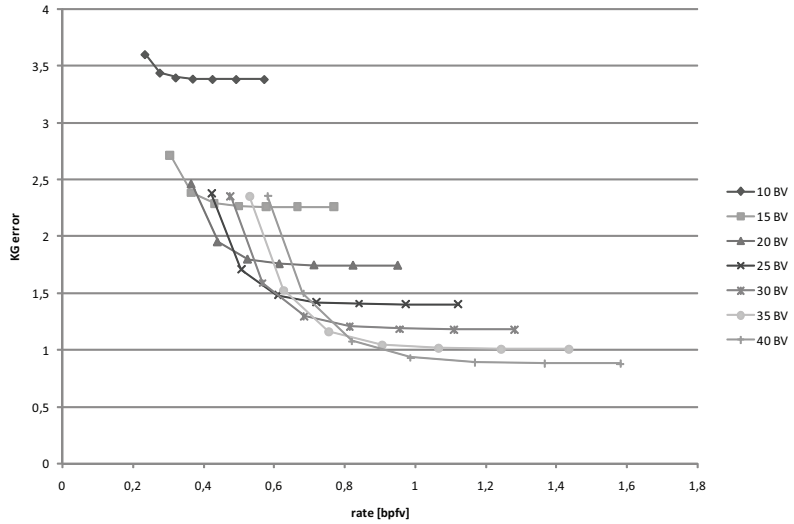

Figure 9.5: Rate/distortion curves for the **jump** sequence using the Coddyac algorithm.

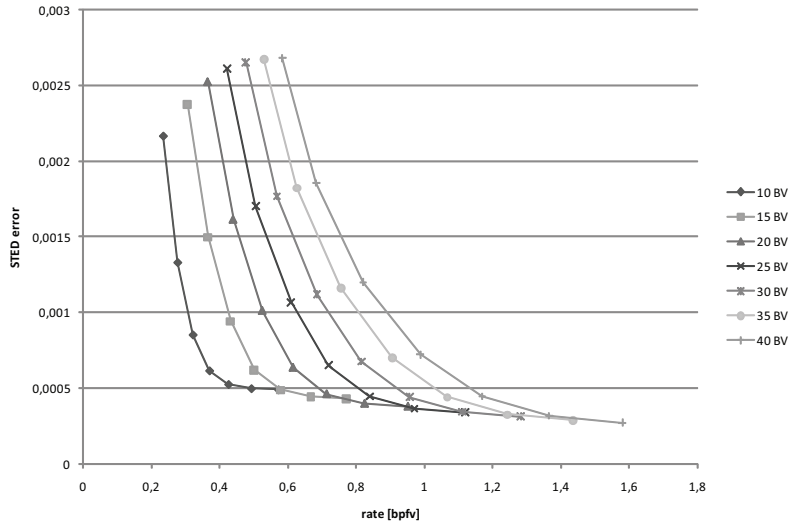

Figure 9.6: Rate/distortion curves for the **jump** sequence using the Coddyac algorithm.

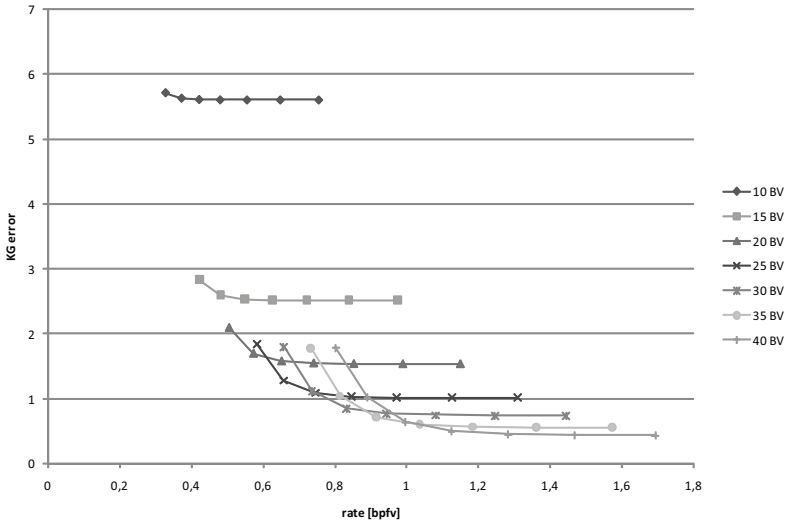

Figure 9.7: Rate/distortion curves for the **cloth** sequence using the Coddyac algorithm.

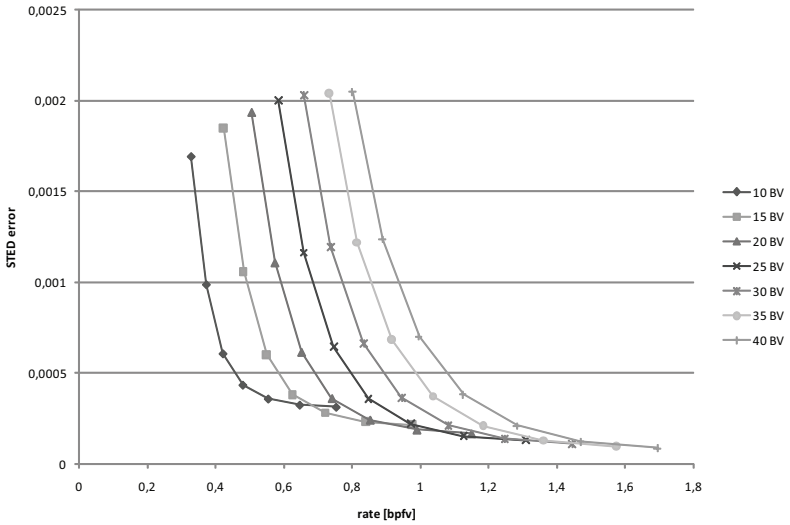

Figure 9.8: Rate/distortion curves for the **cloth** sequence using the Coddyac algorithm.

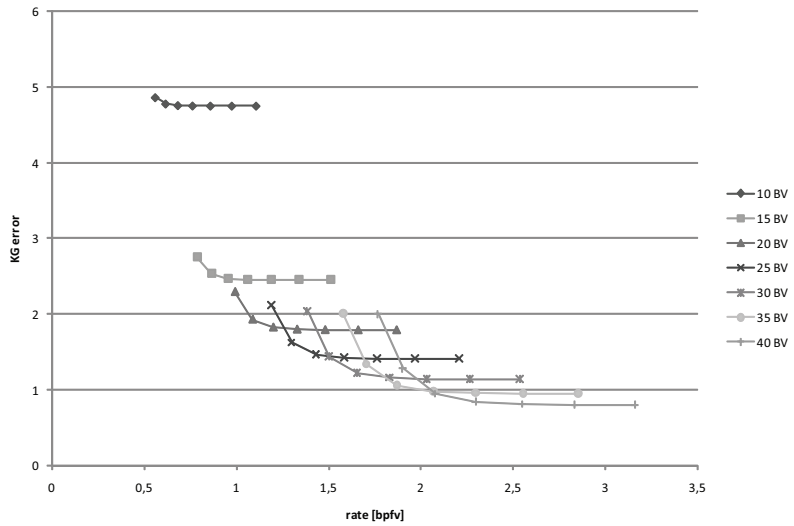

Figure 9.9: Rate/distortion curves for the **cow** sequence using the Coddyac algorithm.

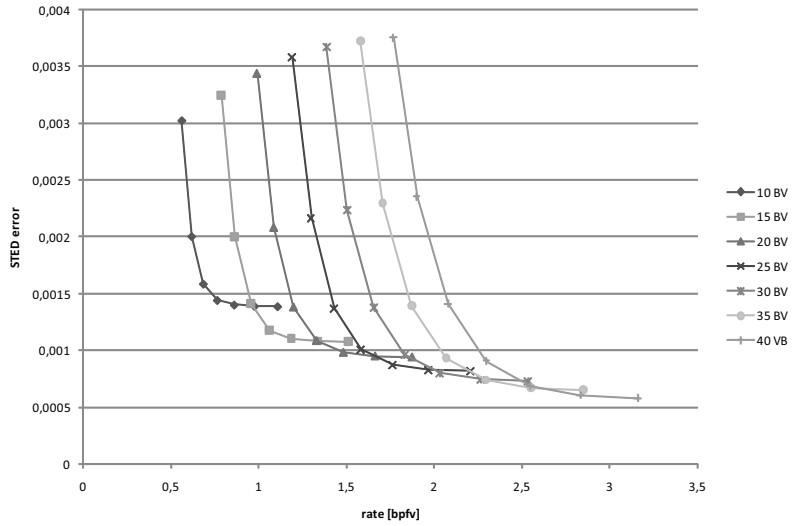

Figure 9.10: Rate/distortion curves for the **cow** sequence using the Coddyac algorithm.

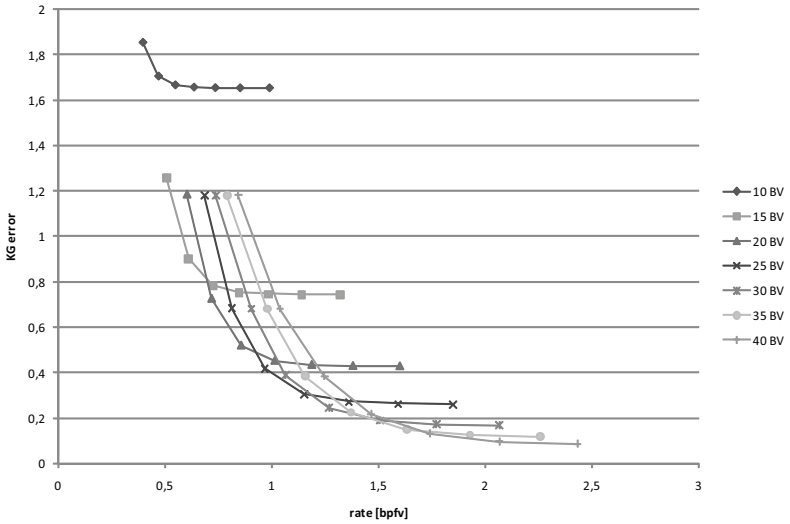

Figure 9.11: Rate/distortion curves for the **snake** sequence using the Coddyac algorithm.

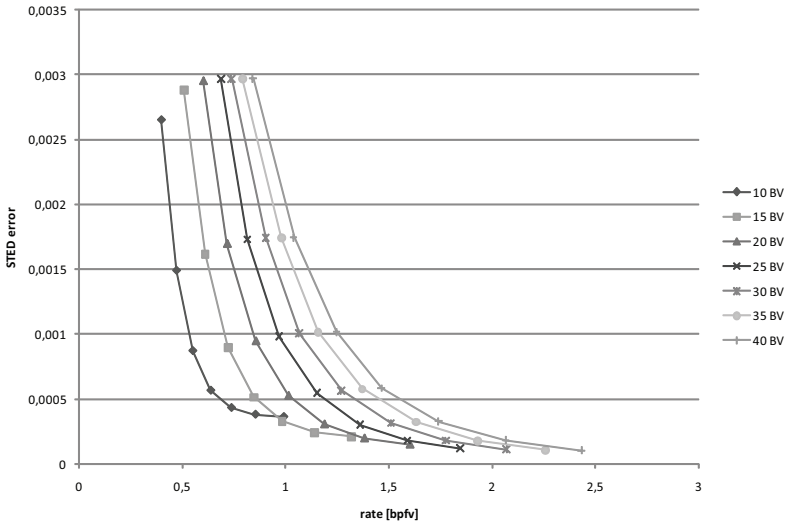

Figure 9.12: Rate/distortion curves for the **snake** sequence using the Coddyac algorithm.

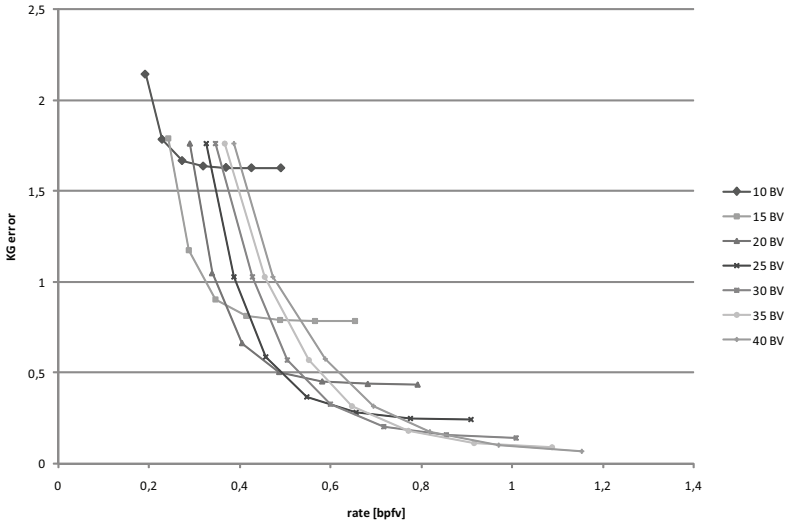

Figure 9.13: Rate/distortion curves for the **walk** sequence using the Coddyac algorithm.

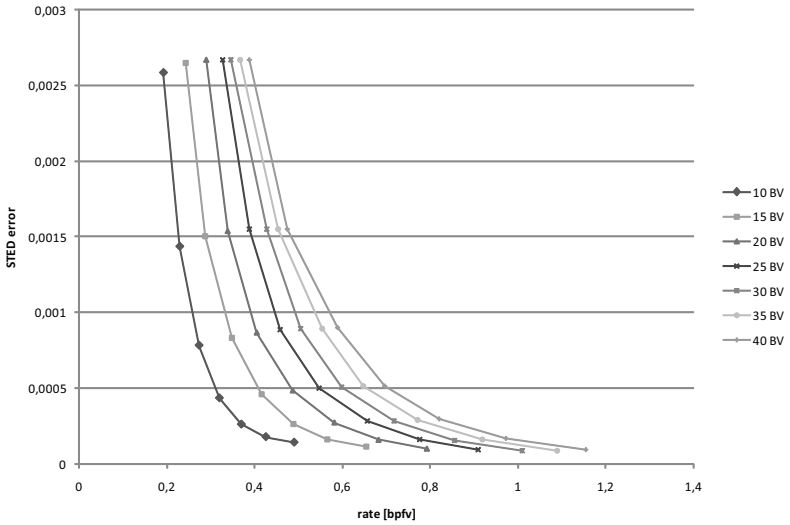

Figure 9.14: Rate/distortion curves for the **walk** sequence using the Coddyac algorithm.

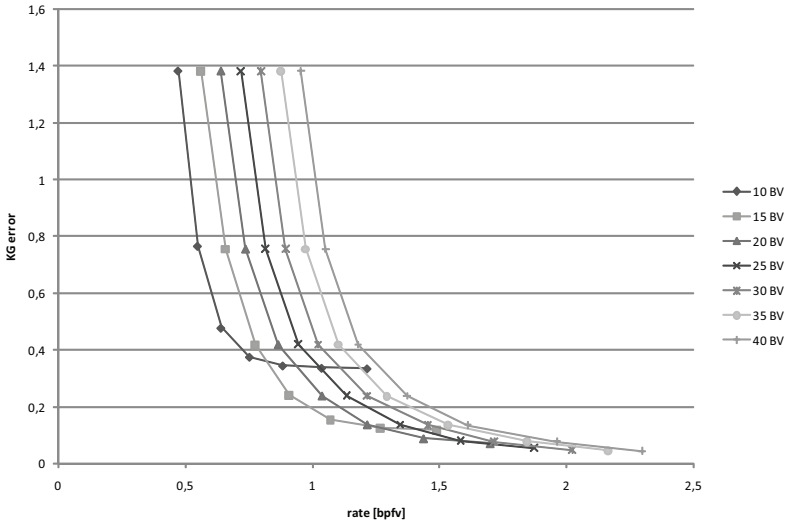

Figure 9.15: Rate/distortion curves for the **dolphin** sequence using the Coddyac algorithm.

