Compression of Static and Dynamic Three-Dimensional Meshes

Dissertation
der Fakultät für Informations- und Kognitionswissenschaften
der Eberhard-Karls-Universität Tübingen
zur Erlangung des Grades eines
Doktors der Naturwissenschaften
(Dr. rer. nat.)

vorgelegt von

Rachida Amjoun

Tübingen
2009

Tag der mündlichen Qualifikation:	21.07.2009
Dekan:	Prof. Dr. Oliver Kohlbacher
1. Berichterstatter:	Prof. Dr.-Ing. Dr.-Ing. E.h. Wolfgang Straßer
2. Berichterstatter:	Prof. Dr. Andreas Schilling

Zusammenfassung

Mit den wachsenden Möglichkeiten von Modellierungssoftware und 3D Scannern ist die Anzahl verfügbarer 3D Modelle stetig gewachsen. Animationen von 3D Modellen sind weltweit verbreitet und bilden heute eine der größten Anwendungen digitaler Medientechnologie. Um einen hohen Realitätsgrad zu erreichen, werden komplexe und hochdetaillierte 3D Objekte verwendet - möglicherweise mit Millionen von Polygonen und Punkten - für lange Sequenzen. Die Ursprungsformate solch großer 3D Modelle benötigen viel Speicherplatz und Bandbreite.

Das Problem des Speicherns und Übertragens ist gut untersucht für statische Meshes. Hierfür gibt es eine große Anzahl an erfolgreichen Kompressionsverfahren. Das Hauptziel dieser Dissertation ist das Entwickeln neuer Verfahren für die Komprimierung statischer und dynamischer 3D Modelle, welche durch Dreiecksnetze repräsentiert werden, und neuer Segmentierungsverfahren, welche notwendig für das effiziente Komprimieren animierter 3D Modelle sind.

Die Arbeit gliedert sich in fünf Teile. Der erste Teil behandelt Definitionen, traditionelle Kompressionsverfahren und Vorarbeiten auf dem Gebiet der Kompression statischer und dynamischer Meshes sowie ihrer Segmentierung.

Der zweite Teil stellt ein Verfahren für statische Geometrie vor. Es werden nicht die drei Koordinaten eines Punktes kodiert, sondern seine tangentiale und normale Komponenten. Neue Vorhersagetechniken werden eingeführt, welche das Normalenkodieren verbessern.

Der dritte Teil behandelt Segmentierungen von sich zeitlich verändernder Meshgeometrie. Drei neue Verfahren werden hier vorgestellt: Ein Region Growing Verfahren, ein statisches Clustering und ein adaptives Clustering. Diese zerteilen die dynamischen Meshes in quasi-rigide Bereiche unter Verwendung einer lokalen Charakteristik.

Der vierte Teil stellt Verfahren für sich zeitlich verändernde Meshgeometrie mit konstanter Konnektivität vor.

Der erste Algorithmus ist eine fast verlustlose Single-Rate Kompression für animierte Meshes. Er verallgemeinert das vorgestellte Verfahren für die Kodierung statischer Meshgeometrie. Er zeigt, dass die lokalen Koordinatensysteme ein größeres zeitliches und räumliches Clustering-Verhalten aufweisen als das Weltkoordinatensystem. Die Kombination beider Clustering-Verfahren führt zu einer erheblichen Reduktion der Bitrate. Verschiedene Prediktoren werden vorgestellt für tangentiale Komponenten und Normalenkomponenten.

Der zweite neue Algorithmus ist eine auf der Relative-Local-Principal-Component-Analysis (RLPCA) basierende Kompression. Er verbindet das vorgestellte Clustern mit LPCA, ausgeführt im lokalen Koordinatensystem. Es wird gezeigt, dass eine einfache Abbildung der originalen Koordinaten in ein lokales Koordinatensystem jede Region quasi-invariant über die Zeit und damit sehr gut komprimierbar mit PCA macht. Um den Algorithmus weiter zu verbessern, wird eine Rate-Distortion Optimierung eingeführt.

Der dritte Kompressionsalgorithmus basiert auf prediktiven Codern und Discrete-Cosinus-Transform-Codern (PDCT). Nach dem Clustern wird ein prediktives Kodieren durchgeführt im lokalen Koordinatensystem jedes Clusters, welches zu sehr kleinen Delta-Vektoren führt (prediktive Fehler). Die Delta-Vektoren werden dann in den Frequenzraum transformiert mit DCT. Die resultierenden DCT Koeffizienten sind besser komprimierbar als die Vektorkoordinaten. Die originale Mesh-Sequenz kann rekonstruiert werden von nur wenigen DCT Koeffizienten, welche ungleich Null sind, ohne großen Verlust an visueller Qualität.

Abschließend diskutiert der fünfte Teil die Ergebnisse und stellt experimentelle Ergebnisse vor. Es wird gezeigt, dass die vorgestellten Verfahren den Vergleich mit anderen aktuellen Arbeiten standhalten.

Abstract

With the advancements and variety of sources to model 3D objects such as scanning technologies and modelling softwares, 3D models are becoming widely available. Animation also attracted worldwide attention and has become one of the most successful application of digital media technology. As a result, it is also becoming easier to acquire animated models. In order to achieve a higher degree of realism, more complex and highly detailed three-dimensional objects – possibly out of millions of vertices and polygons – are created with large sequences. When storing, downloading, or uploading these 3D sequences of objects in their standard forms over a network, large data rows consume large amounts of storage space and network bandwidth.

This problem of storage and transmission has been widely studied for static meshes and wealth of successful compression schemes have been proposed. The main goal of this thesis is to develop new powerful compression techniques to reduce storage requirements and transmission times of static and dynamic 3D models represented by triangulated mesh and introduce new and efficient animation segmentation approaches that are very useful for different purposes, typically 3D dynamic mesh compression.

This work covers five main parts. The first part presents definitions, traditional data compression techniques and reviews the existing techniques in the fields of static and dynamic mesh compression and segmentation.

The second part introduces a new algorithm for static geometry data. Instead of encoding the three coordinates of a vertex, its tangential and normal components are encoded. New advanced prediction techniques are proposed to improve the normal encoding.

The third part concerns segmentation of time varying mesh geometry. Three new approaches have been developed: A region growing based approach as well as static and adaptive clustering based methods. These break down the dynamic meshes into quasi-rigid parts using local characteristic.

The fourth part presents successive contributions to time varying mesh geometry compression where the connectivity remains constant over time.

The first new algorithm is a single rate near-lossless compression for animated meshes. It generalizes the proposed static mesh geometry coding. It shows that the local spaces exhibit higher temporal and spatial clustering behavior than the world space, and the com-

bination of both clusterings yield significant reduction in bit-rates. Different predictors are proposed for tangential and normal components.

The second novel algorithm is a Relative Local Principal Component Analysis (RLPCA) based-compression. It combines the proposed clustering with LPCA, performed in the local space. We will show that simply mapping the original coordinates into local space makes each region quasi-invariant over time and well-compressible by using PCA. To further enhance this algorithm, a rate-distortion optimization is introduced.

The third compression algorithm is based on Predictive and Discrete Cosine Transform coders (PDCT). After, clustering process, predictive coding is performed in the local space of each cluster resulting in very small delta vectors (prediction errors). The delta vectors are then transformed into the frequency domain using DCT. The resulting DCT coefficients are more compressible than the vector coordinates and the original sequence of meshes can be reconstructed from only a few non-zero DCT coefficients without significant loss in visual quality.

Finally, the fifth part discusses and provides the experimental results. We will show that our methods are competitive when compared to the state-of-the-art techniques.

Acknowledgements

I would like to express my gratitude and sincere appreciation to my advisor Prof. Wolfgang Straßer for his guidance, help and continuous support throughout the course of my doctoral studies.

I also want to express my appreciation to Prof. Andreas Schilling for the valuable support and the fruitful discussions.

I would like to thank Prof. Stefan Gumhold from university of Dresden, who directly contributed to the work presented in the third chapter.

I would like to express my gratitude and thanks to PD. Dr. Douglas W. Cunningham from Max Planck Institute of Tuebingen, and Dr. Ralf Sondershaus from Automotive Lighting Reutlingen GmbH, for the proof-reading, valuable advice and helpful discussions, and to Dr. Zein Salah from University of Magdeburg for useful discussions and advice.

Sincere thanks are extended to GRIS's office Manager Ms. Christ Constance who provided help and assistance in numerous ways. Also, my thanks go to all my colleagues at WSI/GRIS for the friendly environment and discussions that helped me complete the present work.

Finally, I would like to thank my family and friends for their complete and unwavering support. Special thanks go to my mother and my father, to whom I dedicate this work, for their incessant support and love, and for being there whenever I needed them. Without their help and encouragement, I would not be able to complete this work successfully.

Contents

Zusammenfassung **i**

Abstract **iii**

1 Introduction **1**

1.1 Problem Statement . . . 2

1.2 Overview of Thesis . . . 5

1.3 Contributions of Thesis . . . 8

2 Background **9**

2.1 Compression . . . 9

2.1.1 Lossless Compression . . . 10

2.1.2 Lossy Compression . . . 10

2.2 General Compression Techniques . . . 11

2.2.1 Huffman Coding . . . 12

2.2.2 Arithmetic Coding . . . 15

2.3 3D Data Representation . . . 17

2.3.1 Static 3D Object . . . 17

2.3.2 Dynamic 3D Object . . . 20

2.4 Prior Mesh Compression Techniques . . . 20

2.4.1 Static Mesh Compression . . . 23

2.4.1.1 Single-Rate Encoders . . . 23

2.4.1.2 Progressive Compression . . . 35

2.4.2 Animated Mesh Compression . . . 37

2.4.2.1 Clustering-based Compression 38
2.4.2.2 Vertex-Prediction based Compression 40
2.4.2.3 PCA-based Compression 43
2.4.2.4 Wavelet-based Compression 44
2.4.3 Level of Details for Dynamic Meshes 44
2.5 Segmentation 45
2.5.1 Static Mesh Segmentation 45
2.5.2 Dynamic Mesh Segmentation 46
2.5.3 Discussion and Summary 48

3 Compression of Static Meshes: Higher Order Predictor 51
3.1 Introduction 51
3.2 Geometry Encoding and Decoding Algorithm 53
3.2.1 Avoidance of Error Accumulation 55
3.2.2 Local Coordinate System 56
3.2.3 Tangential Prediction 57
3.2.4 Binary Coding of Coordinates 57
3.3 Higher Order Prediction 57
3.3.1 Gathering of Fit Vertices 58
3.3.2 Higher Order Surface Fitting 60
3.3.3 Intersecting Higher Order Surfaces with a Tangential Circle . . . 62
3.4 Alternative Approaches 62
3.4.1 Fitting of Implicit Function 63
3.4.2 Sphere Fitting 65
3.5 summary 65

4 Animated 3D Object Segmentation 69
4.1 Introduction 69
4.2 Overview 71
4.3 Definitions 72
4.4 Region Growing based Approach 72
4.4.1 Segment Initialization 73
4.4.2 Seed Selection 73
4.4.3 Mesh Growing Process 73
4.4.4 Results 74
4.5 Clustering based Segmentation 74
4.5.1 Initialization and Seed Selection 74

4.5.2 Local Coordinate Frames Construction 74
4.5.3 Vertex Clustering 75
4.5.4 Results 76
4.6 Adaptive Processing 77
4.6.1 Results 78
4.7 Evaluation of Segmentation Approaches 78
4.8 Computation Time 79
4.9 Discussion and Summary 82

5 Connectivity-Guided Compression of 3D dynamic Meshes 91
5.1 Introduction 91
5.2 Predictive Coding of the animated vertices 93
5.3 System Overview of SplitCoder 94
5.3.1 Mesh Traversal 95
5.3.2 Geometry Coding 95
5.3.2.1 Transforming World into Cylindrical Coordinates . . . 96
5.3.2.2 Tangential and Normal Components Prediction 98
5.3.2.3 Binary Coding of Coordinates 100
5.3.3 Geometry Decoding 101
5.4 summary 101

6 Motion based PCA Compression 103
6.1 Introduction 103
6.2 Overview 104
6.3 Compression pipeline 106
6.3.1 Segmentation 107
6.3.2 Transforming Vertex Positions into the Local Space 107
6.3.3 Compression of Local Coordinate Frames 107
6.3.4 Principal Component Analysis 108
6.3.5 Quantization and Arithmetic Coder 109
6.4 Decompression 110
6.5 Rate-Distortion Optimization for PCA based Coding 110
6.5.1 Description of the Problem 111
6.5.2 Problem Statement 112
6.5.3 Incremental Computation of the Rate-Distortion Optimization . . 113
6.6 Compression Parameters 113
6.7 Conclusion 114

7 Predictive-DCT based Compression 117

7.1 Introduction 117

7.2 Overview 118

7.3 Compression Pipeline 120

7.3.1 Local Coordinate Frames 120

7.3.1.1 Seed Triangles Selection 121

7.3.1.2 Local Coordinate Frame Construction 121

7.3.2 Motion in LCF based Clustering 121

7.3.3 Differential Coding of LCFs 122

7.3.4 Spatial-Temporal Predictive Coding 122

7.3.5 DCT Coding 123

7.3.6 Quantization and Arithmetic Coder 124

7.3.7 Reconstruction 124

7.4 Conclusion 125

8 Evaluation and Comparison 127

8.1 Implementations 127

8.2 Input Datasets 128

8.3 Measurement 128

8.4 Higher Order Prediction for Static Geometry Coding 131

8.5 Near-lossless Predictive Coding for Dynamic Meshes 136

8.5.1 Evaluation of Different Predictors 136

8.5.1.1 Tangential components 136

8.5.1.2 Normal component 138

8.5.2 SplitCoder vs. State-of Art 142

8.6 RLPCA based Compression for Dynamic Meshes 145

8.6.1 Compression Parameters 145

8.6.2 RLPCA vs. LPCA 145

8.6.3 RLPCA and ORLPCA vs. State of Art 147

8.6.4 Timings 149

8.7 Predictive-DCT based Compression 152

8.7.1 Compression Parameters 152

8.7.2 PDCT vs. State of Art
157

8.8 Comparison of the Proposed Methods 161

9 Conclusion and Future Work 177

Bibliography **180**

CHAPTER 1

Introduction

Computer graphics is one of the most exciting sectors in computer sciences. This sector is growing rapidly, possibly more than any other aspect of computer technology and increasedly incorporated in a various domains such as scientific, engineering and medical applications, games, movies for special effects and animated films. In these applications, the use of the computer graphics would not be possible without 3D geometric data or more generally 3D objects. The shape of these objects can be modeled through irregular polygonal meshes, a set of faces connected with points. The most common geometric representation used by the designer either in the static or in the dynamic case, is triangle meshes to suit the requirements of computer graphics visualization systems. There are also other popular representations include parametric surfaces, point sampled surfaces, implicit surfaces and voxel based representation.

Computer animation is one of the most highly regarded parts of computer graphics. It attracted worldwide attention and has become one of the most successful applications of digital media technology. Computer animation has revolutionized the world of movies and TV, and has built the computer games industry. It has also found its way into many other fields such as marketing, arts, sciences, scientific visualization and flight simulation.

Animation allows the creation of more realistic and natural scenes, a more complete comprehension of complex problems, the display of characters that never existed, and access to places that are difficult or impossible to tread. In other words, it transports the users into another world where the impossible becomes possible.

Three dimensional animation simply gives life to static 3D objects, creating a sequence of static meshes each of which represents one frame. Today, animation technology has also become more sophisticated and accessible. Moreover, its applications have become more and more widespread and often demand animated 3D models and scenes with a high degree of realism. As 3D animation becomes more realistic and more complex, the corresponding meshes become bigger and bigger, consuming more and more space. It is therefore indispensable to compress 3D static and animation datasets.

1.1 Problem Statement

With the advancements and variety of sources to generate model 3D objects such as scanning technologies and modelling softwares, 3D models are becoming widely available. In order to achieve a higher degree of realism, more complex and highly detailed 3D objects possibly out of millions of vertices are created. The standard representation of triangle meshes uses a list of floating point values to describe the vertex positions (geometry), a list of integer values that specify the vertex indices (connectivity), and sometimes, properties such as normals and textures coordinates which are specified in similar way. When storing, downloading, or uploading these 3D objects in their standard forms over a network, large data rows consume large amounts of storage space and network bandwidth.

This problem often arises in animation. Today, it is easier to acquire animated models. In parallel, there is rapidly increase in the use of these animated objects in many applications (particulary in computer generated movies, special-effects films, and computer games). Finally, the models are becoming more realistic but more complex. To store an animated objet, i.e. a sequence of meshes, one has to store one mesh for each frame. Assuming that the connectivity is constant over time and only geometry information changes over time, the representation of the geometry information of the sequence will require F times the information of each frame, i.e., $F * V * 3 * 32$ bits, where F and V are the number of frames and vertices, respectively. Each one of the three coordinates of a frame is represented using 32 bits. For large sequences and detailed and high accuracy models, the uncompressed representation results in large files which are expensive to store or to deliver over networks.

The problem of storage and transmission has been widely studied for static meshes and a wealth of successful compression schemes has been proposed. However, the current static techniques for the compression of sequences of meshes independently are inefficient. *Key-frame* animation is one of the most famous traditional and dominant animation representations used in the industry to represent the animation compactly. A set of key frames are chosen to describe certain important key poses in the animation sequence at certain times. Then all frames in-between are generated using interpolation techniques. For such applications, even the number of key-frames can be very large, requiring a large memory space and effective compression techniques. Many techniques have been proposed and there is still room for better encodings of animated geometry.

The goal of this thesis is to develop new powerful compression techniques to reduce storage requirements and transmission times of static and dynamic 3D models represented by triangulated meshes. This type of data is used in many computer graphics applications.

The segmentation of deforming triangular meshes is a new research area. It is rapidly becoming important in many computer graphics applications. In the context of compression, its is used to achieve better compression performance. Typically, it reduces the amount of time needed to handle or transform large datasets. It can also reduce the amount of time needed to extract the coherent rigid parts which leads to a more compact representation. This thesis introduces several new and efficient animation segmentation approaches that can be useful not only for compression but also for different purposes.

Static meshes

Early work in this field concentrated on finding efficient and optimal connectivity coding schemes for static meshes. They encode the connectivity information first then they encode the geometry information in terms of traversal order used to encode the connectivity. Later, the geometry-driven compression techniques emerged.

Most of the proposed paradigms to reduce the amount of vertex positions use the following combination: prediction to exploit the high correlation between the positions of adjacent mesh vertices, quantization to reduce the floating point to finite precision, and entropy encoding to reduce the statistical redundancies. We follow the same paradigm and propose new approaches for the static vertex positions. The new idea is to split the coding into tangential and normal encoding [36, 8]. For tangential encoding, we investigated the current approaches [117, 25]. For the normal encoding, we introduced higher order predictors based on surface and sphere fitting. In surface fitting, we examined two approaches to fit polygonal surfaces to a set of points using explicit and implicit polynomial functions.

Dynamic meshes

During an animation, the vertex positions (geometry) change from frame to frame. Sometimes, the connectivity or the number of vertices also may change in time. Throughout this dissertation, we assume that the sequence of meshes shares the same connectivity and only the vertex positions change in time. The focus on animated meshes with fixed connectivity may be justified by the fact that often the animation creators maintain constant connectivity throughout the animation, in order to allow for easy and efficient manipulation of the sequence. Thus, the connectivity of the mesh is first defined, then the vertices are moved or deformed depending on the way the animation is generated.

Often the meshes differ only slightly between neighboring frames, leading to a large redundancy between frames (temporal redundancy) and between neighboring vertices in the same frame (spatial redundancy). In order to develop a compact representation that significantly reduces the storage space and transmission time of animated models, both spatial and temporal coherence should be exploited. In other words, the large amount of inherent redundancy between frames should be eliminated.

This thesis introduces several compression approaches from *not lossy* or *near-lossless* to *lossy* for animated meshes of constant connectivity. For not lossy compression, we generalize the idea developed for static mesh compression [11, 15]. The local coordinates exhibit a high degree of clustering behavior not only in space but also in the temporal domain. However, a simple predictive-based encoding of each component gives a better bit rate. For lossy compression, two algorithms are introduced. The first algorithm is based on local Principal Component Analysis (PCA) [7, 10] and the second approach is based on a predictive approach and Discrete Cosine Transform (DCT) [12, 13]. Both of these two techniques are combined with a clustering approach

The choice or the design of a compression scheme involves trade-offs along several features. Two of these features are: the compressed file size or the compactness and the amount of distortion introduced by the compression process. The better the quality, the lower the compactness is.

Segmentation

3D Mesh segmentation is a process that partitions the mesh elements that have the same properties into regions. It has become a necessary tool in computer graphics and geometric modelling and it is used for various applications.

Recently, segmentation of deforming triangular meshes has gained much interest and it is used in several contexts in animation, typically, skinning mesh animation [54], ray tracing[37], and compression [77, 38, 102, 84].

While static mesh segmentation aims at detecting meaningful parts and breaks the mesh into sub-meshes of similar features within a specific context, 3D dynamic mesh segmentation approaches exploit the temporal information to partition the mesh into quasi-rigid components.

Compared with the huge number of algorithms proposed for static meshes, segmentation is rarely used in the context of dynamic mesh compression context. Moreover, most of the proposed methods are more suitable for skin meshes but are not efficient to or are even inapplicable for deforming triangular meshes that have lots of motion, particulary in context of compression.

This thesis develops novel segmentation approaches, that are independent of how the animation is generated and can be useful not only for compression but also for other applications.

The objective of partitioning the deforming mesh into near-rigid components is to decrease the computational costs as well as to preserve the global shape of the mesh because some compression algorithms such as DCT and PCA based-coding can destroy important features of the mesh when very few PCA components or very few DCT coefficients are used to recover the original datasets. Thus, it leads to a more compact representation and one can achieve high compression rates with high reconstruction quality. In PCA based coding [7, 10], we want to transform the nonlinear behavior of vertices to a linear fashion by grouping the vertices of similar motion into sets. Then we can efficiently perform the PCA in each group. Thereby, few components can be encoded while the global shape is well preserved. In DCT based-coding that we combine with the predictive coding, the segmentation allows an efficient prediction through time and, thereby, having vertices displacements between two successive frames close to zero. Here also, the clustering will preserve the global shape when DCT coding is performed (spatially) in each cluster.

1.2 Overview of Thesis

This thesis describes new algorithms for static and animated 3D mesh compression, yielding a significant reduction in bit-rates and introduces a new, simple and efficient animation segmentation method that is very useful for compression. These contributions are organized in the following chapters:

Chapter 2 presents some definitions and a description of some general data encoding schemes. These are often used in compression pipelines as a final stage. Then, it gives a description of the geometric representation of 3D objects and of animation – the input data for our algorithms. These are followed by a review and discussion of the most important published works on static and animated 3D objects.

Chapter 3 describes a novel, near lossless geometry compression thechnique [36, 8]. The new geometry encoding strategy follows the predictive coding paradigm, and is based on a region growing encoding order. Only the coordinates of the correction vectors are encoded in a local coordinate system. The vertex coordinates are expressed in term of tangential and normal components. For the tangential components encoding, we investigated the parallelogram or multi-way prediction. For an improved encoding of the normal component, we introduce the so-called higher order prediction. A higher order surface is fit to the so far encoded geometry. Here, two approaches are developed to fit polygonal surfaces to a set of points. In the first approach, we fit the graph $f(x, y)$ of a polynomial function defined over the tangential coordinates x and y. In the second approach, we fit a polynomial implicit function $g(x, y, z)$ such that the zero set represents the surface. This has the advantage that there is no need for guessing the tangential space. As the normal component is encoded as a bending angle, it is found by intersecting the higher order surface with the circle defined by the tangential components. To overcome the high computational time involved in the gathering and weight fitting processes, we develop a faster predictor based on sphere fitting.

Chapter 4 introduces motion-based segmentation for animated meshes [7, 10, 14]. The main idea is to decompose the 3D object into sub-meshes or groups of vertices with similar motions, then compress each group separately. Here, we present three approaches: region growing, clustering, and adaptive clustering based approaches. These algorithms can be applied to different kinds of animated meshes (arbitrary animation) whose connectivity and number of vertices are constant over time. Moreover, we do not need information about how the motion is generated. These approaches can be very useful for other applications.

Chapter 5 introduces single-rate near lossless compression for animated meshes of fixed connectivity based on a simple predictive technique [11, 15]. The algorithm can be seen as a generalization of static mesh compression presented in Chapter 3. The connectivity is encoded once, then the geometry (vertex locations) is encoded by a connectivity traversal of the mesh. Connectivity determines the order of the vertices and provides information for predictions. We are going to show that splitting the coordinate into parametric and geometric information in animation, is a very efficient way of developing simple predictive coding. Indeed, the local spaces exhibit higher temporal and spatial clustering behavior than the world space, and the combination of both clusterings should yield significant reduction in bit-rates. We also going to involve different predictors for tangential and normal components: time-only, space-only and space-time predictors.

Chapter 6 presents a novel Relative Local Principal Component Analysis (RLPCA) based-compression scheme for dynamic meshes [7, 10]. We use Principal Component Analysis (PCA) to represent the original data by very few components and coefficients. Generally, the behavior of the vertices is often non-linear and difficult to extract by a simple global PCA. Therefore, we perform PCA locally. We collect the mesh vertices into groups of similar motion using a region growing based-algorithm or motion based-clustering (Chapter 4), and thus transform the original vertex coordinates into the local coordinate frame of their cluster or segment. This operation leads to a strong clustering behavior of vertices and makes each region invariant to any deformation over time. The local vertex coordinates are then transformed into another basis which allows for very efficient compression.

To improve the PCA-based compression, we introduce a rate distortion optimization that trades off between rate and the quality of the reconstructed animations. Thus, the appropriate number of basis vectors to recover the original data of each group with a certain accuracy is optimally selected using a bit allocation process.

To the best of our knowledge, performing a PCA on the local coordinate system rather than the world coordinates and performing rate distortion optimization has never been performed before.

Chapter 7 combines predictive and spectral techniques [12, 13]. The compression algorithm does not need the connectivity information and belongs to the cluster based prediction approach. The animated mesh is segmented into almost rigid clusters. Then the predictive and DCT coding is performed in each cluster, frame after frame.

Note that the clustering in this approach is used in the prediction phase. We want to have the displacements between two successive frames close to zero, thereby the efficiency of the prediction over time increases. Moreover, the clustering will preserve the global shape when DCT coding is performed in each cluster.

Chapter 8 discusses and evaluates different parameters that affect compression performance, presents the experimental results of the proposed approaches and compares them with the current one.

Chapter 9 concludes this dissertation with a summary of contributions and future research directions.

The work described in this thesis has been published in different conferences and journals [36, 7, 10, 13, 11, 15, 14] as well as technical reports [8, 9, 7, 14].

1.3 Contributions of Thesis

The main contributions of this thesis are summarized as follows

- A Higher order prediction for static mesh compression based on surface fitting and sphere fitting [36].
- Single rate near lossless compression of animated geometry. The approach is vertex based-predictive coding [11, 15]
- Motion based segmentation methods for animated meshes: region growing based approach, static and adaptive clustering [7, 10, 14].
- A novel relative local principal component analysis based compression scheme for dynamic meshes [7, 10].
- A new rate distortion optimization for PCA based-coding [10]
- Predictive-DCT based compression scheme for dynamic meshes [12, 13]

CHAPTER 2

Background

In order to explain and understand the new techniques detailed in the next chapters, it is necessary to go through the main approaches in the literature of static and animated mesh compression and mesh segmentation. This chapter is organized as follows. The first section provides a background on compression. Section 2.2 presents general encoding schemes such as Huffman coding and arithmetic coding which are often used by a compression pipeline as final stage. Section 2.3 describes geometric representations of 3D objects and animations. Section 2.4 covers the most important published works on static meshes and animated meshes. Section 7.3.2 presents some segmentation algorithms developed for static meshes and dynamic meshes.

2.1 Compression

Compression is the conversion of one representation of data into another representation with smaller size. The data reduction results from the elimination of redundancy of the information while preserving content. Data comes in a large variety of forms including written text, speech, 2D still images, video, 3D graphic objects and 3D graphic scenes. Beside compression, there is a decompression process which operates on the compressed representation in order to reconstruct the original data.

Why do we need compression?

With the advances of computer technology, millions of computer-users increasingly produce and consume a lot of data such as text documents, music, videos or graphical data. Day by day a lot of them need to communicate via the Internet or other networks exchanging documents while other users use complex software tools like video editors or video games that need or produce a tremendous amount of data. In addition, all people want to do their job easier and faster. With the increasing data size, efficient storage systems or transmission techniques are necessary. Consequently, compression technologies are required to optimize the amount of memory required to store data and to minimize the required bandwidth of network transmissions.

Compression becomes a part of daily life for millions of people. Digital TV or fast communication via internet would not be possible without sophisticated compression technologies.

The choice or the design of a compression scheme involves trade-offs along several features. Two of these features are the compressed file size (i.e. the compactness of compressed data) and the amount of distortion introduced by the compression process. Often, the better the quality is the lower the compactness is. Basically, compression techniques can be divided into two broad categories depending on whether the original data can be exactly recovered or not, namely *lossless compression* and *lossy compression*.

2.1.1 Lossless Compression

A lossless compression scheme is a scheme that allows the original data to be reconstructed from compressed data without any loss. Lossless compression is necessary for sensitive or very important data when the decompressed data must be identical to the original data. Typical data types for lossless compression are text, executable code or medical images. Typical compression schemes are run-length encoding, dictionary coders (like LZ77 [130] or LZW [122]), and entropy coding such as Huffman coding and arithmetic coding.

2.1.2 Lossy Compression

A lossy compression scheme is a scheme that sacrifices precision in order to achieve high compression ratios. The original data can be recovered with some loss of fidelity. Therefore, a lossy compression scheme often archives higher compression ratios than a lossless compression scheme without greater degradation. The compression ratio is inversely proportional to degree of quality. Usually, applications can use lossy compression schemes if the loss of precision is not perceivable or significant such as video applications, still image editing or audio. Typical lossy compression techniques are MP3, JPEG, Fractal compression and MPEG.

2.2 General Compression Techniques

Various Compression techniques are designed depending on the type of data to be compressed. As mentioned before, these techniques are divided into two classes. Lossless compression schemes which are *reversible* and lossy Compression schemes which accept some loss of data. Note that there are also approaches that combine both lossless and lossy compression which are categorized under lossy category.

Many applications require the decoding process to be able to run in real time like video applications. In contrast, the encoding process is often not required to run in real time and can thus be more computational expensive than the decoding process. But note that this is not always the case as e.g. for speech compression. In addition, the decoding or encoding process must often be implemented with an integrated circuit which has to be considered during the design phase.

This section reviews some lossless techniques that are important for many compression techniques (e.g. for compression of mesh geometry). The next section 2.4 will present some lossy compression techniques that are related to the work of this thesis.

Most information to be encoded is a sequence of symbols which may contain a large redundancy. Entropy encoding reduces the quantity of data without loss of information based on a probability distribution of symbols. It assigns codes to symbols such that the code length matches the probabilities of symbols.

Let $\mathcal{A} = a_1, ..., a_k$ be an alphabet with k different symbols. Each symbol has a probability $P(a_i)$ assigned to it. The amount of information for a single symbol a_i is given by:

$$I(a_i) = log_2 \frac{1}{P(a_i)} \tag{2.1}$$

So, the higher the probability of a symbol is, the lower is its information value. In contrast, the lower the probability of a symbol is, the higher is its information value.

Now, given a sequence S of symbols, i.e. a string of symbols, its entropy H can be calculated as

$$H(S) = \sum_{a \in \mathcal{A}} P(a_i) log_2 \frac{1}{P(a_i)} \tag{2.2}$$

The entropy is a measure for the average number of bits needed to encode strings of symbols of the alphabet $\mathcal{A}$ with the given probabilities $P(a_i)$. Although there can be a string which can be encoded with a bit rate lower than H, the average number of bits for all possible strings cannot drop below H. In that meaning, the information $I(a_i)$ of a single symbol is a measure for the number of bits needed by a minimal code to encode this symbol.

The most popular type of entropy encoding are: arithmetic and Huffman coding. They are lossless and do not offer a good compression ratio but play an important role in a very large number of compression techniques for text, audio, 3D geometric objects, standards such as JPEG and MPEG. They are often used in a final stage to further enhance the compression ratio.

There are also other lossless algorithms such as Run-length encoding (RLE), Lempel-Ziv-Welch (LZW) or Golomb coding which will not be discussed in this thesis and we refer the reader to [103]. We only want to discuss briefly the two techniques above, which are used by a post-processing step in current mesh compression algorithms, and by our approaches presented in the next subsequent chapters.

2.2.1 Huffman Coding

This technique was developed by David Huffman in 1952 [44]. The algorithm uses a variable length code to encode each symbol. The symbols are encoded with lengths that are inversely proportional to the frequencies of the symbols. For example, the symbols that occur more frequently (have a higher probability of appearance), have a shorter code length (are encoded with fewer bits) than the symbols that occur less frequently. To generate the code, a binary tree is created with the symbols at the leaves. The code is then defined by the path from the root of the tree to each leaf. Each symbol is assigned ’0’, if the left child is traversed and ’1’ if the right child is traversed. Now, let’s see how the tree is designed. Given an alphabet $\mathcal{A} = a_1, a_2, a_3, a_4, a_5$ having probabilities $\mathbf{P} = 0.2, 0.25, 0.05, 0.35, 0.15$ (see table 2.1). To generate the Huffman code, first the symbols are sorted in a descending probability order a_4, a_2, a_1, a_5, a_3 (see table 2.2), where each symbol a_i forms a leaf in the first queue as described in 2.1. Then, the two symbols with lowest probabilities are merged producing a single node having a probability equal to the sum of their probabilities. The new node is treated as symbol and inserted into the queue. The nodes are again sorted and the same rule is applied to generate a new parent node for the two symbols with the lowest probabilities. This process is repeated till all nodes are merged and only one is left. The resulting Huffman tree is illustrated in figure 2.1.