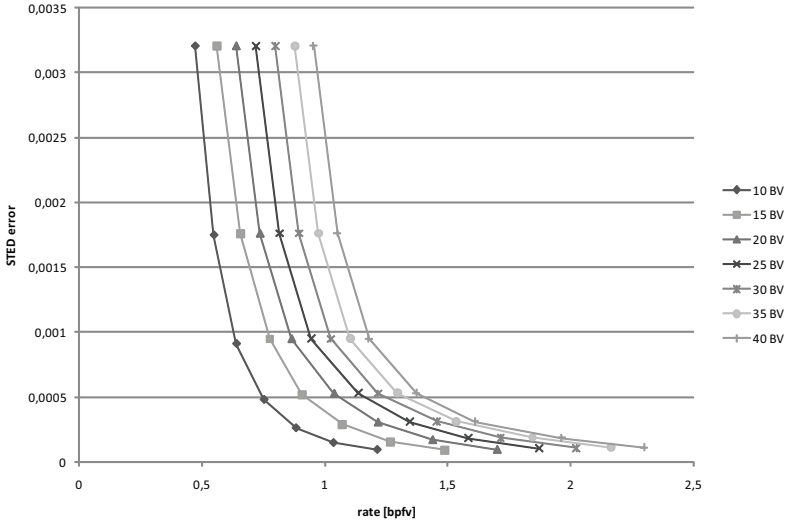

Figure 9.16: Rate/distortion curves for the **dolphin** sequence using the Coddyac algorithm.

9.2 Combined compression and simplification results

The series of figures 9.17-9.32 shows the results of the combined compression and simplification algorithm described in section 8.2. The figures show a reduction of data rate of about 15-25 percent, while the error values remain generally unchanged.

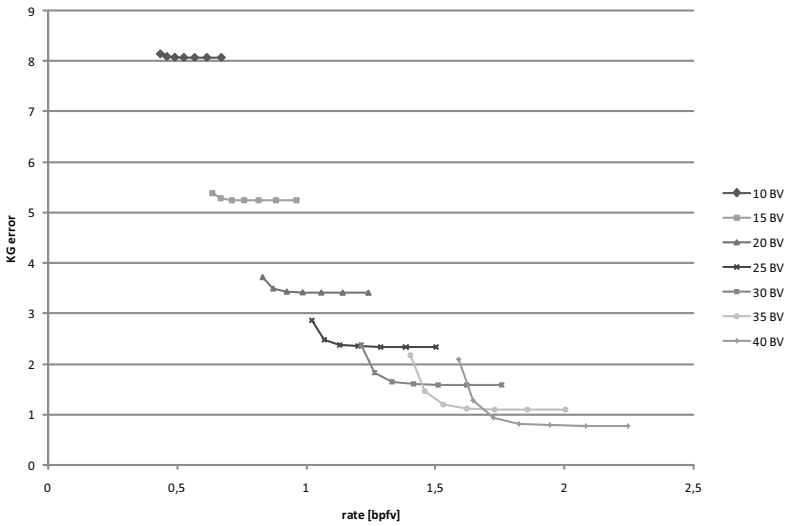

Figure 9.17: Rate/distortion curves for the **chicken** sequence using the combined compression and simplification algorithm.

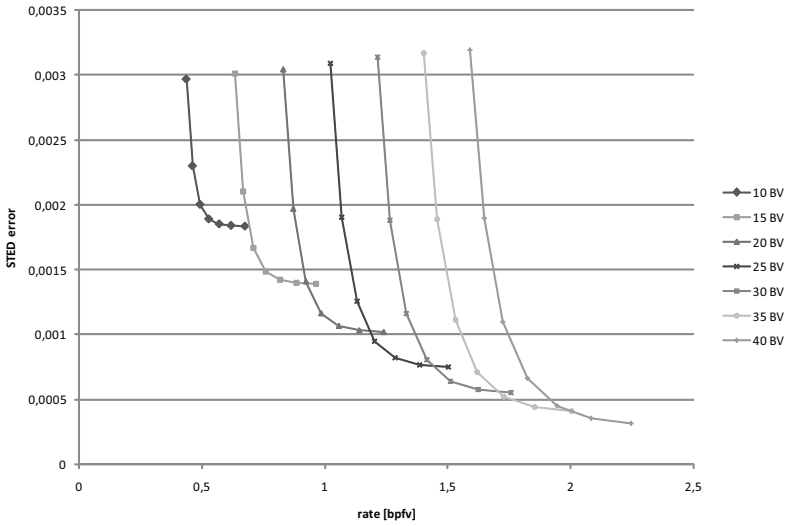

Figure 9.18: Rate/distortion curves for the **chicken** sequence using the combined compression and simplification algorithm.

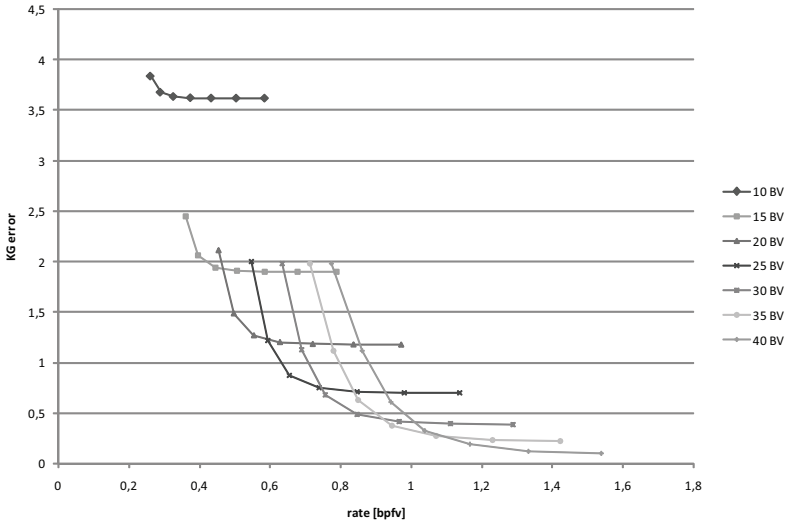

Figure 9.19: Rate/distortion curves for the **dance** sequence using the combined compression and simplification algorithm.

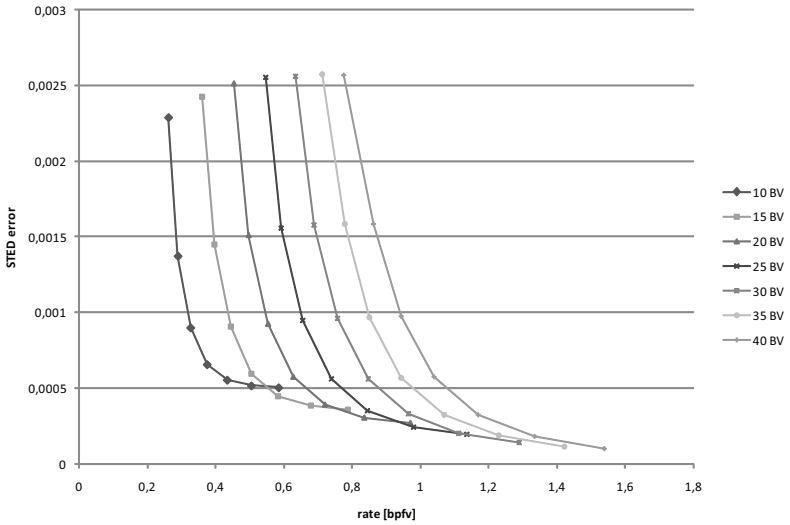

Figure 9.20: Rate/distortion curves for the **dance** sequence using the combined compression and simplification algorithm.

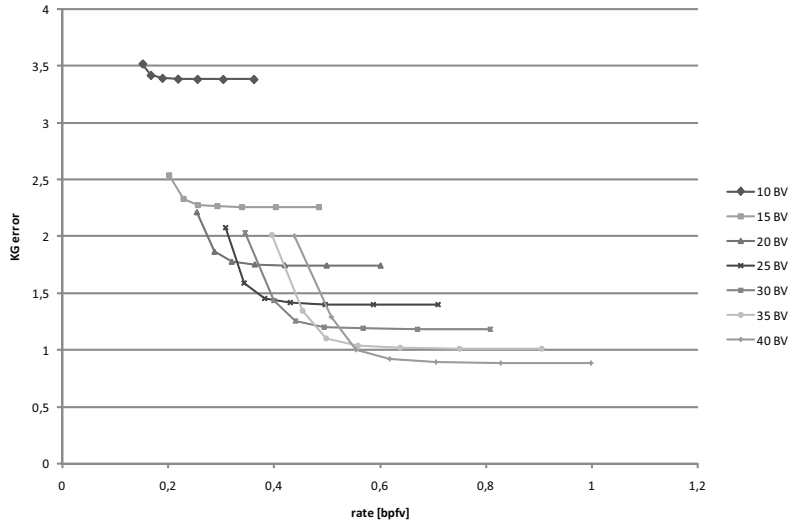

Figure 9.21: Rate/distortion curves for the **jump** sequence using the combined compression and simplification algorithm.

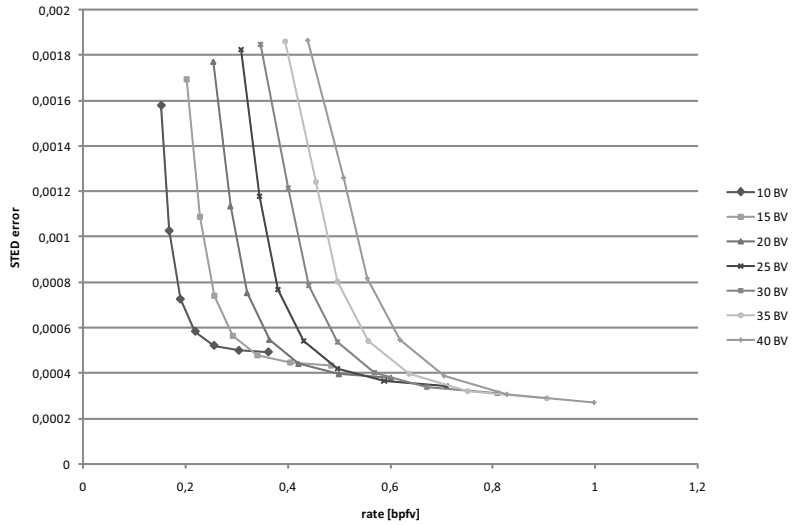

Figure 9.22: Rate/distortion curves for the **jump** sequence using the combined compression and simplification algorithm.

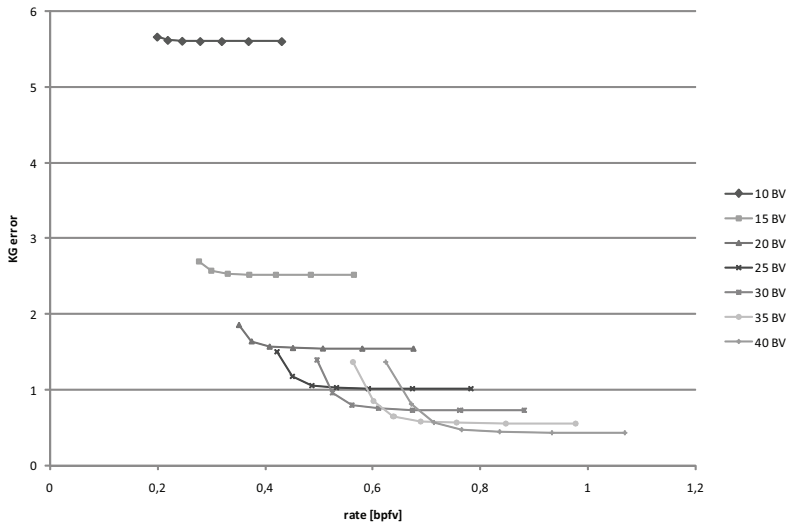

Figure 9.23: Rate/distortion curves for the **cloth** sequence using the combined compression and simplification algorithm.

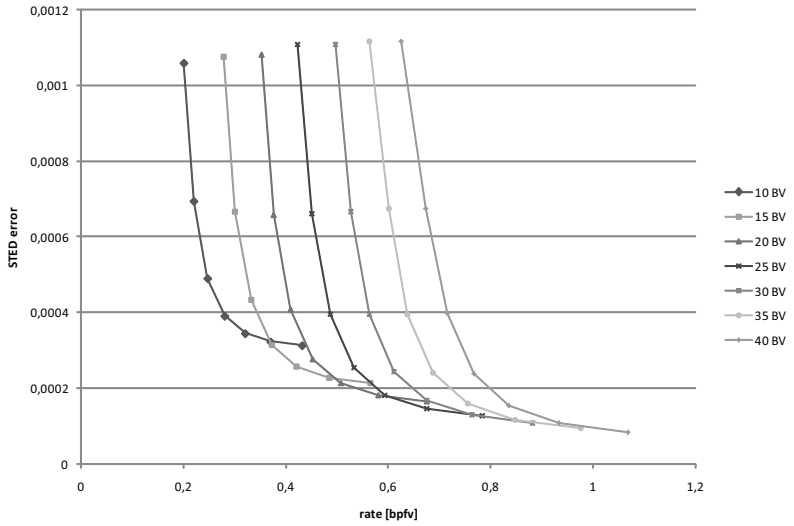

Figure 9.24: Rate/distortion curves for the **cloth** sequence using the combined compression and simplification algorithm.

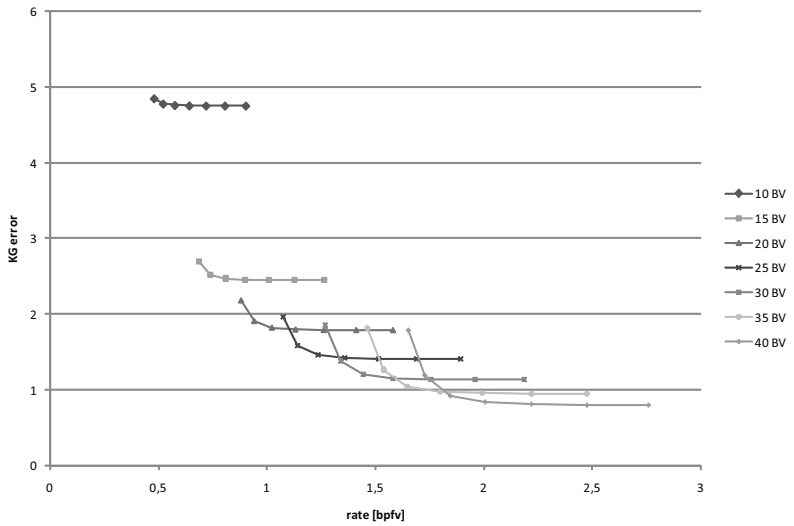

Figure 9.25: Rate/distortion curves for the **cow** sequence using the combined compression and simplification algorithm.

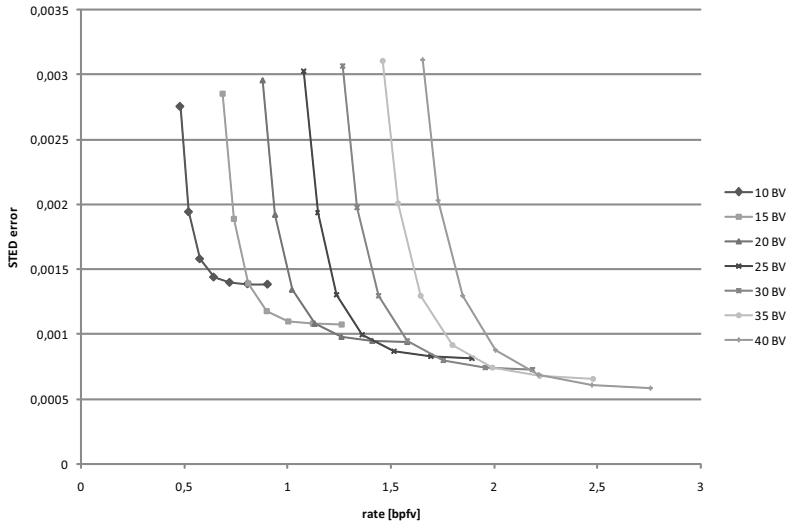

Figure 9.26: Rate/distortion curves for the **cow** sequence using the combined compression and simplification algorithm.

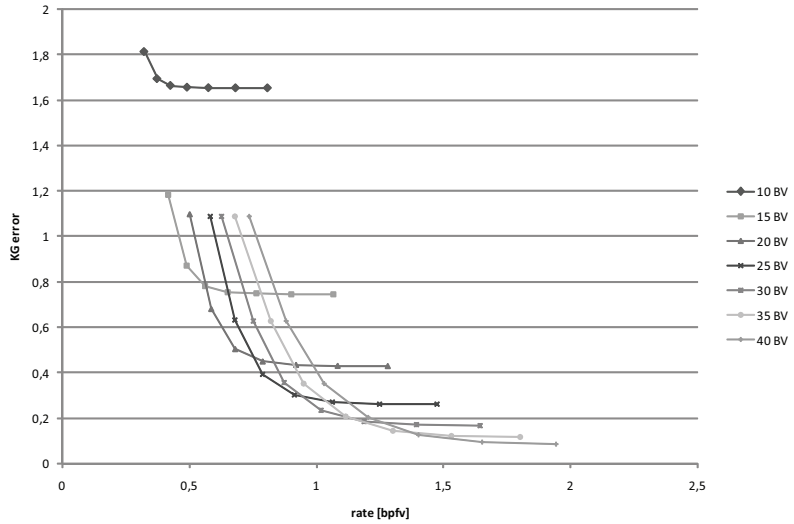

Figure 9.27: Rate/distortion curves for the **snake** sequence using the combined compression and simplification algorithm.