To obtain the code of each symbol, the tree is read backwards, starting at the root

Table 2.1 Symbols with their corresponding probabilities

Symbols	Probabilities
a_1	0.20
a_2	0.25
a_3	0.05
a_4	0.35
a_5	0.15

Table 2.2 Symbols with their corresponding probabilities in descending order

Symbols	Probabilities
a_4	0.35
a_2	0.25
a_1	0.20
a_5	0.15
a_3	0.05

and going to each leaf node, recording '0' if the left branch is crossed and '1' if the right branch is crossed. The codewords of the sequence of symbols a_1, a_2, a_3, a_4, a_5 are then $10, 01, 111, 00, 110$ (see table 2.3).

The Huffman coding is near optimal in the sense of having a minimum code length for the set of symbols. It is prefix code which means that the code of each symbol cannot be a prefix or start of the code of another symbol. And it provides a fast decoding process. According to Gallagher [30] the average code length l_H that Huffman coding can achieve is bounded by the entropy and by the entropy plus $p_{max} + 0.086$:

$$H(S) \leq l_H \leq H(S) + p_{max} + 0.086 \tag{2.3}$$

Thereby, p_{max} is the largest probability in the set of symbols probabilities.

If the alphabet size is large, the probabilities are not skewed, and p_{max} is small. Thus, Huffman coding achieves a rate close to the entropy. If the alphabet is small, the probabilities of different symbols are skewed, p_{max} can be quite large and the amount of the deviation from the entropy is large. Thus, Huffman coding becomes inefficient. The coding becomes quite good if the size of alphabet is increased by grouping several symbols together and generating a single codeword for each group instead of generating a codeword for each symbol. Suppose that the size of the alphabet to be encoded is k and the size of each group is m then the total number of all possible groups of size m is k^m. However, the total number of codewords is k^m. If the sequence of symbols to be encoded is large, then Huffman tree will grow exponentially, causing increase in time consuming procedure and memory usage. In this case, Huffman coding is not the appropriate coding

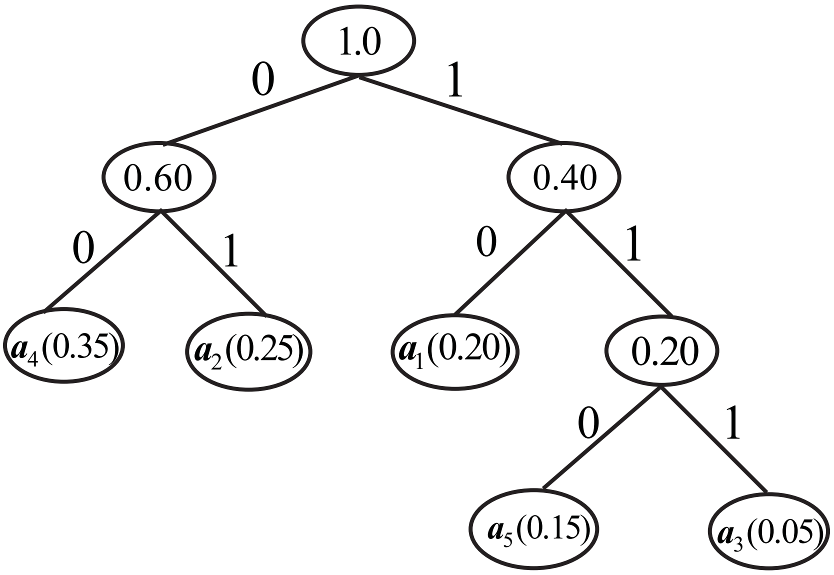


Figure 2.1 A Huffman tree

Table 2.3 Huffman codes

Symbol	Probability	Binary Code
a_4	0.35	00
a_2	0.25	01
a_1	0.20	10
a_5	0.15	110
a_3	0.05	111

(impractical). Arithmetic coding which will be discussed in the next section overcomes this problem. Arithmetic coding encodes the whole sequence of symbols as a block rather than encoding several blocks and can get the coding rate very close to the entropy bound, even for short alphabets or highly skewed symbols's probabilities.

Adaptive Huffman Coding Huffman coding requires a priori knowledge of the probabilities of the symbols. It is then a two pass process: one pass computes the probability of each symbol and constructs the Huffman tree and transmits the tree, and a second pass encodes and transfers the data. In other words, the table code should be sent to the receiver with the encoded stream in order to decompress the data. The size of the coding table therefore becomes large when data size increases. Moreover, in practice, the probabilities are not always known a priori and may change over time. In such cases, a method that compresses data in one pass is need. Therefore, an adaptive Huffman coding is used that allows to build adaptively the Huffman code depending on the sequence of symbols. So, the Huffman code can change if new symbols are consumed. The original algorithm

Table 2.4 arithmetic codes

Symbol	Probability	Cumulative probability	Sub-Interval
a_1	0.4	0.4	$[0.0; 0.4)$
a_2	0.1	0.5	$[0.4; 0.5)$
a_3	0.2	0.7	$[0.5; 0.7)$
a_4	0.3	1.0	$[0.7; 1.0)$

has been developed by Faller [29] and Gallagher [30] and has later been improved by Knuth [68] and Vitter [120]. The main idea is to start coding with an empty Huffman tree, then update the symbol account and the tree as being read and compressed/decompressed. Both the encoding and decoding start with the same tree and modify it in the same way. Thus, both processes are synchronized.

2.2.2 Arithmetic Coding

Unlike Huffman coding which assigns a code word to each symbol and uses an integer number of bits (at least one bit to encode each symbol), arithmetic coding encodes the entire sequence to a rational number in [0,1). All probabilities of the symbols fall into the range [0,1). The algorithm starts with the unit interval [0,1), and transforms the possibilities into subintervals. As each symbol is processed, the algorithm divides the current interval into subintervals, each subinterval represents a certain symbol, and its size depends on the symbol probability. Then, the subinterval of processed symbol is selected to be the new currently interval. For example, let us suppose that we have an alphabet $\mathcal{A} = a_1, a_2, a_3, a_4$ with probabilities $\mathbf{P} = 0.4, 0.1, 0.2, 0.3$ as shown in table 2.4.

We define the cumulative density (*cdf*) function as

$$F(i) = \sum_{k=1}^{i} P(a_k)$$

$P(a_k)$ is the probability of the the symbol a_k. The cumulative probabilities are then given in table 2.4 (column 3). The *cdf* is used to divide the unit interval [0,1) into subintervals of the form $[F(i-1)$, F(i)) (column 4), each symbol has it own subinterval, where the minimum value of *cfd* is zero and the maximum value is one.

Let us for example encode the sequence a_1, a_2, a_3, a_4. The first symbol to be encoded is a_1, the corresponding subinterval $[0.0; 0.4)$ is selected to be the new current interval. This interval is divided similarly as the interval [0,1) yielding the following subintervals: $[0.0; 0.16), [0.16; 0.20), [0.20; 0.28), [0.28; 0.40)$ corresponding to the symbols a_1, a_2, a_3, a_4. The next symbol to encode is a_2. The corresponding subinterval

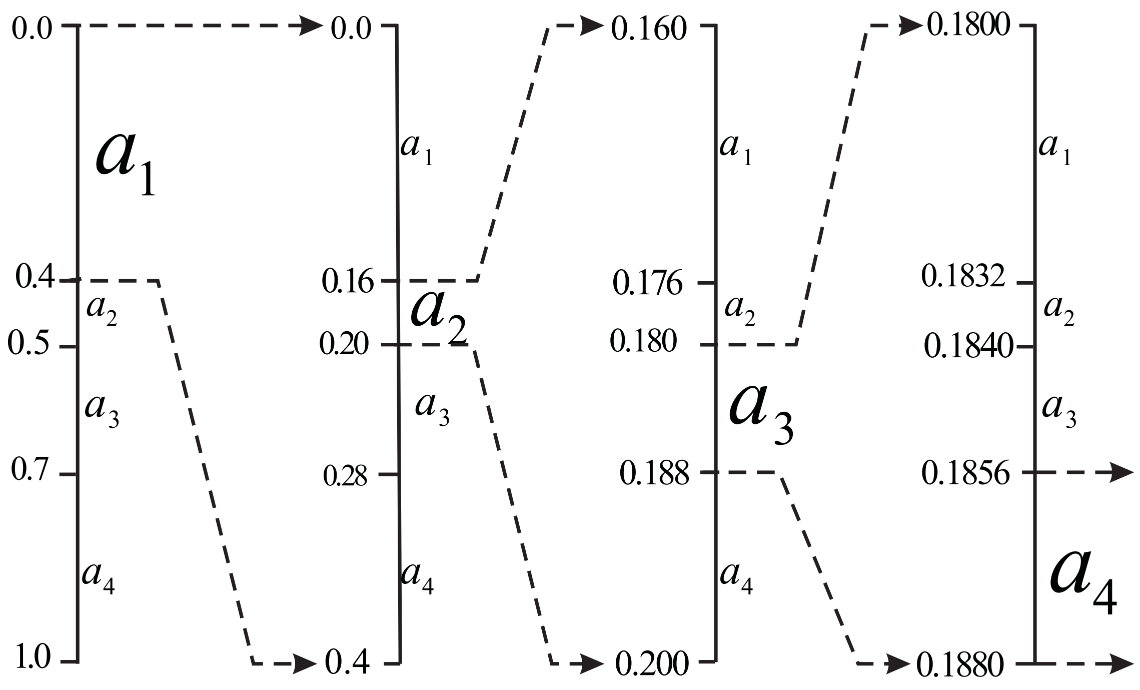


Figure 2.2 Arithmetic coding process

is $[0.16; 0.20)$. We divide this interval in similar way as before producing new subintervals: $[0.160; 0.176), [0.176; 0.180), [0.180; 0.188), [0.188; 0.200)$. To encode the next read symbol a_3, the interval $[0.180; 0.188)$ is partitioned further to give $[0.1800; 0.1832)$, $[0.1832; 0.1840)$, $[0.1840; 0.1856)$, $[0.1856; 0.1880)$. This process of subdivision for the input sequence a_1, a_2, a_3, a_4 is graphically illustrated in Figure 2.2. The final interval which assigned to last symbols a_4 is finally $[0.1856; 0.1880)$.

To encode the sequence of symbols $a_1a_2a_3a_4$, an unique identifier or the tag should be generated. This tag can be any number in the final interval ($[0.1856; 0.1880)$). On can choose for example the lower limit of the interval or midpoint or upper and lower limits of the interval. In our example one can choose 0.1868 as our identifier for the sequence. Notice that the tag appear in the subinterval of each symbols.

In arithmetic coding, the decoding process is similar to the encoding process. To reconstruct the sequence, we need to know two information: the probabilities of the symbols and the tag. First, we calculate the *cdf* for the symbols in the same way as in encoding process. We will get the same values of the cumulative probabilities and subintervals shown in table 2.4. Then, we check in which subintervals the tag belongs. The tag resides in the subinterval $[0.0; 0.4)$ which is assigned to the symbol a_1. Thus the first decoded symbol is a_1. Next, the subinterval becomes the current interval, it is divided into the same subintervals as in figure 2.2. The tag is now reside in the subinterval $[0.16; 0.20)$

corresponding to the symbol a_2. The process continues till all symbols are decoded.

Arithmetic coding is more efficient and can get the coding rate very closer to the entropy bound than Huffman coding but it runs more slowly, particulary for real time data processing since it encodes the sequence of symbols one by one. Therefore, adaptive arithmetic coding which exploits the same idea as adaptive Huffman, is used.

Adaptive Arithmetic Coding Adaptive arithmetic coding reduces the two pass to two to one pass process, where the probabilities of symbols are updated depending on data processed. Thus, eventually, the coding initializes the probability distributions of one or more of symbols to some predefined values, such as 1. Then the cumulative distributions are updated after each symbol is encoded. Adaptive coding can be computationally expensive and complex because of the updating process of the cumulative distributions. To reduce this complexity, different approaches of updating process are developed (see [99]).

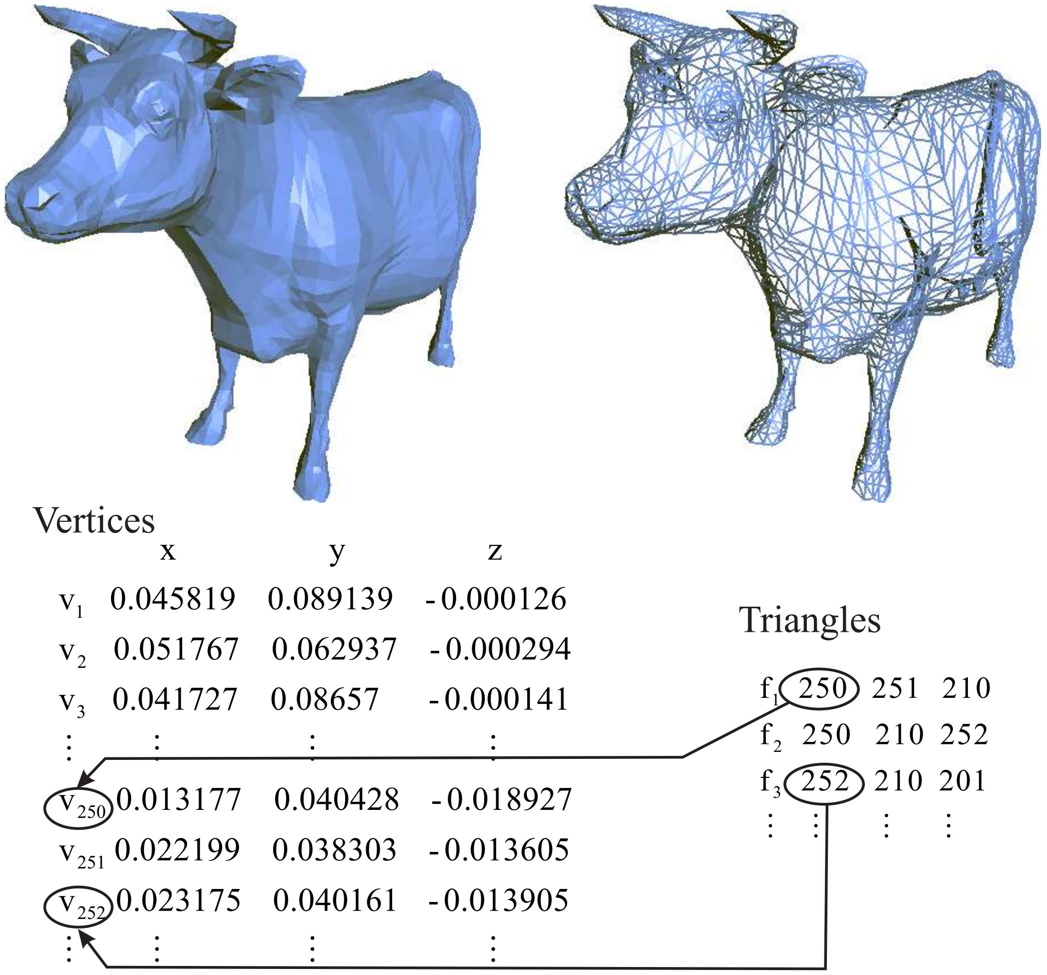
Note that arithmetic Coding is used in many compression standards such as H.264 and JPEG2000.

2.3 3D Data Representation

Text, audio, image, video, etc. are examples of data that computer users almost daily use and need to compress either for storage or transmission. This thesis is being carried out to develop new compression techniques for static and dynamic 3D models represented by triangle meshes, a type of data used in many computer graphics applications.

2.3.1 Static 3D Object

Today, we find 3D objects in many areas such as industrial visualization, medicine applications, video games, movies etc. These objects can be represented in different ways such as boundary representation, parametric surfaces, point sampled surface, implicit surfaces or voxel based representations. The most prevalent representation of an object and the one which will be the input of our all algorithms is the boundary representation. The most common geometric representation for boundary 3D geometric objects is the irregular polygonal meshes which consist of a set *V* of vertices, a set *E* of edges and a set *F* of faces. A vertex represents a node or point. An edge is a straight line that connects two a pair of vertices. A face consists of a set of connected edges defining a triangle, a quad or some higher degree face. Triangle meshes are widely used to represent three-dimensional objects or scenes. One reason for this widespread use of this type of polygons is due to the current graphics hardware which efficiently supports triangles for realtime rendering



Vertices

	x	y	z
v_1	0.045819	0.089139	-0.000126
v_2	0.051767	0.062937	-0.000294
v_3	0.041727	0.08657	-0.000141
⋮	⋮	⋮	⋮
v_{250}	0.013177	0.040428	-0.018927
v_{251}	0.022199	0.038303	-0.013605
v_{252}	0.023175	0.040161	-0.013905
⋮	⋮	⋮	⋮

Triangles

f_1	250	251	210
f_2	250	210	252
f_3	252	210	201
⋮	⋮	⋮	⋮

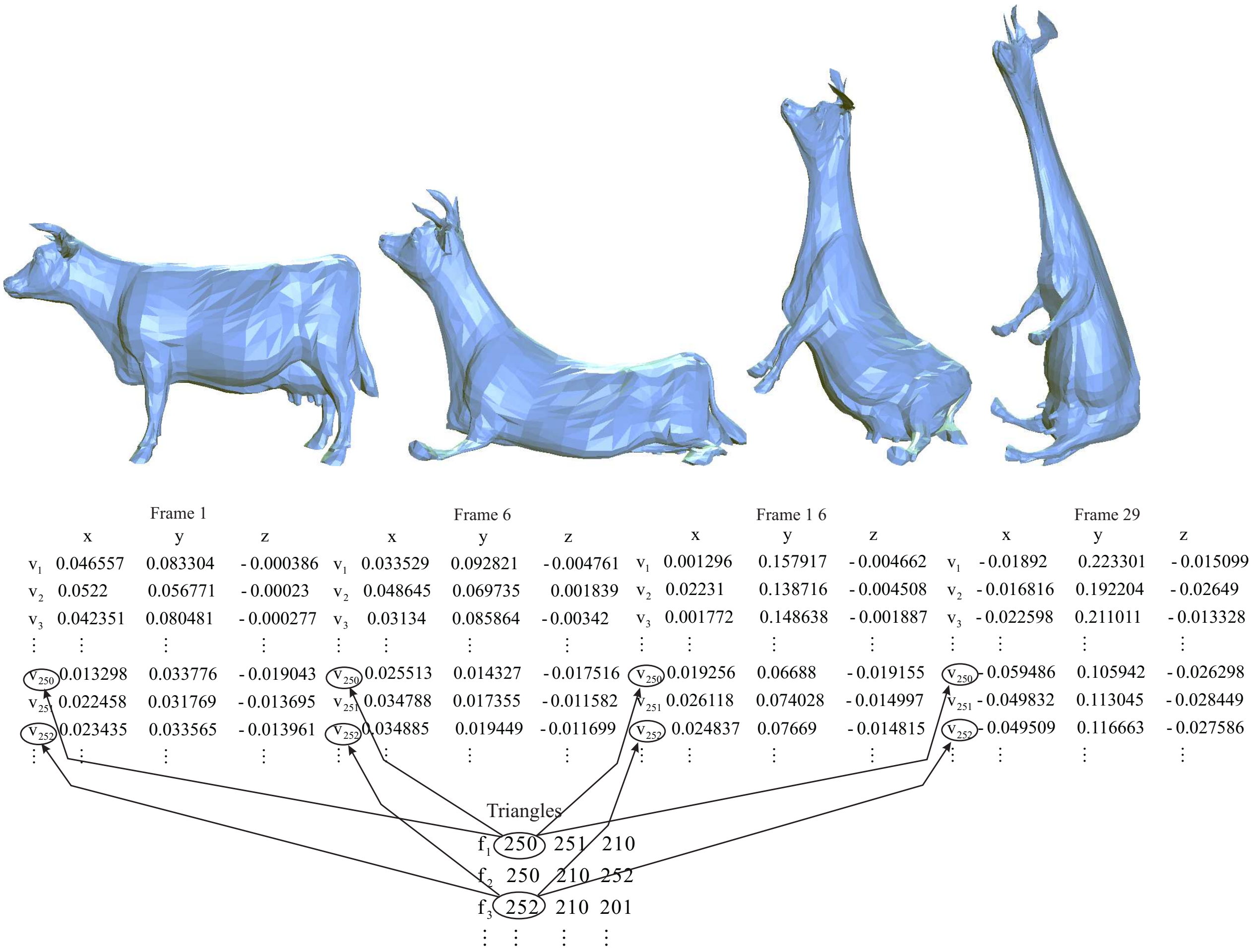
Figure 2.3 A mesh example: Cow model (2904 vertices, 5804 triangles). Top left: shaded display. Top right: wireframe. Bottom: triangle mesh representation (vertex and face arrays

of objects at different complexity. Moreover, all other forms of polygons are typically converted into triangles by the graphics hardware before rendering. Figure 2.3 shows an example of a 3D mesh.

Basically, triangle meshes consist of the *geometry* data and *connectivity* data.

Mesh Geometry

Mesh geometry is represented by an array V of vertex positions in 3D space, i.e. each vertex position is defined by its coordinates (x, y, z) specified by floating point numbers (see figure 2.3).



	Frame 1				Frame 6				Frame 1 6				Frame 29		
	x	y	z		x	y	z		x	y	z		x	y	z
v_1	0.046557	0.083304	- 0.000386	v_1	0.033529	0.092821	- 0.004761	v_1	0.001296	0.157917	- 0.004662	v_1	- 0.01892	0.223301	- 0.015099
v_2	0.0522	0.056771	- 0.00023	v_2	0.048645	0.069735	0.001839	v_2	0.02231	0.138716	- 0.004508	v_2	- 0.016816	0.192204	- 0.02649
v_3	0.042351	0.080481	- 0.000277	v_3	0.03134	0.085864	- 0.00342	v_3	0.001772	0.148638	- 0.001887	v_3	- 0.022598	0.211011	- 0.013328
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮
v_{250}	0.013298	0.033776	- 0.019043	v_{250}	0.025513	0.014327	- 0.017516	v_{250}	0.019256	0.06688	- 0.019155	v_{250}	- 0.059486	0.105942	- 0.026298
v_{251}	0.022458	0.031769	- 0.013695	v_{251}	0.034788	0.017355	- 0.011582	v_{251}	0.026118	0.074028	- 0.014997	v_{251}	- 0.049832	0.113045	- 0.028449
v_{252}	0.023435	0.033565	- 0.013961	v_{252}	0.034885	0.019449	- 0.011699	v_{252}	0.024837	0.07669	- 0.014815	v_{252}	- 0.049509	0.116663	- 0.027586
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮

Triangles

f_1	250	251	210
f_2	250	210	252
f_3	252	210	201
⋮	⋮	⋮	⋮

Figure 2.4 Sample frames of cow sequence (top) and the standard mesh representation (bottom)

Mesh Connectivity

The mesh connectivity is represented by an array F (sometimes noted T) of indexed triangles, describing how the vertices are connected. Each triangle f_i is defined by three indices that point into vertex list, as illustrated in figure 2.3.

Mesh Formats

Many different 3D formats have emerged to import and export the meshes such as: **OBJ**, **WRL**, **PLY** or **DXF**. The use of the format may differ from application to application depending on its content that requires different specification.

2.3.2 Dynamic 3D Object

Animation gives life to static 3D objects. It attracted a surprising amount of attention from marketing, arts, sciences, television and particulary in the computer games industry, and special effects in movies. In animations, we have to take into account that the object may change over time. To model the animation different approaches can be used. Generally, the animation can be either generated using motion capturing systems or simulated by sophisticated software tools like Maya and Max 3D. The popular representation of these animated objects is a triangle mesh to suit the requirements of computer graphics visualization systems. A dynamic 3D object is then a sequence of meshes, each mesh represents one frame, which we call also mesh frame.

During an animation, the vertex positions (geometry) change from frame to frame, eventually, the connectivity or the number of vertices also may change in time. Throughout this dissertation, we assume that the sequence of meshes share the same connectivity and only the vertex positions change over time, because, very often the number of vertices and the connectivity of the mesh is first defined, then the vertices are moved or deformed depending on the way of generating the animation. Figure 2.4 shows sample frames of the cow sequence.

In 3D animation, various formats are provided such as **MAX**, **MDL**, **MS3D** or **FBX**.

2.4 Prior Mesh Compression Techniques

With the advancements and variety of sources to model 3D objects such as scanning technologies and modelling softwares, 3D models are becoming widely available. In order to achieve a higher degree of realism, more complex and highly detailed 3D objects possibly out of millions of vertices are created. As seen in the previous section the standard representation of the triangle meshes uses a list of floating points to describe the vertex positions and a list of integer values that specify the vertex indices and, eventually, a list of floating proprieties attached to the mesh (normals, textures, etc).

The storage cost of uncompressed mesh geometry information is $V * 3 * 32$, where V is the number of mesh vertices, 3 are the three coordinates of each vertex (X, Y and Z) and 32 is the number of bits required to store the floating point value of each coordinate. The cost of the uncompressed mesh connectivity is $T * 3 * 32$, where T is the number of triangles and 3 is the three vertex indices (per triangle) and 32 is the number of bits required to store these three integer indices.

When storing and downloading or uploading these 3D objects over networks, in their standard form, the large raw data sets will consume a large amount of storage space and

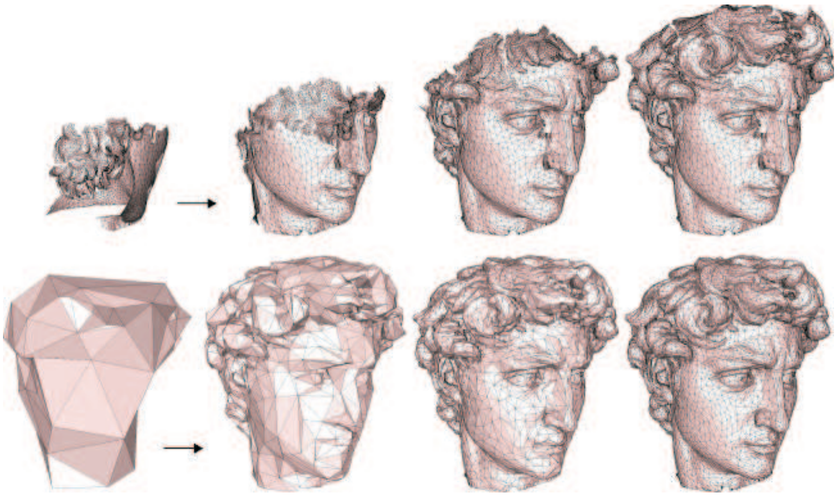


Figure 2.5 Intermediate stage during the reconstruction of mesh using a single rate scheme(top) and a progressive scheme (bottom). (Figure taken from [6])

network bandwidth.

This problem heavily arises in the animation case. Today, it becomes easier to acquire animated models. In parallel, there is unceasing increase in the use of these models in many applications (particulary in computer generated movies, computer games). Moreover, the animation technology is becoming more sophisticated and allowing to make animation more realistic but more complex. To store a sequence of meshes, one has to store one mesh for each frame. Assuming that the connectivity is constant over time and only the geometry information changes over a time, the representation of the geometry information of the sequence will require F times the geometric information of each frame i.e. $F * V * 3 * 32$ bits, where F is the number of frames. For large sequences and detailed and high accuracy models, the uncompressed representation results in large file expensive to store or to deliver over networks.

Different approaches have been developed to reduce the size of the polygonal mesh and time of transmission. These techniques can be classified into *single-rate encoders* and *progressive encoders* (see figure 2.5 illustrates).

Single-rate encoders encode the original mesh as whole and once. These schemes are dedicated to reduce bandwidth between rendering pipeline and local memory. Most of these schemes use a fixed mesh traversal strategy to encode the connectivity, then the geometry is driven. Connectivity encoding is often based on a set of rules that describe a particular traversal order of mesh based on vertices, edges or triangles. Typical, in triangle based traversal, for each new traversed triangle, new symbol is generated and entropy encoded.

Geometry coding is often guided by traversal order used to encode the connectivity. For each new encountered vertex, its 3D coordinates are predicted from the already traversed vertices and the differences between the original and the predicted locations is encoded. The difference is called *residual* or *delta vector* or *prediction error* or sometimes *offset*. The residual can be encoded in different ways. Most single rate compression schemes quantize the residuals into a uniform integer grid and often the user has to specify the number of bits that define the grid resolution. For further compression the resulting integer residuals are entropy encoded.

Progressive encoders first create a simplified version of the original full resolution mesh using simplifications operations (such as vertex elimination, edge collapse). The coarsest version of the mesh is then encoded -using a single rate compression techniques- with a sequence of reversed simplification operations: the refinement operations (vertex insertion,vertex split). Each operation specifies how to add edges and vertices to the mesh of current *level of details* to obtain a new level. After transmission, the base mesh is first decoded and reconstructed, then gradually refined as the bitstream is being received and decoded.

The advantage of progressive compression is that it allows to see a first approximation of the original model *very quickly*, the quality of the model gradually is improved and the transmission can be stopped at any resolution desired by the user.

Single-rate techniques can also be categorized into lossless and lossy techniques depending on whether the decompression process recovers exactly the original mesh (lossless) or only an approximation (lossy). The remeshing and simplification are always lossy.

In recent years, the compression of animated meshes has gained an increasing interest and many techniques have been developed for dynamic meshes of constant connectivity. This topic is a relatively young research area and still attracts much interest. There are still many open problems to be addressed for further research such as the compression of animated meshes of variable number of vertices and distortion measurement of animation sequences.

As in the static case, 3D animation compression schemes can also be categorized into single-rate and progressive compression, taking into consideration the spatial and temporal coherence.

The next sections review the most important compression approaches developed for static and animated meshes. For more details on static and animated mesh compression techniques, we refer the reader to [97, 6, 94, 83].

2.4.1 Static Mesh Compression

The base of the mesh compression has been laid by Deering in 1995. His goal was to reduce the amount of information, which has to be sent to the graphics accelerator, and was constraint to a simple decompression algorithm that can be supported in a hardware implementation. This algorithm was the first step to develop single-rate compression for triangle mesh. Afterwards, mesh compression has undergone rapid developments.

2.4.1.1 Single-Rate Encoders

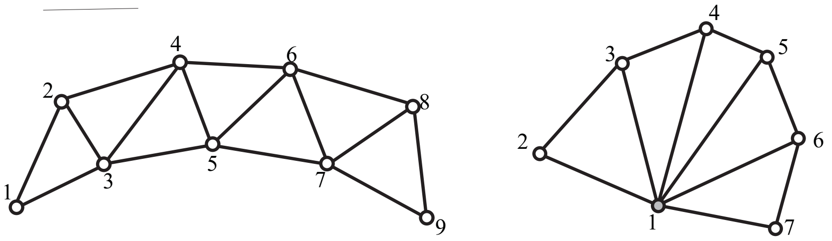
Most of the mesh compression techniques encode two mesh components separately, the connectivity and the geometry information. The early work concentrated on finding an efficient and optimal connectivity coding scheme. However, they encode the connectivity information first before they encode the geometry information in terms of traversal order which is used to encode the connectivity. These approaches are called connectivity driven compression schemes. In contrast, the geometry-driven compression techniques have emerged. These approaches encode the geometry first then the connectivity is driven.

The number of contributions to static geometry compression is very low compared with the large number of published work on connectivity compression. In contrast, in the animation case, most contributions are in animated geometry mesh compression.

2.4.1.1.1 Connectivity Coding Typically, the mesh connectivity is described by a list of faces, each face represented by three vertex indices. If the number of vertices is V then the number of faces is approximated by $2V$, requiring about $6\lceil log_2(n)\rceil n$ bits of storage [42]. If a vertex is shared by six triangles and for each triangle three vertices should be transmitted then the transmission of each vertex become expensive. Indeed, in average, each vertex should be transmitted six times.

Triangle Strip based Techniques

In graphics API such as OpenGL or Direct3D, the triangle are rearranged in *strips* for compact representation and efficient rendering of 3D polygonal meshes. Triangle strips use a buffer of two vertices and allow to reduce the number of times to transmit each vertex. Since two consecutive triangles share one edge, to render a new triangle, only one new vertex is needed and added to the two vertices of the joined vertex. Thus , for each triangle only one vertex is transmitted. Figure 2.6 (a) shows an example of a triangle strip. A *triangle fan* is another structure in computer graphics that reduces storage space and processing time. As shown in 2.6 (b), all adjacent triangles share the same vertex.



Triangle strip: 1 2 3 4 5 6 7 8 9

Triangle fan: 12 3 4 5 6 7

Figure 2.6 Example of triangle strip and triangle fan.

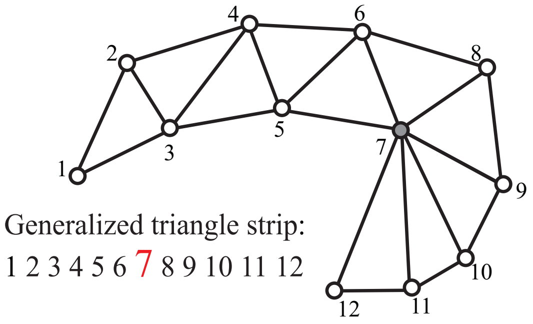


Figure 2.7 Example of generalized triangle strip

A more general structure is the *generalized triangle strip* which is a mixture of triangle strips and triangle fans.

To further reduce the cost of the connectivity information, Deering [28] proposed the *generalized triangle mesh*, well-suited for hardware implementation. He suggested in his pioneering algorithm to use generalized triangle strips with an on-line buffer in the graphics pipeline where sixteen vertices can be stored. To render each triangle one vertex is specified either from the stack buffer or a new vertex is picked out and pushed onto the stack replacing an existing vertex and referenced in the future if it is needed. Figure 2.7 shows a generalized triangle strip and a generalized triangle mesh. Chow [23] further improved this approach. He introduced local and global meshing algorithms to decompose an arbitrary triangulated meshes into generalized triangle meshes.

Turan [118] proposed to encode planar graphs in constant number of bits per vertex using a spanning tree. Taubin and Rossignac [116] extended Turan's work and developed the *topological surgery* method. They first cut the mesh through a *vertex spanning tree*, producing a connected triangulated surface without internal vertices whose dual graphic is *triangle spanning tree*. Then, they encoded the triangle and vertex-spanning tree separately. The Topological Surgery scheme has been included into the three dimensional Mesh Coding (3DMC) algorithm in MPEG-4, the ISO/IEC standard developed by MPEG

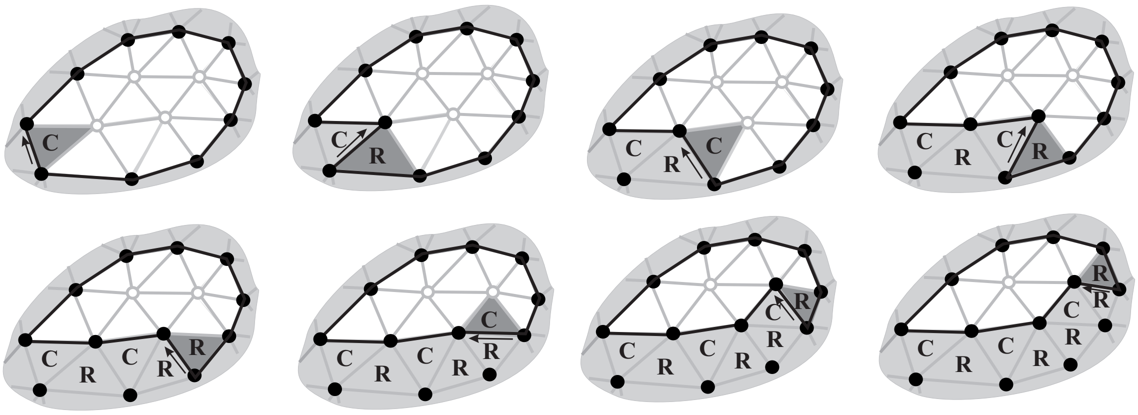


Figure 2.8 Face based coding:Edgebreaker yields the sequence of symbols: CRCRRCRR...

(Moving Picture Experts Group), charged with the development of video and audio encoding standards.

Growing Region based Techniques

Alternative approaches to triangle strip and spanning tree based techniques are *region growing* methods. The basic idea is to start with single triangle then grow the region over the mesh and continuously encode the new adjacent triangle and edges, producing a sequence of symbols or labels that describe the connectivity information. This sequence is entropy encoded. The region growing techniques can be classified into three classes: *face-based*, *edge-based* and *vertex-based*, depending on the mesh elements based traversal.

Face-based methods. The Cut Border Machine (CBM) of Gumhold and Straßer [35] and Edgebreaker of Rossignac [96] are two examples of face-based methods. Both methods are quite similar. They perform either depth-first traversal order or breadth-first traversal order. The encoding process starts with mesh border or an arbitrary triangle then grows the region over the mesh. This region is bounded by a border of edges which enclose the inner region containing the set of triangles already processed and the outer region of triangles which have to be processed. This border is called the cut border. At least one edge in the cut border is incident to one triangle in the inner region and the other incident triangle that have to be added to the growing region. This edge is called gate. The coding starts with the initial triangle as inner region, and one of its three edges is selected to be the active gate. Then in each step, the inner region grows by the unprocessed adjacent incident to the active gate. The triangle is encoded by specifying a label that defines the way in which the new triangle is located relative to the gate edge and the inner part and it is called cut border operation. The label is assigned a code, the triangle is inserted into inner part and marked as visited, the cut border is updated, and a new gate is selected. The process of iteration stops when all triangles in the mesh are processed. Note that all vertices contained in the cut border are buffered to avoid to send the vertex more than once.

To define how the new triangle is incorporated into region growing, Edgebreaker uses the five labels C, R, L, S and E, each label corresponds to one operation. CBM uses additional *offset* associated with the split operation or label S.

Both techniques reconstruct the connectivity using the same traversal used during the encoding process. The decoding in the Cut border machine is simple, fast and well-suited for hardware implementation. The Edgebreaker decoding uses a look-ahead procedure during split operation (S) and exhibits a non-linear time complexity. Therefore, two decoding schemes of *clers*: Wrap&Zip [98] and Spiral Reversi [52], have been proposed to improve the worst case time complexity from $O(n^2)$ to $O(n)$. The Wrap&Zip needs multiple traversals during the encoding and decoding process for meshes with handles and/or boundary. The Spiral Reversi performs only a single traversal and reconstructs the mesh in reverse order as well in linear time.

Edge-based methods. In contract to the face-based methods which encode triangles, the edge-based schemes encode edges. Indeed each edge is assigned one label. Face Fixer of Isenburg and Snoeyink [51] is an example of edge-based method, inspired by the face based Edgebreaker method and it is developed for polygonal meshes.

The encoding process (in Triangle Fixer), first defines an active boundary. One of its edges is selected to be active gate. In each iteration the active gate is labeled with either T, R, L, S, E, H, or M depending on its adjacency relation to the boundary. Then this boundary is updated and new active gate is selected. The sequence of labels are compressed using order-3 adaptive arithmetic coder [123].

In opposite to the Edgebreaker and CBM schemes which use the same traversal order during compression and decompression, the Fixer Face processes the label sequence in reversed order to reconstruct the connectivity of triangle meshes.

Vertex-based methods These techniques are also called vertex degree-based and valence-based. They encode the connectivity information as a stream of vertex valences in specified order. The valence or the degree of a vertex is the number of connected edges to it.

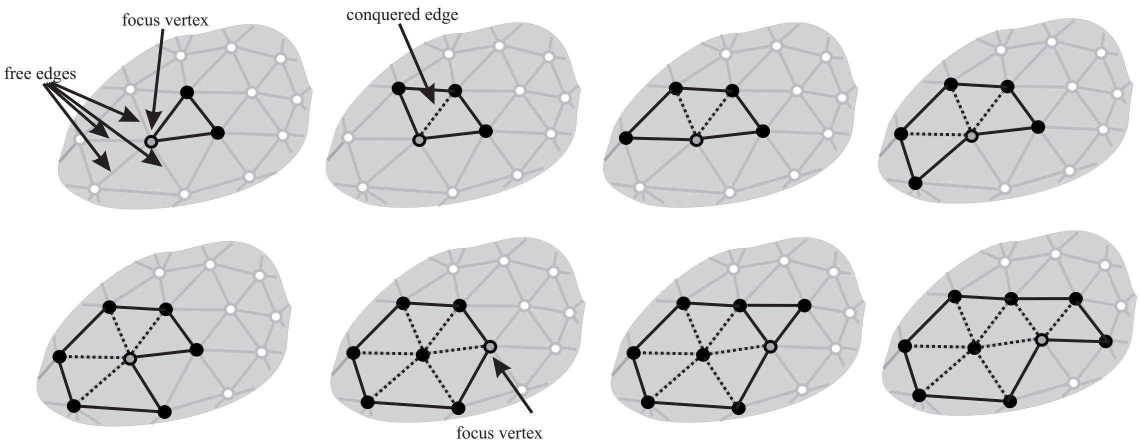


Figure 2.9 Vertex based coding: TG coder yields 66657566...

The method *Triangle Mesh Compression* of Touma and Gotsman [117] (TG) is one of these schemes that encodes the vertex degrees in spiral way. The encoding process starts with an initial triangle, constructs an initial cut border with its three edges and an active list with its three vertices. One of these vertices is chosen to be a focus vertex. The region grows by conquering the free edges -untraversed edges- incident on the focus vertex in counter-clockwise order, pushing the free vertices down into the active list and recording their valence. The free vertex is the one connected to the focus vertex by the free edge. When all free edges (connected to focus vertex) have been processed the focus is conquered and the focus is moved to the next vertex in the active list. The process continues until all vertices are full, i.e. all edges are processed.

During the conquest procedure, special cases may arise and for such cases additional code are needed. Indeed, it can happen that the free edge of the focus vertex is connected to another vertex in the active list, in such case the active list is split into two separated active lists and a new command is generated: *split* with the vertex valence. Another case may arise is when the free vertex is connected to a vertex in another active list. In this case, both active lists are merged to generate one active list and the symbol *merge* is output. If the mesh has boundaries then before the encoding process a *dummy* vertex is temporally added to each boundary and connected to all boundary vertices, in order to have a closed mesh. When dummy vertex is hit then the number of its valence and the code "dummy" are output. Note that the dummy vertices are removed after mesh decoding.

For large number of triangle meshes, the distribution of the valance is very law and often the mesh contains a large number of vertices with the valence of six. In such a mesh, the sequence of vertices valences can be well encoded with the entropy encoding. For regular meshes, TG compress the connectivity down to 0.2 bvp and between 2 and 3.5 bvp otherwise.

TG's algorithm is further improved by Alliez and Desbrun [4] by using an adaptive conquest over the mesh to avoid as much as possible the *split* operations. The sequence of valences and the few additional codes are encoded using an adaptive arithmetic encoder. For large, arbitrary meshes, they achieve an upper bound of 4.24 bpv.

Isenburg and Snoeyink [50] performed a series of edge contraction and division to collapse the whole mesh into a single vertex. For each edge contract operation, the degree of vertex is output and for each edge division, a start and an end is output. The entropy encoding of the resulting code sequence yields a compact bitstreams of 1 to 4 bpv.

There are many others algorithms have been proposed for compressing the mesh connectivity and often in lossless way, such as [17, 53, 59]. An efficient coding of the symbol stream such as Huffman coding or arithmetic coding allows coding the connectivity with often less than 2 bits per vertex and never more than 4 bits per vertex.

2.4.1.1.2 Geometry Coding

Geometry Coding is the compression of vertex coordinates (x, y, z). Traditionally, each vertex coordinate is represented with an IEEE 32-bit floating-point number. The early research on mesh compression focused mostly on efficient lossless coding of triangle mesh connectivity, as mentioned before, achieving the best bit-rates of 1.5-4 bits per vertex on an average (as reported in [70]). In contrast, the compression performance of geometry has not been as well impressive. Indeed, the vertex positions are mostly encoded in traversal order induced by the connectivity code. This leads usually to near or non-optimal geometry coding. Moreover, the geometry code dominates the total compressed data size. Therefore, the researches later shifted more to geometry coding, where also geometry-driven compression schemes have been proposed.

Most of the proposed paradigms to reduce the amount of vertex positions use the following combination: prediction to exploit the high correlation between the positions of adjacent mesh vertices, quantization to reduce the floating point to finite precision, and entropy encoding to reduce the statistical redundancies. Spectral methods are another type of methods for geometry coding that generalize 1D and 2D signals to 3D geometry. It employs as well quantization and entropy encoding, in order to achieve a higher compression rates.

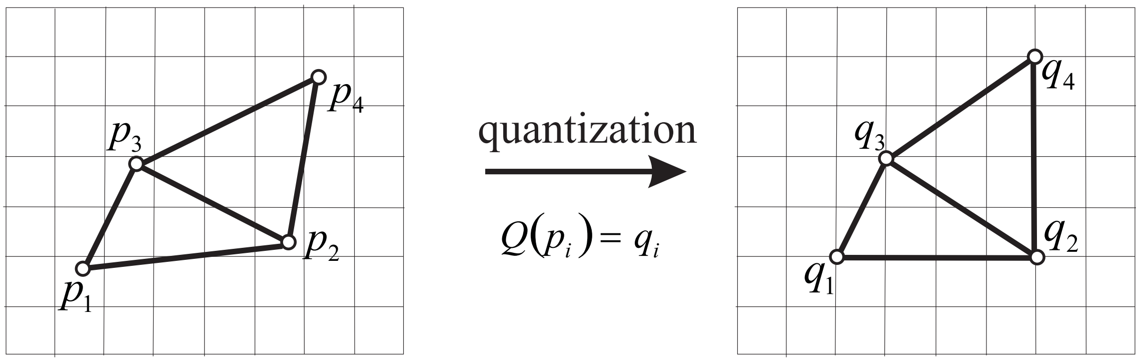


Figure 2.10 Quantization grid

Quantization

It is a standard technique, commonly used to compress numerical data. It reduces the precision of data prior to that data being encoded with finite precision. The resulting values are represented with a limited number of bits and they can be compressed efficiently. Quantization is an irreversible process, i.e. it is impossible to recover the original coordinates once encoded. The inverse process is called *dequantization*. Quantization allows for higher compression ratio at the expense of distortion in the reconstructed data. We distinguish between two type of quantization: scalar and vector quantization.

Scaler quantization (SQ). SQ truncates *uniformly* the vertex coordinate of 32- or 64-bits floating point values to desired accuracy and converts them into integers of, typically, 16 bits with very negligible loss in data fidelity. This reduces the range of input dataset before the encoding stage. To do this, first the bounding box of a mesh is computed. Given a number of bit length for each coordinate n, the bounding box is divided into $2n$ grids along X, Y and Z axis. The side length of each cell is $L_{max}/2^n$, where $l_{max} = (max - min)$ is the tight axis-aligned bounding box, defined by the maximum max and the minimum min of the coordinates x, y and z. Then, each vertex is aligned with the nearest grid intersecting point. The new positions in the grid system is the new integers coordinates. The parameters of bounding box (max and min) must also be sent with the compressed data to be able to construct the bounding box and to recover the original quantized points at decoding step.

Figure 2.10 shows quantization of 8 two dimensional points to uniform 8×8 grid. Since the original coordinates cannot be recovered and only an approximation is reconstructed, the quantization then occurs an error. The lower the quantization level (the coarser the grid resolution), the more compact data, and the larger the error between the original and reconstructed data, resulting in model with blocky structure.

Quantization can be uniform or non-uniform. In uniform quantization, a given the number of quantization, the bounding box is partitioned into a uniformly spaced grid structure, meaning that all grid cells are of equal side length, as describe before. In opposite, in non-uniform quantization, the cells have different lengths.

To encode vertex coordinates, Deering [28] suggests to quantize uniformly each vertex coordinate to at most 16 bits. At this precision, the reconstructed mesh vertices are visually indistinguishable from the true ones. To achieve higher compression, Chow [23] subdivides the mesh into different regions based on triangle size and the curvature. The regions are then uniformly quantized with different precision depending on the level of detail present. The highly detailed regions are assigned more bits than the less detailed ones.

Vector quantization (VQ). VQ is extension of the scalar quantization. It operates on vectors instead of individual values. It maps the input data set into small set of vectors or representatives called the *codebook*, minimizing the total distortion incurred by the quantization. The advantage of VQ over SQ is that it exploits the correlation between the vertex coordinates and results in lower distortion than SQ, but its complexity increases with the codebook size. Since the aim is to generate a codebook that minimizes the distortion between the input data and the codebook, the performance of compression then depends on the codebook generation method. The most popular codebook design is the Linde-Buzo-Gray (LBG) algorithm [79], also known as Generalized Lloyd Algorithm (GLA). The algorithm is an iterative algorithm. Its starts with one codevector chosen as the average of the entire all training vectors. Then, at each step the algorithm refines the codebook using splitting procedure until the number of desired codevectors is obtained.

VQ has also been proposed for geometry coding [73, 22, 18]. In the literature, it is often used after prediction process. Therefore, sometimes are categorized under predictive techniques. VQ based techniques will then be reviewed later in the next paragraph.

Prediction Coding

There is a large correlation between the positions of the adjacent mesh vertices. To exploit this high correlation and to reduce the redundancy, most techniques use prediction rules. The prediction uses a set of known positions of vertices to a decoder, to predict the position of a new vertex. The differences between the original and the predicted values have distribution which is close to zero. Then, instead of encoding the true values, it is sufficient to store and to encode these differences. These can be well encoded with entropy encoding, resulting in fewer bits than would be results from the entropy encoding of the quantized original positions.

The type of this technique is called predictive method. As aforementioned before, the connectivity information plays a crucial role during the encoding geometry. It defines not only the order of vertices in which are encoded but also the set of vertices that are used for prediction. The better prediction is, the smaller error is and the better compression performance we achieve. Today it becomes common for geometry compression to quantize the coordinates with a number between 10 and 16 bits.

In the literature, different predictions have been proposed such as delta predictor, spanning tree predictor and parallelogram predictor. The satisfactory predictor is the one who predict a position close to its current position. Thereby, the prediction error will be small and requires fewer bits.

Delta Prediction. Often, the coordinates of two successive vertices are highly correlated. The difference between their positions is quite small. Therefore, Deering suggested to use the previously transmitted vertex position p_{i-1} to predict the position of the new vertex p_i and encoded the differences instead of the original coordinates. The first step in his algorithm is to normalize the mesh into an unit cube and quantize each vertex coordinate to 16 bits precision. Then, the difference between the previous and the current quantized positions is computed and further compressed with the entropy coding.

Spanning tree predictor. A more efficient predictor is the spanning tree predictor [114] of Taubin and Rossignac. As a preprocessing, Taubin and Rossignac quantize the original vertex locations to a user specified number of bits per coordinates (typically 8, 10 or 12 bits). Then they constructed a vertex spanning tree and exploited for the prediction of each vertex location p, the locations p_i of the vertices in the path from p to the root of the vertex spanning tree.

$$p = \sum_{i=1}^{K} \lambda_i p_i \tag{2.4}$$

The weights for the locations p_i are chosen to minimize the squared lengths of the delta vectors over all vertices $\sum \|\epsilon\|^2$. The delta vectors are computed and can be encoded with Huffman coding or arithmetic coding, resulting in about 13 bits per vertex at 8 bit quantization level (as reported in [117]).

Spanning tree predictor can be seen as generalization of delta prediction. This one uses only one ancestor i.e. $K = 1$ and $\lambda = 1$, while Spanning tree predictor uses weighted linear combination of K ancestors in the vertex spanning tree.

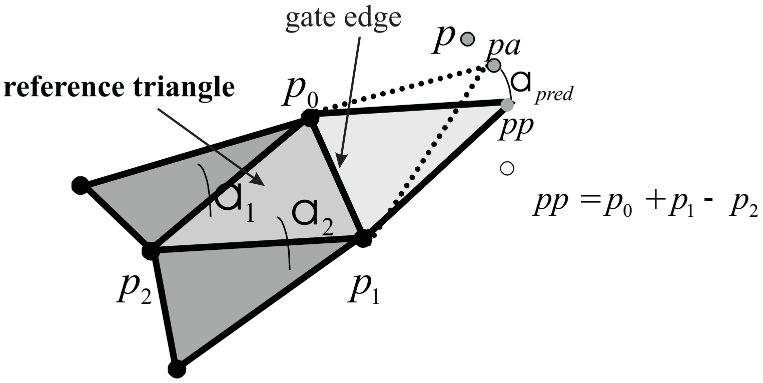


Figure 2.11 Parallelogram prediction

Parallelogram Prediction. More sophisticated prediction scheme is parallelogram prediction introduced by Touma and Gotsman [117]. Each newly encountered vertex v_{new} during the traversal of the connectivity constructs with two vertices (u, v) in the active list a triangle that is adjacent to the already encoded triangle (reference triangle) (u, v, w) of the growing region (see figure 2.11). Based on the assumption that two adjacent triangles tend to form a parallelogram, the position p_{new} of the vertex v_{new} can then be predicted as

$$p_{pred} = p_v + p_u - p_w \tag{2.5}$$

A good predictor is also the one that performs well for large class of meshes, meaning that prediction errors are very small. Parallelogram provides a good prediction if the shape is almost planar (meaning that the two triangles are (or near) co-planar), and gives a poor prediction if the shape has high curvature (meaning that the two triangles are highly non-planar). Therefore, for more accurate prediction, Touma and Gotsman combined this linear prediction with a curvature estimate. After the first prediction, the *crease angle* between the two adjacent triangles along the gate (u, v) is predicted as average of the available crease angles of two other edges incident to the reference triangle. At 8 bit quantization level, this approach requires approximately 9 bits per vertex.

Parallelogram predictor has extensively been used and extended up to now. It has also been adopted for the MPEG-4 standard for mesh geometry coding [2]

Vector quantization. Lee and Ko [73] applied vector quantization to geometry coding. They used a local coordinate frame defined upon the previously visited triangle. The resulting set of model-space vectors is encoded with a typical vector quantization scheme, achieving in average 6.7 bits per vertex for a quantization to 8 bits per coordinate, which is

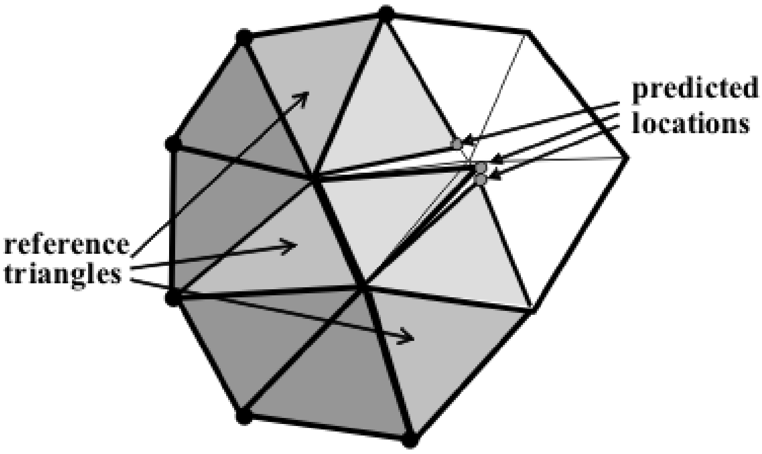


Figure 2.12 Multi-way prediction

seldom sufficient. Chou and Meng [22] proposed Predictive Vector Quantization scheme (PVQ), where the vertex position is first predicted then the prediction error is quantized by vector quantization. Very recently, Bayazit et al. also developed predictive vector quantization, where the prediction error is represented in the local coordinate system prior to vector quantization. They employ an extension to the entropy constrained of the predictive vector quantization, unlike the other VQ based approaches [73, 22] which aimed to design a code book vector with the minimum distortion without constraints on the rate.

Other techniques

Kronrod and Gotsman [70] suggested to encode the connectivity in such way that optimizes the parallelogram prediction rule. Unlike the previously described approaches that are connectivity-driven compression, this methods is geometry-driven approach. They constructed *prediction tree* where the mesh is traversed in some order to produce a good prediction and simultaneously encode the mesh connectivity. The entropy coding of the geometry is reduced by up to 50% on TG's algorithm [117]. However, its disadvantage is the complexity of the encoder.

As opposite to the parallelogram prediction which based on a single triangle in one direction, Cohen-Or et al. suggested *k-way predictor* (or *Multiway predictor*) that uses a multiple directions to predict the location of a vertex. In order to allow more than one way prediction, the mesh is traversed in a special order such that the vertex with highest degree of prediction is selected. Figure 2.12 shows an example of 3-way prediction that uses three triangles. Each triangles (already encoded) predicts a point, and the average is the final predicted position of a vertex. Typically, this yields smaller prediction error than the 1-way prediction.

Isenburg and Alliez [47] generalized the geometry coder of Touma and Gotsman to polygonal meshes. They let the polygonal connectivity dictate where to apply the parallelogram rule. As polygons tend to be flat in typical models, they tried to predict vertex locations within polygons rather than across polygons. Lee et al. [74] proposed to encode the dihedral angle along the gate, between the two neighboring faces and to transform the tangential coordinates into the interior angles of the spanning triangle. All angles are uniformity quantized then compressed with an arithmetic coder. This angle based scheme results in approximately 20% better compression ratio for geometry only than TG's scheme [117], at the same level of geometric distortion.

In predictive techniques, the coordinates are quantized to a number of bits typically lying between 10 and 14 with a negligible loss of accuracy. For low compression ratio, a coarse quantization level can be used, resulting in significant loss in data possible causing high visible degradation. Therefore these technique are not well-suited for very low bit-rate. Therefore, using *High-pass quantization* algorithm [112], the high-frequency distortion can be transformed into low-frequency which are almost invisible. The vertices coordinates are then first transformed into another space by applying the Laplander operator. Then the new transformed coordinates are quantized. This allows aggressive quantization without introducing visually disturbing artifacts.

There are several other geometry compression techniques such as [59, 109, 91, 49]. More deeply survey about static mesh compression, can be found in [94, 6].

Spectral Coding

Spectral methods are widely used for lossy compression of images such as the popular JPEG which is based on the DCT. In these methods, the data are expressed as linear combination of orthogonal basis functions, each basis function is weighted by a coefficient.

Karni and Gotsman [61] introduced the spectral theory method for geometry compression purpose. They computed the eigenvectors of the mesh Laplacian matrix then projected the mesh geometry onto the new orthonormal basis vectors. The spectral coefficients are quantized to typically between 10 and 16 bits followed by the quantization of the coefficient vector. The resulting integers are entropy coded with Huffman or arithmetic coder. For a mesh of large number of vertices, the computation of the eigenvectors of the Laplacian is prohibitively expensive. Therefore, it is more practical to decompose the mesh into submeshes and encode each mesh separately.

The results showed to be significant for relatively smooth models. The algorithm achieves about a half to a third of the bitrate of TG's algorithm [117] at the same visual quality.

This approach enables the progressive transmission. The spectral coefficients can be sorted then transmitted from low frequency to high frequency to the decoder. At the client side, an approximation of the mesh may be reconstructed using a few spectral coefficients, and it is incrementally refined by using more coefficients.

Geometry Image. It is a technique that remeshes an irregular triangle meshes onto regular grid which called *geometry image*. This can simply be encoded using traditional 2D image compression schemes. Geometry image representation is simple and well-suited for hardware rendering.

Basically, the mesh is first cut along a set of edge paths to produce a mesh that has the topology of a disk. This allows then to parameterize the cut surface onto the square domain of the image and sampled on 2D regular grid in this domain. This 2D grid forms a geometry image where the vertex coordinates (x,y,z) are encoded as an RGB image. The geometry images is compressed using wavelet-based coders.

2.4.1.2 Progressive Compression

Hoppe is the first who introduced the concept of progressive mesh representation (PM) [42]. This is another scheme for storing and transmitting arbitrary triangle meshes. As described before, progressive scheme encode a base mesh or a coarse version of the mesh with a sequence of refinement operations. After the base mesh has been sent, it is first reconstructed and gradually refined as the bitstream of refinement operation is being received, decoded and parallel rendered until the original mesh or its close approximation is recovered. In progressive scheme, the transmission or decompression of the mesh can be stopped at any accuracy.

Progressive techniques can also be categorized into lossy and lossless depending on that if the original connectivity and geometry information are recovered or not. In lossless techniques the mesh is continuously simplified into a coarse mesh, and at each step the simplification operation is stored. To retrieve the original mesh, the inverse of these operations are applied to the base mesh simplification. In lossy techniques often the remeshing and the wavelet are used. The mesh is decomposed into coarse mesh with hierarchy of fine detail. The distortion in lossy techniques is measured as the geometric distance between the surfaces.

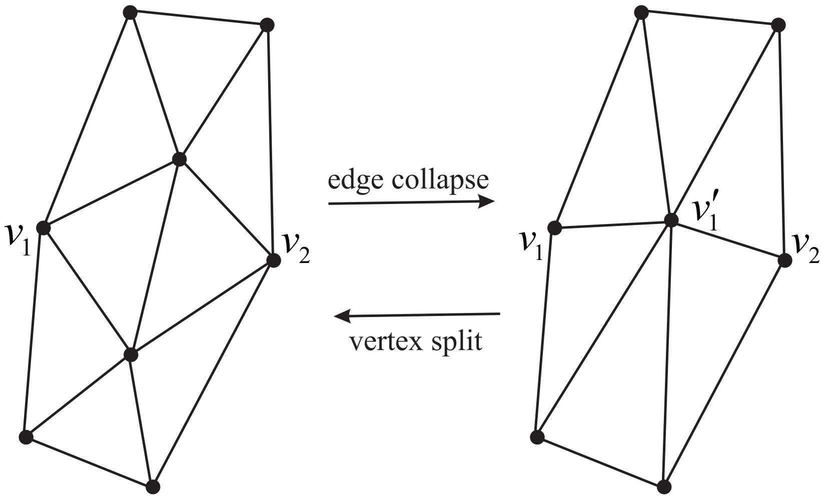


Figure 2.13 Edge collapse and vertex split operation.

In his progressive scheme [42], Hoppe showed that only one single simplification operation which is *edge collapse*, is sufficient to effectively simplify a mesh. This operation involves eliminating an edge by merging its two end vertices v_1 and v_2 into a single vertex v_1' (see figure 2.13. Thus, it reduces the mesh by one vertex and one triangle if the edge is on the boundary or two triangles otherwise. The inverse of edge collapse operation is called *vertex split*. It split the vertex v_1' back into v_1 and v_2, recovering the original connectivity.

To obtain the coarse mesh, a successive collapse operations $ecol_i$ are applied to the original mesh. At each iteration an edge that minimizes the distortion is selected to preserve the overall appearance of the mesh during the reconstruction. The progressive representation of the original mesh is then the base mesh and the sequence of vertex split operations: $(M_0, split_1, split_2, ..., split_k)$. To generate finely progressive coding, PM encodes the mesh by collapsing only one edge and encoding only one vertex, at time. This operation uses a large number of bits per vertex. Thus, it was not efficient for compression. To enhance the coding efficiency, PM was extended by several researchers.

Taubin et al. developed [115] compression approach. Instead of using single vertex split operation, they introduced the *forest split* operation which split a group of edges at the same time. At each step, the mesh is cut along a forest of edges, then the resulting crevice is triangulated and finally, the new vertices are displaced into their new position. This operation achieves much higher compression ratios than PM. Pajarola and Rossignac [89] proposed the *compressed progressive mesh*. To minimize the cost of each vertex split, they group the mesh edges into batches. At each batch the edges are collapsed, then the information to reverse these steps are encoded. At the batch refinement, the number of vertices is increased by up to 50%. Khodakovsky et al. [67] developed Progressive Geometry Compression for highly detailed and densely sampled meshes. First, the input mesh is remeshed into a semi-regular mesh using Maps algorithms. Then, a loop wavelet transform is applied on the semi-regular mesh outputting a coarsest base mesh and a sequence of wavelet coefficients which represent the difference between successive levels. The coarse base mesh is encoded with non-progressive technique while the sequence of wavelet coefficients is progressively encoded using zerotree whose concept can be found in [107].

Later, Khodakovsky and Guskov [66] developed another wavelet based approach based on the the normal mesh representation [40] which designed to produce detail coefficients with one scalar value only instead of 3D vectors. Furthermore, they used unlifted Butterfly wavelets as predictor. Alliez and Desbrun [5] proposed an new algorithm based on the valence of vertices in a mesh. They observed that the entropy encoding depend on the valence distribution. In their approach, the decimation conquest subdivides the mesh into patches (1-rings). Then, each patch center vertex is removed, it valence is output and the resulting patch hole is re-triangulated.

There exist a long series of improvement in the progressive compression and transmission of static meshes such as [43, 95, 31, 93, 78]. In this section, only few approaches are reviewed and for more detail we refer the reader to [6, 111].

2.4.2 Animated Mesh Compression

As animation technologies have become more sophisticated and accessible, their applications become more widespread. Application such as computer games, Movies, education, medicines, etc. often demand animated 3D models, and scenes with highly degree of realism. As animation becomes more realistic and more complex, the corresponding frame meshes become bigger and bigger, consuming more and more space. It is therefore indispensable to compress the animation datasets. *Key-frame* animation is one of the most famous and dominant animation representations used in the industry to represent the animation compactly. A set of key frames are chosen to describe certain important key poses in the animation sequence at certain times. Then all frames in between are generated using interpolation techniques. For such applications, even the number of key-frames can be very large, requiring a large memory space and need for effective compression techniques.

During more than one decade, extensive research has been done on static mesh compression, producing a large number of schemes as cited before. While research still focuses on efficient compression for huge static meshes [48], animated meshes have become more and more important and useful every where. However, the current static techniques for the compression of sequences of meshes independently are inefficient.

Often the meshes differ only slightly between neighboring frames, leading to a large redundancy between frames (temporal redundancy) and between neighboring vertices in the same frame (spatial redundancy). In order to develop a compact representations that significantly reduce the storage space of animated models, both the space and time coherence should be exploited.

The current coders are dedicated to compress the triangular meshes of fixed connectivity so that the connectivity needs to be encoded, stored or transmitted once, then the geometry coding comes into play. The focus on animated meshes with fixed connectivity may be justified by the fact that often the animation creators maintain the connectivity constant throughout the animation, in order to allow for easy and efficient manipulation of the sequence.

The previous section has shown how to compress static 3D meshes. This section will review the current compression algorithms for compression of animated meshes. Here, we distinguish between four approaches, according to the scheme adopted for compression: predictive based methods [JCS02, IR03], PCA based representations [AM00, KG04, SSK05], wavelet based techniques [GK04, PA05] and clustering-based approaches [Len99, ZO04]. These techniques can also be are classified into single-rate and progressive coders.

Lengyel provided a description of several *animation primitives* used to create the animated model: Free Form Deformations, Key-Shapes, Weighted Trajectories, and Skinning. The encoding of the primitive used in generating the animation yields the best compact representation of all. This is possible if the primitive is determined a priori. However, there are many animation tools. Finding the way in which the animation of each object is generated, is difficult or unfeasible. Therefore, it is more practical to develop compression tool that compresses animated object independently from how the animation is generated or even how complex (linear or non linear) it is. The efficiency of the method depend on how much the redundancy is removed, eventually, on the speed of the compression and decompression algorithm. Yet, we need compression algorithms that allow for small compressed representations that maintain good visual fidelity.

2.4.2.1 Clustering-based Compression

The basic idea of a clustering-based compression approach is to split the mesh vertices into several groups of similar motion and to encode the motion of each group using few representative vectors or parameters.

Affine Transformation. In his approach, Lengyel [77] partitioned the mesh into submeshes and described the motion of the submeshes by rigid body transformations. The rigid body transformation of a submesh was thereby estimated to best match the trajectories of its vertices. The compression was achieved by encoding the base submeshes, the parameters of the rigid body transformations, and the differences between the original and the estimated locations. This approach is very effective when large parts of an animated model can be described well by rigid body transformations. The other four modeling primitives can also replace the rigid transformation.

Lengyel's approach is improved [26], by encoding the mesh animation with a sequence of rigid transforms without residuals. A base mesh is segmented into rigidly transforming segments, using a new weighted least squares segmentation process. Then, to exploit the temporal coherence, these transformations are aggregated. The reconstructed animation may not be enough smooth in time and space. To rectify this effect, a spatial-temporal smoothing scheme is applied. Affine Transformation is also used to develop a Level of Details for Dynamic Meshes (see section 2.4.3).