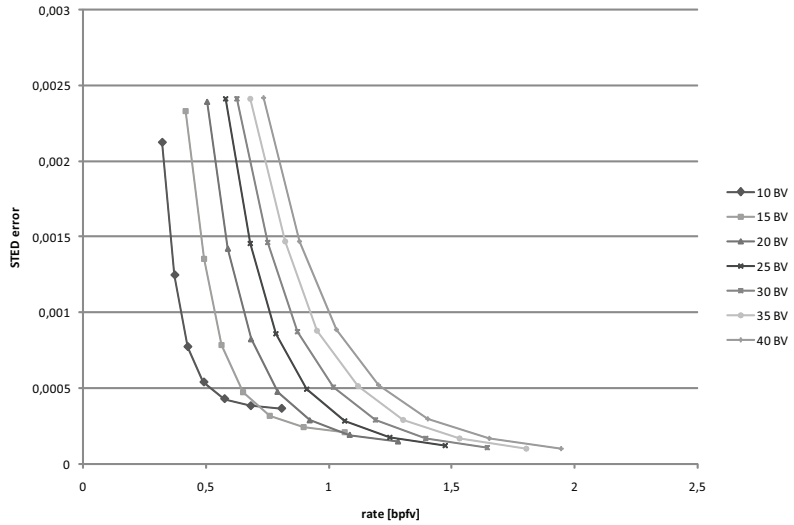

Figure 9.28: Rate/distortion curves for the **snake** sequence using the combined compression and simplification algorithm.

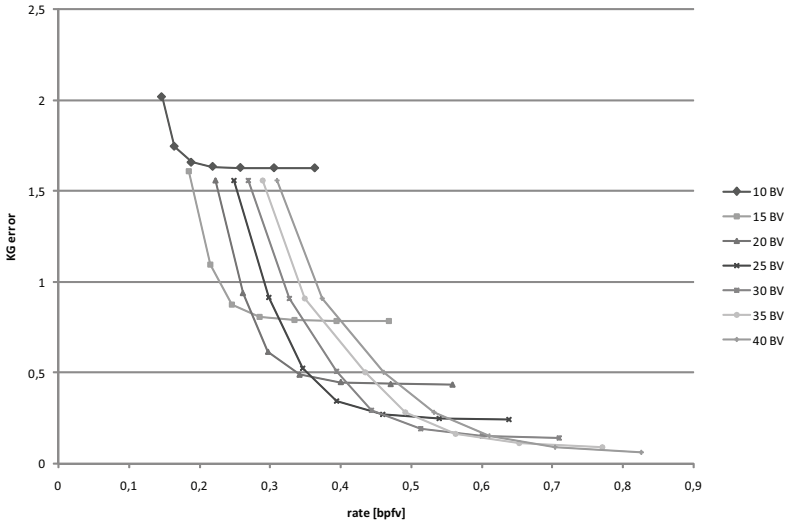

Figure 9.29: Rate/distortion curves for the **walk** sequence using the combined compression and simplification algorithm.

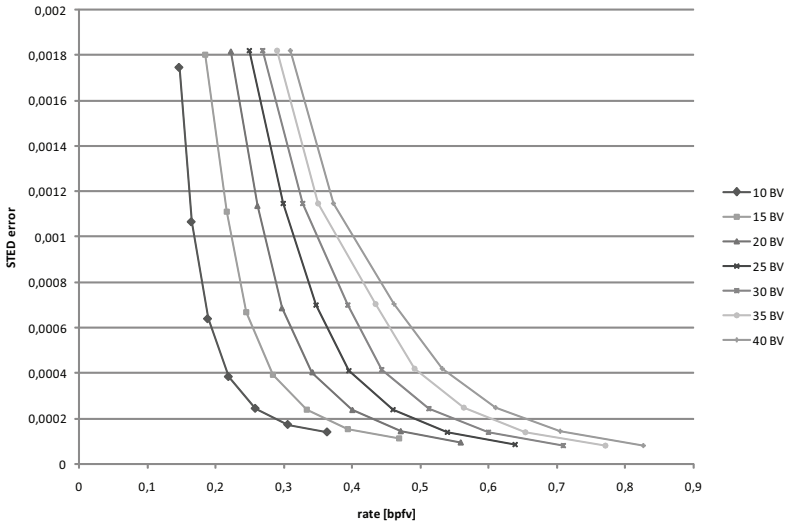

Figure 9.30: Rate/distortion curves for the **walk** sequence using the combined compression and simplification algorithm.

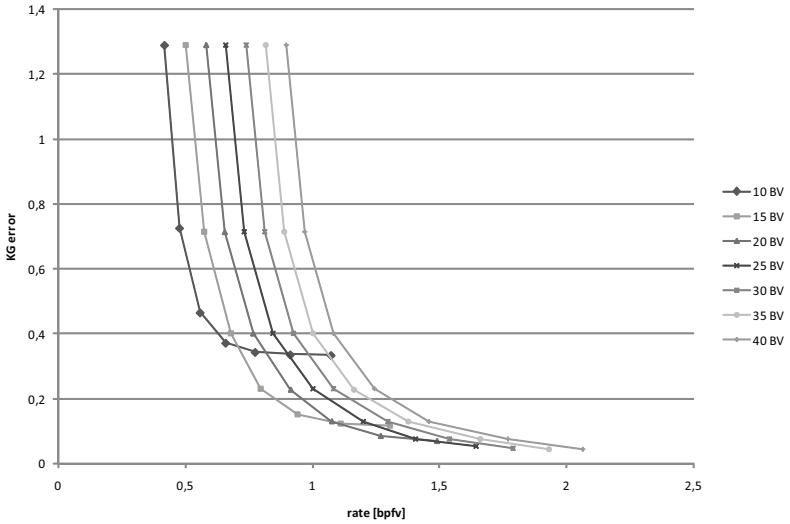

Figure 9.31: Rate/distortion curves for the **dolphin** sequence using the combined compression and simplification algorithm.

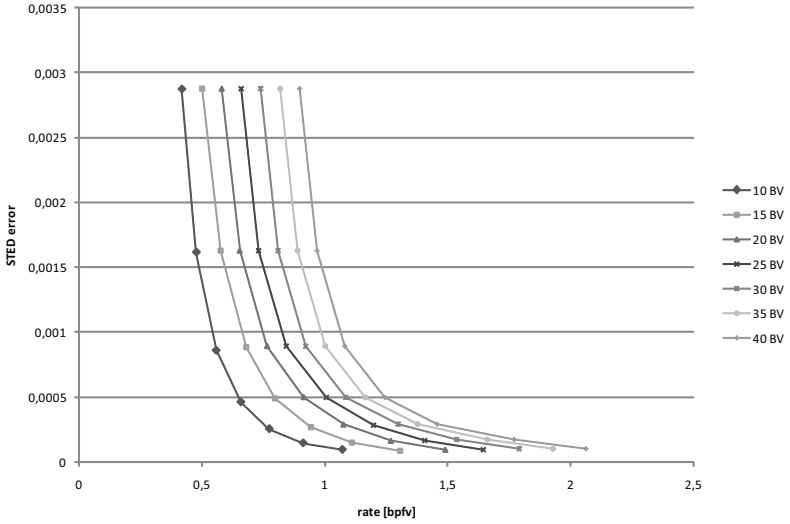

Figure 9.32: Rate/distortion curves for the **dolphin** sequence using the combined compression and simplification algorithm.

9.3 Progressive predictors results

The series of figures 9.33-9.48 shows the results of the RBF based prediction algorithm described in section 8.3.2. The figures show a further reduction of data rate of about 5-15 percent.

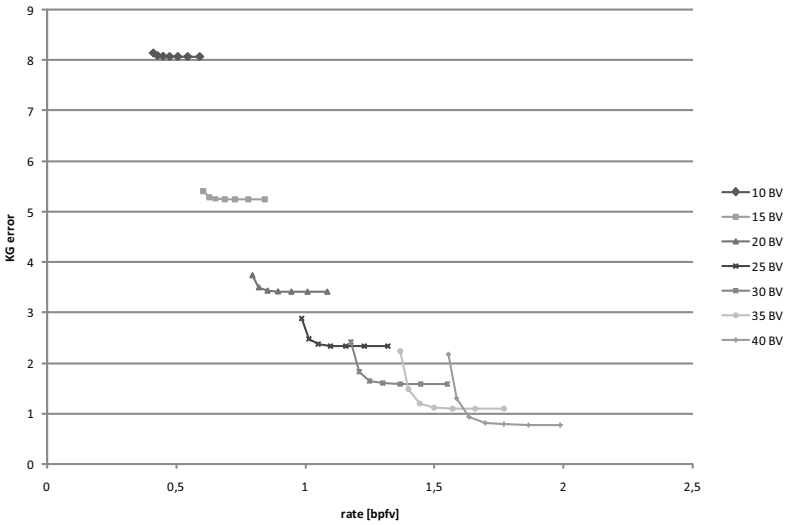

Figure 9.33: Rate/distortion curves for the **chicken** sequence using the RBF predictor based algorithm.

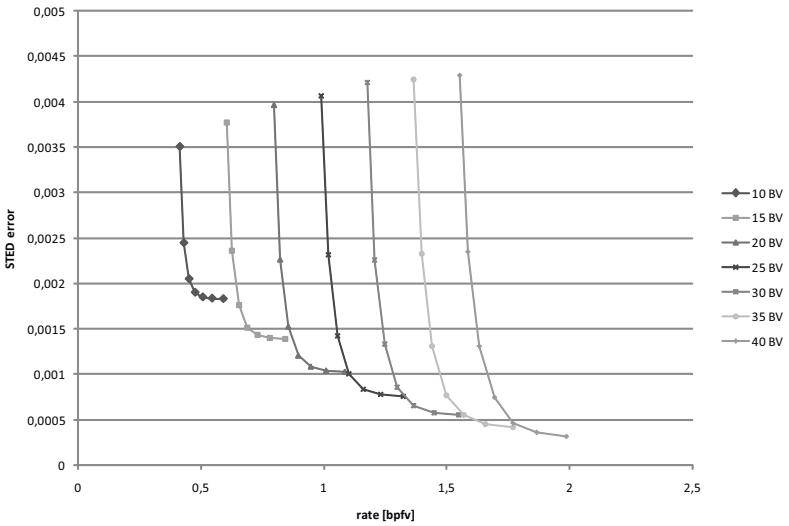

Figure 9.34: Rate/distortion curves for the **chicken** sequence using the RBF predictor based algorithm.

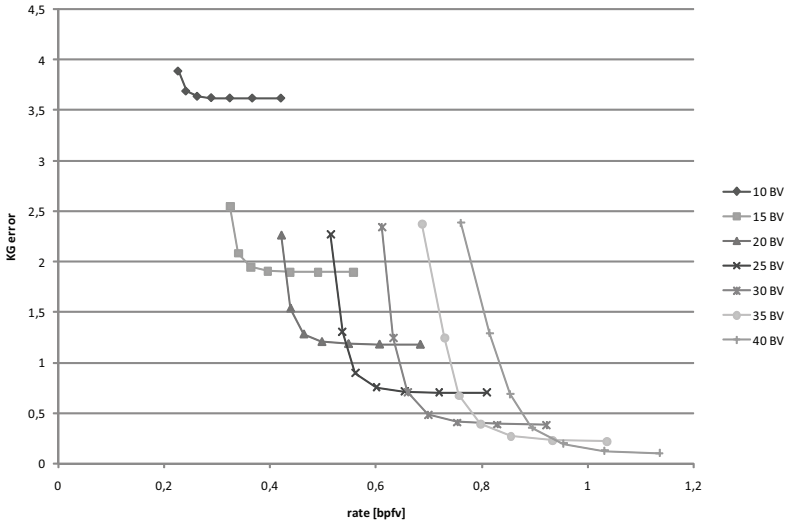

Figure 9.35: Rate/distortion curves for the **dance** sequence using the RBF predictor based algorithm.

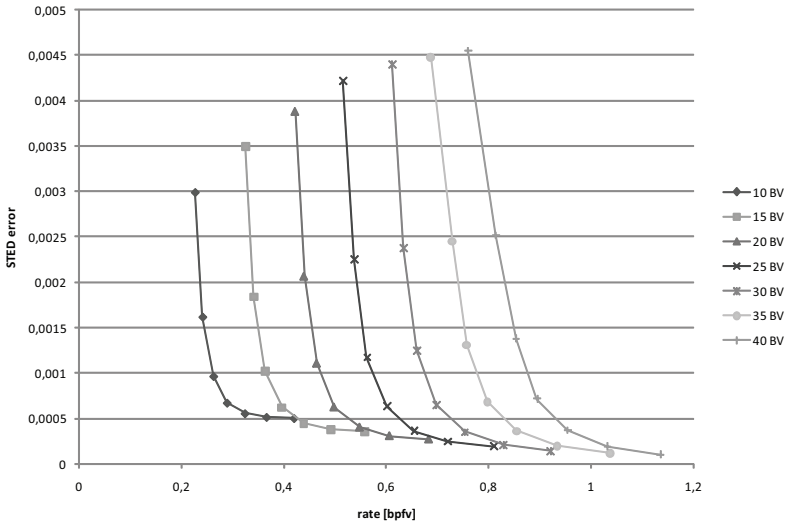

Figure 9.36: Rate/distortion curves for the **dance** sequence using the RBF predictor based algorithm.

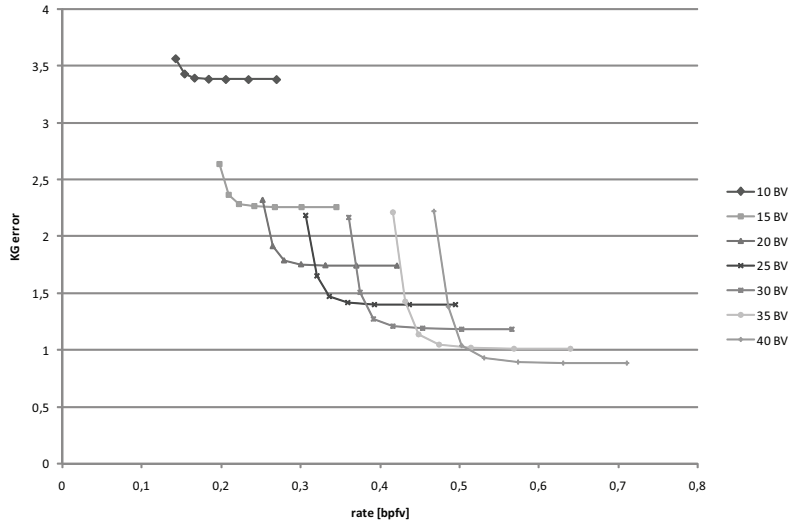

Figure 9.37: Rate/distortion curves for the **jump** sequence using the RBF predictor based algorithm.

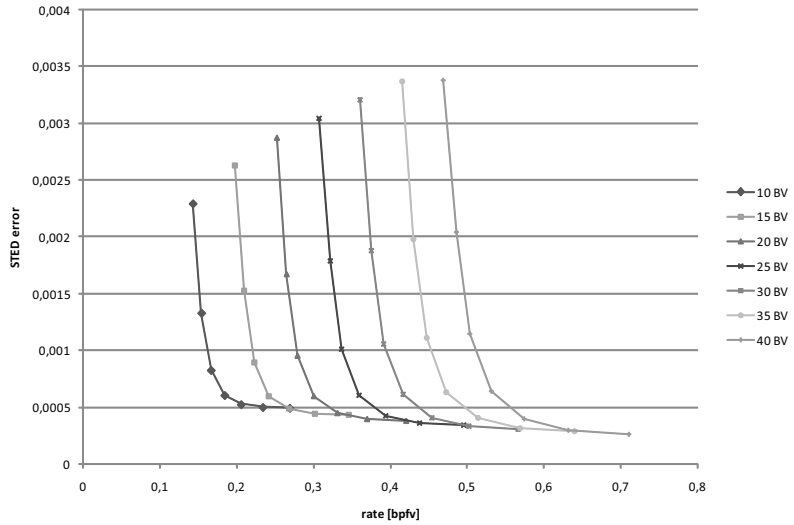

Figure 9.38: Rate/distortion curves for the **jump** sequence using the RBF predictor based algorithm.