Octree based Method. This approach is based on spatial clustering, based on an octree decomposition of the object. The basic idea is to represent all motion vectors of the vertices enclosed in each cell with only few representative motion vectors.

Assuming that the previous frame is already encoded, the motion vectors are computed as the differences between the position of vertices in the current and the previous frames $\Delta\nu$. Starting with a cube bounding box surrounding the 3D object as initial cell, eight representative motion vectors $m_1, ..., m_8$ are associated with the eight corners of the cell. The motion vectors of the enclosed vertices within the cell are then predicted by *tri-linear interpolation* in the form of weighted sum of the eight representative vectors.

$$\Delta\nu = \sum_{i=1}^{8} w_i m_i \tag{2.6}$$

The representative vectors for a given cell are computed using least square estimation. If the motion of the enclosed vertices is well approximated, meaning that the error is below a specified threshold, the representatives for the octant are quantized and entropy encoded. If the error exceeds the specified threshold, the cell is then refined into 8-octants and for each new octant 8-motion vectors are computed to estimate the motion of its enclosed vertices. The process continues until the error is below the threshold. The accuracy of the approximation can be measured in terms of maximum or average Euclidean distances between the original and the reconstructed motion vectors of the vertices within a cell. The compression is achieved by quantizing and entropy encoding the representative motion vectors.

This approach is improved to the so called Differential 3D Mesh Coder (D3DMC) [63], which is followed by a rate-distortion optimized version (D3DMC-RD) [87].

Iterative Closest Point. The idea of ICP is to find point matches between two meshes. In [38], the ICP is used to estimate the motion of the vertices in each cluster in terms of affine parameters and residuals. The mesh is initially segmented into clusters by the topological partitioning algorithm [41]. Then the clustering is refined to separate the vertices that can be encoded with the affine transforms only and those that need further encoding the residuals.

2.4.2.2 Vertex-Prediction based Compression

Prediction techniques assume that the connectivity of the meshes does not change over time and use the previously recovered vertex locations to predict the location of each new vertex. The vertex path (or vertex displacements) may be estimated using linear or non-linear predictors in space and (or) in time. The delta vectors are compressed up to a user-defined error. In such technique, the mesh triangles are traversed in an order suitable for the predictor and the first frame is encoded using static compression schemes. Here, we distinguish three type of predictors: spatial predictor, temporal predictor and space-time Predictor.

Spatial predictor exploits the coherence between neighboring vertices in each frame separately using for example the parallelogram predictor (see figure 2.14).

Temporal Predictor exploits the redundancy between the positions of the vertex in subsequence frames. The position of the vertex $\mathbf{p}^f$ is simply predicted from the position of the vertex $\mathbf{p}_i^{f-1}$ in the previous frame $f-1$ or as a linear or quadratic combination, taking into consideration the vertex positions in the two or three previous frames ($\mathbf{p}^{f-1}, \mathbf{p}^{f-2}, \mathbf{p}^{f-3}$).

Spatio-Temporal Predictor takes into account both the spatial and temporal correlation. For instance dynapack uses two space-time predictors:

Extended Lorenzo Predictor (ELP)
This predictor uses a parallelogram prediction to exploit spatial coherence, and then performs temporal prediction on the spatial details. It is a prefect predictor for a subset of the

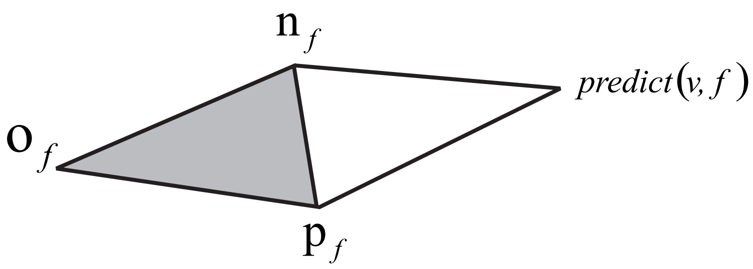


Figure 2.14 Space-only predictor: parallelogram prediction.

mesh undergoing pure translation from the previous frame.

$$predict(\mathbf{v}, f) \quad = \quad \mathbf{n}_f + \mathbf{p}_f - \mathbf{o}_f + \mathbf{v}_{f-1} - \mathbf{n}_{f-1} - \mathbf{p}_{f-1} + \mathbf{o}_{f-1}$$

Replica Predictor

This predictor expresses the location of the vertex relative to the locations of three vertices of an adjacent triangle as local coordinate system. The vertex location in the new frame is estimated by its relative coordinates from the previous frame. This predictor replicates perfectly the local geometry undergoing any combinations of translations, rotations, and uniform scaling (see figure 2.15).

$$predict(\mathbf{v}, f) = \mathbf{o}_f + aA' + bB' + cC' \tag{2.7}$$

$$\begin{aligned} A' &= \mathbf{p}_f - \mathbf{o}_f \\ B' &= \mathbf{n}_f - \mathbf{o}_f \\ C' &= \frac{A' \times B'}{\sqrt{\| A' \times B' \|^3}} \end{aligned} \tag{2.8}$$

The position of the $\mathbf{v}_{f-1}$ can be written as : $\mathbf{p}_{f-1} + a\mathbf{A} + b\mathbf{B} + c\mathbf{C}$. Then, the coefficients a, b and c are computer from the previous frame as.

$$\begin{aligned} a &= \frac{A.D * B.B - B.D * A.B}{A.A * B.B - A.B * A.B} \\ b &= \frac{A.D * A.B - B.D * A.A}{A.B * A.B - B.B * A.A} \\ c &= D.\frac{A \times B}{\|A \times B\|} \end{aligned} \tag{2.9}$$

where

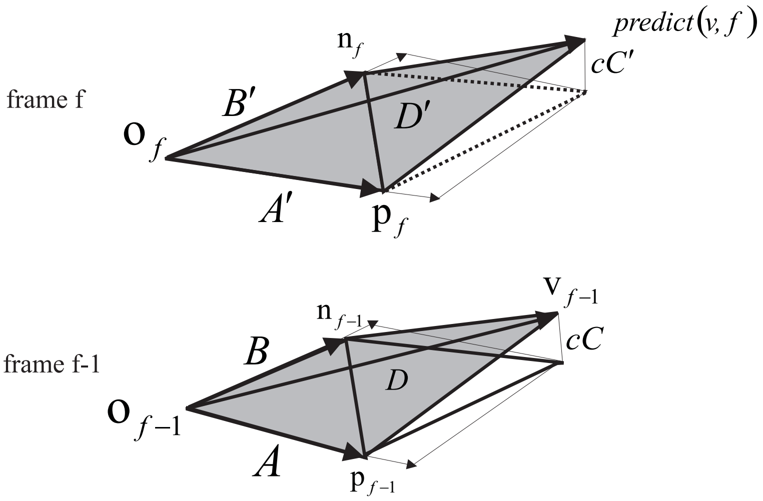


Figure 2.15 Replica predictor.

$$\begin{aligned} A = &\quad \mathbf{p}_{f-1} - \mathbf{o}_{f-1} \\ B = &\quad \mathbf{n}_{f-1} - \mathbf{o}_{f-1} \\ C = &\quad \frac{A \times B}{\sqrt{\| A \times B \|^3}} \end{aligned} \tag{2.10}$$

A similar predictor, introduced by Stefanoski and Ostermann [113], is a perfect predictor that preserves the angle between the reference triangle and the spanning triangle. The difference between this predictor and the Replica Predictor lies in the coordinate system used to express the vertex location. Yang et al. proposed, before, first and second order predictors for the vertices displacements within some Lagrange Multiplier scheme.

The interpolation compression (AFX-IC) [55] is a tool adopted by the MPEG-4 standard and aims at exploiting the redundancies between the the set of key frames. The main concept is to reduce the number of keys trough an interpolator process and compress the remaining key frames using the differential coding and entropy coding.

Predictive methods are simple, efficient and have a low computational cost which makes them well suited for real time compression and decompression. The drawback is that they do not support progressive compression and that they are not efficient at very low bit rates which would require a coarse level quantization, which leads to blocky structure as discussed before.

Vertex based predictive approaches, focus on frame to frame changes to exploit local coherence. They process vertex after vertex. In opposite, There are other approaches that analyze the global coherence by using, for example, a PCA. This examines the entire mesh sequence and represents the sequence by a few principal components and coefficients.

2.4.2.3 PCA-based Compression

Principal Component Analysis (PCA) is a statistical technique that can reduce the dimensionality of a dataset. It determines linear combinations of the original dataset which contain maximal variation and represents them in an orthogonal basis. The original data can then be compactly represented by a few principal components and coefficients. If we have for example, F frames of $3V$ dimension each and V is the number of the vertices, PCA produces a reduced number $L \ll F$ of principal components that represent the original dataset. On can then see the PCA as a transformation of the original coordinates to a new coordinate system such that the direction of the first axis is pointed in the direction of the greatest variance in the datasets, the direction of the second axis in the direction of the second greatest variance and so on.

Alexa and Müller [3] were the first who suggested PCA to achieve a compact representation of animation sequences. First, all frames are translated so that the center of mass of the model coincides with the origin. Then, an affine transformation is computed minimizing the squared distance of corresponding vertices with respect to the first frame. All frames are gathered to form a matrix of dimension three times the number of vertices times the number of frames. To find out a compact representation for the sequences, they perform PCA on that Matrix using singular value decomposition (SVD) to extract the eigenvectors and the coefficients. This method was improved [62] by applying second-order Linear Prediction Coding (LPC) to the PCA coefficients such that the large temporal coherence present in the sequence is further exploited.

Animated meshes exhibit highly nonlinear behavior, which is globally difficult to capture using standard PCA. Therefore, for more efficient compression, the clustered PCA (CPCA) was introduced [102]: the mesh is segmented into meaningful clusters using Lloyd's algorithm [82] in combination with principal component analysis. These clusters are then compressed independently using a few PCA components only. This technique outperforms the standard PCA and the combined PCA with LPC scheme, since they explore a robust segmentation which is based on a data analysis technique. But it remains expensive.

PCA based approaches support progressive transmission and level-of-detail. The quality of the animation increases with the number of PCA coefficients. From the computational viewpoint it is expensive, but the decompression process is very fast.

2.4.2.4 Wavelet-based Compression

Wavelet transform aims to decorrelate geometric data and to generate a sequence of detail coefficients. In irregular meshes, the wavelet detail is computed every time a vertex is removed from the current level of progressive mesh hierarchy and defined as the difference between the actual location of the removed vertex and the predicted location from the coarse level.

Guskov et al. [39] used wavelets for a multiresolution analysis and exploited the parametric coherence in animated sequences. The resulting wavelet detail coefficients were progressively encoded with predictive coding scheme.

The idea of this algorithm is to separate the geometric and the parametric information for parametrically coherent mesh sequences. The sequence of meshes sharing the same connectivity, also share a similar local parametrization. Hence, the connectivity and the parametrization need to be encoded and transmitted only once then use it to decorrelate the geometry of each frame. Indeed, the encoder first process a specific mesh which is called parametric mesh, then uses the information of this mesh to compress the geometry of the remaining meshes, frame by frame.

Payan et al. [92] introduced the *lifting scheme* to exploit the temporal coherence. The wavelet coefficients are thereby optimally quantized by minimizing the reconstructed mean square error for specific user-given target bitrate.

Video Geometry [19] is an alternative way that treats the animated meshes as video sequence. It is based on the *Geometry Image* representation developed for static mesh. The sequence of meshes are then transformed into geometry images which are then compressed using standard video compression.

2.4.3 Level of Details for Dynamic Meshes

Shamir et al. [106] introduced a multiresolution model for dynamic geometry sequence of meshes called TDAG. It supports spatial and temporal level of detail. They first extract the affine transformation relative to the first frame then encode the residual in the TDAG structure. Through the TDAG structure, the construction approximation of each mesh is governed by a metric function that combines spatial constrains and temporal constraints.

2.5 Segmentation

3D Mesh segmentation is a process that breaks mesh elements having the same properties into regions. It has become a necessary tool in computer graphics and geometric modelling. And it is used for various applications such as metamorphosis [33, 131, 110], compression [61, 84, 10, 102], parametrization [100, 69] simplification [132, 32] and skeleton extraction [54]. For detailed survey in literature on segmentation, we refer the reader to [105].

2.5.1 Static Mesh Segmentation

Segmentation of static mesh partitions the mesh into regions using the mesh attributes as segmentation criteria, depending on the applications. This may impose different requirements and criteria. The most criteria functions used for partitioning process are geodesic distance [65, 64], curvature [104, 34, 72], normal [125], etc. Of course, the decision to assign each vertex or triangle to the same segment affects heavily on the results of the segmentation. Thus, the quality of the method is strongly related to its application.

A various approaches have been proposed to decompose the static meshes into visually meaningful parts such as region growing [72, 119, 90, 57, 127, 86, 20], watershed [88, 132], hierarchical clustering [32, 16, 46, 108, 71], iterative clustering [110, 101, 58, 124], spectral analysis methods [61, 80, 81, 128, 129], implicit methods, etc.

For instance, the **region growing** approach collects the elements of similar feature. These elements can be vertices, triangles or region. The strategy of region growing starts with a number of elements which are either selected randomly or using geometrical criteria. Then, it grows sub-meshes incrementally (eventually, in parallel) under a set of criteria that determines if the new element can be added to the current region and the growing stop criteria. The main differences between various methods arises typically on growing criteria, seeds selection process and dealing with the small regions or merging criteria. In this approach the result of segmentation depends heavily on the number of the choice of initial seeds and its number.

The **clustering** approaches can be either hierarchical or iterative. **Hierarchical clustering** initializes all elements as clusters, then merges the clusters of low cost to one cluster. The number of final clusters in this approach is unknown. **Iterative clustering** is introduced to find an optimal segmentation. Often, this category uses the popular Lloyd's algorithm [82], some time referred to as k-means.

Given a number of clusters, the approach searches iteratively the best segmentation. The algorithm starts with k representatives of k clusters and assigns each element to one of these clusters. Then, the iterative procedure updates the k representatives from the clusters until they stop changing. Generally, the iterative process can be automatically stopped when stopping criterion is met, typically, the estimated improvement stays below a predefined threshold value or the maximum of iterations is reached.

The **Spectral analysis** approach uses generally the eigenvectors of the affinity matrix to break the mesh into parts. Spectral clustering is one of these well-known spectral approaches. It uses the eigenvectors of weighted graph Laplacian matrix to construct a low dimensional embedding in which the clustering would be easier than clustering the original data points.

2.5.2 Dynamic Mesh Segmentation

Recently, segmenting dynamic 3D meshes gained much interest and it is used in several contexts in animation [37, 102, 54, 64, 75, 76], typically, skinning mesh animation [54], ray tracing[37], and compression [77, 38, 102, 84].

While static mesh segmentation aims at detecting meaningful parts and breaks the mesh onto sub-meshes of similar specific features within a specific context, 3D dynamic mesh segmentation approaches exploit the temporal information to partition the mesh into quasi-rigid parts. In other words, group the vertices that undergo similar motion. Beside, the spatial information or features can also be exploited.

The literature on 3D dynamic mesh partitioning is poor compared with the large literature on static mesh partitioning. For instance, in skinning deformable mesh animation, James and Twigg [54] used **mean shift** algorithm [21, 27] to cluster triangles with simple rotation sequences to identify the near-rigid structures of deformable meshes and estimates their transformations. This kind of clustering performs well for virtual characters but not for extremely deformed animation where the most component are not near-rigid.

To ray trace animations, Günter e al. [37] decomposed the mesh into rigid parts underlying a similar transformation by clustering the triangles. Residual motion is then captured in a single **fuzzy kd-tree** for the entire animation. Note that kd-tree subdivides a region enclosed by a bounding box into irregular areas.

Lee et al. [76] introduced a method that find near-rigid sub-meshes. The algorithm initially extracts feature triangles on the mesh. Then, the remaining triangles are assigned to different partitions, depending on the distance between the face and the feature faces. The distance metric which is used combines geodesic distance with the deformation distance. The drawback of this algorithm is computationally costly for practical purpose.

For dynamic mesh compression purpose, segmentation is rarely used compared with the huge number of algorithms proposed for static meshes. The most proposed approaches assumes that the sequence of meshes have the same connectivity and only geometry change over time.

Usually, the objective of using partitioning the mesh in the compression context to decrease the computational costs as well as to preserve the global shape of the mesh because some compression algorithms such as some spectral based techniques (DCT, PCA, etc.) can destroy important features of the mesh. Another objective is to find near-rigid components whose motion can be described by affine transforms which lead to more compact representation. In all cases, the algorithm employed to partition mesh vertices into near-or rigid parts heavily affects compression performance.

To find the vertices that have similar motion, Lengyel [77] proposed that one select a set of seed triangles randomly and compared their trajectories. Triangles with a similar motion are combined. Then the vertices are associated with the triangle whose trajectory best fit theirs. Gupta et al. [38] use multilevel ***k-way*** partitioning algorithm [41] on the basis of proximity in the connectivity and the number of parts given by the user.

Mamou et al. [85] proposed to partition the mesh vertices into clusters whose motion can be described by unique 3D affine transforms, by applying k-means [60]. Amjoun et al. [7] proposed region growing based approach. Starting from several seed points. The regions grow uniformly around the set of selected seed points by first traversing the closest neighboring vertices over time until all vertices of the mesh are visited (see chapter 4).

These approaches use the connectivity information for segmentation, eventually with the geometry information. Generally, the approach that is connectivity-based clustering only is not well-suited for time geometry-variant.

Alternative approaches use the geometry information only, and they are characterized as motion clustering based techniques [102, 10] which analyze the vertex motion over time.

Sattler et al. [102] proposed to cluster the trajectories of vertices using Lloyd's algorithm in combination with PCA. Then, they compress each cluster independently. In this approach, unlike [54], they don't analyze the motion of each triangle but the motion each vertex. However, the experiment results showed that the cutting boundary may deviate from the deformable regions. Thus, the near-rigid is not always obtained as cited in [76].

Amjoun and Strasser [10] proposed another scheme based on the motion of vertices relative to the local coordinate system defined for each cluster.(see more details in chapter 4).

As mentioned in static case, in animation the segmentation process also depends on the applications which may impose different requirements, and the quality of the results depends strongly on the applications.

Note that the static mesh segmentation can also be well used in animation case as an initial segmentation which is then refined based on the motion information.

2.5.3 Discussion and Summary

This chapter began with an overview of compression data. Two categories of compression were described: lossless and lossy compression. General compression techniques were presented. Two entropy techniques are described: Huffman coders and arithmetic coders. These techniques are lossless techniques often used in mesh compression. A description of 3D geometric models for the static and animated case are presented. Triangle meshes are a frequently used representation to model three-dimensional object or scene. Then the most popular techniques of mesh compression have been reviewed. These techniques can be classified into into *single-rate encoders* and *progressive encoders*. Single-rate encoders encode the original mesh as whole and once. They are dedicated to reduce bandwidth between rendering pipeline and local memory.

A wealth of successful compression schemes have been proposed. Earlier works focused on the efficient coding of the connectivity data driving the encoding of the geometry data, i.e. a vertex location is encoded at the time, when the connectivity-coding scheme encounters the vertex for the first time. As connectivity coding techniques became near optimal within the last years, the researchers started to concentrate on the encoding of the geometry whose code dominates the total compressed data size. The geometry driven coding have then been emerged to encode the geometry in near optimal way, independently of connectivity coding. This is then guided by the geometry coding.

Progressive schemes have also been widely studied. Progressive approach first creates a simplified version of the original full resolution mesh using simplifications operations. After transmission, the base mesh is first decoded and reconstructed, then gradually refined as the bitstream is being received and decoded. In geometry driven compression, the geometry is progressively encoded without restraint of connectivity. Then the connectivity changes between two levels of details are encoded.

While research still focuses on efficient compression for huge static meshes, animated meshes have become more and more important and useful every where. However, the current techniques for the compression of sequences of meshes independently are inefficient.

The current coders are dedicated to compress the animated triangular meshes of fixed connectivity so that the connectivity needs to be encoded, stored or transmitted once, then the geometry coding comes into play.

There are several criteria by which developed coding techniques can be distinguished. One of these criteria is the methods used to encode dynamic geometry. In PCA based techniques, the global linear behavior of the vertices through all frames is approximated in terms of linear space. The animation sequence can be reduced to a few principal components and coefficients. The efficiency of this technique increases when the datasets are segmented or clustered, so that each group is individually encoded by PCA. This type of method supports progressive transmission. The drawback of this approach is it is computationally expensive. In predictive methods, for each frame, the difference between the predicted and the current locations is encoded with very few bits. These approaches are simple, not expensive and well-suited for real-time applications. The drawback of these methods is that they do not support progressive transmission. Affine transformations well approximate the behavior of sets of vertices relative to the initial position (the first frame, eventually the I-frame). This type of method is very effective for animations based on motion capturing, if the mesh is well partitioned into almost rigid parts, since the vertices are attached to the bones and move according to their representative joints. Therefore, exploiting the coherence in this animation and finding the transformation that best matches each group of vertices is easier than finding a transformation that approximates each part in deformed meshes (like a cow animation). The drawback of this technique is that it can be computationally expensive depending on the splitting process or the affine transformation optimization.

Wavelet based approaches which also support the progressive transmission, have showed to be efficient. There is a method that separates the parametrization and the geometry. It is assumed that the connectivity and the parametrization component are constant throughout time. Thus, they encoded and transmitted only once and then use it use the parametric information to de-correlate the geometry of each frame. Lifting scheme exploit the temporal coherence by transforming the vertex position into high and low frequencies. The wavelet coefficients are thereby optimally quantized. Geometry images is also extended to geometry video which are compressed using standard video compression

We end up the chapter with mesh segmentation. We discussed different methods proposed for static and dynamic meshes in different application. Particulary, we presented the most proposed approaches in the 3D dynamic mesh compression context. The goal of almost all methods is to partition the mesh into near or rigid parts, taking advantage of the motion correlation property. Many of them focus on articulated characters and may be less efficient for deforming meshes.

Basically, There are methods which we call geometry-based segmentation techniques. They use the geometry information only to cluster the mesh vertices, independent of the mesh connectivity. The connectivity-based segmentation techniques use the vertex adjacency to partition the mesh. These approaches are not well-suited for partitioning, since the geometry is changing over a time. There are methods that use both geometry and the connectivity information for partitioning, often they assume that the connectivity is constant over time and only geometry changes over a time.

CHAPTER 3

Compression of Static Meshes: Higher Order Predictor

This chapter describes a new geometry encoding strategy that follows the predictive coding paradigm, which is based on a region growing encoding order. Only the delta vectors between original and predicted locations are encoded in a local coordinate system, which splits into two tangential and one normal component.

We introduce so-called higher order prediction for an improved encoding of the normal component. The tangential components are encoded with parallelogram prediction. Then, a higher order surface is fit to the so far encoded geometry. As the normal component is encoded as a bending angle, it is found by intersecting the higher order surface with the circle defined by the tangential components. Because of computational time of gathering vertices during fitting process, we come up with higher order predictor based on sphere fitting to speed up higher order prediction.

3.1 Introduction

There are two main criteria by which static coding techniques can be distinguished. The first criterion is whether the method is *progressive*, i.e. allows for incremental transmission, or not. In the latter case the scheme is called *flat*. Progressive methods tend to be a bit less efficient as flat methods and in this paper we propose a flat compression scheme. The second criterion is whether the geometry is encoded *lossy* or *“not lossy”*. In the lossy setting it is allowed to move vertices over the surface as long as the L_2-norm or the Hausdorff distance between encoded and decoded mesh is less than a prescribed limit. It is even allowed to change the connectivity. Typically, tools like *Metro* [24] are used to measure the distance between original and decoded mesh. The proposed method is "not lossy", where we quote this notion as it also introduces some loss. In the "not lossy" setting the coordinates of the original vertex locations are quantized to a user specified number of bits q relative to the maximum extend $e_{\max}$ of the bounding box of the model, or a bit more general: the error between the original vertex location $\mathbf{p}_{\rm i}$ and the decoded vertex location $\tilde{\mathbf{p}}_i$ may not exceed

$$\|\mathbf{p}_{\rm i} - \tilde{\mathbf{p}}_i\| \leq e_{\max}/2^q. \tag{3.1}$$

This is the setting that we used in this chapter.

As most other proposed methods for geometry coding we also follow the predictive coding paradigm. Here the triangular mesh is traversed in a region growing order, which is either driven by the connectivity coding algorithm or can be chosen, as we do, as a breadth-first traversal of the connectivity after the connectivity has been decoded. One initializes the growing region, which contains the so far encoded geometry, with one triangle and encodes the incident vertex locations in uncompressed form. The three edges of the initial triangle are pushed onto a FiFo. The traversal loop pops the currently first edge from the FiFo and defines it as the so called *gate*, at which the region grows. The gate is incident to at least one triangle in the growing region. The other incident triangle is added to the growing region if it is not already part of it and new potential gate edges are pushed onto the FiFo. Every time a new vertex is encountered during the traversal, one predicts its location from the so far encoded geometry and only encodes the delta vectors

$$\delta_i \overset{def}{=} \mathbf{p}_{\rm i} - \mathbf{p}_{\mathrm{pred},i}$$

between the original vertex location $\mathbf{p}_{\rm i}$ and the predicted location $\mathbf{p}_{\mathrm{pred},i}$. The traversal loop is iterated until the FiFo is empty and all vertices have been encoded. The decoding algorithm just performs the same traversal and does the same prediction, but decodes the delta vector and reconstructs the original vertex location.

Previous work [73] has shown that it is advantageous to split the coding in tangential and normal components by expressing the delta vectors in a coordinate system aligned with the so far encoded geometry. We follow this approach as described in more detail in section 3.2.2. We also split the prediction into a tangential and a normal prediction. For the *tangential prediction* of the two tangential components $\mathbf{p}_{\mathrm{pred,tan}}$, we investigated the two existing methods of parallelogram prediction and multi-way prediction. But our

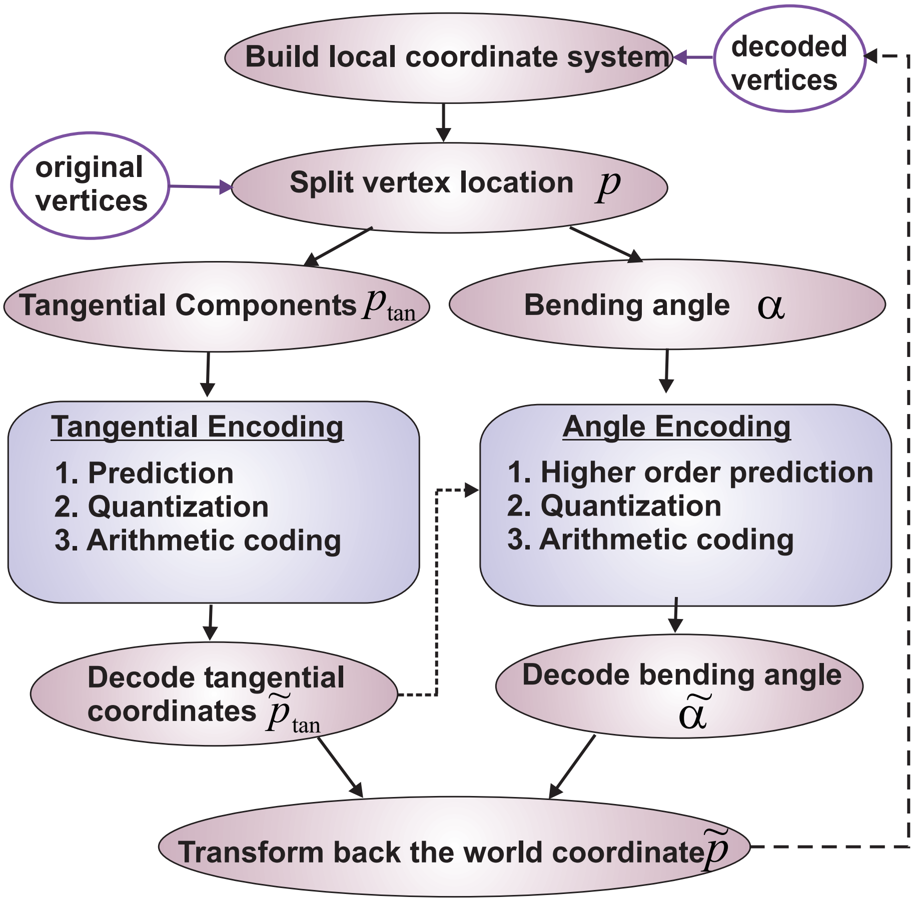

Build local coordinate system
decoded vertices
original vertices
Split vertex location p
Tangential Components $p_{\tan}$
Bending angle α
Tangential Encoding
1. Prediction
2. Quantization
3. Arithmetic coding
Angle Encoding
1. Higher order prediction
2. Quantization
3. Arithmetic coding
Decode tangential coordinates $\tilde{p}_{\tan}$
Decode bending angle $\tilde{\alpha}$
Transform back the world coordinate $\tilde{p}$

Figure 3.1 Geometry coding process.

main contribution is the so called *higher order prediction* for the prediction of the normal component. The normal component is encoded as the bending angle between the triangles incident to the gate. After the encoding of the tangential components we fit a higher order surface to the so far encoded geometry and intersecting it with a circle defined by the tangential components.

3.2 Geometry Encoding and Decoding Algorithm

Our geometry-coding scheme is based on a breadth-first region growing traversal of the mesh as described before. Here, we detail the coding algorithm for the vertex locations of vertices that are newly encountered during the traversal of the connectivity.

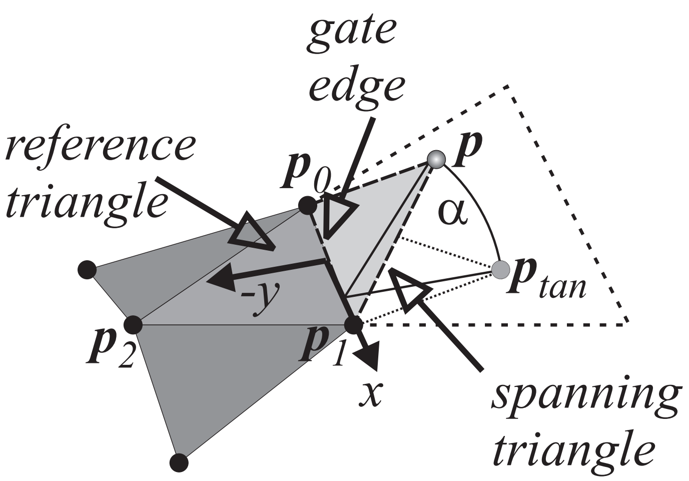
The block diagram of the encoder is shown in figure 3.1. We decompose the geometry encoding into the following steps.

Geometry Encoding Algorithm

1. build local coordinate system
2. transform original vertex location into tangential coordinates and bending angle
3. tangential prediction (parallelogram or multi-way)
4. compute, quantize and encode tangential delta vector
5. decode tangential coordinates
6. higher order prediction of bending angle with decoded tangential coordinates
7. compute, quantize and encode bending delta angle
8. decode bending angle
9. replace original vertex location with decoded location transformed back to the world coordinate system

Firstly, we compute the local coordinate system by splitting the coordinates into two tangential components and a normal component represented by a bending angle (see subsection 3.2.2). In the second step, we transform the original vertex into the local coordinates. Next we predict the two tangential components as described in section 3.2.3 and compute, quantize and encode the delta vector with an arithmetic coder as will be described in section 3.2.4. Steps 5 and 8 are crucial for the avoidance of error accumulation. By simulating the decoding process, we make sure that we use during encoding exactly the same information also available to the decoding algorithm. In step 6 the main contribution of the paper comes into play, when we predict the bending angle. This is detailed in section 3.3. Again we compute the delta angle, i.e. the difference to the bending angle measured from the original point, quantize and encode the bending delta angle. Finally, we decode the bending angle also known by the decoder, transform the local decoded coordinates back to the world system and replace the original vertex location with the decoded one, what ensures avoidance of error accumulation.

The decoding algorithm uses the same traversal of the connectivity and performs the following steps:



$$\begin{aligned} x &= (\mathbf{p}-\mathbf{o})\cdot\hat{\mathbf{x}} \\ t &= (\mathbf{p}-\mathbf{o})\cdot\hat{\mathbf{y}} \\ z &= (\mathbf{p}-\mathbf{o})\cdot\hat{\mathbf{z}} \\ y &= \sqrt{t^2+z^2} \\ \alpha &= \mathrm{atan2}(z,t) \end{aligned}$$

Figure 3.2 Illustration and computation of local coordinates: tangential coordinates x, y and normal component given as bending angle α.

Geometry Decoding Algorithm

1. build local coordinate system
2. tangential prediction (parallelogram or multi-way)
3. decode and undo quantization of tangential delta vector
4. compute tangential components
5. higher order prediction of bending angle
6. decode and undo quantization of bending angle
7. compute bending angle
8. transform local coordinates back to world coordinates

3.2.1 Avoidance of Error Accumulation

As we use predictive coding and do the quantization in a local coordinate system, the quantization errors normally accumulate. Lee and Ko [73] had the same problem with their vector quantization strategy and proposed to encode additional correction vectors every time the accumulated quantization error exceeded the error tolerance. As we only encode correction vectors, a careful design of the encoding algorithm is necessary to avoid error accumulation. We simply simulate the decoding process also during the encoding, and store the decoded vertex locations of each vertex. From these we compute the local coordinate system (step 1 of encoding algorithm) and the prediction (step 3). The deltas of the tangential components (step 4) and of the bending angle (step 7) are computed between the original vertex locations and the prediction computed from the decoded vertex locations, such that only one quantization step can introduce error and error accumulation is avoided.