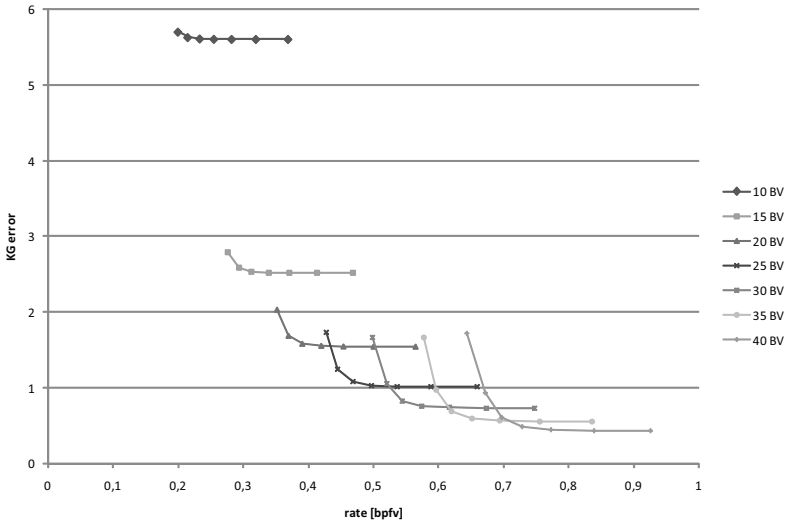

Figure 9.39: Rate/distortion curves for the **cloth** sequence using the RBF predictor based algorithm.

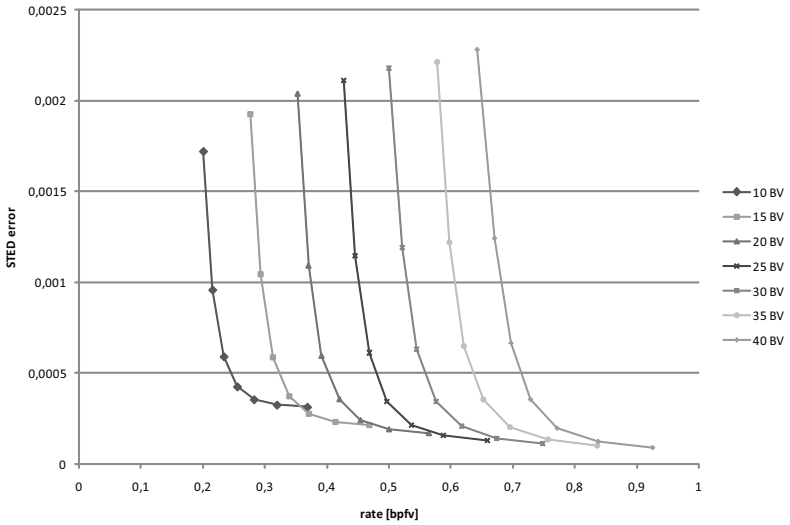

Figure 9.40: Rate/distortion curves for the **cloth** sequence using the RBF predictor based algorithm.

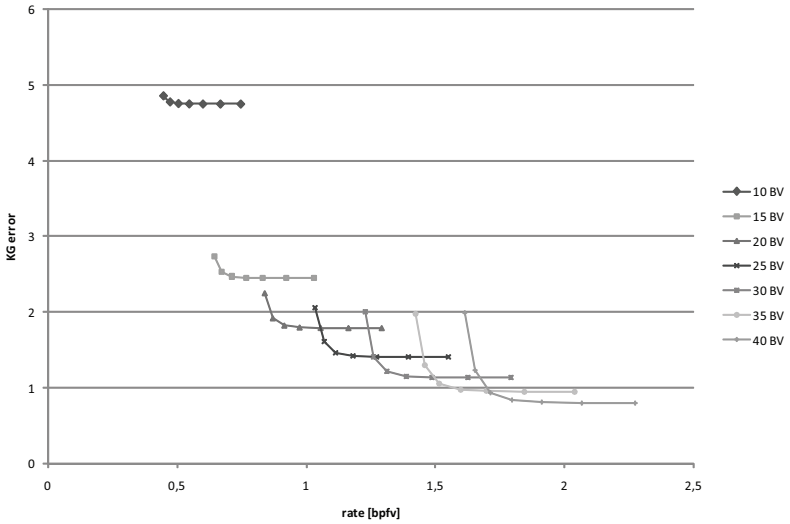

Figure 9.41: Rate/distortion curves for the **cow** sequence using the RBF predictor based algorithm.

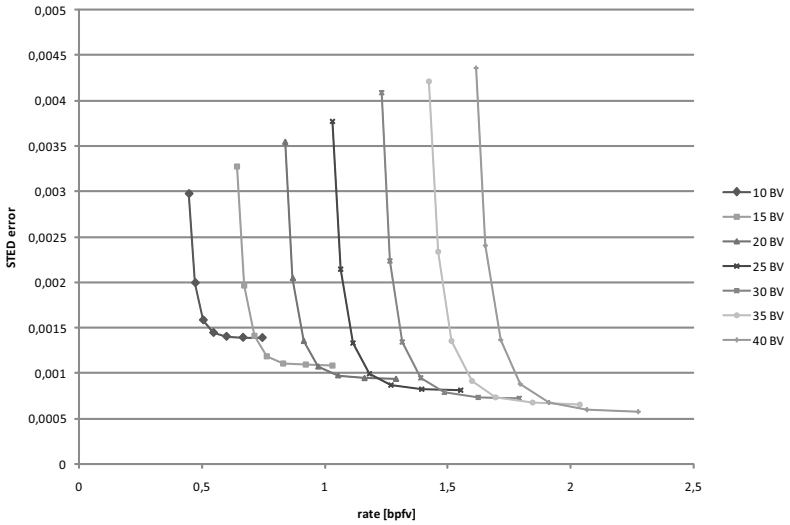

Figure 9.42: Rate/distortion curves for the **cow** sequence using the RBF predictor based algorithm.

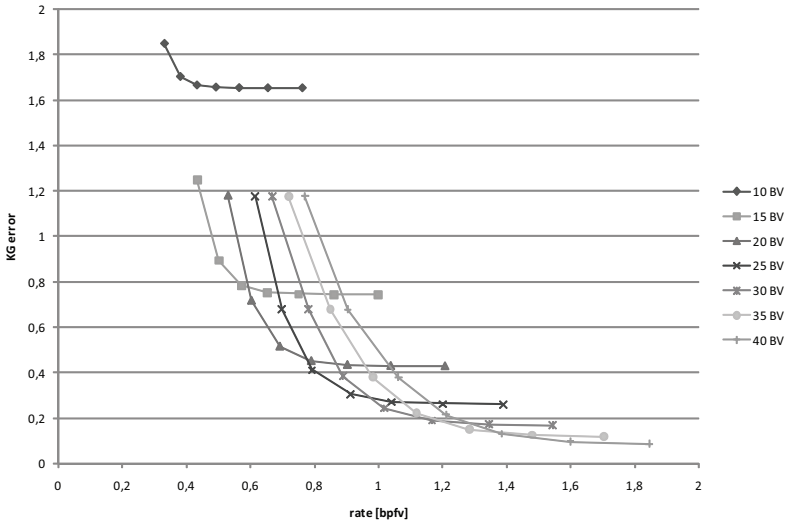

Figure 9.43: Rate/distortion curves for the **snake** sequence using the RBF predictor based algorithm.

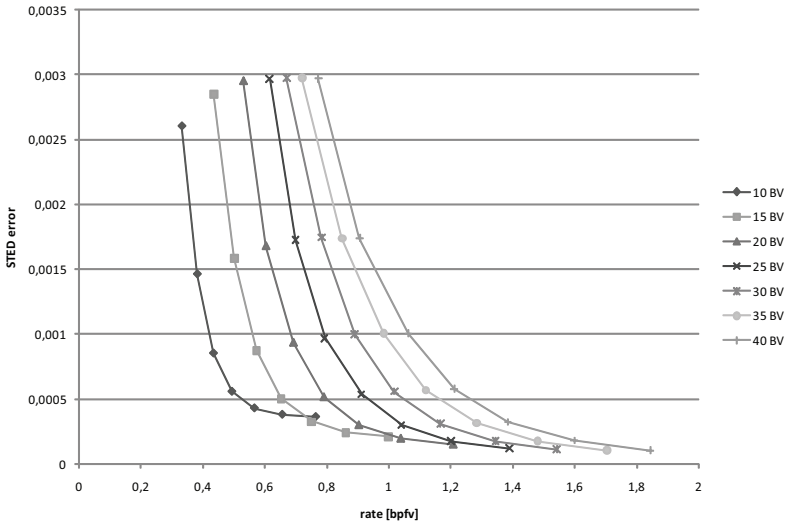

Figure 9.44: Rate/distortion curves for the **snake** sequence using the RBF predictor based algorithm.

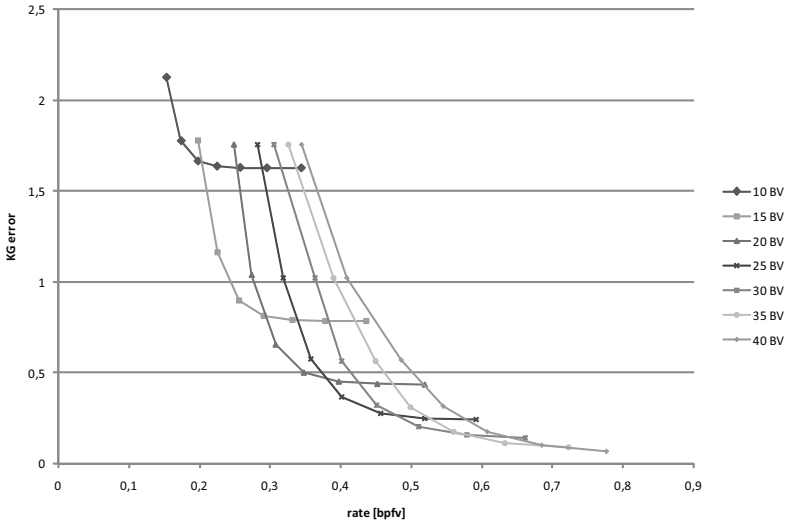

Figure 9.45: Rate/distortion curves for the **walk** sequence using the RBF predictor based algorithm.

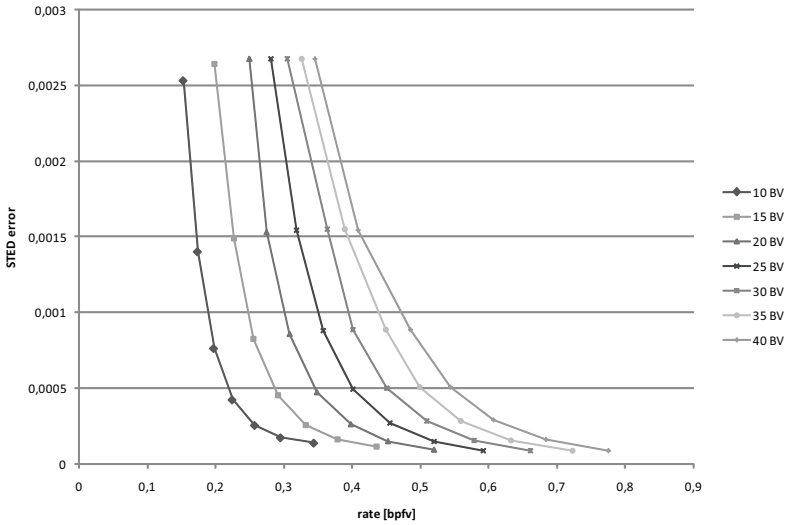

Figure 9.46: Rate/distortion curves for the **walk** sequence using the RBF predictor based algorithm.

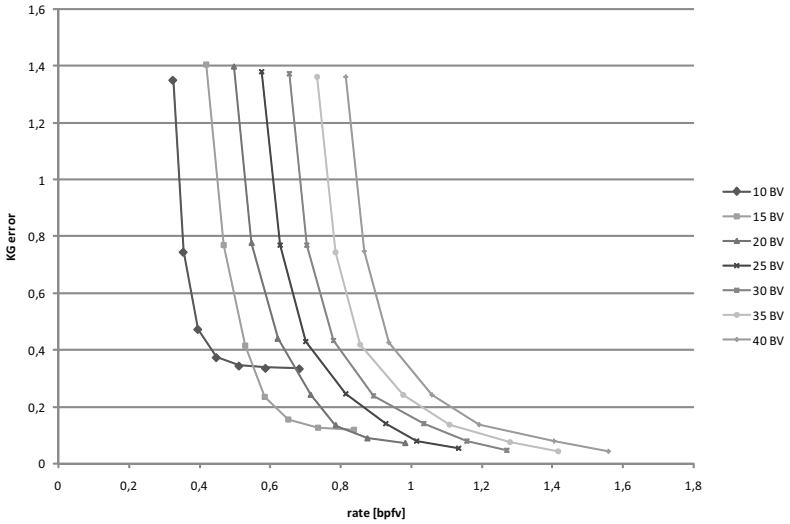

Figure 9.47: Rate/distortion curves for the **dolphin** sequence using the RBF predictor based algorithm.

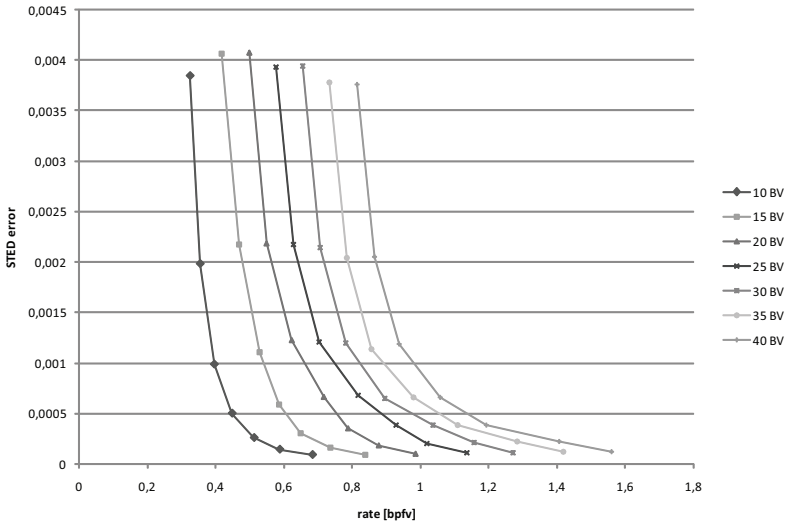

Figure 9.48: Rate/distortion curves for the **dolphin** sequence using the RBF predictor based algorithm.

9.4 Cobra results and considerations

We have tested the basis compression algorithm derived in section 8.4 with an implementation of the Coddyac compression scheme. We have performed experiments with direct encoding of the basis (64 bits per double precision value), LPC predictors of order from 1 to 4 and the four physical predictors defined by (8.18), (8.19), (8.21) and (8.22). We have also used both uniform and non-uniform quantization to test the efficiency of our scheme.

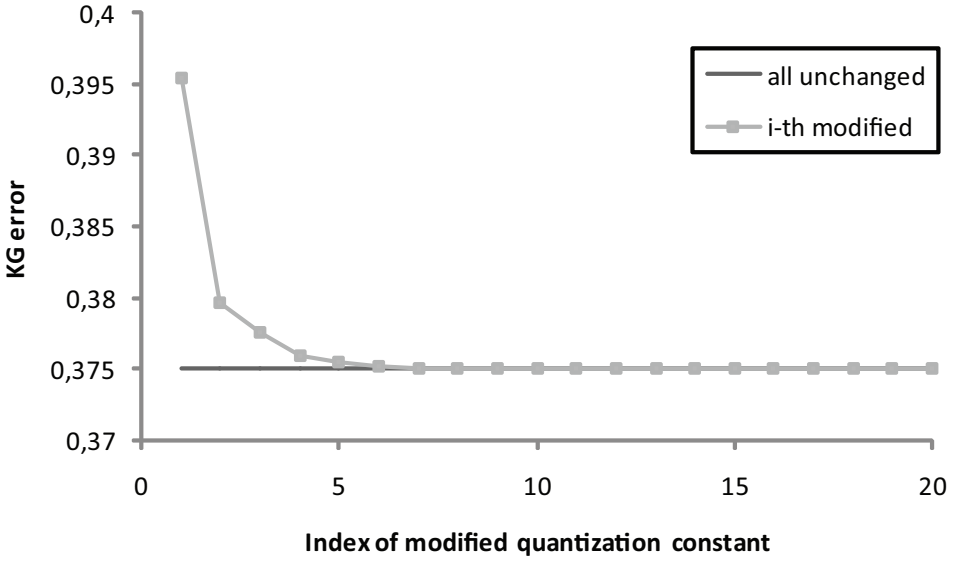

Figure 9.49: Experiment with uniform quantization.

First, we will test the efficiency of the non-uniform quantization using the chicken run sequence. We have configured the Coddyac algorithm so that the quantization of coefficients is very fine, and thus the error is mainly introduced by basis quantization. First, we have evaluated the error for the uniform and non-uniform quantization. Subsequently, we have observed what happens, if the quantization constant used for i-th basis vector is tripled.

Figure 9.49 shows the dependence of KG error on the index of artificially increased quantization for the case of uniform quantization. The error value constantly decreases with the index, showing that tripling the quantization constant used for the first basis vector has a radical effect on the error, while tripling for example the tenth quantization constant has virtually no effect at all. This means that the **basis vectors with higher indices are quantized too finely**, their quantization can be much coarser without significant influence on the error.

On the other hand, figure 9.50 shows the result of the experiment using the determined non-uniform quantization constants. It shows that tripling any of

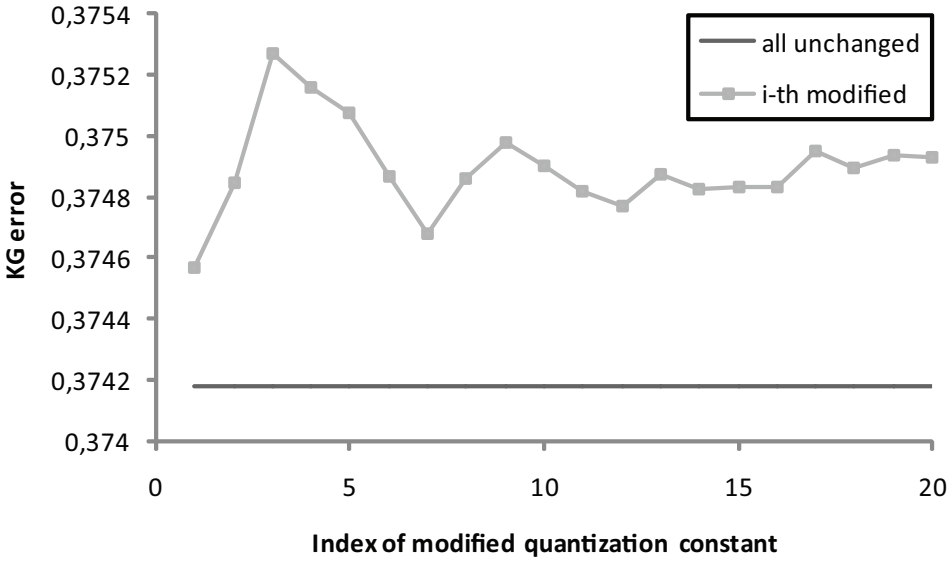

Figure 9.50: Experiment with non-uniform quantization.

the constants introduces error of roughly the same magnitude, which proves that the quantization constants are well balanced and no basis vector is quantized too finely.

Now we will compare the efficiency of the predictors. We will compare the entropies of residuals, using the non-uniform quantization, which has been shown to work well. Table 9.2 shows the entropies for the LPC predictor of order from 1 to 4 (LPC1-LPC4) and the physical predictors (Cob1-Cob4). The table shows that the physical predictors outperform the LPC predictors for all the available datasets. Note that the sum of squared residuals has decreased with the order of LPC predictor, and the physical predictor has always produced residuals with higher sum of squares than the LPC of corresponding order, which is the expected behavior.

Usually the best results are provided by linear motion or accelerated motion predictors, with only two exceptions: for the dolphin dataset at 25 basis vectors, the fourth order predictor works best, and for the snake sequence at 50 basis vectors the first order predictor works best. The dolphin case is caused by the character of the dataset, where there are basically only two important basis trajectories of the sine and cosine shape. These trajectories are very smooth, and thus well predicted by a higher order predictor. The snake case is the exact opposite - the movement of the snake is very fast and shaky, the basis trajectories are not smooth at all, and therefore they are best predicted by simple delta coding. The most efficient overall approach is to test the performance of the physical predictors in the encoder, and then spend two bits to determine the predictor to

dataset	basis	KG	LPC1	LPC2	LPC3	LPC4	Cob1	Cob2	Cob3	Cob4
jump	50	0,78	9,22	8,46	8,21	8,22	9,11	8,14	7,96	8,30
jump	25	1,42	9,12	8,18	7,69	7,66	9,01	7,82	7,41	7,59
chicken	50	0,48	9,87	9,90	9,87	9,87	9,10	8,11	8,21	8,70
chicken	25	2,36	9,38	9,30	9,29	9,30	8,86	7,77	7,83	8,32
cow	50	0,66	10,77	10,59	10,57	10,57	10,69	10,45	10,62	11,09
cow	25	1,42	11,06	10,75	10,66	10,66	11,00	10,55	10,56	10,90
dolphin	50	0,24	6,07	6,06	6,07	6,08	5,71	5,55	5,55	5,79
dolphin	25	0,24	7,12	6,97	6,76	6,73	6,68	6,01	5,61	5,54
dance	50	0,36	10,08	10,07	10,06	10,07	9,93	9,61	9,84	10,43
dance	25	0,76	10,22	9,60	9,47	9,48	10,14	9,43	9,41	9,89
cloth	50	0,38	9,61	9,53	9,49	9,50	8,93	7,78	7,78	8,33
cloth	25	1,03	9,63	9,26	9,07	9,11	9,26	7,72	7,50	7,95
snake	50	0,22	10,30	10,20	10,11	10,11	10,04	10,23	10,87	11,62
snake	25	0,31	10,38	10,42	10,21	10,15	10,18	9,99	10,46	11,14
walk	50	0,32	8,93	8,80	8,82	8,83	8,78	8,40	8,79	9,49
walk	25	0,37	9,06	8,53	8,55	8,57	9,00	8,29	8,57	9,22

Table 9.2: Residual entropies for various predictors. The green color coded values denote low entropy, the red values denote high entropy within the given row.

be used.

We also give the relations of different parts of encoded data for all the datasets in figure 9.51. All the datasets are compressed at KG error of cca 0.3, and we are giving results for direct encoding (8 bytes per float number), encoding suggested by Amjoun (uniform quantization) and our algorithm.

The final set of figures 9.52-9.63 shows the results of the basis compression algorithm presented in section 8.4 combined with RBF based predictor from section 8.3.2. The most interesting graph is the result of the final dolphin test in 9.62 and 9.63, where using cobra basis compression there is almost no increase of bitrate with increasing number of basis vectors. This is caused by the fact that there is practically only one sinusoidal trajectory, which drives most of the vertices, and the remaining trajectories are unimportant, and therefore highly compressed. This graph nicely demonstrates the efficiency of the cobra compression, and of the PCA in general.

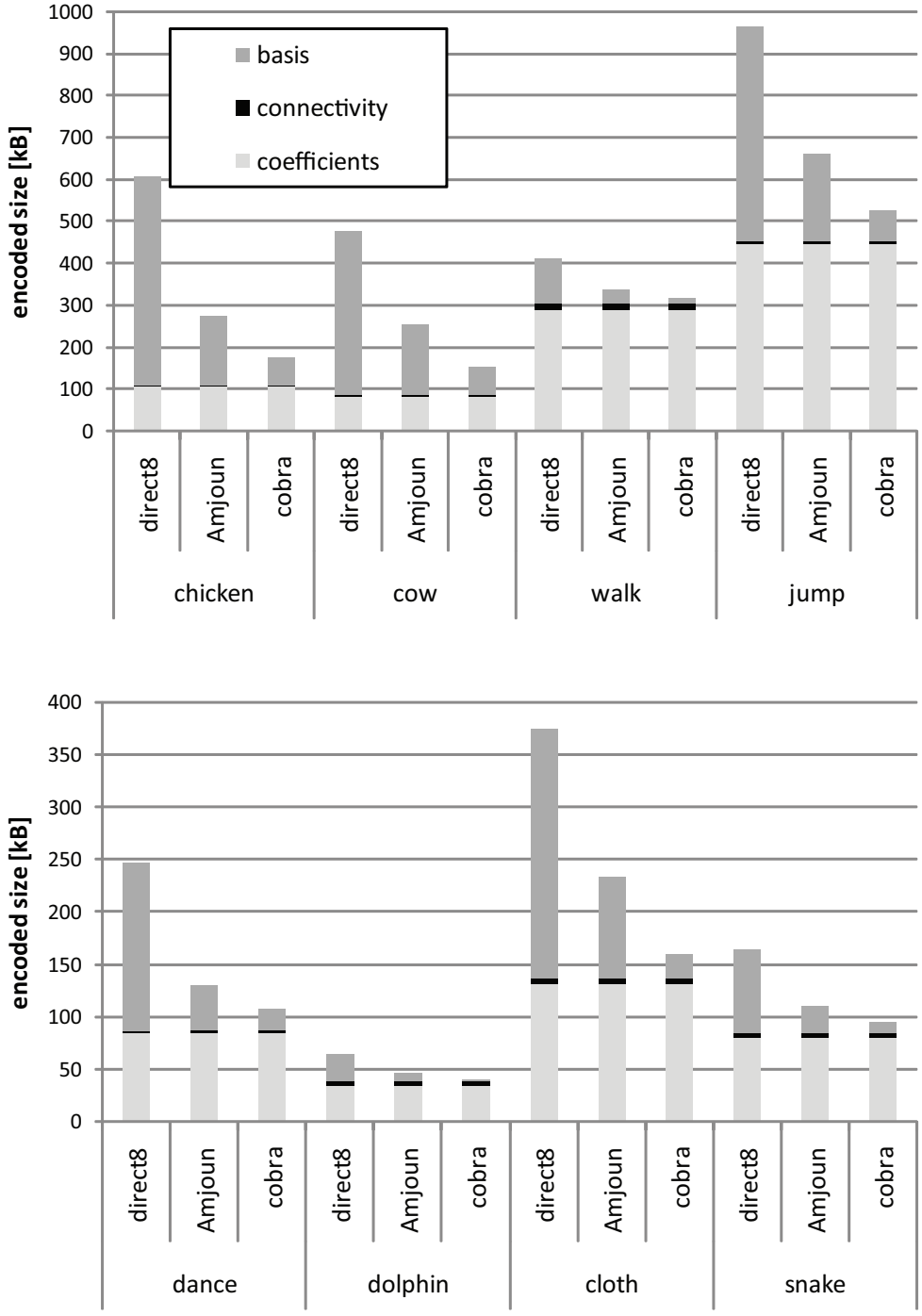

Figure 9.51: Relations of coefficients, basis and connectivity for various datasets. Direct8 denotes direct encoding of 8-bytes per double value, Amjoun denotes the result of quantization without prediction, and cobra denotes our result.

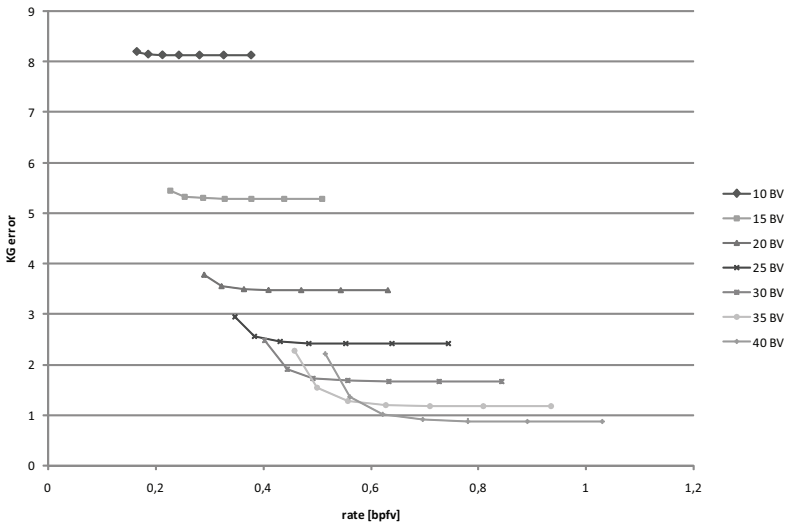

Figure 9.52: Rate/distortion curves for the **chicken** sequence using the RBF predictor based algorithm with cobra basis encoding.

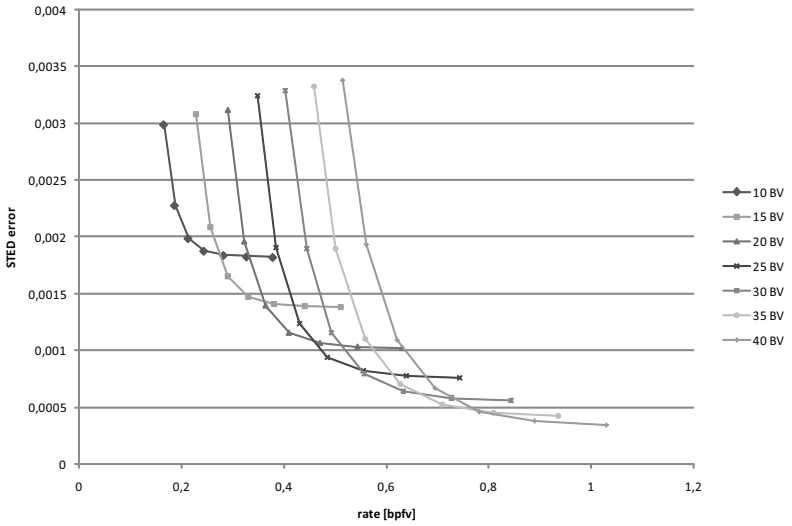

Figure 9.53: Rate/distortion curves for the **chicken** sequence using the RBF predictor based algorithm with cobra basis encoding.

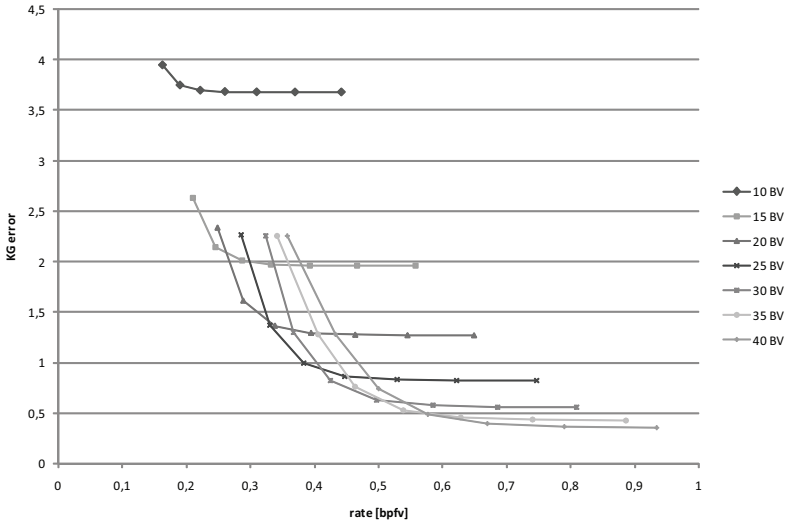

Figure 9.54: Rate/distortion curves for the **dance** sequence using the RBF predictor based algorithm with cobra basis encoding.

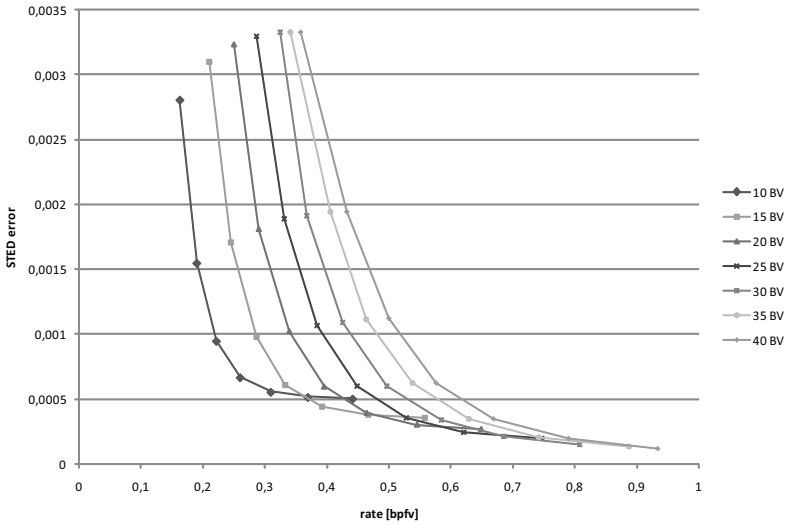

Figure 9.55: Rate/distortion curves for the **dance** sequence using the RBF predictor based algorithm with cobra basis encoding.