3.2.2 Local Coordinate System

Figure 3.2 illustrates the local coordinate system, that we use in our geometry coding algorithm. When a new vertex is encountered the gate edge is always incident to an already encoded triangle in the growing region, which we call the *reference triangle*, and to a triangle incident to the new vertex, which is called the *spanning triangle*. The local coordinate system is defined on the reference triangle with origin $\mathbf{o}$ at the center of the gate, x-axis along the gate edge and y-axis orthogonal to x-axis in the plane of the reference triangle. As third coordinate we use the bending angle α between the normals of reference and spanning triangle resulting in a cylindrical coordinate system with y as radius. We kept the notation x and y because they refer to tangential components. To compute the local coordinates (see Figure 3.2) we also determine the $\hat{\mathbf{z}}$-axis orthogonal to $\hat{\mathbf{x}}$ and $\hat{\mathbf{y}}$. The transformation back to world coordinates is simply

$$\mathbf{p} = x\hat{\mathbf{x}} + y\cos(\alpha)\hat{\mathbf{y}} + y\sin(\alpha)\hat{\mathbf{z}} + \mathbf{o}.$$

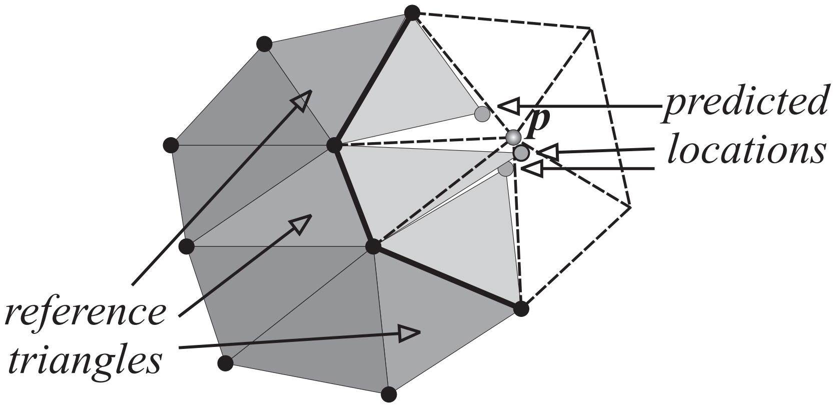


Figure 3.3 In the multi-way prediction mode all possible reference triangles are exploited for parallelogram predictions.

3.2.3 Tangential Prediction

In parallelogram prediction mode for the tangential components we use the formula of Touma and Gotsman but use the decoded vertex locations in order to avoid error accumulation. In the multi-way prediction mode we exploit the ideas proposed by Cohen-Or et al. [25]: when a new vertex is encountered there can be more than one possible reference triangle as illustrated in Figure 3.3. The idea is to use all possible reference triangles for parallelogram predictions and average the resulting predicted locations in world coordinates. This is exactly what we do before we transform the averaged predicted location to the local coordinate system of the actually selected gate edge.

3.2.4 Binary Coding of Coordinates

For the final encoding of the local coordinates we have to quantize the coordinates according to the error bounds as given by equation 3.1. For the tangential components this quantization step is straightforward. Integer values are derived according to

$$\begin{aligned} i_x &\overset{def}{=} \lfloor x/e_{\max} \cdot 2^q + 1/2 \rfloor \\ i_y &\overset{def}{=} \lfloor y/e_{\max} \cdot 2^q + 1/2 \rfloor \end{aligned}$$

For the angular component, one has to account for the radius given by the y coordinate of the cylindrical coordinate system. Computing the arc length yields

$$i_\alpha \overset{def}{=} \lfloor y\alpha/e_{\max} \cdot 2^q + 1/2 \rfloor$$

The inversion of the quantization process is simple as our algorithm makes sure that y is known before α needs to be decoded.

The resulting signed integer values are encoded with an adaptive arithmetic coder [123], which generates new symbols for every newly encountered index. We use two different encoding contexts one for x and y and one for α.

3.3 Higher Order Prediction

In this section we describe our new approach for the prediction of the normal direction, i.e. the bending angle, of the local coordinate system. Figure 3.4 shows an example. The red arrow shows the gate edge and illustrates the local x-direction. The green arrow is the local y-direction and the blue one the virtual z-direction. The vertices illustrated by blue spheres are the already encoded vertices close to the gate. These vertices are used to fit a

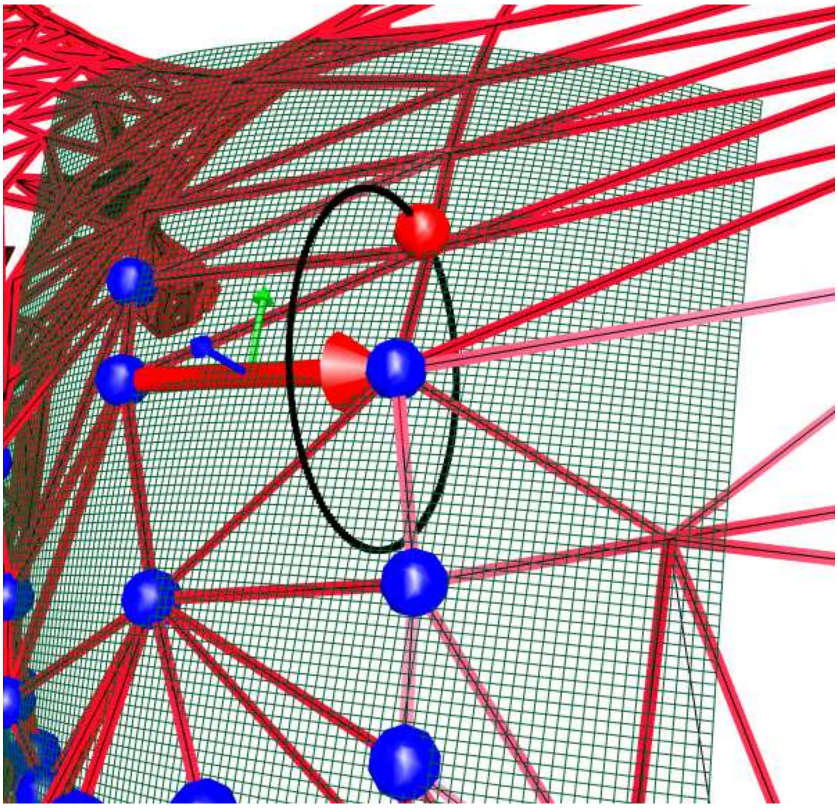


Figure 3.4 Snapshot during the higher order prediction of the cow model (polynomial graph function).

higher order surface as shown in cyan and are therefore called *fit vertices*. The dark circle around the gate illustrates all points corresponding to the decoded tangential components x and radius y and is called the *tangential circle*. The intersection of the circle with the higher order surface defines two or more possible prediction angles α. The angle closest to $\alpha = 0$ is chosen and the resulting predicted locations is illustrated by the red ball in Figure 3.4. In the following three subsections we detail the gathering of fit vertices, the fitting of higher order surfaces and the intersection of the higher order surface with the tangential circle.

3.3.1 Gathering of Fit Vertices

As shown in figure 3.5, the fit vertices were gathered in a region growing strategy starting at the reference triangle. In order to ensure that the decoder could collect the same vertex locations, we collected the decoded locations of already encoded vertices only.

As it does not make sense to fit a smooth surface in the presence of sharp edges, we additionally restricted the search for fit vertices to a flat region around the reference trian-

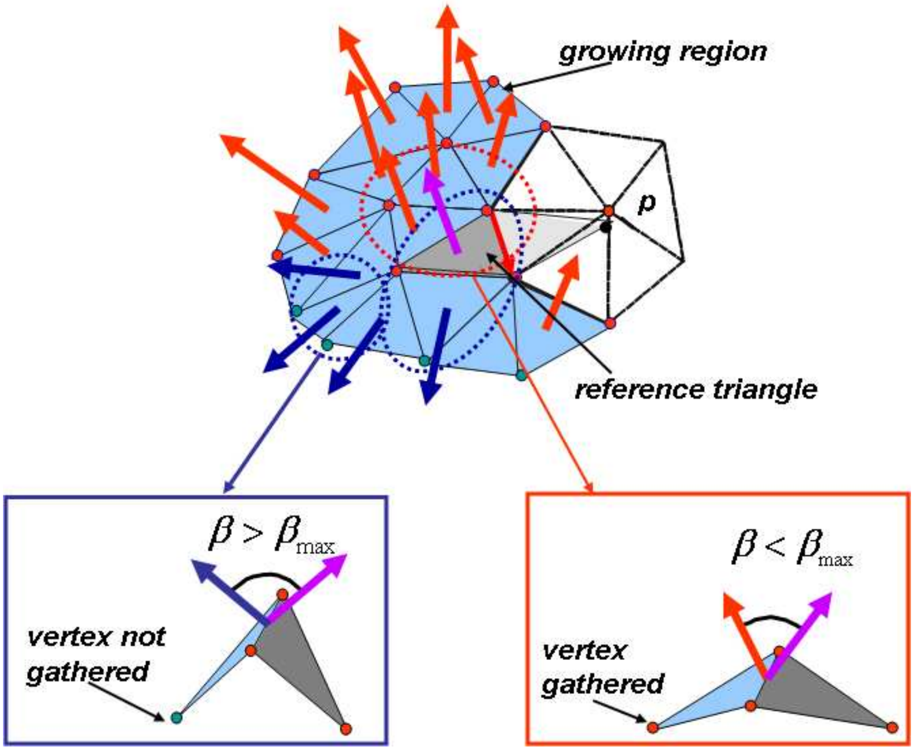


Figure 3.5 Gathering process. If the angle between the triangle normal and the reference normal is smaller than a given threshold then the decoded incident vertex is collected (right) otherwise it is not used for surface fitting(left).

gle. First we computed the so called *reference normal* of the reference triangle, gathered its three vertices and initialized the growing region to the reference triangle by placing its three edges onto a stack. As long as the stack was not empty, we popped an edge from it and checked if the incident triangle outside of the growing region could be incorporated. This check included whether all vertices incident to the triangle had already been encoded and whether the angle between the triangle normal and the reference normal was smaller than a given threshold. We used a threshold of sixty degrees in all our measurements. If a triangle succeeded all tests, we incorporated it into the growing region and pushed all new edges on the region border onto the stack. If the third vertex of the triangle was newly encountered, we collected its decoded location. The normal check also ensures that we can always fit a graph $f(x, y)$ to the gathered vertices with the x- and y-axes in the plane of the reference triangle.

During fitting we weighted the fit vertices by their 3D distance r from the center of the gate via $1/r^e$, where e is the so-called *weighting exponent*.

To find out the best number of to be gathered vertices and the best weighting exponent, we plotted the number of bits per vertex consumed by the normal component for different weighting exponents over the number of gathered fit vertices. The bits per vertex were av-

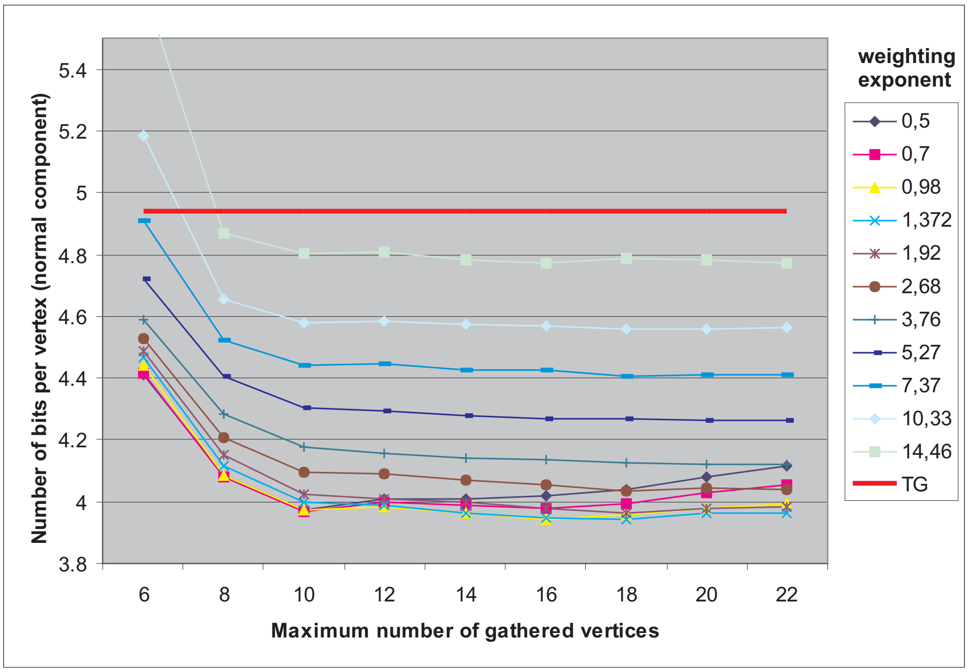


Figure 3.6 Plot of bits per vertex consumed for the normal component with different number of gathered fit vertices and different weighting exponents. The straight red line illustrates the performance of the angle prediction via Touma and Gotsman.

eraged over the collection of sample models (see chapter 8, Figure 8.1). The resulting plot in Figure 8.6 shows a minimal coding cost for eighteen gathered vertices and a weighting exponent of approximately 1.3. More results will be reported later in 8

3.3.2 Higher Order Surface Fitting

For higher order surface fitting we fit the graph $f(x, y)$ of a polynomial function defined over the tangential coordinates x and y, where in this case y is not the radius but the actual direction in the plane of the reference triangle. $f(x, y)$ is the virtual z component of the local coordinate frame. An arbitrary polynomial function in two variables with maximum degree d is given by its Taylor representation

$$f(x, y) = \sum_{j=1}^{m} \alpha_j \phi_j(x, y),$$

with $m = \frac{(d+1)(d+2)}{2}$ basis functions $\phi_i(x, y) = (1, x, y, x^2, xy, y^2, \ldots, y^d$ and the m parameters α_i. For example for degree $d = 2$ there are $m = (d + 1)(d + 2)/2 = 6$ basis functions, and for degree 3 there are ten basis functions. The vector notation

$$\begin{aligned}\Phi(x,y) &= (\phi_1(x,y),\ldots,\phi_m(x,y))^t\\ A &= (\alpha_1,\ldots,\alpha_m)^t\end{aligned}$$

allows us to simplify the expression for f to

$$f(x,y) = A^t\Phi(x,y).$$

Suppose we are given a set of k 3D points $\mathbf{p}_i = (x_i,y_i,z_i)^t$, $i = 1\ldots k$ with weights ω_i resulting from the gathering phase. Fitting of f corresponds to the minimization of the squared z-distances

$$A_{\text{fit}} = \min_A \arg\left[E_f^2(A) = \sum_{i=1}^{k}\omega_i\left(A^t\Phi(x_i,y_i) - z_i\right)^2\right].$$

As A is independent of the summation index i, it can be taken out of the sum resulting in

$$\begin{aligned}E_f^2(A) &= A^t(\mathbf{F}A - 2\mathbf{f}),\\ &\mathbf{F} = \sum_{i=1}^{k}\omega_i\Phi(x_i,y_i)\Phi(x_i,y_i)^t,\\ &\mathbf{f} = \sum_{i=1}^{k}\omega_i\Phi(x_i,y_i)\end{aligned} \tag{3.2}$$

with the symmetric $m \times m$ matrix $\mathbf{F}$. The vector A_{fit} minimizing the squared distances can be found by setting the gradient $\nabla_A E_f^2(A_{\text{fit}})$ to zero, what yields

$$\mathbf{F}A_{\text{fit}} = \mathbf{f}.$$

To solve these equations independent of a potentially singular symmetric matrix $\mathbf{F}$, we decomposed $\mathbf{F}$ with an eigenvalue decomposition into an orthogonal matrix $\mathbf{O}$ and a diagonal matrix $\mathbf{\Lambda}$

$$\mathbf{F} = \mathbf{O}\mathbf{\Lambda}\mathbf{O}^t$$

If the matrix $\mathbf{F}$ has full rank m, A_{fit} computes to $O\mathbf{\Lambda}^{-1}\mathbf{O}^t\mathbf{f}$. In the case of a rank r smaller than m suppose the columns of $\mathbf{O}$ are arranged such that

$$\mathbf{\Lambda} = \text{diag}(\lambda_1,\ldots,\lambda_r,\overbrace{0,\ldots,0}^{m-r}),$$

and we set

$$A_{\mathrm{fit}} = \sum_{i=1}^{r} \mathbf{O}_i \frac{1}{\lambda_i} \left(\mathbf{O}_i^t \mathbf{f}\right).$$

3.3.3 Intersecting Higher Order Surfaces with a Tangential Circle

Now that we are able to fit a higher order surface to a set of weighted vertex locations, we finally have to find out how to compute the intersection of the tangential circle with the higher order surfaces. The tangential circle corresponding to the coordinates x_{tan} and y_{tan} is defined by the two equations

$$\begin{aligned} x_{\mathrm{tan}} &= x \\ y_{\mathrm{tan}}^2 &= y^2 + z^2. \end{aligned}$$

The points on the higher order surface obey $z = f(x, y)$, such that we have to solve the polynomial of maximum degree $2d$

$$y_{\mathrm{tan}}^2 = y^2 + f^2(x_{\mathrm{tan}}, y) \tag{3.3}$$

for y. In the case of $d = 2$ the resulting quartic equation can be solved in closed form. For higher degree an iterative solver is necessary. In our experiments we restricted ourselves to the case with $d = 2$.

After having solved equation 3.3 for y, we computed for each solution y_i the corresponding z value $z_i = f(x_{\mathrm{tan}}, y_i)$. Finally, we computed up to four potential prediction angles α_i and selected the one minimizing the absolute value. In the case of no solution, we used the bending angle prediction strategy of Touma and Gotsman. In all measurements we performed this happened in less than one percent of the cases.

3.4 Alternative Approaches

We develop alternative approaches for the prediction of the normal component that could replace the height field fitting. The first is fitting of implicit function that are more expensive to compute but do not depend on a good estimate of the tangential space of the surface. The second is fitting of spheres, what neither depends on a tangential space but is much faster to compute as implicit fitting.

3.4.1 Fitting of Implicit Function

In this approach, we make the fitting process independent of the local coordinate system, we propose to fit implicit function g(x, y, z) such that the zero set represents the surface.

A polynomial function in three variables with maximum degree d can be written as

$$g(x,y,z) = \sum_{j=1}^{\frac{d(d+1)(d+2)}{6}} \beta_j \gamma_j(x,y,z),$$

with the basis functions $\gamma_j(x,y) = (1, x, y, z,\ x^2, xy, y^2, yz, z^2,\ zx,\ x^3, \ldots, xyz, \ldots, z^d$ and the parameters β_j. With the vector notation

$$\begin{aligned} \Gamma(x,y,z) &= \left(\gamma_1(x,y,z), \ldots, \gamma_{\frac{d(d+1)(d+1)}{6}}(x,y,z)\right)^t \\ B &= \left(\beta_1, \ldots, \beta_{\frac{d(d+1)(d+2)}{6}}\right)^t \end{aligned}$$

allows to simplify the expressions of g to

$$g(x,y,z) = B^t \Gamma(x,y,z)$$

Suppose we are given a set of k three dimensional points $\mathbf{p}_i = (x_i, y_i, z_i)^t, i = 1 \ldots k$ with weights ω_i. Fitting g corresponds to the minimization of the squared analytic distances $g^2(x_i, y_i, z_i)$:

$$B_{\text{fit}} = \min_B \arg \left[E_g^2(B) = \sum_{i=1}^{k} \omega_i \left(B^t \Gamma(x_i, y_i, z_i)\right)^2 . \right]$$

As B independent of the summation index i, it can be taken out of the sum resulting in

$$\begin{aligned} E_g^2(B) &= B^t \mathbf{G} B, \\ & \quad \mathbf{G} = \sum_{i=1}^{k} \omega_i \Gamma(x_i, y_i, z_i) \Gamma(x_i, y_i, z_i)^t, \end{aligned}$$

with the symmetric $k \times k$ matrices $\mathbf{G}$. The minimum of the squared distances is achieved by setting $\nabla_B E_g^2(B_{\text{fit}}) = 0$ which yields

$$\mathbf{G} B_{\text{fit}} = \mathbf{0}$$

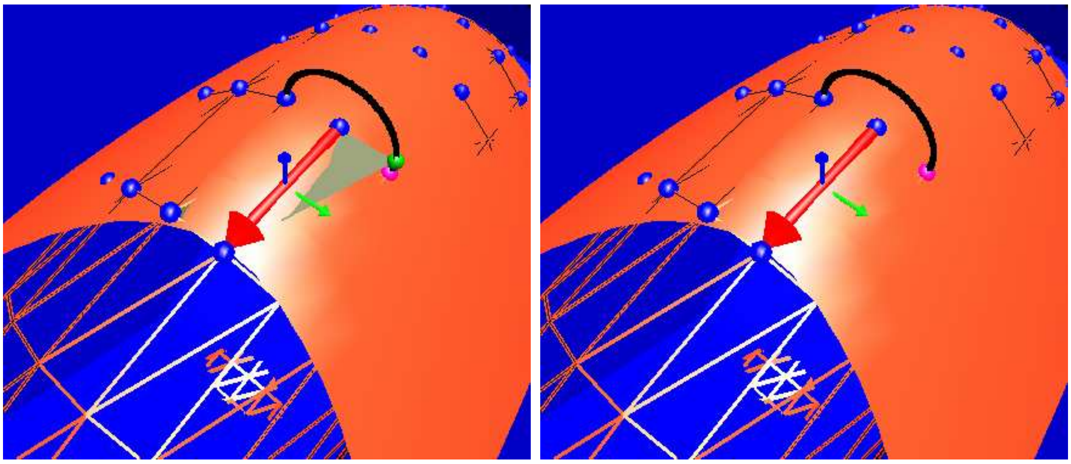


Figure 3.7 Snapshot during the higher order prediction of the torus model (polynomial implicit function).

To solve these equations independent of potentially singular symmetric matrices $\mathbf{G}$ we decompose them with an eigenvalue decomposition into an orthogonal matrix $\mathbf{O}_g$ and a diagonal matrix $\mathbf{\Lambda}_g$

$$\mathbf{G} = \mathbf{O}_g \mathbf{\Lambda}_g \mathbf{O}_g^t$$

If the matrix $\mathbf{G}$ has full rank m, B_{fit} computes to $O_g \mathbf{\Lambda}_f^{-1} \mathbf{O}_g^t$. In the case when of a rank m smaller than k, we suppose the columns of $\mathbf{O}_g$ are arranged such that the eigenvalues $\lambda_{g,i}$ are sorted by decreasing absolute value. Then the eigenvector $\mathbf{O}_{g,k}$ corresponding to the eigenvalue $\lambda_{g,k}$ with the smallest absolute value will minimize the squared error and we set

$$B_{\text{fit}} = \mathbf{O}_{g,k}.$$

The same strategy of surface fitting described before is followed. To find out the predicted bending angle, we compute the intersection between the tangential circle and the higher order surface. Figure 3.8 shows an example. The intersection point is represented by the violet ball.

3.4.2 Sphere Fitting

Sphere based prediction is another predictor that estimates the curvature from the triangles adjacent to the reference triangle. Sphere fitting is based on the assumption that four neighboring vertices (p1, p2, p3, p4) which are noncoplanard, form an unique sphere whose radius r and and its center coordinates (x_0, y_0, z_0) can be found by resolving the following cartesian set of equations:

$$(x_i - x_0)^2 + (z_i - z_0)^2 + (z_i - z_0)^2 = r^2 \qquad i = 1, ..., 4$$

Figure shows 3.8 higher order prediction based sphere fitting. To the three points of the reference triangle and the incident point in the adjacent triangle (left), we fit a sphere with radius R_l and center C_l. The radius of this sphere estimates the inverse of the curvature at reference triangle. Similarly, on the right side we construct another sphere with radius R_r and center C_r. We compute the average of both radii, and with the points of the reference triangle, we reconstruct a new sphere with the average radius r. Finally, we find out the predicted angle by intersecting the tangential circle with this new sphere (see the bottom row of figure 3.8). Also, here, we choose the intersection point corresponding to the angle closer to zero.

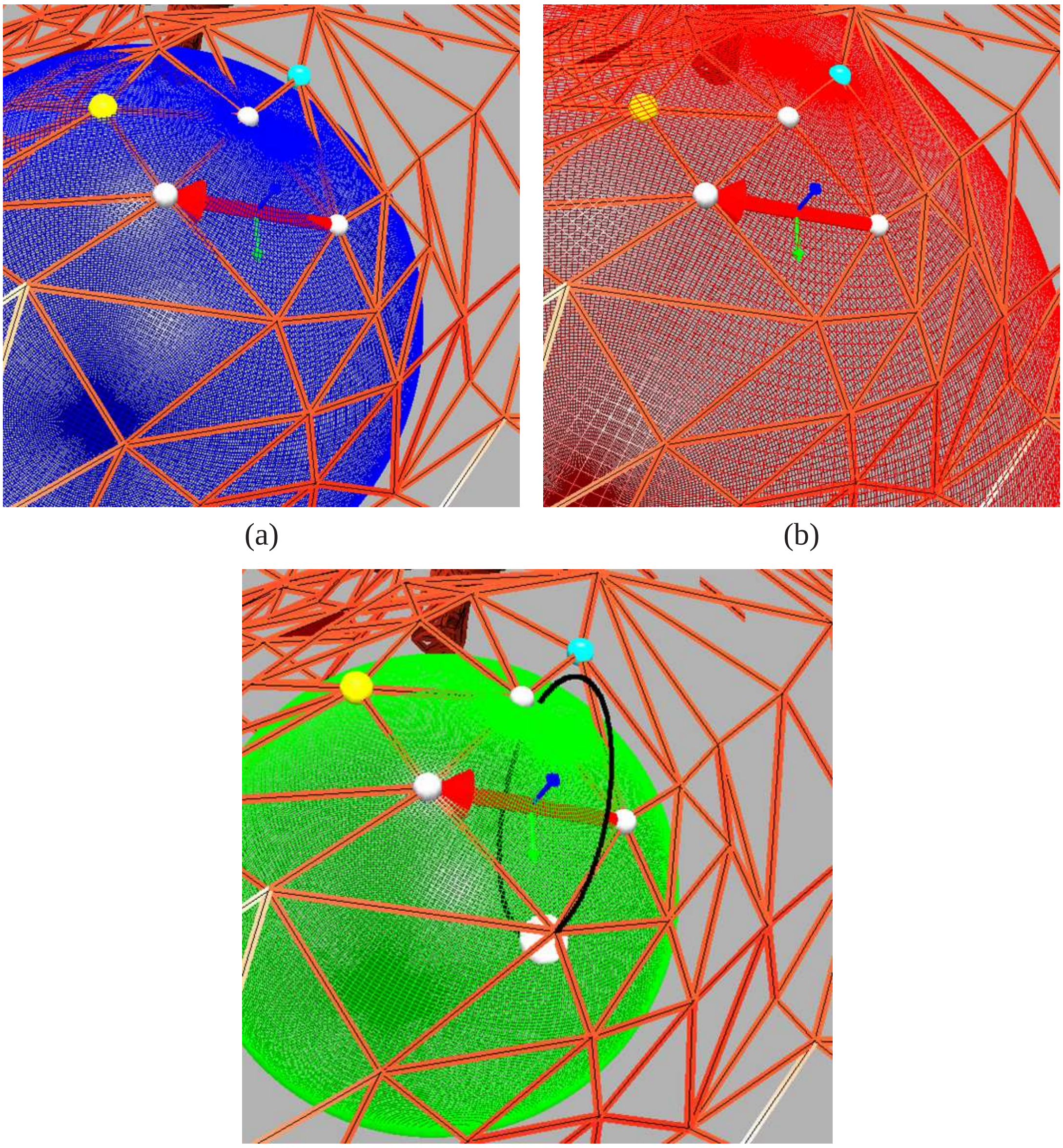
Similar to the surface fitting, before fitting, we consider the following condition: if the adjacent triangle is available and if the angle between its normal and the reference normal is smaller than a given threshold. The radius is then defined by:

$$r = \left\{ \begin{array}{ll} \frac{r_l + r_r}{2} & \text{if two adjacents triangles are availble and fit the condiction} \\ r_r \text{ or } r_l & \text{if only one sphere is available (right or left side)} \end{array} \right\} \tag{3.4}$$

In the case of no reference spheres are fit, then we use the bending angle prediction method of Touma and Gotsman.

3.5 summary

We have presented a higher order prediction scheme for geometry compression, which is based on the splitting of the vertex locations in its tangential and normal components in the local coordinate system. The normal component is encoded as bending angle. For its prediction, we first fit a polynomial surface to the previously encoded vertices in the vicinity of the current gate edge. Then, we intersect the tangential circle given by the tangential components, which are encoded in advance, with the polynomial surface yield-



(a)

(b)

Figure 3.8 Higher order prediction process based on sphere fitting. Top: fit sphere to four points that connect the reference triangle with its adjacent decoded triangle. (a) shows the right side and (b) the left side. Bottom: given the average radiuses of the above spheres,we fit a new sphere to three previously decoded points.

ing the prediction for the bending angle. We examined two approaches to fit polygonal surfaces to a set of points. In the first approach we fit the graph $f(x, y)$ of a polynomial function defined over the tangential coordinates x and y. In the second approach we fit a polynomial implicit function $g(x, y, z)$ such that the zero set represents the surface. This has the advantage that there is no need for guessing the tangential space.

We will see in chapter 8 the gain of our method is best for smooth objects, the proposed gathering strategy for the fit vertices ensures an improvement of bending angle prediction for all models. We showed that our approach can be combined with different prediction schemes for the tangential components, in specific the parallelogram prediction rule and the multi-way prediction proposed by Touma and Gotsman [117] and Cohen-Or et al. [25] respectively, and allows to save in average one bit per vertex for the normal component for smooth objects (see chapter 8). Higher-order prediction can also be combined with the Angle Analyze [74] and we believe that it would also improve the compression rates by one bit per vertex. We also introduced a new normal encoding algorithm based on sphere fitting to speed up higher order prediction. Hence, we fit sphere to a small number of vertices as a fast compromise between graph fitting and simple angle prediction.

CHAPTER 4

Animated 3D Object Segmentation

Segmentation of dynamic 3D objects has recently gained much interest and is used in various contexts such as compression parametrization, ray tracing, morphing and skinning mesh animation.

The aim of this chapter is to introduce new segmentation strategies to be used in the compression of 3D dynamic meshes. We propose three approaches based on region growing, clustering, and adaptive clustering. The aim is to decompose the 3D deforming triangular mesh into near-rigid components, i.e. to group vertices with similar motions, then compress each group separately and thereby, to reduce the computation complexity and to achieve better compression performance.

These algorithms can be well applied to different kinds of deforming meshes whose connectivity and the number of vertices does not change over time, and no information about how the motion is generated, is necessary.

4.1 Introduction

3D Mesh segmentation has become a necessary operation for many applications in computer graphics and geometric modelling. It is a process that divides a mesh into components depending on the applications. The quality of the method is strongly related to its application which may impose different requirements and criteria. Various approaches have been proposed for static meshes, including the region growing, watershed, clustering and spectral analysis methods.

While static mesh segmentation aims at detecting meaningful parts and subsequently breaking the mesh into sub-meshes of similar specific features within a specific context, 3D dynamic mesh segmentation approaches exploit temporal information to partition the mesh into quasi-rigid parts. In other words, 3D dynamic mesh segmentation approaches group the vertices that undergo similar motion.

Different approaches have been proposed (see chapter 2). The central aim is how to partition the deforming mesh into quasi-rigid components. Of course, this relies heavily on the application and its objective. Generally, a partitioning scheme that works well in one application, may not even be applicable in another application. Indeed, some methods were introduced specifically for special kinds of input meshes. For example, James and Twigg's approach [54] performs well for virtual characters but not for extremely deformed animation sequences where most components are not rigid.

In the context of 3D dynamic mesh compression, segmentation is rarely used compared with the huge number of algorithms proposed for static meshes (see chapter 2). Lengyel [77] proposed that one select a set of seed triangles randomly and compared their trajectories. Triangles with a similar motion are combined. Then the vertices are associated with the triangle whose trajectory best fit theirs. Gupta et al. [38] use a multilevel ***k-way*** partitioning algorithm [41]. Mamou et al. [85] proposed that one partition the mesh vertices into clusters whose motion can be described by unique 3D affine transforms, by applying k-means [60]. These approaches use the connectivity information for segmentation, potentially also using the geometry information. Generally, any approach that only uses connectivity-based clustering is not well-suited for geometry that changes over time.

An alternative approach has been proposed by Sattler et al. [102]. They analyze the motion of each vertex independent of the connectivity information. They clustered the trajectories of vertices using Lloyd's algorithm in combination with PCA. Then, they compress each cluster independently. However, the experimental results showed that the cutting boundary may deviate from the deformable regions. Thus, the property of near-rigidity is not always obtained as cited in [76].

Most of the current approaches are either computationally expensive, or are not efficient for our problem, due to using other criteria which may not fit our objective.

This chapter presents new segmentation methods for dynamic mesh compression. These approaches are designed so they can be combined with the proposed compression schemes to achieve high compression rate with high quality reconstruction.

4.2 Overview

Usually, the objective of partitioning a mesh in the context of geometry compression is to decrease the computational costs as well as to preserve the global shape of the mesh because some compression algorithms such as some spectral based techniques (DCT, PCA, etc.) can destroy important features of the mesh.

Another objective is to find near-rigid components to decorrelate the sequence of meshes during a pre-processing stage in the compression pipeline. This leads to a more compact representation. In all cases, the algorithm employed to partition mesh vertices into near-rigid parts heavily affects compression performance.

Our goal is to gather the vertices which have similar movement. Therefore, performing segmentation for one frame using the spatial information only and then applying it to all frames in an animation would not make sense. Moreover, criteria based segmentation process that works well for one frame may not necessarily be appropriate or efficient for the other frames. We want to take into consideration both the temporal and spatial information, i.e. consider the motion of the vertex and of its neighbors over all frames. This would allow us to gather the vertices into groups of near-rigid motion, and consequently achieve better compression performance by using, for example, PCA. Thus, segmentation is a crucial step in the compression pipeline. Its effectiveness lies in its ability to segment an animated object into rigid-bodies.

This chapter presents new segmentation approaches for dynamic meshes. The first method is based on region growing. We involve both connectivity information and the vertex positions over time to gather the vertices of similar motion. The algorithm starts with several selected seed triangles and grows a region incrementally. The growing criteria, or cost function that decides if a newly encountered vertex can be added to the current region, is defined by the Euclidian distance of the vertex position to the seed triangle over time.