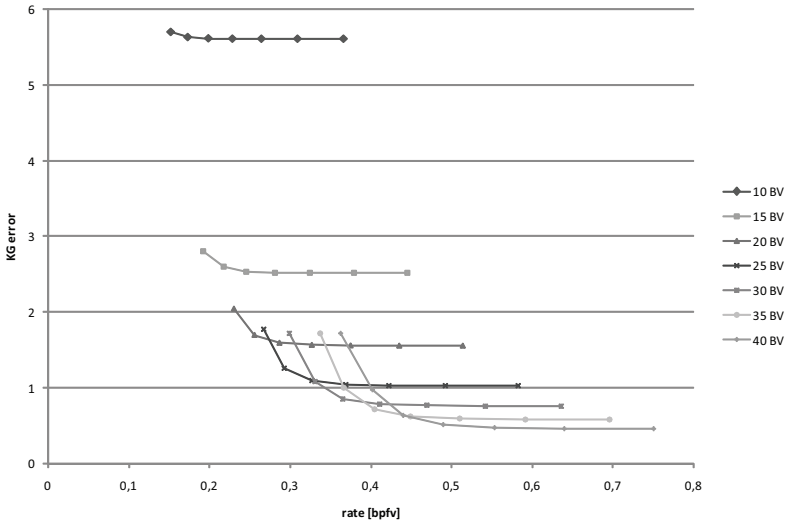

Figure 9.56: Rate/distortion curves for the **cloth** sequence using the RBF predictor based algorithm with cobra basis encoding.

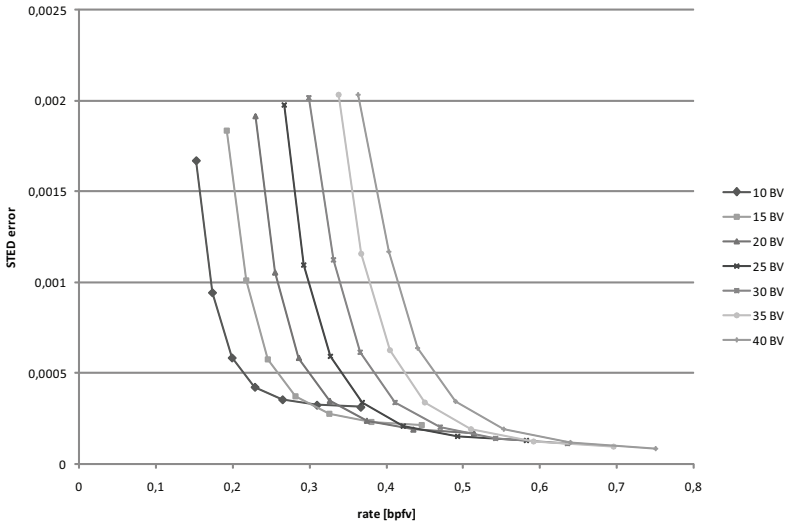

Figure 9.57: Rate/distortion curves for the **cloth** sequence using the RBF predictor based algorithm with cobra basis encoding.

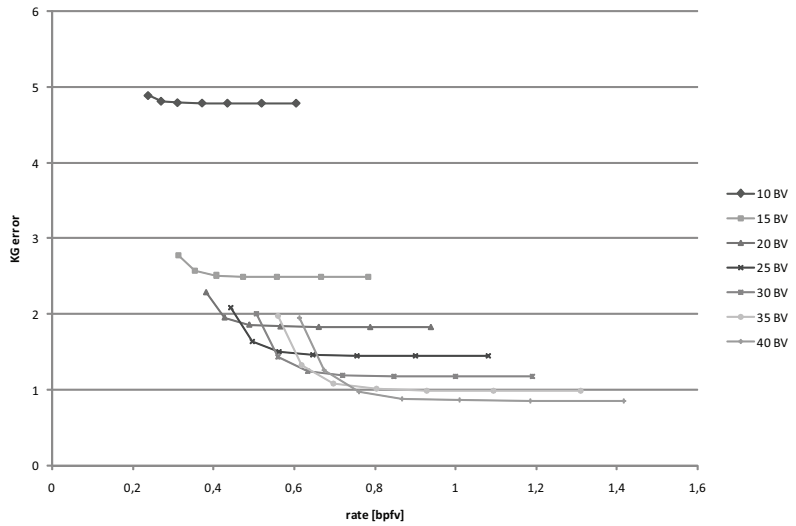

Figure 9.58: Rate/distortion curves for the **cow** sequence using the RBF predictor based algorithm with cobra basis encoding.

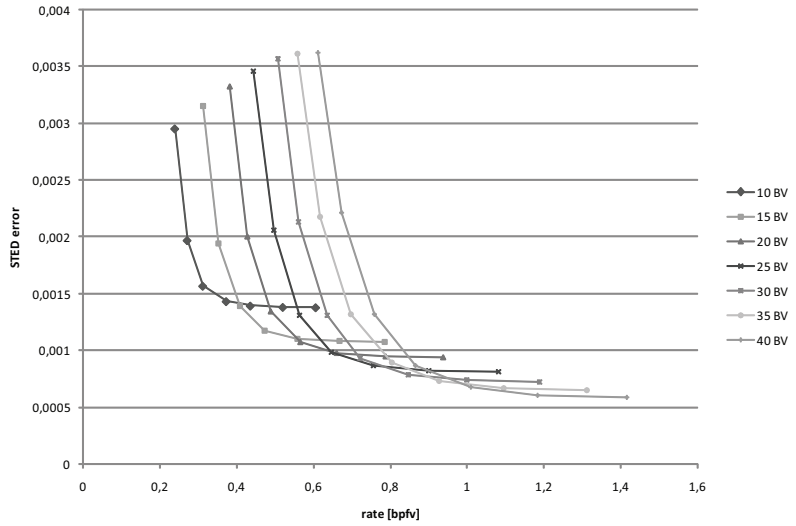

Figure 9.59: Rate/distortion curves for the **cow** sequence using the RBF predictor based algorithm with cobra basis encoding.

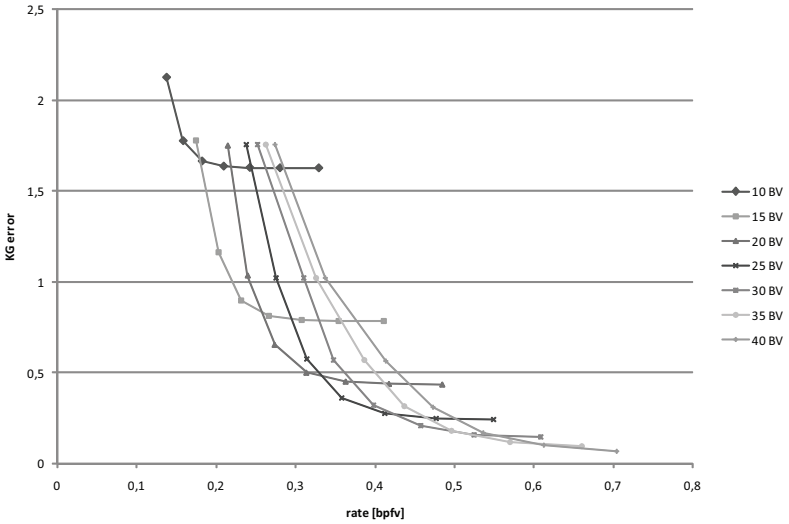

Figure 9.60: Rate/distortion curves for the **walk** sequence using the RBF predictor based algorithm with cobra basis encoding.

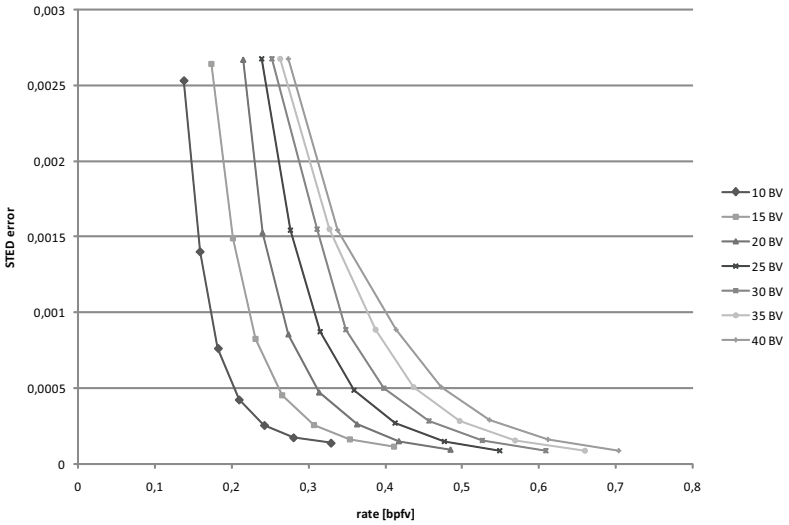

Figure 9.61: Rate/distortion curves for the **walk** sequence using the RBF predictor based algorithm with cobra basis encoding.

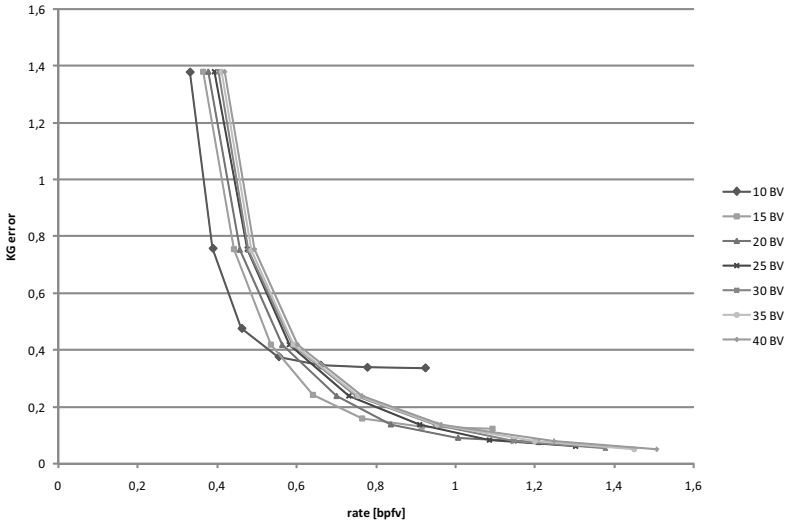

Figure 9.62: Rate/distortion curves for the **dolphin** sequence using the RBF predictor based algorithm with cobra basis encoding.

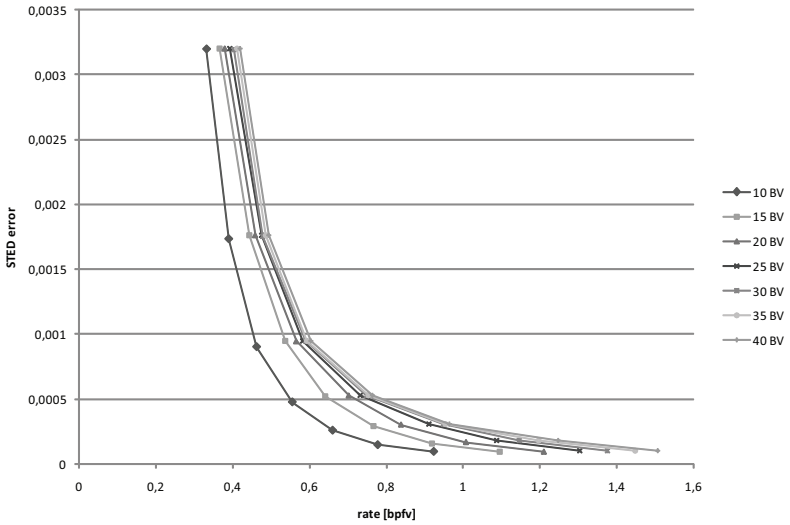

Figure 9.63: Rate/distortion curves for the **dolphin** sequence using the RBF predictor based algorithm with cobra basis encoding.

	NA	RBF	Cobra
chicken	4,7%	12,5%	60,4%
dance	22,2%	36,1%	53,0%
dolphin	10,7%	30,1%	36,2%
cow	13,4%	23,8%	57,4%
snake	19,3%	19,5%	39,0%
cloth	35,9%	40,1%	44,8%
walk	28,8%	30,3%	36,4%
average	**19,3%**	**27,5%**	**46,8%**

Table 9.3: Comparison of the proposed methods. The numbers represent the relative decrease of the bitrate with respect to the Coddyac algorithm, while assuming that the distortion has not changed significantly.

9.5 Comparison of the proposed compression methods

The table 9.3 shows the comparison of the improvement of the proposed methods with respect to plain original Coddyac algorithm. The percentages shown were obtained by averaging the relative reduction of the data rate in all the experiments presented in the previous sections, i.e. with quantization constant spanning from 7 to 13 and with basis sizes of 10-40 basis vectors.

We assume that the distortion in the experiments has not changed, which is roughly true, because the error most significantly depends on the compression parameters such as the quantization constant and basis length. However, in some of the NA and RBF experiments we have observed some error reduction, which is due to the higher robustness of interpolation with respect to extrapolation. On the other hand, Cobra experiments have produced a slight increase of the error, which is due to the approximative representation of the PCA basis. However, all the error offsets were lower than 2%.

9.6 Performance comparison against the state of the art

Finally, we give a comparison with existing methods. It is difficult to make such comparison, due to two reasons:

1. There is no consensus in the literature about the error measure that should be used. Quite often the exact measure used is not exactly defined. Therefore we only compare our results to papers that have shown results using the KG error measure. However, as shown in chapter 7, such measurement says very little about the actual behavior of the decompressed animations. Moreover, even with papers that do use KG error, we have found out that

some authors misinterpret the definition of KG error, and thus their measures cannot be used for comparison.

2. There is no consensus about what is the area of interesting rates and distortions that should be investigated. Therefore we have performed measurements in such areas where we have results of competing methods, although we believe that in some cases the reconstruction is far too fine. Most of the papers focus on the bitrates of 1-5 bpfv, while our experiments have shown that it is possible to achieve results that are hard to distinguish from the original at bitrates between 0.5 - 0.75 bpfv.

Figures 9.64-9.68 show the comparison with the FAMC coder by Mamou and Stefanoski[1], described in section 3.1.14. This state of the art coder is based on techniques described in sections 3.1.7 and 3.1.13, and it has been adopted by the MPEG consortium as a standard in 2007. The tables show that the combination of RBFP predictor and COBRA basis compression scheme outperforms the FAMC coder in all the testing sequences. Because the FAMC coder has been previously shown to outperform all the existing techniques in the rate/distortion ratio, we assume that our final combination of RBFP and COBRA represents the most efficient algorithm to date. The rest of the comparisons is summarized by table 9.4.

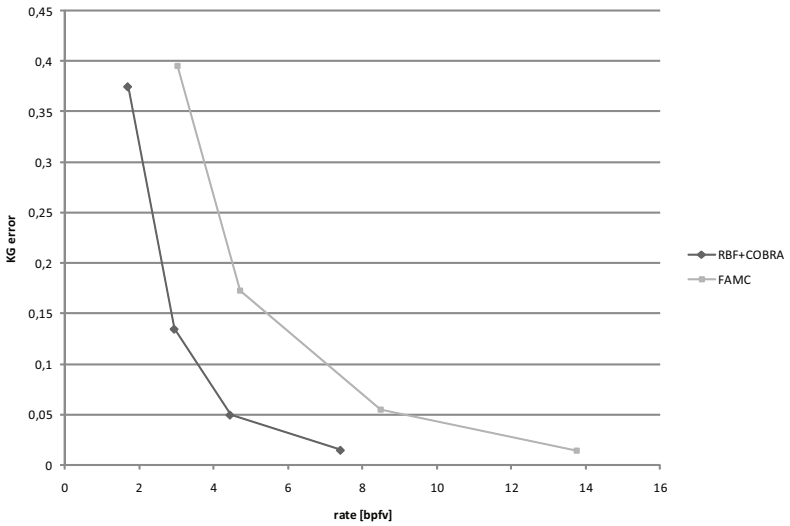

Figure 9.64: Comparison with FAMC encoder for the cow model.