The second approach involves the geometry information only. The process of clustering starts with several seed points and defines for each cluster one LCF. Then, it groups the mesh vertices into clusters by analyzing the local motion relative to a local coordinate system defined for each cluster, in which the cluster motion will be encoded. The relationship between the proposed segmentation and compression strategies is an another reason that let us to develop new clustering method that is more suitable for our coding. Indeed, we are minimizing the vertex displacement relative to the LCFs, while the existing methods try to minimize certain criterion function using global feature or distance function in the global coordinate frame. Thereby, they are not efficient for our problem. The results of both of the proposed segmentation methods depends on the initial seed selection.

The third approach, which uses an adaptive process, is introduced to find a more flexible clustering process. We want to minimize the deviation in the LCF to obtain *low-motion partitioning*.

The idea is to start with an initial partitioning of very small number of clusters then iteratively find new see point, add a new cluster and update the partitioning until the cost function converges or the predefined number of clusters is attended.

4.3 Definitions

In this section, we introduce a set of definitions and notation which will be used throughout the current and the next chapters.

Let $M_1, M_2, M_3, ..., M_F$ be a sequence of F meshes. Each mesh M_i is called frame or mesh frame. V and T are the numbers of vertices and triangles in the mesh, respectively. Both V and T are constant over time. Let $\mathbf{G}$ be the set of all vertices (vertex position and index) in a mesh. The segmentation of the mesh M means partitioning the mesh into N subsets of element, i.e.:

$$\bigcup_{n=1}^{N} \mathbf{G}_i = \mathbf{G}, \qquad \mathbf{G}_i \subset \mathbf{G}, \quad \mathbf{G}_i \cap \mathbf{G}_j = \emptyset, \quad i,j = 1,...,N, \quad i \neq j$$

$\mathbf{G}_i$ is characterized by V_i vertices

For a sequence of meshes, let G_i^f be the $i-th$ segment of the $f-th$ frame, $i = 1,...,N$ and $f = 1,...,F$. A single segment $\mathbf{G}_i$ thus consists of F segments (one for each frame):

$\mathbf{G}_i = \{G_i^1, G_i^2, G_i^3, ..., G_{V_i}^F\}$

The segmentation process depends on the criteria used to associate vertices to segment $\mathbf{G}_i$. This criteria is chosen as an objective function that depends on the application or the context of segmentation. The segmentation can then be treated as an optimization problem of given criteria.

4.4 Region Growing based Approach

This approach assumes that all meshes have the same connectivity. The basic idea is to grow regions starting from several seed points. The regions grow uniformly around the set of selected seed points by first traversing the closest neighboring vertices over time until all vertices of the mesh are visited. Each region is also called segment.

4.4.1 Segment Initialization

The segmentation process initializes N sets $\mathbf{G}_i$ of V_i vertices, where $i = 1, ..., N$. All sets are empty and all vertices are marked as unvisited vertices.

4.4.2 Seed Selection

To collect the mesh vertices into N regions, it is necessary to select N seeds using, for example, the far distance approach [126]:

1. The first seed is the vertex corresponding to the largest euclidian distance from the geometrical center of all vertices in the first frame.

2. The next seeds are selected sequentially until all N seeds are selected. After selecting $i-1$ seeds, the $i-th$ seed is the vertex with the farthest distance from the set of $i-1$ already selected seeds, that satisfy : max(min(distance(vertex v, seed j))), where j is a seed vertex in the set of $i-1$ seeds, v is a vertex from the remaining mesh vertices.

3. The process above is iterated until all N seeds are selected

We associate with each seed one of its incident triangles and call this triangle the seed triangle. The regions are initialized as $(v^f_{i,1}, v^f_{i,2}, v^f_{i,3})$ the three vertices of seed triangle of i-th segment.

4.4.3 Mesh Growing Process

This algorithm grows the regions starting from the seed triangles. Every region has an interior, i.e. all triangles that belong to the region, and an exterior, i.e. all triangles that have not been visited. Every region has a queue associated with it which consists of edges who separate between the interior of the region and the exterior.

The queues drive the growing process. Every edge connects two triangles, one inside the region and one outside the region, which is called a *candidate triangle*. The vertex of the outside triangle that does not lie on the edge is called a *candidate vertex*. The queue of every region $\mathbf{G}_i$ is initialized to the (three) edges of its seed triangle. The edges of the queues are sorted by the distance of their candidate vertices to the seed vertex; we use the average of all position of a vertex in all frames as vertex position.

We iterate over all regions and for every region we add the candidate triangle whose candidate vertex has the lowest distance to the region. This candidate vertex can easily be found because it is the candidate vertex of the top-most edge of the region's queue. The iteration stops if no more edges are in the queues. (i.e. no more candidate vertices exist and all triangles are sorted into the regions).

When a candidate triangle is added to a region, it is marked as visited. The queue is updated by removing the edge and by adding the two remaining edges of the candidate triangle.

4.4.4 Results

The region growing based segmentation algorithm is tested on several different animations. Figures 4.6 and 4.7 illustrate some results for segmentation into different number of regions. Figure 4.6 shows sample frames from the dance and chicken animations segmented into 14 and 20 segments, and 10 and 18 segmnets, respectively. Figure 4.7 shows samples frames from the elephant, dolphin, and cow animations segmented into 10 and 20, 9 and 20, and 6 and 20 segments, respectively.

4.5 Clustering based Segmentation

The basic idea of this algorithm is to transform the original vertex coordinates into several *LCFs* defined by seed triangles. One *LCF* (one seed triangle) is associated with each cluster. Then the clustering is obtained by assigning the vertices to the cluster where they have minimal local coordinate variation across the F frames. Minimal coordinate variation means that the vertex and the LCF have very similar motion. Note that this approach needs the connectivity only once to construct one LCF. The clustering process consists of the following steps:

4.5.1 Initialization and Seed Selection

Similar to region growing based segmentation, the algorithm initializes the N sets $\mathbf{G}_i$, where $i = 1, ..., N$, to be empty and all vertices are unvisited. Then it selects a number of vertices using the approach described above. After choosing N seeds, we associate with each seed one of its incident triangles (seed triangle). Then, each cluster is initialized with its three vertices $(v_{i,1}, v_{i,2}, v_{i,3})$ of seed triangle of i-th cluster.

4.5.2 Local Coordinate Frames Construction

Figure 7.2 illustrates the LCF which was used in the clustering process. We assume that each cluster is initialized with a seed triangle $(v_{i,1}, v_{i,2}, v_{i,3})$, and the positions of its

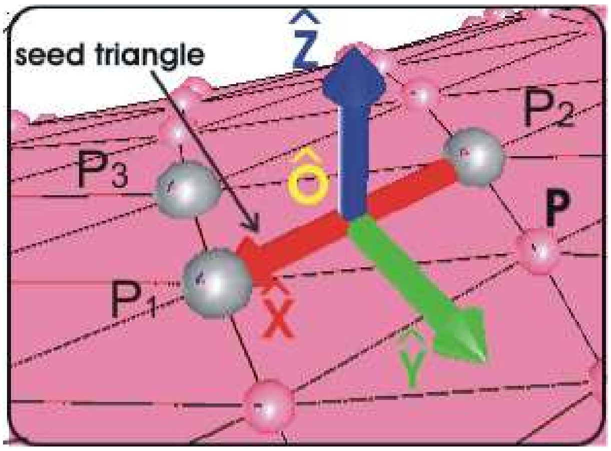


Figure 4.1 Illustration of the local coordinate frame

vertices are $(\mathbf{p_1}, \mathbf{p_2}, \mathbf{p_3})$, respectively. Each cluster $\mathbf{G}_i$ has its own *LCF* defined on the seed triangle. The origin $\mathbf{o}$ is the center of one of its three edges (typically $(\mathbf{p_1}, \mathbf{p_2})$), the x-axis (red arrow) points down the edge $(\mathbf{p_1}, \mathbf{p_2})$, the y-axis (green arrow) is orthogonal to the x-axis in the plane of the seed triangle and the z-axis is orthogonal to the x- and y-axis. The transformation of a point $\mathbf{p}$ to its local coordinate system $\mathbf{q}$ can be accomplished by an affine transformation with a translation $\mathbf{o}$ and a linear transformation $\mathbf{T}$:

$$\hat{\mathrm{q}} = \mathbf{T}(\mathbf{p} - \mathbf{o})$$

$\mathbf{T}$ is an orthonormal matrix, it means $\mathbf{T}^{-1} = \mathbf{T}^t$.

For a sequence of meshes, for each frame f $(1 \leq f \leq F)$ and for each frame cluster $G_i^f \in \mathbf{G}_i$ $(1 \leq i \leq N)$, we computed $\{\mathbf{T}_i^f, \mathbf{o}_i^f\}$ from the points of the seed triangle $(\mathbf{p}_{i,1}^f, \mathbf{p}_{i,2}^f, \mathbf{p}_{i,3}^f)$.

Note that there is no restriction on how the local coordinates are reconstructed upon the seed triangle. The origin also can be the center of the seed triangle or one of its three vertices.

4.5.3 Vertex Clustering

Given an unvisited vertex $\mathbf{p}_k^f$, we do the following: First, we transform its world coordinates into the N local coordinate frames constructed in each frame f, so: $\{\mathbf{q}_k^{1,f}, \mathbf{q}_k^{2,f}, ..., \mathbf{q}_k^{N,f}\}$, $(f = 1, ..., F)$. Second, we compute the total deviation (motion) of the vertex between each two adjacent frames f and $f-1$ in euclidian space, which is also equivalent to the displacements in the LCF:

$$\theta_{k,i} = \sum_{f=1}^{F} \|\mathbf{q}_k^{i,f} - \mathbf{q}_k^{i,f-1}\|^2 \tag{4.1}$$

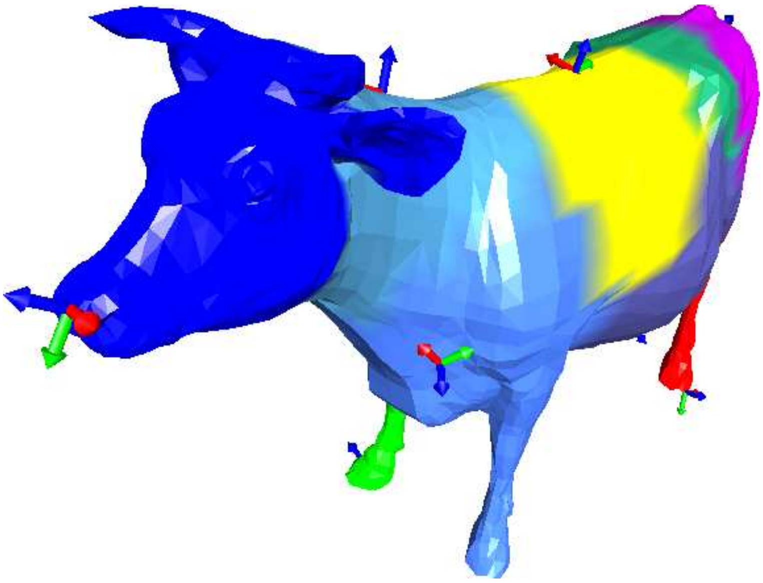


Figure 4.2 Illustration of the local coordinate frame

$\theta_{k,i}$ represents the total motion differences of the vertex k in the *LCF* associated with the cluster i. A small value means that the vertex position has motion that is similar to $\mathcal{C}_i$. Thus the vertex should belong to the cluster i for which the deviation is very small, note i_{min}. Equivalently, we want to minimize the vertex displacement in the LCF:

$$i_{min} := argmin_{1 \leq i \leq N}\{\theta_{k,i}\} \qquad (4.2)$$

We iterate over all vertices, adding the unvisited vertex whose local coordinates are almost invariant in the *LCF* to the cluster $\mathcal{C}_i$.

The iteration stops if no more candidate vertices exist. When a vertex is added to a cluster, it is marked as visited. We end up with N clusters that have V_i vertices each.

4.5.4 Results

The output of the clustering process is shown in the figures 4.8 and 4.9. Figure 4.8 shows sample frames of dance, and chicken animations clustered into 14 and 20, and 10 and 18 clusters respectively . Figure 4.9 shows samples frames from the elephant, dolphin and cow animations clustered into 10 and 20, 9 and 20, and 6 and 20 clusters, respectively.

4.6 Adaptive Processing

The clustering approach basically consists of two passes. Given a desired number N of fixed clusters, N seed points are selected using the distance approach. Then, the vertices are grouped into hierarchical clusters so that the vertices belonging to a cluster have a smaller deviation in the LCF of their cluster than they would in the other clusters. In order to find better clusters and to make the process more flexible, we designed an adaptive process. This is an enhancement to minimize the deviation in the LCF so as to obtain *low-motion partitioning*.

The idea is to start with an initial partitioning of k clusters (typically 1 or 2) then iteratively add a new cluster and update the partitioning until the cost function converges or the predefined number of cluster is attained.

Initial Seeding and Clustering

Given k seed points, we partition the vertices into k clusters by minimizing the cost of function 4.2 as described before.

New Cluster Insertion

In order to find better partitioning, in each iteration step, we first go through all clusters i $(1 \leq i \leq N)$ in the current partitioning $\mathcal{R}_i$ and find the cluster i_{max} with maximum average deviation-maximum cost function:

$$i_{max} := argmax_{1 \leq i \leq N} \{ \frac{1}{V_i} \sum_{k=4}^{V_i - 3} \theta_{k,i} \} \tag{4.3}$$

This means that the cluster i_{max} contains some vertices whose motion is not similar to the cluster motion. Therefore, within it, we pick the vertex v_{max} with the large deviation in the local coordinate frame.

$$v_{max} := argmax_{4 \leq k \leq V_{i_{max}}} \{ \theta_{k, i_{max}} \} \tag{4.4}$$

v_{max} is chosen as a new seed that creates a new cluster, and one of its incident triangle is selected as seed triangle.

To obtain a new partitioning $\mathcal{R}_{i+1}$, we update the clustering as follows: upon the new seed triangle we construct a new LCF, we add a new cluster, and initialize it with the three incident vertices of the seed triangle. The existing clusters are newly initialized with the vertices of their seed triangles. Then, by minimizing the cost function 4.2, the vertices are assigned to the correct cluster.

The iteration process stops when the maximum number of cluster is achieved or when the overall average deviation is below the specified threshold.

For further improvement, a pair of clusters can also be merged when the average deviation of the resulting cluster is less than the average deviation of both individual clusters. Likewise, a cluster with very few vertices (typically 3 or 4 vertices) can be also deleted and its vertices newly clustered.

4.6.1 Results

Figures 4.10 shows sample frames of dance, and chicken animations adaptively clustered into 14 and 20, and 10 and 18 clusters respectively . Figure 4.11 shows samples frames from the elephant, dolphin and cow animations adaptively clustered into 10 and 20, 9 and 20, and 6 and 20 clusters, respectively.

4.7 Evaluation of Segmentation Approaches

It currently says that the evaluation of the segmentation process is dependent on whether or not the segmentation process makes the right segments. This is a tautology. In the context of compression, a good segmentation process is the one which leads to the best bit-rate compression – of course, if the complexity of segmentation and process time is excluded. The goal of the segmentation in this thesis is to gather the vertices of similar motion. Thereby, the redundancy existing between the sequence of frames is reduced a priori. The process should assign each vertex to the segment/cluster where its deviation (motion) is relatively very small or it motion with the group is almost rigid.

To evaluate the quality of the segmentation approaches, we need to define some metric or error. Therefore, we decided to use the cost function used in the clustering approach. This function provides a very efficient indication of how similar the motion within a group of vertices is. The idea is to assign to each group one LCF (as described in section 4.5). The coordinates of vertices are then converted into LCF of their segment/cluster and the motions between each two successive frames are computed. The motion deviation is also defined as residual motion.

The average deviation of each group $\mathbf{G}_i$ is defined to be the average of motion vertices between two successive instants over all frames, in the LCF.

$$Deviation_{aver}(\mathbf{G}_i) = \frac{1}{F(V_k-3)}(\sum_{f=1}^{F}\sum_{i=4}^{V_c}\|\mathbf{q}_k^{i,f} - \mathbf{q}_k^{i,f-1}\|^2)$$

F is the number of frames, V_i is the number of vertices in G_i. i is the cluster index. $\mathbf{q}_k^{i,f}$ is the local coordinates of the vertex k in frame f. It starts with the index 4. The first three vertices 1, 2 and 3 were used to construct the LCF.

The accuracy of segmentation is then defined by the average deviation over all segments/clusters.

$$Deviation_{aver}(\mathbf{G}) = \frac{1}{N}\sum_{i=1}^{N} Deviation(G_i)$$

where N is the number of segments or clusters.

Intuitively, the vertices that belong to the cluster should have minimum residual motion in the LCF of their cluster, meaning that their motion is relatively almost invariant. Thus, better segmentation is the one that leads to small deviation over all groups (the differences between two frames tend to be zero) and thereby better compression performance can be achieved by using for example predictive coding (as we will see in the next chapters).

Figure 4.3 illustrates the average displacements or the average error of all clusters (or segments) of the chicken, dance and cow animations. The clustering approaches are more efficient than growing region based approach. The performance of the clustering comes close to the adaptive clustering approach where we obtain less variation of the vertices in the LCF (almost rigid) than in the region growing approach (less rigid). Thus, the removal of the redundancy a priori seems to be very efficient, allowing the predictive and spectral methods to be efficiently performed for further compression (also see section 4.9).

Note that since we use a metric which is similar to the one used for the clustering approaches, it is obvious that their results will be better than the results of the region growing approaches. However we found that this metric is the best one that can be used for our measurement for our algorithms and fit the goal of segmentation and compression.

4.8 Computation Time

Table 8.6 shows the run time for different segmentation approaches. The second, third and fourth columns show the number of frames, vertices and triangles in each model, respectively. The column RG lists the time required for region growing segmentation. The column Clu lists the time needed for clustering processes and column AdaptClu list the time needed for adaptive clustering. The timing results were measured on AMD Athlon(TM)XP 3000+, 2.10 GHz, 1.00GB of Ram.

Region growing is intuitively the fastest approach. The computation is done in the

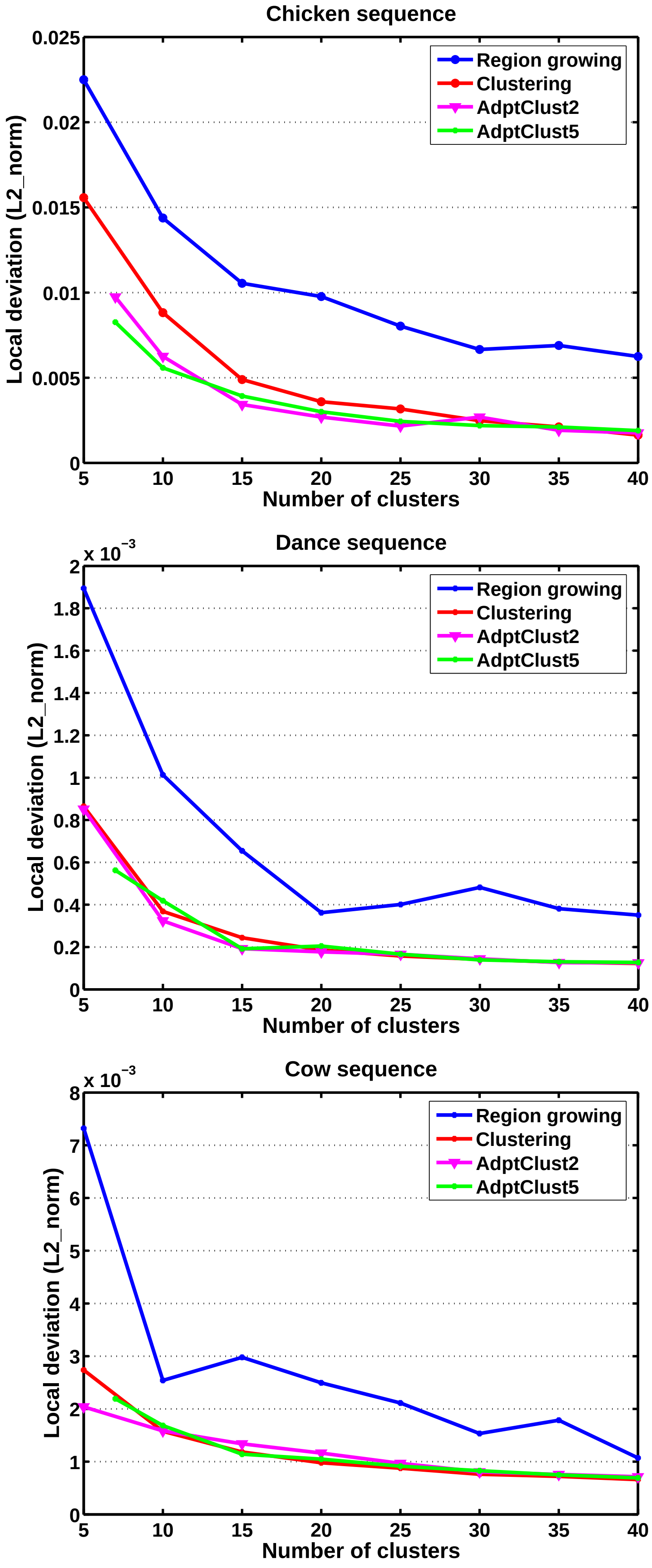


Figure 4.3 Evaluation of segmentation approaches.

world coordinate system while clustering approaches require extra time for transformation. Indeed, in order to determine the correct cluster to which each vertex should belong, we have to computer $N \times F$ times transformations from WCF (World Coordinate Frame) into $N \times F$ LCF per vertex. In the adaptive process, which starts with an initial number of seeds k selected by distance approach as described before, the rest of seeds $(N - k)$ are selected incrementally and new clusters are created. The number of transformations per vertex is about $k \times F$ + $F \sum_{j=k}^{N} j$ making the run-time longer.

Table 4.1 Timing statistic for segmentation results on different animations in seconds (sec). RG: Region Growing, clu: clusters and adaptClu: adaptive clustering.

Models	vertices	triangles	frames	N	RG (sec)	clu(sec)	adaptClu (sec)
chicken	3030	5664	400	5	0.14	3	14
				10	0.14	7	45
				20	0.30	15	184
				30	0.36	23	385
				40	0.41	31	644
dolphin	6179	12337	101	5	0.23	1	6
				10	0.28	3	23
				20	0.33	7	80
				30	0.39	10	173
				40	0.45	14	299
cow	2904	5804	204	5	0.2	1	6
				10	0.24	3	22
				20	0.27	6	77
				30	0.30	10	164
				40	0.33	15	288
dance	7061	14118	201	5	0.5	4	16
				10	0.56	12	54
				20	0.63	16	187
				30	0.7	24	398
				40	0.78	36	702
elephant	42321	84638	48	5	0.92	6	25
				10	1.31	12	78
				20	1.63	23	270
				30	1.95	35	578
				40	2.34	47	995

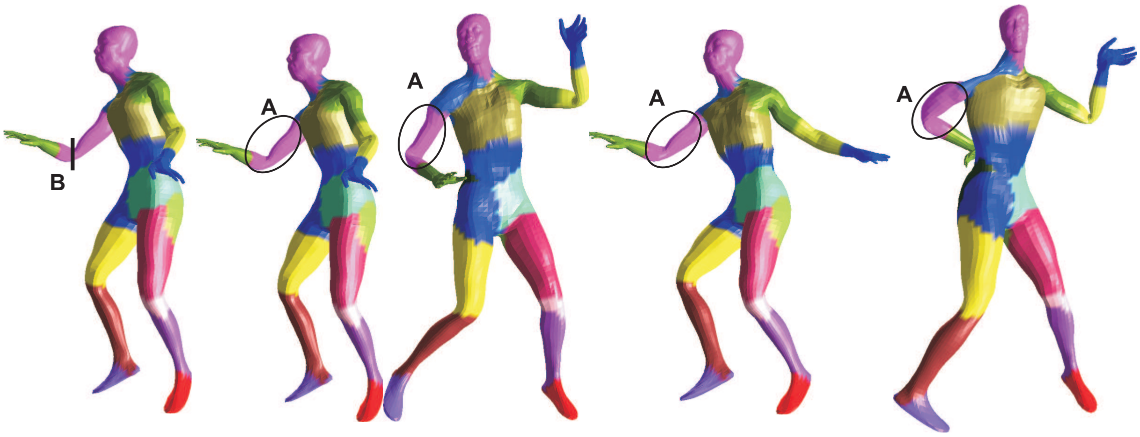


Figure 4.4 Clustered dance animation. The region (A) shows a cluster that is almost rigid. However, it is not necessary to group the vertices int two clusters at (B)

4.9 Discussion and Summary

In this chapter, robust segmentation approaches were designed for all types of animation including high deformable animations.

These algorithms are tested on different mesh animations generated in different ways. Figure 4.5 summarizes the results of the three segmentation approaches. For more results see 4.4.4, 4.5.4 and 4.6.1.

The region growing based strategy is simple and fast. It allows one to group the vertices through connectivity and their distance to the seed triangles assuming that the connectivity remains constant over a time.

In contrast, the second strategy is based on the clustering process and does not need the connectivity information, except during LCF construction. The process assigns each vertex to the cluster where its displacement is relatively very small. In both techniques, the number of segments is given a priori and the seed points are fixed using the distance approach. In order to overcome this limitation and to obtain low-motion partitioning, an adaptive clustering strategy is introduced. The process of clustering incrementally creates seeds and updates the clustering.

Dynamic 3D mesh segmentation aims to group the vertices of similar motion for all types of animation. However, the vertices may not always be grouped into visually meaningful parts but can be grouped into groups of similar motion, such as dance animation. For example, in figure 4.4, the region (A) of the figure visually may be not well segmented visually and it would be better if the vertices are segmented in such way that two regions are created at (B). In contrast, in our context, this animation is well segmented because the vertices in this region have similar motion over time as seen in the four frames

$(3, 56, 92, 137)$: both sides of (B) move almost rigidly over time.

In order to evaluate the segmentation approaches the displacement between frames is measured. We found that this measurement, which is derived from the cost function (used in the clustering), is well-suited for evaluation. For example in the prediction phase, we want to have a prediction that produces a small prediction error. Good segmentation, is then, the one that produces small clusters of similar motion, meaning that if we predict the vertex coordinates from the previous frame (displacement) relative to the LCF, the error would be very small.

Visually as well as metrically, clustering approaches exhibit better partitioning than the region growing based algorithm.

For the computation time, the growing region based algorithm is very fast compared with the clustering approach. This needs to transform the world coordinates into local coordinates, thereby more time is required. The processing time becomes longer when adaptive clustering is used.

Note that the processing time grows with the number of clusters and segments as shown in figure 8.6

Unfortunately, we could not compare our approaches with existing techniques for several reasons. First, their implementations are not available. Second, our algorithms are designed with respect to the proposed compression techniques, meaning they are tailored to fit the encoding algorithms to achieve better compression performance.

Each cluster or region is initialized with a seed triangle, from which we construct an LCF. By assigning the vertex to the cluster in which its deformation over time is small, we are able to segment the dynamic mesh into approximately rigid components, and quasi invariant to their LCF. Then each cluster will be encoded in its LCF.

It is important to note that the process of clustering can be seen as minimization of the vertex displacements relative to LCF. Consequently, performing for example the predictive coding leads to prediction errors that are very small (see chapter 7). And performing PCA in quasi-invariant region will lead to more compact representation than quasi-rigid component. The algorithm will require fewer principal components at similar reconstruction error than with global PCA (where PCA is performed on the whole animated meshes) (see chapter 6).

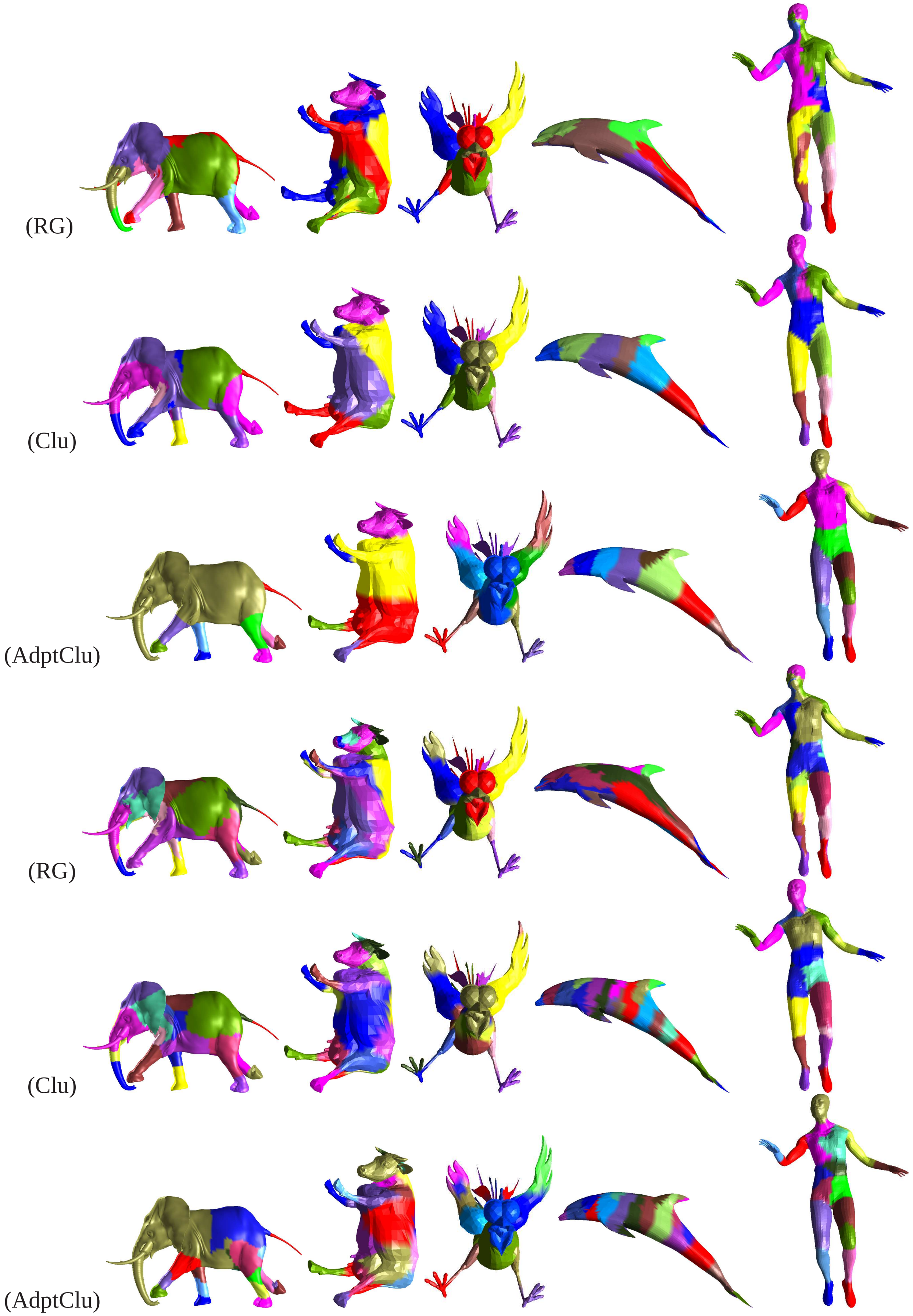


Figure 4.5 From top to bottom: results of region growing (RG), clustering (Clu) and adaptive clustering (AdptClu) approaches. Elephant(10 and 20 clusters), cow (6 and 20 clusters), chicken (106 and 18 clusters), dolphin (9 and 20 clusters) and dolphin (14 and 20 clusters) .

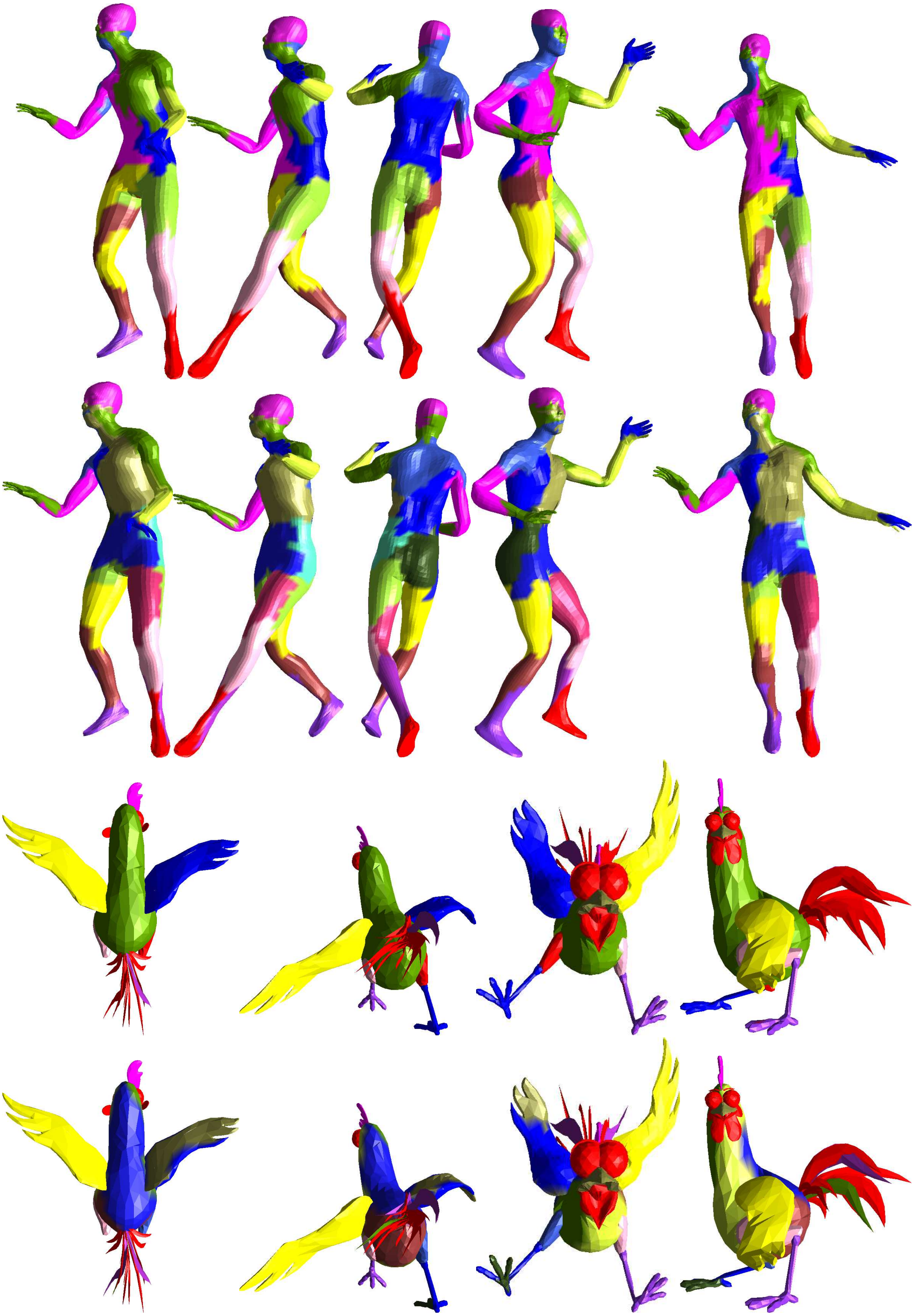


Figure 4.6 Results of region growing based segmentation. From top to bottom: sample frames from the dance (14 and 20 clusters) and chicken (10 and 18 clusters) animations.