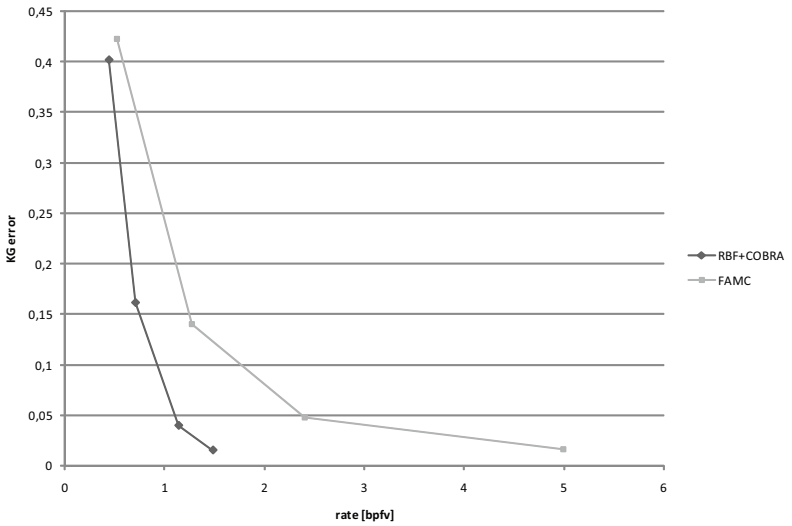

Figure 9.65: Comparison with FAMC encoder for the dance model.

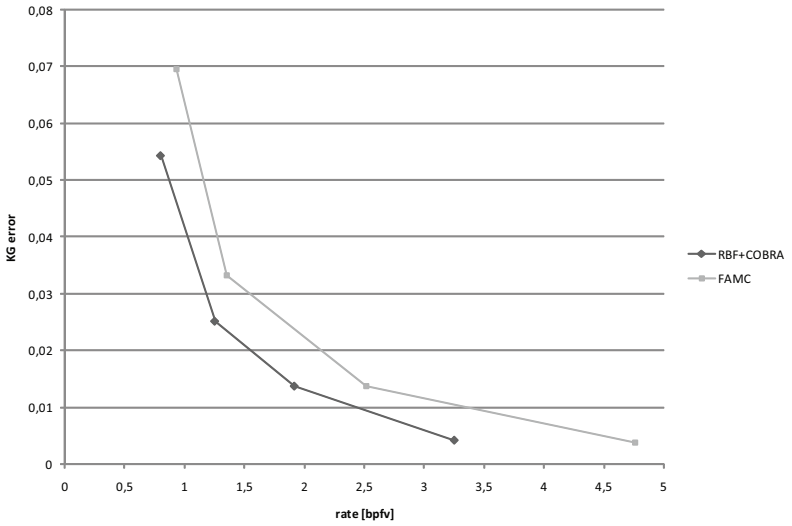

Figure 9.66: Comparison with FAMC encoder for the dolphin model.

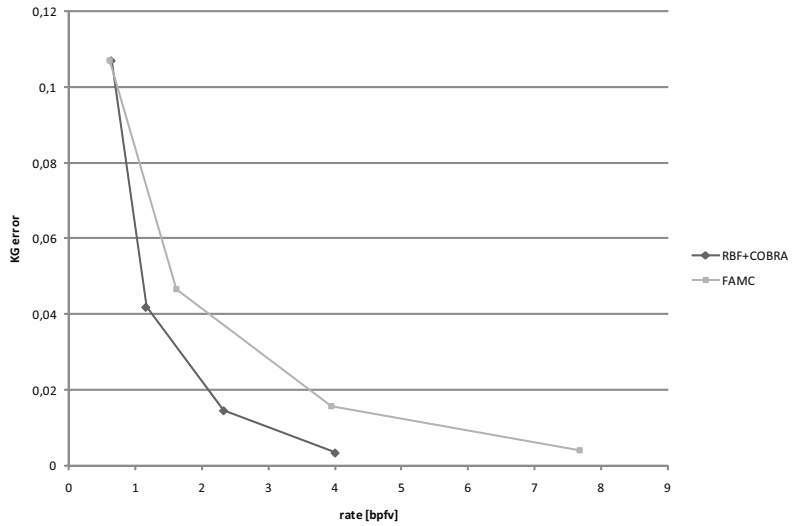

Figure 9.67: Comparison with FAMC encoder for the snake model.

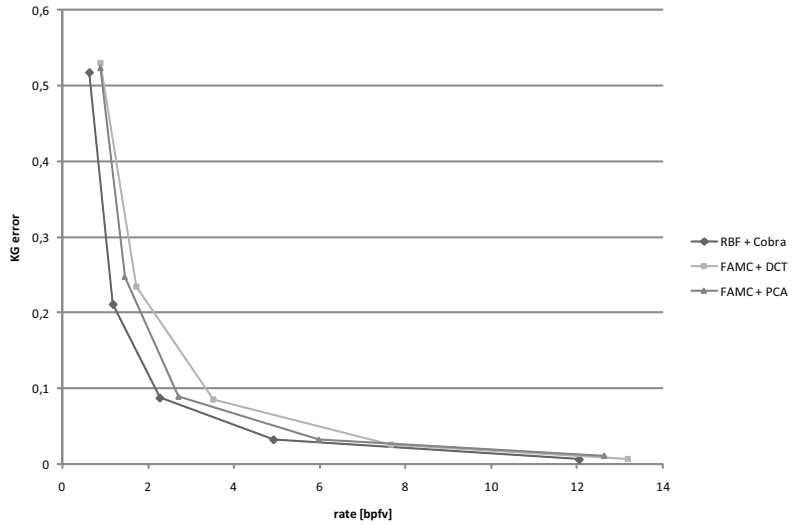

Figure 9.68: Comparison with FAMC encoder for the jump model.

Dataset	Method	Entropy	Residuals	Basis	bpfv	KG error
dance	NA	2,235	881,3	358,8	0,639	0,090
dance	LSP	1,293	715,7	358,8	0,519	0,089
dance	RBFP	1,190	694,9	358,8	0,504	0,090
dance	RLPCA	N/A	N/A	N/A	3,300	0,350
chicken	NA	4,677	985,3	724,8	1,470	0,371
chicken	LSP	2,470	810,9	724,8	1,320	0,371
chicken	RBFP	1,881	770,9	724,8	1,285	0,371
chicken	STSLPC	N/A	N/A	N/A	1,650	0,250
human	NA	2,050	1874,8	488,9	2,031	0,698
human	LSP	1,470	1587,8	488,9	1,785	0,698
human	RBFP	1,163	1470,8	488,9	1,684	0,698
cow	NA	4,044	665,7	341,1	1,700	0,598
cow	LSP	2,365	490,1	341,1	1,403	0,598
cow	RBFP	1,957	447,1	341,1	1,331	0,598
cow	STSLPC	N/A	N/A	N/A	2,950	0,600
cow	RLPCA	N/A	N/A	N/A	5,200	1,200
dolphin	NA	1,28613	521,9	297,3	1,326	0,071
dolphin	LSP	1,11149	484,6	297,3	1,265	0,070
dolphin	RBFP	1,04235	475,8	297,3	1,251	0,070
dolphin	RLPCA	N/A	N/A	N/A	3,100	0,110

Table 9.4: Predictor tests. NA denotes the original neighborhood average, LSP denotes the proposed algorithm based on weights and solving linear equations set, RBFP denotes the proposed algorithm based on radial basis functions. STSLPC denotes the algorithm by Stefanoski et al.[57], the values were taken from the paper. RLPCA denotes the algorithm by Amjoun et al.[5], the values were taken from the paper.

ATI FireGL V5200

trajectories\faces	5664	14118	70590	141180
26	1729	1273	448	251
36	1607	1124	364	202
50	1457	964	287	153

NVIDIA 8800 GTS

trajectories\faces	5664	14118	70590	141180
26	5772	4705	1488	799
36	4706	4222	1176	621
50	4585	3825	1081	573

Table 9.5: GPU playback tests. The numbers show achieved framerates on a common consumer graphics card, and on a high-end GPU. The used dataset is the chicken run (first column) and dance (second column, replicated five times in the third column, replicated ten times in the last column).

9.7 GPU implemetation of decompression

With a significant help of Vojtěch Hladík we have implemented on GPU the final step of decompression, i.e. the restoration of the original trajectories from the PCA basis and coefficients. Since this step is based on vector multiplication, it can be simply performed by any GPU with shader model 2.0 (or higher) in real-time without the necessity to store the decompressed meshes in main memory. The whole algorithm can be done by a vertex shader which has sixteen four-dimensional input registers to store up to 64 trajectory weights for each vertex. The basis trajectories have to be stored as constant arrays of floats (GPUs with Shader Model 2.0 or 3.0 must have at least 256 four-dimensional registers for vertex shader constants) for every frame.

Although with increasing number of trajectories also rises the memory consumption for the compressed mesh, the influence of this on the frame rate is minimal. Decompression slowdown is caused by the limited number of instructions in the vertex shader and mainly by the number of vertices.

Note that this approach can be used as a final step with any of the suggested compression schemes.

Chapter 10

Conclusions and future work

In this thesis we have presented the state of the art methods for compression and simplification of dynamic meshes. We have identified problems of the current methods, and suggested improvements.

We have derived new compression scheme, Coddyac, based on PCA and Edgebreaker. Subsequently, we have enhanced this scheme by combining it with connectivity driven simplification, reaching lower bitrates and better scalability. We have added new predictors, specifically developed for dynamic mesh compression and decompression, utilizing the properties of PCA represented animations, namely the length of the coefficient vector.

Our methods provide excellent rate/distortion ratio, while the computational cost is low for the basic approaches (coddyac and combined compression and simplification). The drawback of higher computational cost of the progressive decoders can be reduced in the future by further optimizing the computation. Our PCA-based methods are suitable for GPU decompression, which implies that our method reduces not only the transmission and storage costs, but also the main memory consumption.

Finally, we have developed a scheme for compression of the PCA basis, which also needs to be transmitted. Based on our observations we have used physical predictors and non-uniform quantization which exploits the knowledge of the distribution of the error introduced into the basis.

The results of the combined RBF and Cobra method significantly outperform all the known schemes to date in the KG error measure, however we have shown that such test tells only little about the perceived quality of the result.

We have also developed a novel error measure, the STED error, which has been tuned to fit the results of subjective testing. This measure now provides correlation with subjective results of more than 0.9 even for data sets that have not been used during the development. Moreover, the measure has predicted

some counterintuitive phenomena (such as error increase with increasing number of basis vectors) which have been subsequently confirmed by further subjective testing.

Still, there is a lot of space for future work. We can easily reach even lower bitrates by using more advanced entropy coders, such as adaptive arithmetic coders[45, 39]. However, such a decrease is beyond the scope of this work and has more to do with compression techniques in general. Our focus in this work has been to find a representation of low entropy.

Still it is necessary to note that such improvement cannot be applied to the best performing state of the art methods, such as the octree encoding or FAMC, because in these cases the high performance version of arithmetic coding - CABAC - is already an internal part of the algorithms. The authors of FAMC state that employing CABAC has lead to a reduction of data rate of cca 20-30% with respect to arithmetic coding.

The newly developed error measure also offers a new field for optimizations - we intend to develop new compression schemes which aim to reduce STED error instead of KG error, which will naturally lead to radically different algorithm design.

The STED error itself can be improved in the future, specifically to detect some of the singular cases, and also to detect global transformation of the dynamic mesh.

Appendix A

Subjective testing questionnaire

This is the original text of the questionnaire used in subjective testing mentioned in section 7.4:

Vážené subjekty,

Předem Vám děkuji za vyplnění následující tabulky. kolem dotazníku je subjektivní hodnocení kvality (nekvality) dynamických sítí. Na projekční stěně běží originální sekvence "chicken run", Vaším kolem je posoudit míru poškození této sítě v jednotlivých variantách přehrávaných na počítačích. Jinými slovy, Vaším kolem je určit která z degradovaných variant by pro vás jako uživatele kompresního algoritmu byla nejpřijatelnější. Pro zachování stálých podmínek prosím dodržte následující postup:

1. Prohlédněte si všechny animace, nejlépe v celé délce trvání. Soustředte se na možné artefakty a na rozdíly proti originálu
2. Určete animaci s nejhorší degradací a udělte jí známku 10 (největší problém)
3. Projděte znovu všechny sekvence a udělte jim známky 0-10 podle toho jak přijatelná je jejich distorze. Snažte se dodržet proporce, tj. dvojnásobná známka značí dvojnásobně nepřijatelnou distorzi
4. Známku 0 udělte jen takové sekvenci, u které nedokážete rozpoznat žádný rozdíl proti originálu
5. Udělené hodnocení vzájemně nekonzultujte, neupozorujte se navzájem na druh nebo lokaci problematických míst, snažte se pracovat "dle svého nejlepšího vědomí a svědomí".

Děkuji za spolupráci
L. Váša

Appendix B

List of authors publications

- VÁŠA,L.; SKALA,V. Combined Compression and Simplification of Dynamic 3D Meshes. In Computer Animation And Virtual Worlds. New York : John Wiley & Sons, Ltd., 2008. ISSN 1546-4261. To appear.
- SMOLIC,A.; SONDERSHAUS,R.; STEFANOSKI,N.; VÁŠA,L.; MUELLER,K.; OSTERMANN,J.; WIEGAND,T. A Survey on Coding of Static and Dynamic 3D Meshes. Berlin : Springer, 2008. pp. 239-311. ISBN 978-3-540-72531-2.
- VÁŠA,L.; SKALA,V. Coddyac: connectivity driven dynamic mesh compression. In 3DTV-CON 2007. Piscataway : IEEE, 2007. pp. 1-4. ISBN 1-4244-0721-4.

  Cited by:

  - KHAN,M.; OHNO,Y.: Compression of Temporal Video Data by Catmull-Rom Spline and Quadratic Bezier Curve Fitting. In WSCG'2008 Full Papers Proceedings. Pilsen : UNION Agency, 2008. ISBN 978-80-86943-15-2.
  - SMOLIC,A.; MUELLER,K.; STEFANOSKI,N.; OSTERMANN,J.; GOTCHEV,A.; AKAR,G.B.; TRIANTAFYLLIDIS,G.; KOZ,A. Coding Algorithms for 3DTV: A Survey. Circuits and Systems for Video Technology, IEEE Transactions on, 2007.
- VÁŠA,L. Methods for dynamic mesh size reduction, Technical report no. DCSE/TR-2006-07 at University of West Bohemia, October 2006.

- FRANK,M.; VÁŠA,L.; SKALA,V. MVE-2 Applied in Education Process. In .NET Technologies 2006. Pilsen : UNION Agency, 2006. pp. 39-45. ISBN 80-86943-10-0.

- VÁŠA,L.; SKALA,V. A spatio-temporal metric for dynamic mesh comparison. In Lecture Notes in Computer Science. Heidelberg : Springer Verlag, 2006, vol.4069, pp. 29-37. ISSN 0302-9743.

- FRANK,M.; VÁŠA,L.; SKALA,V. Pipeline approach used for recognition of dynamic meshes. In Computer graphics and artificial intelligence. Limoges : University of Limoges, 2006. pp. 219-224. ISBN 2-914256-08-6.

- VÁŠA,L.; HANÁK,I.; SKALA,V. Improved super-resolution method and its acceleration. In EUSIPCO 2005. Intanbul : Bogazici University, 2005. pp. 1-4. ISBN 975-00188-0-X.

  Cited by:

  – Hawkridge,S. Development of a Superresolution image enhancement plugin for the GIMP. Bachelor's Thesis. Rhodes University. 2006.

- VÁŠA,L.; SKALA,V. Resolution improvement of digitized images. In Algoritmy 2005. Bratislava : Slovak University of Technology, 2005. pp. 270-279. ISBN 80-227-2192-1.

  Cited by:

  – Shu-Fan Chang: High Frequency Compensated Super-Resolution Algorithm. Master's Thesis. National Central University. Taiwan. 2007.

- VÁŠA,L. Resolution Improvement of Digitized Images. Diploma thesis at University of West Bohemia. 2004.