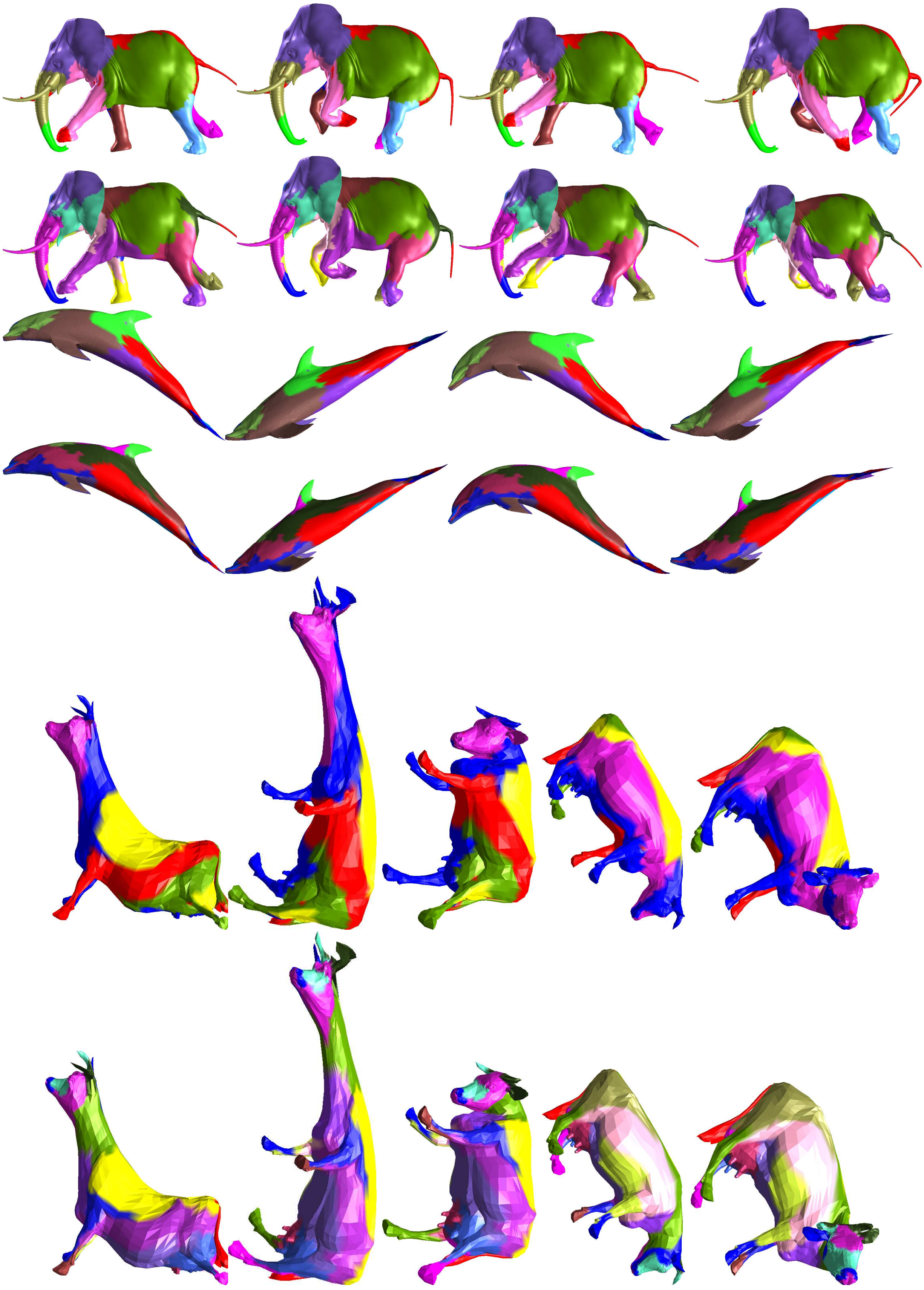


Figure 4.7 Results of region growing based segmentation. From top to bottom: sample frames from the elephant(10 and 20 clusters), dolphin (9 and 20 clusters) and cow (6 and 20 clusters) animations.

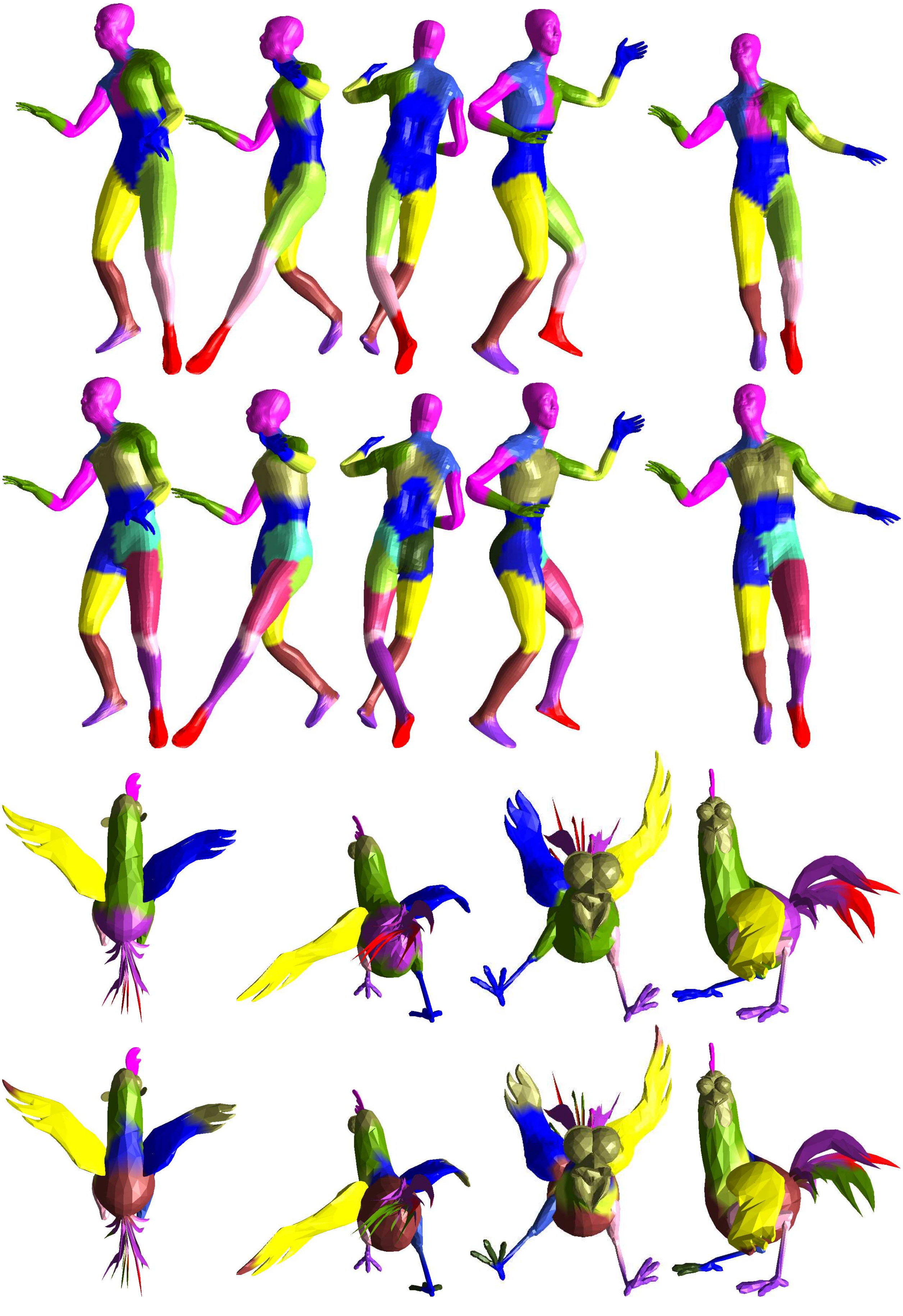


Figure 4.8 Results of clustering approach. From top to bottom: sample frames from the dance (14 and 20 clusters) and chicken (10 and 18 clusters) animations.

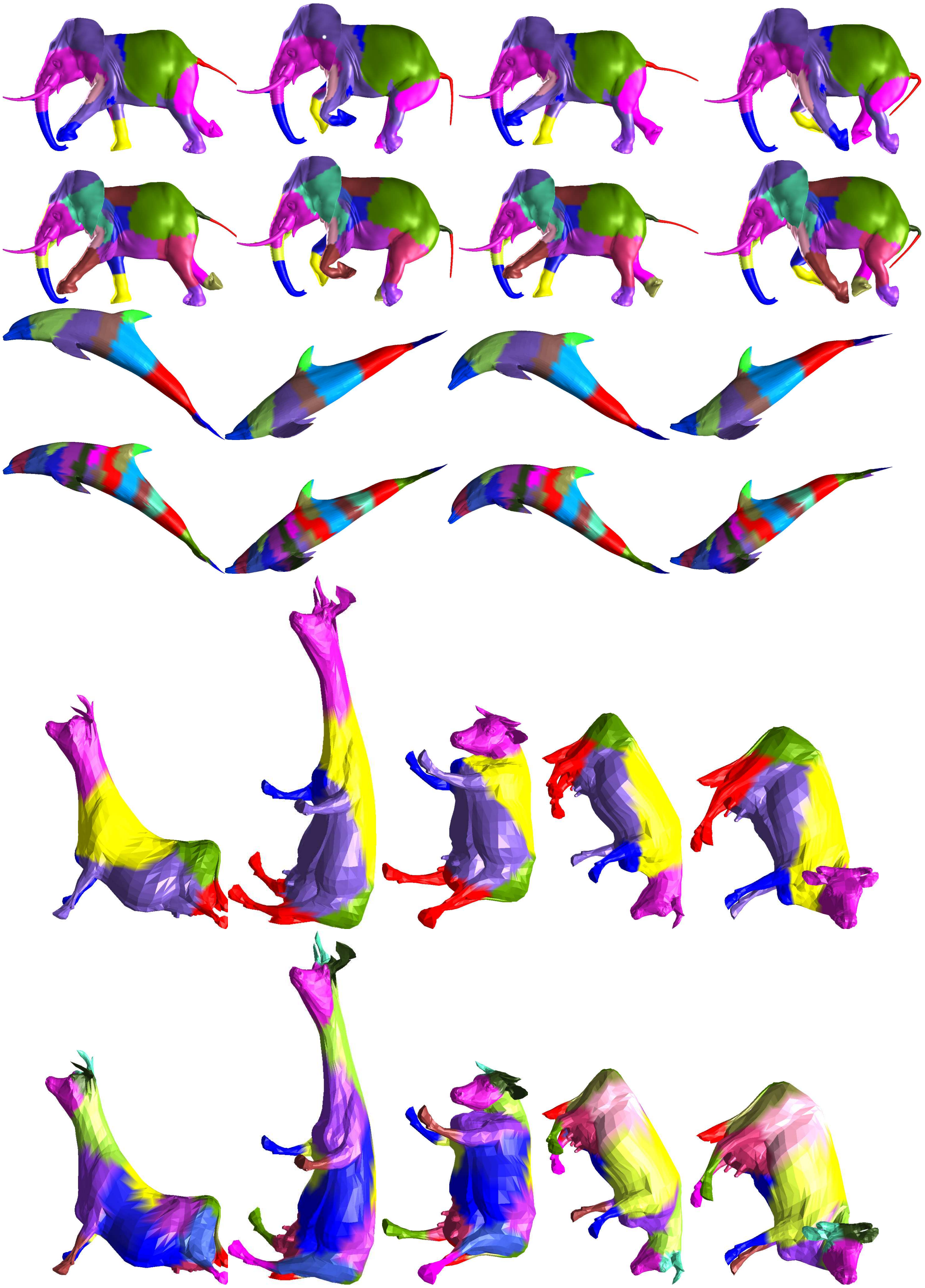


Figure 4.9 Results of clustering approach. From top to bottom: sample frames from the elephant(10 and 20 clusters), dolphin (9 and 20 clusters) and cow (6 and 20 clusters) animations.

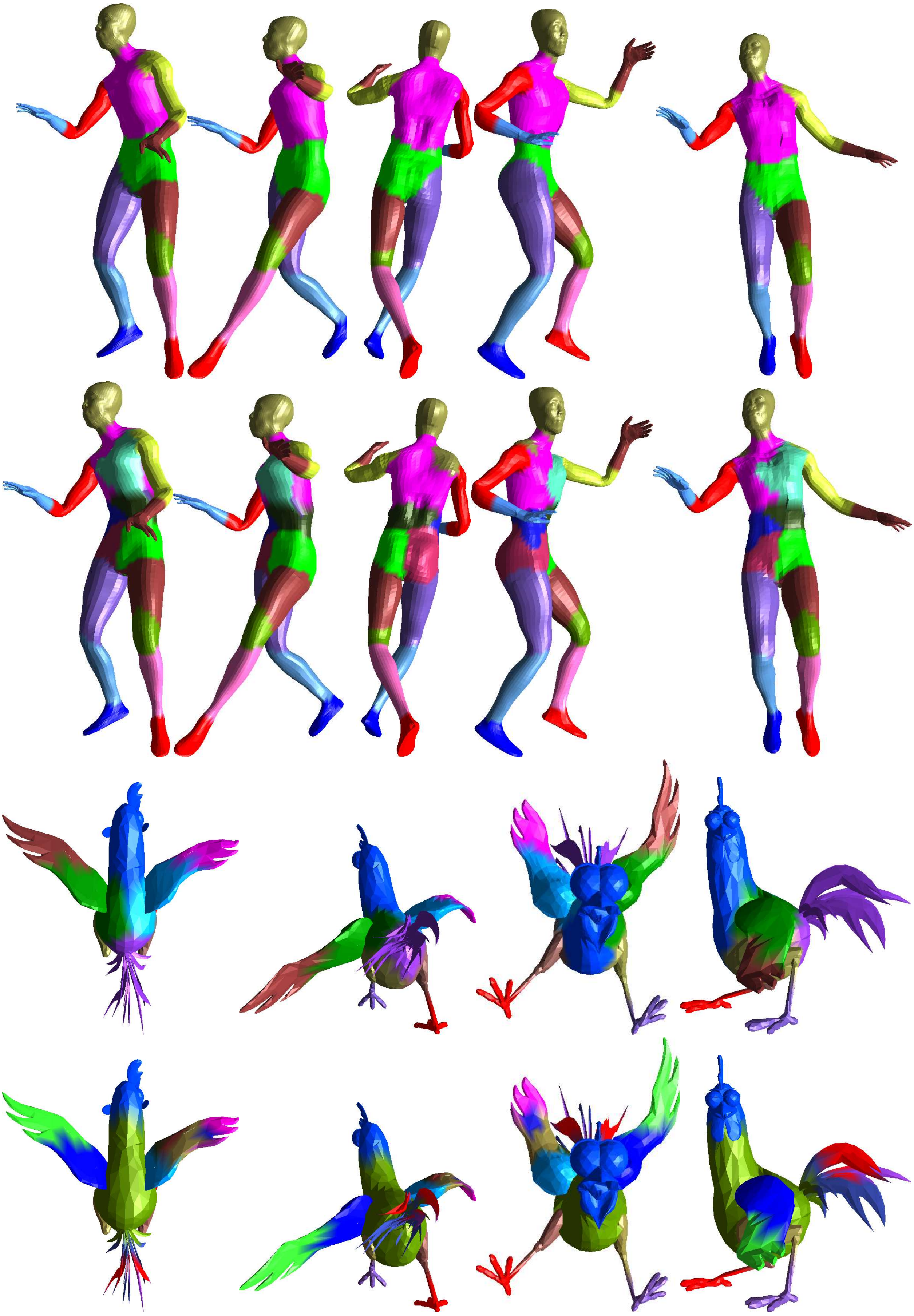


Figure 4.10 Results of adaptive clustering approach. From top to bottom: sample frames from the dance (14 and 20 clusters) and chicken (10 and 18 clusters) animations.

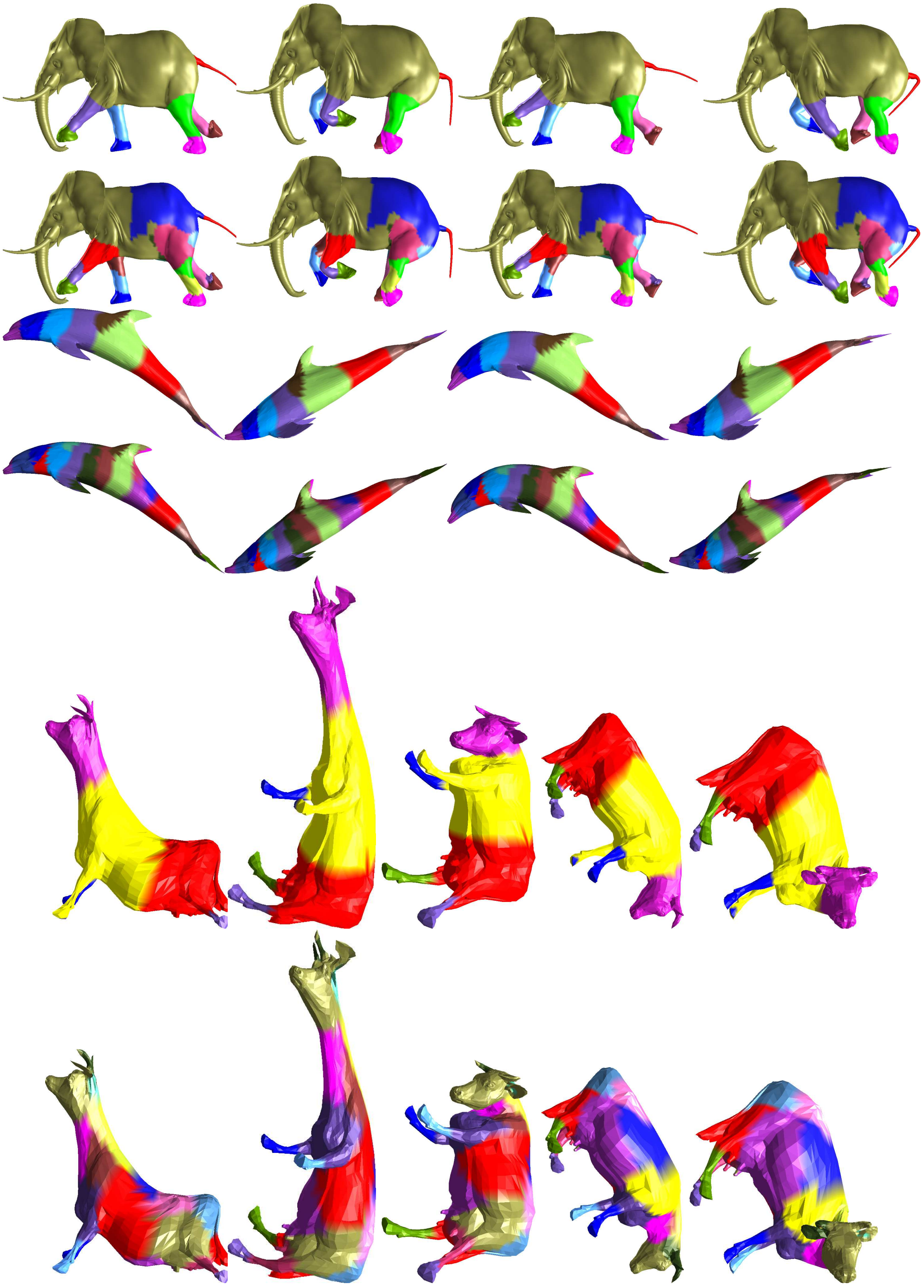


Figure 4.11 Results of adaptive clustering approach. From top to bottom: sample frames from the elephant (10 and 20 clusters), dolphin (9 and 20 clusters) and cow (6 and 20 clusters) animations.

CHAPTER 5

Connectivity-Guided Compression of 3D dynamic Meshes

This chapter aims at compressing animated objects represented by triangular meshes of fixed connectivity using predictive technique. We propose a new connectivity-driven coding which we call SplitCoder. The connectivity is encoded once then the geometry (vertices locations) is encoded by a connectivity traversal of the mesh. Connectivity determines the order of the vertices and provides information for predictions. The algorithm is near-lossless, simple, efficient, fast and well-suited for real time applications.

5.1 Introduction

Often, the meshes differ only slightly between neighboring frames, leading to large amounts of inherent redundancy between frames and between neighboring vertices in the same frame. Thus, the static mesh compression techniques for the compression of sequences of meshes independently are inefficient. Therefore, for efficient coding, one should also exploit inter-frame coherence. There are several criteria by which developed coding techniques can be distinguished. One of these criteria is whether coherence is globally or locally analyzed. In a global approach, on might examine the entire mesh (or submesh) sequence by using, for example, a principal component analysis (PCA) transform. Locally, one might focus on frame to frame changes to exploit local coherence by using, for example predictive coding. In this work, we focus on local compression schemes based on the predictive coding. Such schemes tend to be simple, fast, and well-suited for real time application (unlike PCA based approaches). We categorized this approach vertex based-predictive coding.

Given a number of frames M_f, $f = 0, 1, ..., F$, a predictive method assumes that connectivity does not change over time. For each newly encountered vertex in the current frame, its location is predicted from the locations of neighbor vertices in the current frame or/and in the previous frame(s). Then, the error between the original vertex location and the predicted vertex location $\tilde{\mathbf{p}}_i^f$ is compressed to a user-defined error.

Prediction step is the key of the good compression rate and good prediction lie in its accuracy. Thus, the more the predicted point is close the actual position of the vertex, the more the prediction good is and the lower entropy of prediction errors is achieved.

Current predictive methods predict the position of the vertex either in global coordinates or in local coordinates. However, almost all methods, to our knowledge, compute and quantize the coordinates of the residuals in world space.

Local coordinate frames have many properties that are useful for many 3D animation processing applications. One of these properties is clustering behavior. Generally, the 3D positions of vertices are scattered over wide area and the behavior of the vertices is often non-linear. Fortunately, the correlation between the neighboring vertices, as well as between the successive positions of a single vertex, is very high. In other words, neighboring vertices tend to move together. Thus, we might exploit this property to guess the location of the vertex relative to its neighbors, to obtain a better prediction than space, time or space-time predictions in world space would provide. Therefore, one can construct a local coordinate frame upon the locations of three of these neighbors. The coordinates of each individual vertex tend to concentrate around one point over time, which we call *temporal clustering*. Another clustering, which we call *spatial clustering*, arises in each frame: the model tends to form very few clusters (depending on the model). The combination of both clusterings should yield significant reduction in bit-rates.

In this chapter, we introduce new and simple predictive scheme for single-rate compression for animated meshes of fixed connectivity (guided), it can be seen as generalization of static mesh compression presented in chapter 3 to animation case. Our geometry encoding strategy is based on a region growing encoding order and only the delta vectors between the original and the predicted locations are encoded in a local coordinate system, which splits into two tangential and one normal components. We call this approach *SplitCoder*.

5.2 Predictive Coding of the animated vertices

We distinguish three ways to encode the animated vertex positions. One way considers each frame independent of other frames and uses only the existing static compression techniques such as the popular parallelogram and multi-way prediction. The second way is to encode the trajectory of each vertex through time independent of the motion of its neighbors, so only the temporal redundancy is exploited. Typical predictor is a first-order time only predictor that returns the position of the vertex in the previous frame, eventually the predictor can be a higher order predictor. A more sophisticated way is to join the coherence in both space and time domains to approximate the vertex motion such as Extended Lorenzo Predictor (ELP), Replica Predictor (RP) and Angle Preserving Predictor.

ELP is an extension of the parallelogram predictor to incorporate coherence over time. Replica predictor expresses the location of the vertex in the previous frame as in a coordinate system derived from the adjacent triangle. Then, the same coordinates are used in the current frame to predict the position of the vertex with respect to the new location of the same adjacent triangle. An algorithm similar to Replica Predictor is Angle Preserving Predictor that preserves the angle between the reference triangle and the spanning triangle. Our algorithm belongs to this category of predictive based-compression methods. We let the connectivity information dictate the compression of animated vertices.

As aforementioned before, in world space, the coordinates of vertices scatter in a wide range. In a local coordinate frame constructed upon each reference triangle, however, the coordinates of each mesh vertices tend to cluster around a few points and the trajectory of each vertex tend to cluster around one points.

Since a good prediction is when the residual is small as much as possible, the estimation can be done in the local space producing residual close to zero, rather in the world space. Thereby, the entropy of the sequence of the residuals is very small.

Moreover, a large of animated meshes are highly complex and very irregular, thus it is more advantageous to exploit the prediction and the compression of the residual in the local than world space for example as is done here. This may avoid a poor prediction that may be produced using parallelogram prediction only, or combination with the temporal linear prediction in the global coordinates.

We present a new single rate compression algorithm for animated meshes of constant connectivity. We propose to move the coding of the vertex positions into a new local coordinate frame that splits the coordinates into two tangential and one normal components. The tangential components describe the parametric information of the shape while the normal components hold the geometric information. Local spaces exhibit higher temporal

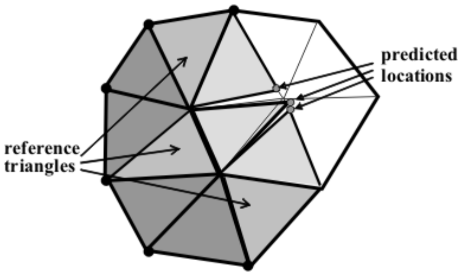


Figure 5.1 Multi-way prediction.

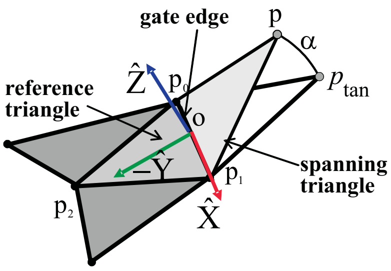


Figure 5.2 Local coordinates represented as tangential coordinates p_{tan} and normal component given as bending angle α.

and spatial clustering behavior than the world space. Looking at a sequence of animated meshes in model space, we observe that the meshes share the same tangential components and that the surface animation can then be viewed as a very low-magnitude movement of the geometric information only. The geometric information is quite constant, at least over a few frames. Therefore, our algorithm, after transformation, predicts and quantizes each component in the new LCF and entropy is encoded.

Another advantage of splitting is that moving the tangential coordinates in tangent plane of the surface may not alter visual error, while moving the point on the normal direction can cause visible artifacts on the surface. In other words, visual distortion is more influenced by the coding of the normal component than the coding of the tangential components. This observation can be very useful when a lossy compression is used. In this chapter, we only consider *near-lossless* compression.

5.3 System Overview of SplitCoder

Given a sequence of triangle meshes M_f; $f = 1, ..., F$ with V vertices and F frames (meshes), we encode the first frame separately from the rest of the frames in the sequence using static mesh compression described in previous chapiter 3. Then we encode frame by frame and vertex by vertex.

Our coding needs to store only the decoded local coordinates of the vertices in the

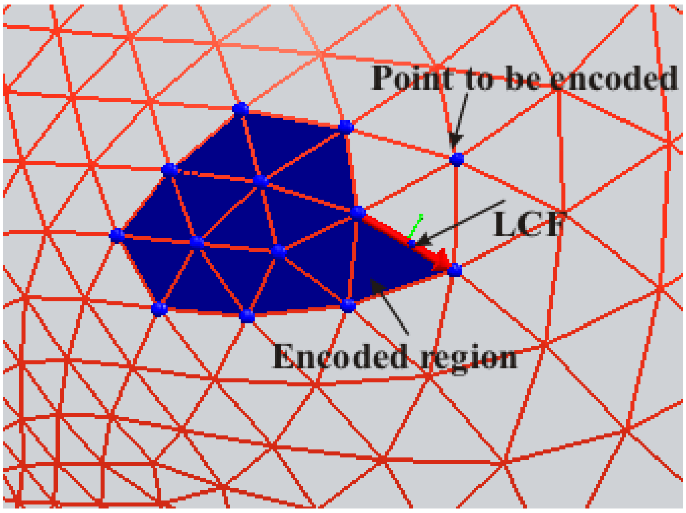


Figure 5.3 Region growing.

previous frame and the *decoded* locations in the current frames, in buffer memory.

5.3.1 Mesh Traversal

The triangular mesh of each frame is traversed in a region growing order (see Figure 5.3) as described in previous chapter 3, which is driven by the connectivity coding algorithm using a breadth-first traversal of the connectivity after the connectivity has been decoded. One initializes the growing region, which contains the so far encoded geometry, with one triangle and encodes the incident vertex locations in uncompressed form. The three edges of the initial triangle are pushed onto a FiFo. The traversal loop pops the current first edge from the FiFo and defines it as the so called gate, at which the region grows. The gate is incident to at least one triangle in the growing region. The other incident triangle is added to the growing region if it is not already part of it and new potential gate edges are pushed onto the FiFo. Every time a new vertex is encountered during the traversal, one predicts its location from the so far encoded geometry in the previous and the current frames and only encodes the delta vectors between the predicted and the original locations.

5.3.2 Geometry Coding

For the first three vertex positions in each mesh component have no local predictor. Therefore, we predict them from their position in the previously decoded frame in world space (delta coding), we quantize and encode delta vectors and reconstruct the three vertex positions to then encode all following vertices in local space.

First, we construct the local coordinate frame by splitting the coordinates into two tangential components and a normal component represented by a bending angle (see Figure 5.2). In the second step, we transform the original vertex into the local coordinates. Next, we predict the two tangential components from their coordinates in the previous frame and compute, quantize, and encode the delta vector with an arithmetic coder. We simulate the decoding process during the encoding to make sure that we use exactly the same information that will be available to the decoding algorithm during encoding. Similarly, the bending angle is predicted from the bending angle in the previous frame. Again we compute the delta angle, i.e., the difference from the bending angle measured from the original point, quantize, and encode the bending delta angle. Finally, we decode the bending angle also known by the decoder, transform the local decoded coordinates back to the world system, and replace the original vertex location with the decoded one, which avoids error accumulation. Note that the decoded tangential components and bending angle stored and then may be used to encode the location of the vertex in the next frame.

5.3.2.1 Transforming World into Cylindrical Coordinates

We follow the same strategy described in chapter 3 to construct the local coordinate frame. We defined the local coordinate system on the reference triangle with origin o at the center of the gate, x_axis along the gate edge and y_axis orthogonal to x_axis in the plane of the reference triangle. As a third coordinate, we use the bending angle α between the normals of reference and spanning triangle resulting in a cylindrical coordinate system with r as radius. We also determine the z-axis orthogonal to x_axis and y_axis.

$$\begin{aligned} x &= (\mathbf{p}-\mathbf{o})\cdot\hat{\mathbf{X}} \\ y &= (\mathbf{p}-\mathbf{o})\cdot\hat{\mathbf{Y}} \\ z &= (\mathbf{p}-\mathbf{o})\cdot\hat{\mathbf{Z}} \\ r &= \sqrt{y^2+z^2} \\ \alpha &= \mathbf{atan2}(z,y) \end{aligned}$$

The transformation back to world coordinates is simply

$$\mathbf{p} = x\hat{\mathbf{X}} + rcos(\alpha)\hat{\mathbf{Y}} + rsin(\alpha)\hat{\mathbf{Z}} + \mathbf{o}$$

The results of the local space transformation are illustrated in Figures 5.4 and 6.1. Figure 5.4 illustrates the behavior of two different classes of meshes in local space.