Bibliography

[1] MPEG4 part 16 AMD2: Frame-based animated mesh compression. ISO/IEC JTC1/SC29/WG11, 2007. [cited at p. 24, 152]

[2] Marc Alexa and Wolfgang Müller. Representing animations by principal components. *Computer Graphics Forum*, 19(3):411–418, 2000. [cited at p. 18, 21, 22]

[3] Pierre Alliez and Mathieu Desbrun. Valence-driven connectivity encoding of 3d meshes. *Computer Graphics Forum*, 20:480–489, 2001. [cited at p. 33]

[4] Pierre Alliez, Giuliana Ucelli, Craig Gotsman, and Marco Attene. Recent advances in remeshing of surfaces. Research Report, 2005. [cited at p. 37]

[5] Rachida Amjoun. Efficient compression of 3d dynamic mesh sequences. In *Journal of the WSCG*, pages 99–106, February 2007. [cited at p. 22, 155]

[6] Nizam Anuar and Igor Guskov. Extracting animated meshes with adaptive motion estimation. In *Vision, Modeling, and Visualization*, pages 63–71, 2004. [cited at p. 68]

[7] Nicolas Aspert, Diego Santa-Cruz, and Touradj Ebrahimi. Mesh: measuring errors between surfaces using the hausdorff distance. *Proceedings of the IEEE International Conference on Multimedia and Expo*, 1:705–708, 2002. [cited at p. 61, 62]

[8] Marco Attene, Bianca Falcidieno, Michela Spagnuolo, and Jarek Rossignac. Swingwrapper: Retiling triangle meshes for better edgebreaker compression. *ACM Transactions on Graphics*, 22(4):982–996, 2003. [cited at p. 40, 41]

[9] Hector M. Briceno, Pedro V. Sander, Leonard McMillan, Steven Gortler, and Hugues Hoppe. Geometry videos: a new representation for 3d animations. In *SCA '03: Proceedings of the 2003 ACM SIGGRAPH/Eurographics symposium on Computer animation*, pages 136–146, Aire-la-Ville, Switzerland, 2003. Eurographics Association. [cited at p. 50]

[10] Michael M. Bronstein, Alexander M. Bronstein, Alfred M. Bruckstein, and Ron Kimmel. Matching two-dimensional articulated shapes using generalized multidimensional scaling. In *AMDO Lecture Notes on Computer Graphics*, 4096, pages 48–57, 2006. [cited at p. 89]

[11] Michael M. Bronstein, Alexander M. Bronstein, and Ron Kimmel. Face2face: an isometric model for facial animation. In *AMDO Lecture Notes on Computer Graphics*, 4096, pages 38–47, 2006. [cited at p. 89]

[12] Robert Calderbank, Ingrid Daubechies, Wim Sweldens, and Boon-Lock Yeo. Wavelet transforms that map integers to integers. *Applied and Computational Harmonic Analysis (ACHA)*, 5(3):332–369, 1998. [cited at p. 18]

[13] Martin Čermák and Václav Skala. Polygonization by the edge spinning. In *Algoritmy 2002*, pages 245–252, Slovakia, 2002. Univ.of Technology. [cited at p. 40]

[14] Prashant Chopra and Joerg Meyer. Tetfusion: an algorithm for rapid tetrahedral mesh simplification. In *VIS '02: Proceedings of the conference on Visualization '02*, pages 133–140, Washington, DC, USA, 2002. IEEE Computer Society. [cited at p. 53]

[15] Paolo Cignoni, Claudio Rocchini, and Roberto Scopigno. Metro: measuring error on simplified surfaces. Technical report, Paris, France, France, 1996. [cited at p. 61, 62]

[16] Volker Coors and Jarek Rossignac. Delphi: geometry-based connectivity prediction in triangle mesh compression. *The Visual Computer: International Journal of Computer Graphics*, 20(8-9):507–520, 2004. [cited at p. 31]

[17] Jean Duchon. Splines minimizing rotation-invariant semi-norms in sobolev spaces. In *Constructive Theory of Functions of Several Variables*, pages 85–100. Springer-Verlag, Berlin-Heidelberg, 1977. [cited at p. 103]

[18] Michael S. Floater. Parametrization and smooth approximation of surface triangulations. *Computer Aided Geometric Design*, 14(3):231–250, 1997. [cited at p. 42]

[19] Martin Franc. Methods for polygonal mesh simplification. Technical Report DCSE/TR-2002-01, University of West Bohemia, 2002. [cited at p. 37]

[20] Martin Franc and Václav Skala. Parallel triangular mesh decimation without sorting. In *SCCG '01: Proceedings of the 17th Spring conference on Computer graphics*, page 22, Washington, DC, USA, 2001. IEEE Computer Society. [cited at p. 39]

[21] Milan Frank, Libor Váša, and Václav Skala. Pipeline approach used for recognition of dynamic meshes. In *Proceedings of 3IA 2006*, pages 219–224, 2006. [cited at p. 76]

[22] Michael Garland and Yuan Zhou. Quadric-based simplification in any dimension. *ACM Transactions on Graphics*, 24(2):209–239, 2005. [cited at p. 51]

[23] Craig Gotsman, Stefan Gumhold, and Leif Kobbelt. *Simplification and Compression of 3D Meshes*, pages 319–362. Mathematics and Visualization. Springer. [cited at p. 37]

[24] Xianfeng Gu, Steven J. Gortler, and Hugues Hoppe. Geometry images. In *SIGGRAPH '02: Proceedings of the 29th annual conference on Computer graphics and interactive techniques*, pages 355–361, New York, NY, USA, 2002. ACM Press. [cited at p. 42, 43, 50]

[25] Stefan Gumhold. Entropy extremes for coding and visualization. Technical report, Wilhelm-Schickard-Institut für Informatik, University of Tübingen, Germany, July 2003. [cited at p. 9]

[26] Stefan Gumhold, Stefan Guthe, and Wolfgang Straßer. Tetrahedral mesh compression with the cut-border machine. In *VIS '99: Proceedings of the conference on Visualization '99*, pages 51–58, Los Alamitos, CA, USA, 1999. IEEE Computer Society Press. [cited at p. 34, 35]

[27] Stefan Gumhold and Wolfgang Straßer. Real time compression of triangle mesh connectivity. *Computer Graphics*, 32(Annual Conference Series):133–140, 1998. [cited at p. 28]

[28] David A. Huffman. A method for the construction of minimum-redundancy codes. *Proceedings of the IRE*, 40(9):1098–1101, September 1952. [cited at p. 9, 94]

[29] Lawrence Ibarria and Jarek Rossignac. Dynapack: space-time compression of the 3d animations of triangle meshes with fixed connectivity. In *SCA '03: Proceedings of the 2003 ACM SIGGRAPH/Eurographics symposium on Computer animation*, pages 126–135, Aire-la-Ville, Switzerland, Switzerland, 2003. Eurographics Association. [cited at p. 13, 14, 50]

[30] Euee S. Jang, James D. K. Kim, Seok Yoon Jung, Mahnjin Han, and Sang Oak Woo. Interpolator data compression for mpeg-4 animation. *IEEE Transactions on Circuits and Systems for Video Technology*, 14(7):989–1008, 2004. [cited at p. 60]

[31] Chang-Su Kim Jeong-Hyu Yang and Sang-Uk Lee. Compression of 3d triangle mesh sequences based on vertex-wise motion vector prediction. *IEEE Transactions on Circuits and Systems for Video Technology*, 12(12):1178–1184, December 2002. [cited at p. 14]

[32] Zachi Karni and Craig Gotsman. Compression of soft-body animation sequences. In *Computers & Graphics*, volume 28, pages 25–34, 2004. [cited at p. 20, 59, 106]

[33] Ladislav Kavan, Steven Collins, Jiri Zara, and Carol O'Sullivan. Skinning with dual quaternions. In *2007 ACM SIGGRAPH Symposium on Interactive 3D Graphics and Games*, pages 39–46. ACM Press, April/May 2007. [cited at p. 69]

[34] Scott Kircher and Michael Garland. Progressive multiresolution meshes for deforming surfaces. In *SCA '05: Proceedings of the 2005 ACM SIGGRAPH/Eurographics symposium on Computer animation*, pages 191–200, New York, NY, USA, 2005. ACM Press. [cited at p. 56]

[35] Patrick Klie, Eugen Okon, Nikolce Stefanoski, and Jörn Ostermann. A framework for scene-flow driven creation of time consistent dynamic 3d objects using mesh parametrizations. In *3DTV-CON, The True Vision - Capture, Transmission and Display of 3D Video*, volume 0. IEEE, May 2007. [cited at p. 89]

[36] Peter Lindstrom and Greg Turk. Fast and memory efficient polygonal simplification. In *IEEE Visualization*, pages 279–286, 1998. [cited at p. 46]

[37] Peter Lindstrom and Greg Turk. Evaluation of memoryless simplification. *IEEE Transactions on Visualization and Computer Graphics*, 5(2):98–115, 1999. [cited at p. 46]

[38] Khaled Mamou, Titus Zaharia, and Francoise Preteux. A skinning approach for dynamic 3d mesh compression: Research articles. *Computer Animation and Virtual Worlds*, 17(3–4):337–346, 2006. [cited at p. 23]

[39] Detlev Marpe, Thomas Wiegand, and Heiko Schwarz. Context-based adaptive binary arithmetic coding in the h.264/avc video compression standard. *IEEE Transactions on Circuits and Systems for Video Technology*, 13(7):620–636, 2003. [cited at p. 16, 25, 158]

[40] Andrew Mason. Mushra (multi stimulus test with hidden reference and anchor). ITU-R BS.1534, BBC R&D White Paper WHP 038. [cited at p. 82]

[41] Prasun Mathur, Chhavi Upadhyay, Parag Chaudhuri, and Prem Kalra. A measure for mesh compression of time-variant geometry: Research articles. *Computer Animation and Virtual Worlds*, 15(3-4):289–296, 2004. [cited at p. 49]

[42] Nelson Max. Weights for computing vertex normals from facet normals. *J. Graph. Tools*, 4(2):1–6, 1999. [cited at p. 50]

[43] Gary H. Meisters. Polygons have ears. *American Mathematical Monthly*, pages 548–551, June 1975. [cited at p. 97]

[44] Paul S. Heckbert Michael Garland. Surface simplification using quadric error metrics. In *SIGGRAPH*, pages 209–216, 1997. [cited at p. 45, 49, 51, 53]

[45] Alistair Moffat and Radford M. Neal. Arithmetic coding revisited. In *DCC '95: Proceedings of the Conference on Data Compression*, page 202, Washington, DC, USA, 1995. IEEE Computer Society. [cited at p. 9, 158]

[46] Alex Mohr and Michael Gleicher. Deformation sensitive decimation. Technical report, University of Wisconsin, 2003. [cited at p. 55]

[47] Karsten Müller, Aljoscha Smolic, Matthias Kautzner, Peter Eisert, and Thomas Wiegand. Predictive compression of dynamic 3d meshes. In *ICIP05*, pages I: 621–624, 2005. [cited at p. 15]

[48] Karsten Müller, Aljoscha Smolic, Matthias Kautzner, and Thomas Wiegand. Rate-distortion-optimized predictive compression of dynamic 3d mesh sequences. *SP:IC*, 21(9):812–828, October 2006. [cited at p. 16, 60]

[49] Frederic Payan and Marc Antonini. Wavelet-based compression of 3d mesh sequences. In *Proceedings of IEEE ACIDCA-ICMI'2005*, Tozeur, Tunisia, November 2005. [cited at p. 17]

[50] Remi Ronfard and Jarek Rossignac. Full-range approximation of triangulated polyhedra. In *Proceeding of Eurographics, Computer Graphics Forum*, volume 15(3), pages C67–C76. Eurographics, Blackwell, August 1996. [cited at p. 43, 47]

[51] Jarek Rossignac. Edgebreaker: Connectivity compression for triangle meshes. *IEEE Transactions on Visualization and Computer Graphics*, 5(1):47–61, 1999. [cited at p. 28]

[52] Peter Sand, Leonard McMillan, and Jovan Popovič. Continuous capture of skin deformation. In *SIGGRAPH '03: ACM SIGGRAPH 2003 Papers*, pages 578–586, New York, NY, USA, 2003. ACM Press. [cited at p. 68]

[53] Pedro V. Sander, Steven J. Gortler, John Snyder, and Hugues Hoppe. Signal-specialized parametrization. In *EGRW '02: Proceedings of the 13th Eurographics workshop on Rendering*, pages 87–98, Aire-la-Ville, Switzerland, Switzerland, 2002. Eurographics Association. [cited at p. 42]

[54] Mirko Sattler, Ralf Sarlette, and Reinhard Klein. Simple and efficient compression of animation sequences. In *SCA '05: Proceedings of the 2005 ACM SIGGRAPH/Eurographics symposium on Computer animation*, pages 209–217. ACM Press, 2005. [cited at p. 21, 22]

[55] William J. Schroeder, Jonathan A. Zarge, and William E. Lorensen. Decimation of triangle meshes. In *SIGGRAPH '92: Proceedings of the 19th annual conference on Computer graphics and interactive techniques*, pages 65–70, New York, NY, USA, 1992. ACM Press. [cited at p. 37, 49]

[56] Václav Skala and Vít Ondračka. Fast maximum distance of points in e2 & e3. Technical report, University of West Bohemia, 2008. [cited at p. 81]

[57] Nikolče Stefanoski, Xiaoliang Liu, Patrick Klie, and Jörn Ostermann. Scalable linear predictive coding of time-consistent 3d mesh sequences. In *3DTV-CON, The True Vision - Capture, Transmission and Display of 3D Video*, volume 0. IEEE, May 2007. [cited at p. 16, 59, 155]

[58] Nikolče Stefanoski and Jörn Ostermann. Connectivity-guided predictive compression of dynamic 3d meshes. In *Proceedings of ICIP '06 - IEEE International Conference on Image Processing*, October 2006. [cited at p. 15, 59]

[59] Andrzej Szymczak and Jarek Rossignac. Grow & fold: compression of tetrahedral meshes. In *SMA '99: Proceedings of the fifth ACM symposium on Solid modeling and applications*, pages 54–64, New York, NY, USA, 1999. ACM Press. [cited at p. 34]

[60] Gabriel Taubin and Jarek Rossignac. Geometric compression through topological surgery. *ACM Transactions on Graphics*, 17(2):84–115, 1998. [cited at p. 26]

[61] Costa Touma and Craig Gotsman. Triangle mesh compression. In *Graphics Interface*, pages 26–34, June 1998. [cited at p. 13, 31]

[62] William T. Tutte. A census of planar triangulations. *Canadian Journal of Mathematics*, 14:21–38, 1962. [cited at p. 26]

[63] Karel Uhlíř and Václav Skala. Radial basis function use for the restoration of damaged images. *Computer vision and graphics*, 32:839–844, 2006. [cited at p. 103]

[64] Libor Váša, Vojtěch Hladík, and Václav Skala. Geometry driven local neighborhood based predictors for dynamic mesh compression. Submitted for publication, 2008. [cited at p. 91]

[65] Libor Váša and Václav Skala. Coddyac: Connectivity driven dynamic mesh compression. In *3DTV-CON, The True Vision - Capture, Transmission and Display of 3D Video*. IEEE, May 2007. [cited at p. 91]

[66] Libor Váša and Václav Skala. Cobra: Compression of basis of pca represented animations. Submitted to review in Computer Graphics Forum, 2008. [cited at p. 91]

[67] Libor Váša and Václav Skala. Combined compression and simplification of dynamic 3d meshes. *Computer Animation and Virtual Worlds*, 2008. to appear. [cited at p. 91]

[68] Libor Váša and Václav Skala. Perception based comparison method for dynamic meshes. Submitted to review in Computer Animation and Virtual Worlds, 2008. [cited at p. 78]

[69] Libor Váša and Václav Skala. A spatio-temporal metric for dynamic mesh comparison. In *AMDO Lecture Notes on Computer Graphics*, 4096, pages 29–37. Springer-Verlag Berlin Heidelberg, 2006. [cited at p. 72, 74]

[70] E. W. Weisstein. Least squares fitting, 2002. From Mathworld – a Wolfram Web Resource. http://mathworld.wolfram.com/LeastSquaresFitting.html. [cited at p. 108]

[71] Zhidong Yan, Sunil Kumar, and C. C. Jay Kuo. Error-resilient coding of 3-d graphic models via adaptive mesh segmentation. *IEEE Transactions on Circuits and Systems for Video Technology*, 11(7):860–873, 2001. [cited at p. 22]

[72] Jeong-Hyu Yang, Chang-Su Kim, and Sang-Uk Lee. Semi-regular representation and progressive compression of 3-d dynamic mesh sequences. *IEEE Transactions on Image Processing*, 15(9):2531–2544, 2006. [cited at p. 89]

[73] František Zadražil. Methods of triangular dynamic meshes comparison. Diploma thesis at University of West Bohemia, 2007. [cited at p. 62, 78]

[74] Jinghua Zhang and Charles B. Owen. Octree-based animated geometry compression. In *DCC '04: Proceedings of the Conference on Data Compression*, page 508, Washington, DC, USA, 2004. IEEE Computer Society. [cited at p. 16]

[75] Jinghua Zhang and Jinsheng Xu. Optimizing octree motion representation for 3d animation. In *ACM-SE 44: Proceedings of the 44th annual Southeast regional conference*, pages 50–55, New York, NY, USA, 2006. ACM Press. [cited at p. 16]