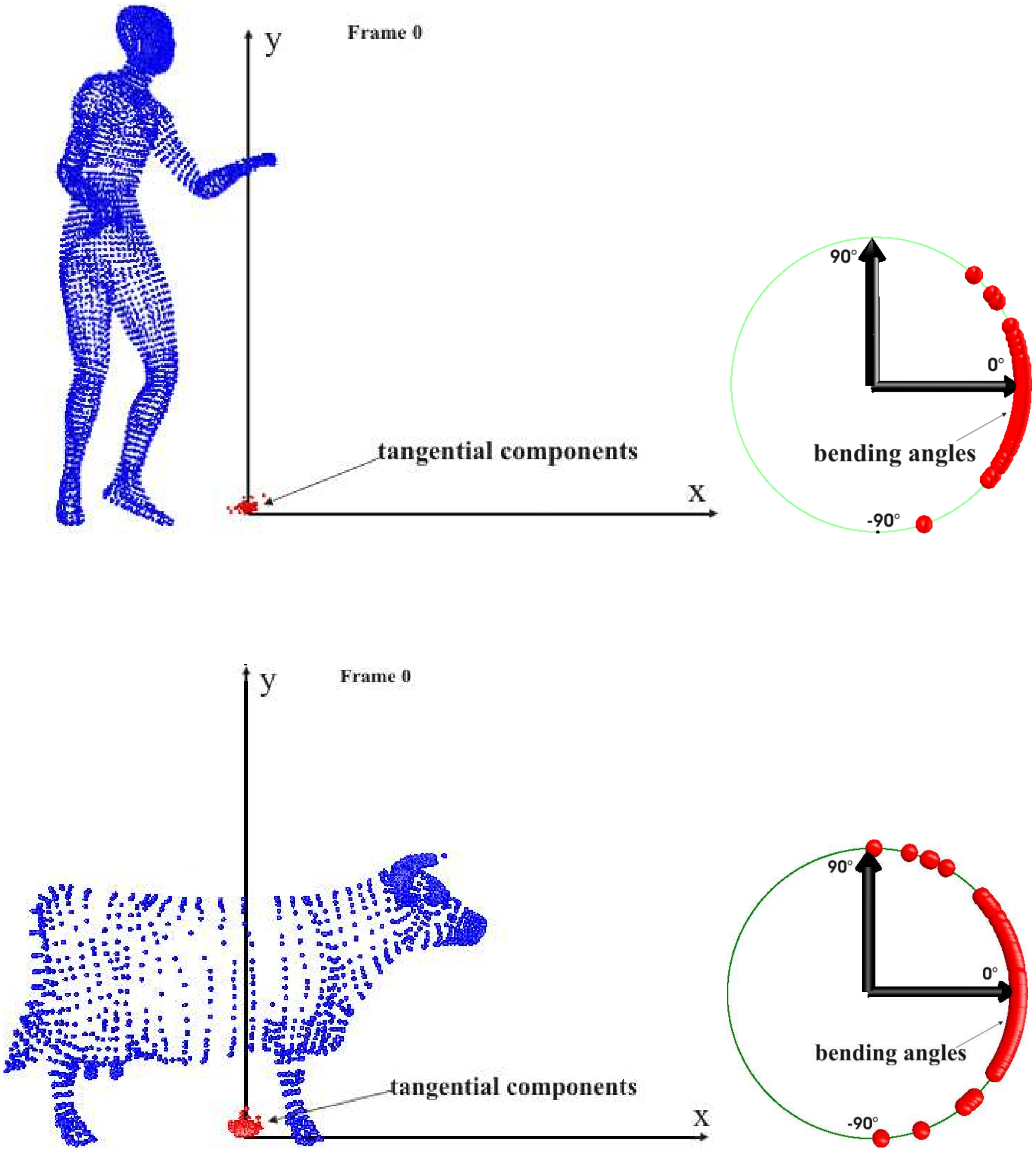


Figure 5.4 Spatial clustering of two different type of model in space model: cow (2029 vertices) and dance (7061 vertices) models. Left: the red points correspond to tangential components of the vertices of the first frame and the blue points correspond to their world coordinates x and y. Right: the corresponding bending angles.

The world coordinates x and y (blue points) of the vertices are distributed over large interval while the tangential components have a tendency to gather together to form clusters.

Figure 6.1 shows the locations of a single vertex (dance animation) over time in world and local spaces. The world x and y-coordinates and z-coordinates are represented by the blue points in Figure 6.1 (a) and (b) respectively, while the tangential and the bending angle are represented by the red points in (a) and (b) respectively.

The world coordinates are scattered across a wide area and have non linear behavior. In the local space, the tangential components are highly concentrated around one point (the red point in Figure 6.1(a)). This means that parametric information does not change over time and it is shared by all frames. Therefore, the prediction from the previous frame would be very efficient and produces delta vectors that are very small or even zero. The bending angles cluster around few points (the red points in Figure 6.1(b)). Their clustering behavior is less pronounced than the tangential components but the bending angle is still preserved between the adjacent frames and the prediction from the previous frame is very efficient. Of course that clustering degree varies depending on the model.

5.3.2.2 Tangential and Normal Components Prediction

Once the vertex location $\mathbf{p}_i^f$ is transformed into local coordinates defined as tangential components and bending angle $(p_{tan,v}^f, \alpha_v^f)$, each component is encoded separately using predictive coding.

The prediction assumes that the tangential components of the current point do not change relative to the LCF, and that the curvature at the gate will be preserved independently of the tangential components coding as mentioned above.

5.3.2.2.1 Tangential Components

For each new vertex v in the frame f, one predicts its tangential components from the decoded tangential component in the previous frame $f-1$ by:

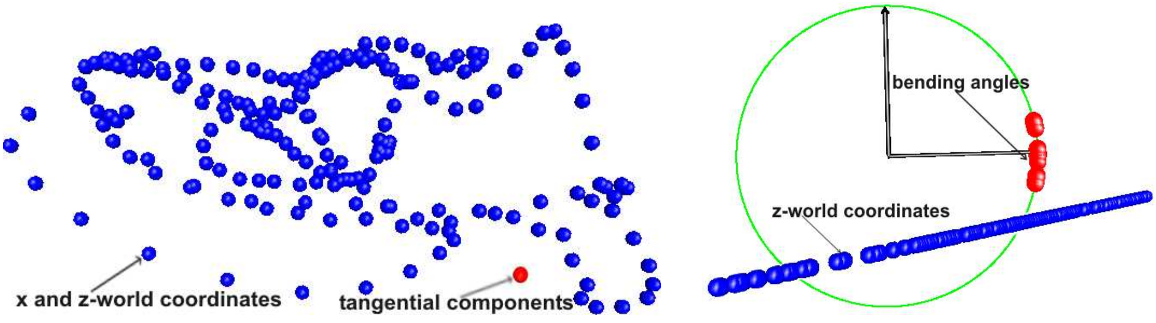
$$predict_{tan}(v, f) = \tilde{p}_{tan,v}^{f-1}$$

The delta vectors are computed:

$$\delta_v^f = p_{tan,v}^f - predict_{tan}(v, f)$$

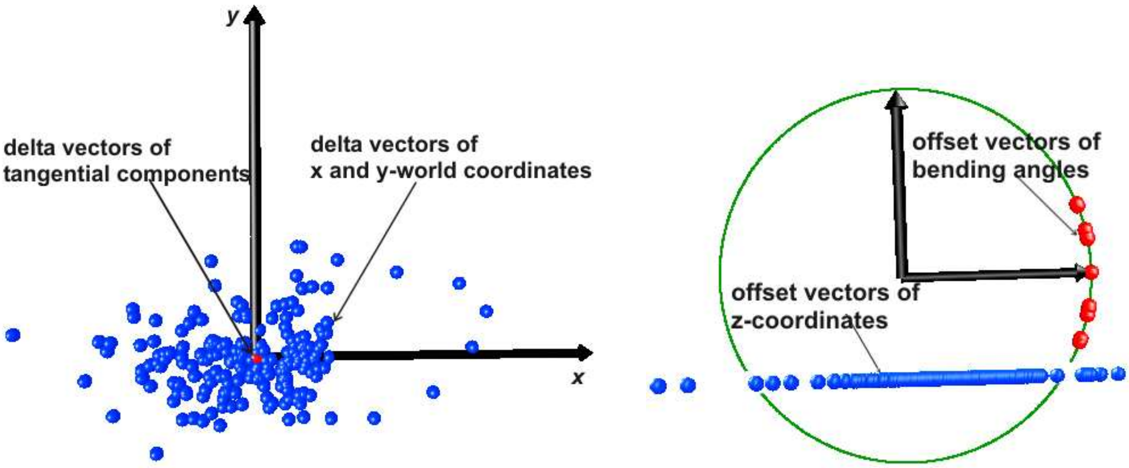
5.3.2.2.2 Normal Component

Similarly, the bending angle is predicted by:



(a) (b)

Figure 5.5 Temporal clustering of a single vertex. The blue points in (a) represent the x and y world coordinates and in (b) z world coordinates over time. The red points in (a) represent the tangential components and in (b) the bending angles over time.



(a) (b)

Figure 5.6 The resulting differences between the original and the predicted locations of a single vertex over time, when temporal prediction is used in the world and in the local space. In the world space, the vectors are represented by three coordinates: x and y-coordinates (a) and z-coordinates (b), (blue points). In the local space, the differences are represented by 2D delta vectors (a) of tangential components and delta angles (b) of the bending angle, (red points) .

$$predict_{ang}(v,f) = \tilde{\alpha}_v^{f-1}$$

The delta angles are computed:

$$\theta_v^f = \alpha_v^f - predict_{ang}(v,f)$$

The predictor we introduced for both components duplicates perfectly the parametric and the curvature information from frame to frame. It is good for meshes with smooth or/and sharp creases, and whatever the motion the vertices undergo.

Figure 5.6 illustrates the results of the delta vectors and angles of a single vertex over time. The blue points represent the differences between the original and the predicted coordinates from the location in the previous frame in the world space. The red points define the delta vectors and angles in the local space. The delta vectors tend to be zero ((0,0)), which tells us again that the parametric information does not change and that the prediction is (or near) optimal.

The predictor described in this section duplicate exactly the parametric and the curvature. Thus, we call it *Duplicata Predictor* (DP). We also implemented other predictors such us space-only predictor and space-time predictor, which will be described in chapter 8 with the experiment results.

Note that unlike the current predictive animated mesh compression techniques [56, 45, 63, 113] where the delta vectors are encoded in a world coordinate frame, here they are computed in the local coordinates and each component is encoded separately.

5.3.2.3 Binary Coding of Coordinates

For further compression, the coordinates (32 or 64 bits) are often quantized to a user specified number of bits per coordinate relative to the maximum extent of the bounding box of the model. In the case of an animation, the quantization is often performed according either to the tight axis-aligned bounding box for each frame e^f_{max} or to the largest bounding box for all frames $e_{max} = max\{e^f_{max}, f = 0, .., F\}$. In our algorithm, we consider the largest bounding box e_{max} for all previously visited frames. The tangential delta vector is then quantized to a user specified number of bits q:

$$\begin{aligned} \bar{x} &= \lfloor x/e_{max} \cdot 2^q + 1/2 \rfloor \\ \bar{y} &= \lfloor y/e_{max} \cdot 2^q + 1/2 \rfloor \end{aligned}$$

For the angular component, one has to consider the radius given by the y coordinate of the cylindrical coordinate system. Computing the arc length yields

$$\bar{\theta} = \lfloor y\theta/e_{max} \cdot 2^q + 1/2 \rfloor$$

At reconstruction, the tangential components are first decoded, since y needs to be known before α can be decoded.

as illustrated in figure 5.6 the delta vectors tend to be smaller than delta angles. Thereby, the entropy of the delta vectors will be lower than that of the delta angles. For entropy coding, it is more advantageous beneficial to encode the data of lower and higher entropy separately. Therefore, we encode the resulting signed integer values of delta vectors and angles separately with an adaptive arithmetic coder [123], using different coding contexts.

5.3.3 Geometry Decoding

The decoding algorithm uses the same traversal of the connectivity. First we decode the first frame and the locations of the first three vertices of each component. Then for each new vertex in the current frame we do the following: we first build the local coordinate system, we predict the tangential components, we undo the quantization of tangential delta vectors and we compute tangential components. Then for the normal component, we predict the bending angle, we undo the quantization of delta angle and compute bending angle. Finally, we transform the local coordinates back to world coordinates.

5.4 summary

We have presented a new compression scheme SplitCoder for animated meshes with constant connectivity. Our coding traverses the triangular mesh of each frame in a region growing order. Every time a new vertex is encountered during the traversal, we split its position into its tangential and normal components. Then, we encode each component separately using prediction and quantization coding.

We showed the coordinates in tangential and normal spaces exhibit high temporal clustering behavior that is well-suitable for temporal prediction in model space rather than in the world space. For both components, we implemented different predictors: space-only predictor, time-only predictor and space-time predictor. We will see in the experiment results that time-only predictor (relative to the local space) out performs the other predictors.

Unlike the current predictive animated mesh compression techniques where the delta vectors are encoded in world coordinate frame, here they are computed in the local coordinates and each component is encoded separately.

This approach exploits the coherence frame by frame and vertex by vertex, the next chapter will propose an new approach that exploits the coherence over all frames once and represents the set of vertices by very few components and coefficients.

.

CHAPTER 6

Motion based PCA Compression

This chapter aims at compressing animated object using principal component analysis. The approach first segment a mesh into several segments or clusters using region growing based algorithm and motion based clustering described in previous chapter. The goal behind the segmentation is to gather the vertices which have similar motion. Each set of vertices is then efficiently encoded using the PCA in the local coordinate frame. We also introduce the rate distortion optimization for PCA coding. Our main objective is to achieve an optimal tradeoff between the bitrate and the quality of the reconstructed animations.

6.1 Introduction

In previous chapter we presented near-lossless compression based on predictive coding. The coding of the geometry is dictated by the connectivity. The coding is classified with the local compression techniques which exploit the coherence frame by frame. This chapter present a new technique based on PCA. This technique is lossy compression and belongs to the global techniques that exploit the coherence over all frame once.

The advantage of using PCA is that it captures the linear correlations present in the datasets. The set of vertices can be represented by very few components and coefficients depending on the user's desired visual quality. The PCA is a good compressor for rigid motion and provides a more compact representation for temporally-invariant meshes. In many applications, however, animated meshes exhibit highly nonlinear behavior, which is globally difficult to capture using standard PCA. Locally, the neighboring vertices have a strong tendency to behave and to move in a similar way. The nonlinear behavior can therefore be described in a linear fashion by grouping the vertices of similar motion into clusters or by segmenting the mesh into meaningful parts. Then PCA is performed in each group. The process to construct this representation is called Local Principal Component Analysis (LPCA).

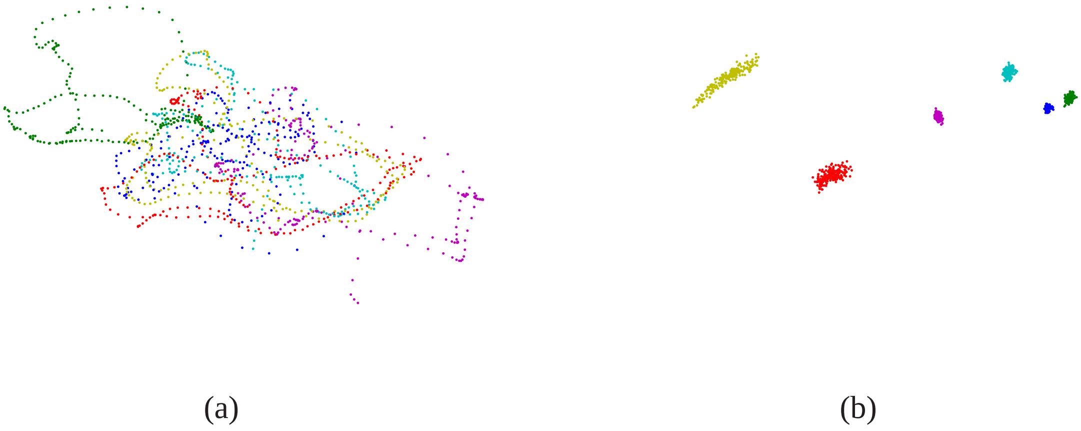
On the other hand, introducing a local coordinate frame (*LCF*) in each cluster may lead to extra clustering of the coordinates before performing the PCA. If the segmentation or clustering process is efficient then it would be highly probable that these coordinates change very slightly relative to the coordinate frame of their cluster. Of course, the number of clusters/segments will also affect the compression. If the number of clusters is very small, then a cluster might contain vertices that have different behaviors. To overcome this problem one might possible improve on the present approach by automatizing the selection of the number of clusters.

Figure 6.1 demonstrates the idea of using local coordinate systems. Figure 6.1 (a) shows the path of six points of a dance animation in the world coordinate system. Note the highly nonlinear behavior of the trajectories. Figure 6.1 (b) shows the path of the points using a local coordinate system. Note the relative small changes and the tendency of the trajectory of individual points to cluster. In previous chapter, we used for each new vertex one local coordinates frame constructed upon previously traversed frame. Here, we construct one local coordinate frame for each group of vertices. Thus, if we use clustering based segmentation, we will need the connectivity information only once at reconstruction of the LCF for each segment or cluster.

In our approach, we perform a PCA on the local coordinate system rather than the world coordinates. The advantage of combining PCA with the *LCF* is now obvious: if the motion of a group of vertices is rigid in the world coordinates, the positions of the vertices are slightly invariant relative to their *LCF*. Therefore, performing a PCA in these invariant groups of vertices leads to a more compact representation than the original data, and a large number of PCA coefficients are close to zero. In order to achieve an optimal tradeoff between the bitrate and the quality we have introduced the rate distortion optimization for PCA based on an incremental computation of the convex hull [121]

6.2 Overview

For animated mesh compression, we present a new technique based on the local PCA. The basic idea is to aggregate the vertices of similar trajectories using region growing



(a) (b)

Figure 6.1 The position of six different vertices over time (illustrated with different colors) are represented with global coordinates (left) and local coordinates system (right).

algorithm or a clustering approach, then transform the original positions of the vertices into the local coordinate frame of their segment/cluster.

This automatically "transforms" the nonlinear behavior of the original vertices into the clustering behavior which is very well compressible. The vertex positions will tend to cluster around the same position over time (see Fig. 6.2). Thus, the segments themselves are almost invariant to any deformation. A PCA is then performed on each segment such that the (local) vertex coordinates are transformed into another basis which allows for very efficient compression. An error accumulation scheme ensures that the decompression does not introduce severe artifacts.

Our clustering process (eventually the region growing approach), produces clusters of different sizes. If one chooses a fixed number of basis vectors for all clusters, then there may be too few eigenvectors to recover the clustered vertices at a desired accuracy and eventually too many eigenvectors for other clusters which we call *underfitting* and *overfitting*, respectively. Moreover, the number of bits needed to encode the unnecessary basis vectors in *overfitting* cases may be better allocated for other clusters in *underfitting* cases. Therefore the selection of the best number of basis vectors to be extracted from animation data is necessary to properly recover the original data of each cluster with a certain accuracy. We introduce a rate distortion optimization that trades off between rate and the total distortion. To our knowledge the *combination of local coordinates and PCA with the optimization process* has never been performed before. We call our approach Relative Local Principal Component Analysis (RLPCA) compression. We use the term *Relative* as the LPCA is performed in local coordinates.

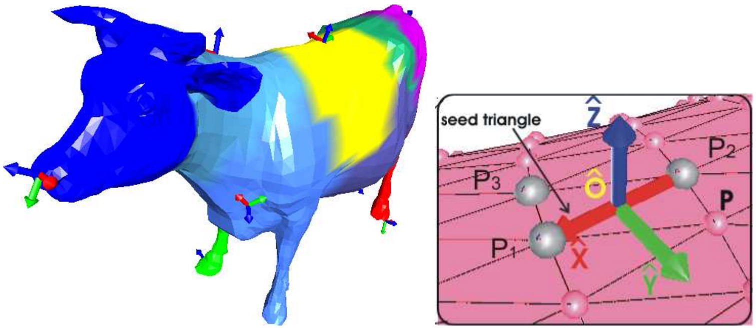


Figure 6.2 Illustration of the local coordinate frame

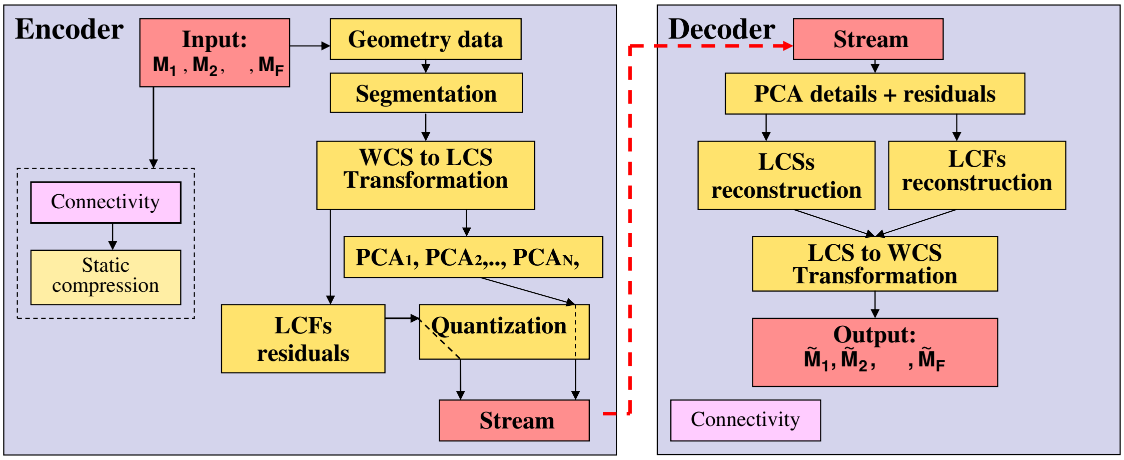


Figure 6.3 Compression and decompression pipeline.

As the connectivity of the animated meshes remains constant over time, we encode the connectivity once.

6.3 Compression pipeline

In this section, we describe in detail the core of our compression algorithm for the motion of vertices of animated triangle meshes. An overview of compression and decompression pipeline is illustrated in Figure 6.3.

Given a sequence of triangle meshes $M_f, f = 1, .., F$ of constant connectivity with V vertices and F frames (meshes), we first group the mesh vertices into N segments, where each segment contains $V_i, i = 1, .., N$, vertices.

6.3.1 Segmentation

Segmentation is an important step in our algorithm. Indeed, the compression performance depends on the segmentation into correct group, meaning that the better the vertices can be partitioned into almost rigid parts the better PCA based compression will be.

Basically, two simple and fast segmentation algorithms described in chapter 4 were used:

- Region Growing: This approach assumes that all meshes have the same connectivity. The basic idea is to grow regions starting from several seed points. The regions grow uniformly around the set of selected seed points by first traversing the closest neighboring vertices over time until all vertices of the mesh are visited.

- Clustering: The basic idea of this algorithm is to transform the original vertex coordinates into several *LCFs* defined by seed triangles. One *LCF* (one seed triangle) is associated with each cluster. Then the clustering is obtained by assigning the vertices to the cluster where they have minimal local coordinates variation across the F frames. Minimal coordinate variation means that the vertex and the LCF have almost similar motion. The clustering is also developed in adaptive way but the process becomes more slower.

6.3.2 Transforming Vertex Positions into the Local Space

Expressing the vertex locations in a LCF is an near optimal way of exhibiting clustering behavior. It makes the clusters quite invariant over time to any rotation and/or translation. This representation can be very compressible with the PCA. This is one of the key features of this new algorithm.

After segmentation, the world coordinates of the vertices of each segment are transformed into local coordinate frame of their cluster as described in chapter 4. In the case of the clustering approach, we transformed the vertex coordinates into LCFs to find out its true corresponding cluster. To avoid a second transformation into LCFs during compression, we store the local coordinates once during the clustering for further compression.

6.3.3 Compression of Local Coordinate Frames

Once the mesh vertices are clustered, their coordinate systems need to be encoded using PCA. In order to be able to transform back to the world coordinates during the decoding step, we also have to encode the world coordinate of the points of seed triangles (used to construct the transformations). The affine transformation should then be correctly computed (at decoding) without loss of information. We propose to encode the seed triangle points separately with delta encoding.

Given the sequence of the seed triangle points $(\mathbf{p}_1^{i,f}, \mathbf{p}_2^{i,f}, \mathbf{p}_3^{i,f})$, we first encode their world coordinates in the first frame. Then, the differences between each two adjacent frames in the sequence are computed. To avoid error accumulation during animation, these residuals are computed between the coordinates of the point $\mathbf{p}_j^{i,f}$ in the current frame and their recovered coordinates $\tilde{\mathbf{p}}_j^{i,f-1}$ in the previous frame:

$$\delta_j^{i,f} = \mathbf{p}_j^{i,f} - \tilde{\mathbf{p}}_j^{i,f-1}, \quad (j = 1, 2, 3)$$

where $i = 1, ..., N$ and $f = 1, ..., F$.

6.3.4 Principal Component Analysis

We adopt the PCA to compress the resulting local coordinate systems of each segment. The objective of the PCA is to reduce the dimensionality of a dataset. It determines linear combinations of the original datasets and represents them in an orthogonal basis.

For a sequence of F frames of $3V$ dimension each, PCA produces a reduced number $L \ll F$ of principal components that represent the original datasets.

We now consider how a cluster evolves over the frames of the animation. Let G_i^f be the i-th cluster in the f-th frame, $i = 1, ..., N$ and $f = 1, ..., F$. A single cluster $\mathbf{G}_i$ thus consists of F clusters (one for each frame) $\mathbf{G}_i = \{G_i^1, G_i^2, ..., G_i^F\}$ where G_i^f represents the vector with the geometry of the cluster i in frame f

$$G_i^f = (\mathbf{q}_{i,4}^f, \mathbf{q}_{i,5}^f, .., \mathbf{q}_{i,V_i}^f)^t,$$

whose elements are the local coordinates of corresponding vertices (except the coordinate of the seed triangle). All these vectors $\mathbf{G}_i^f$ have the same length $3(V_i - 3)$, and construct a geometric matrix $\mathbf{A}_i$ with $3V_i - 9$ rows and F columns (V_i is the number of vertices of the cluster $\mathcal{C}_i$).

$$\mathbf{A}_i = \left[\begin{array}{c} G_i^1 G_i^2 ... G_i^F \end{array} \right]$$

A singular value decomposition on $\mathbf{A}_i$ is

$$\mathbf{A}_i = \mathbf{U}_i \mathbf{D}_i \mathbf{V}_i^t$$

where $\mathbf{U}_i$ is a $(3V_i - 9) \times F$ column-orthogonal matrix that forms an orthogonal basis and contains the eigenvectors of the $\mathbf{A_i}\mathbf{A_i}^t$. $\mathbf{D}_i$ is a diagonal matrix whose nonzero elements represent the singular values and are sorted in decreasing order. Thus $\mathbf{D}_i = diag\{\lambda_1, \lambda_2, ..., \lambda_F\}$. $\mathbf{V}$ is a $F \times F$ orthogonal matrix.

To reduce the datasets, we pick only the first L eigenvectors (L is a user specified number). So, $\mathbf{U}_i^{'} = \{\mathbf{u}_{i,l}, l = 1, ..., L\}$ contains the most important principal components $\mathbf{u_i}$ that correspond to the largest eigenvalues $\lambda_1, ..., \lambda_L$. Then each cluster G_i^f is projected into the new basis $\mathbf{U}_i^{'}$ to get a new matrix of coefficients $\mathbf{C}_i^{'}$ of size $L \times F$.

$$\mathbf{C}_i^{'} = \mathbf{U}_i^{'t}\mathbf{A}_i$$

After performing the PCA for all N clusters $\mathcal{C}_i$, we get N new sets $\{\mathbf{U}_1^{'}, \mathbf{U}_2^{'}, ..., \mathbf{U}_N^{'}\}$ and coefficient matrices $\{\mathbf{C}_1^{'}, \mathbf{C}_2^{'}, ..., \mathbf{C}_N^{'}\}$ with different sizes.

6.3.5 Quantization and Arithmetic Coder

For further compression, the floating-point values (32 or 64 bits) are often quantized to a user specified number of bits per coordinate relative to the maximum extend of the bounding box of the model. The quantized values are encoded with an arithmetic coder [123].

In the case of an animation, the quantization is often performed according either to the tight axis-aligned bounding box for each frame or to the largest bounding box for all frames. Since we have to encode the basis vector values and the coefficients rather than the vertex coordinates, we use two different encoding contexts. The first concerns the matrices and the second the delta vectors. The basis matrix $\mathbf{U}_i^{'}$ and the coefficient matrix $\mathbf{C}_i^{'}$ of each cluster $\mathcal{C}_i$ are truncated using a fixed number of bits q_u and q_c respectively (typically $q_u = q_c$).

We first compute the minimum and the maximum values $(u_{min,i}, u_{max,i})$, $(c_{min,i}, c_{max,i})$ of $\mathbf{U}_i^{'}$ and $\mathbf{C}_i^{'}$ respectively. Let

$$\begin{aligned} e_{max,u_i} &= u_{max,i} - u_{min,i} \\ e_{max,c_i} &= c_{max,i} - c_{min,i} \end{aligned}$$

The integer values are straightforwardly derived according to

$$\begin{aligned} u_{iq}(m,j) &= \lfloor u_i(m,j)/u_{max,i} - u_{min,i} \cdot 2^{q_u} + 1/2 \rfloor \\ c_{iq}(j,f) &= \lfloor c_i(j,f)/c_{max,i} - c_{min,i} \cdot 2^{q_c} + 1/2 \rfloor \end{aligned}$$

where $1 \leq m \leq 3V_i - 9$, $1 \leq j \leq L$ and $1 \leq f \leq F$.

The resulting signed integer values of the matrices are encoded with an adaptive arithmetic coder and sent with the extreme numbers.

For delta vectors, the coordinates are encoded according to the bounding box of each frame. Using a fixed number of bits q_Δ, the coordinates of the delta vectors are mapped into integers as follows:

$$\begin{aligned}
\delta^{i,f}_{x_q} &= \lfloor \delta^{i,f}_{x} / e_{max\Delta} \cdot 2^{q_\Delta} + 1/2 \rfloor \\
\delta^{i,f}_{y_q} &= \lfloor \delta^{i,f}_{y} / e_{max\Delta} \cdot 2^{q_\Delta} + 1/2 \rfloor \\
\delta^{i,f}_{z_q} &= \lfloor \delta^{i,f}_{z} / e_{max\Delta} \cdot 2^{q_\Delta} + 1/2 \rfloor
\end{aligned}$$

for $i = 1, 2, ...N$ and $f = 1, 2, ...F$

The resulting signed integer values are encoded separate from PCA details with an adaptive arithmetic coder.

We assume that the quantization errors of PCA details are negligible up to 12 bits quantization. Note that the total number of bits needed for storing delta vectors is very small. It ranges between 0.01 and 1 bit per vertex per frame when the quantization ranges between 12 and 16 bits depending on the number of eigenvectors, the level of quantization, and the number of clusters.

6.4 Decompression

Figure 6.3 illustrates the decoding process. After receiving the sequences of the PCA details and the delta vectors (or residuals), we decode and undo quantization of delta vectors, we reconstruct the points of the seed triangles of each cluster in each frame, then reconstruct the LCFs. In the second stage, we undo the quantization of all basis vector values and coefficients, we reconstruct the local coordinates of all vertices in each cluster and transform them back to world coordinates. Finally, we collect all clusters to reconstruct the sequence of meshes.

6.5 Rate-Distortion Optimization for PCA based Coding

Our clustering process (eventually the region growing approach), produces clusters of different sizes. If one chooses a fixed number of basis vectors for all clusters, then there may be too few eigenvectors to recover the clustered vertices at a desired accuracy and eventually too many eigenvectors for other clusters which we call *underfitting* and

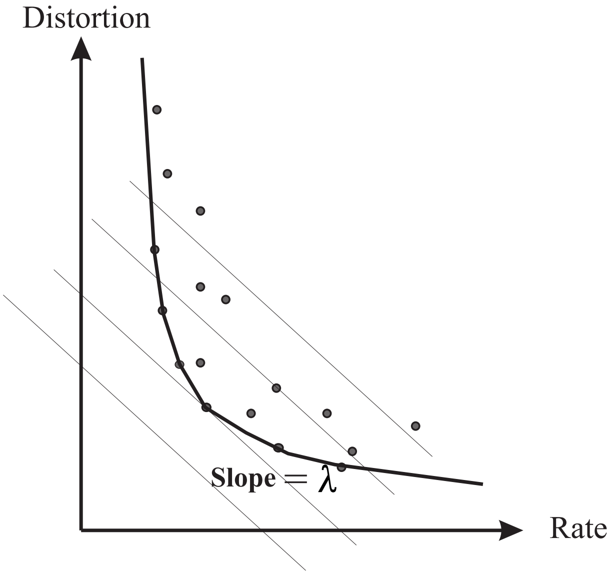


Figure 6.4 Lagrangian optimization

overfitting, respectively). Moreover, the number of bits needed to encode the unnecessary basis vectors in *overfitting* cases may be better allocated for other clusters in *underfitting* cases. Therefore the selection of the best number of basis vectors to be extracted from animation data is necessary to properly recover the original data of each cluster with a certain accuracy. We introduce a rate distortion optimization that trades off between rate and the total distortion.

6.5.1 Description of the Problem

In LPCA-based techniques often PCA is performed using a fixed number of components per cluster, neglecting the fact that whole mesh sequences are often not rigid and the different parts can have different behavior (i.e., their motion is not similar). Thus, using a fixed number of components per cluster may result in an insufficient number to represent a given cluster at the desired accuracy while having too many for the representation of other clusters.

To improve the PCA based compression and avoid this overfitting and underfitting, we introduce the Rate-Distortion Optimization (RDO) which is also known as *bit allocation*. The objective is to find the best tradeoff between the bitrate and the distortion of coordinates of the vertices.

6.5.2 Problem Statement

Given N clusters $\mathbf{G}_i$, $i = 1, ..., N$ that we have to encode separately, and a set of eigenvectors $\mathcal{I} = \{l_0, l_1, ..., l_L\}$. For each cluster $\mathbf{G}_i$, let $(R_i^{l_i}, D_i^{l_i},)$ denote the rate-distortion (RD) point for each number $l_i \in \mathcal{I}$, (typically $l_i = 1, ..., 40$ components). The rate $R_i^{l_i}$ represents the number of bits required to encode the basis vector values and the coefficients. The distortion $D_i^{l_i}$ is the root square error between the original and the reconstructed coordinates of all vertices in the cluster. (Thus we can define for each cluster a set of rate distortion points $\mathcal{P}_i = \{(R_i^{l_i}, D_i^{l_i}) : l_i \in \mathcal{I}\}$.

Let R_{target} be the given total bit rate for all clusters. Then the optimization problem is to find the optimal number of components l_i for the cluster i, $(i = 1, ..., N)$ that minimize the overall distortion:

$$D = \sum_{i=1}^{N} D_i^{l_i} \tag{6.1}$$

subject to the constraint

$$\sum_{i=1}^{N} R_i^{l_i} \leq R_{target}. \tag{6.2}$$

This constrained optimization problem can be reformulated into an unconstrained problem by using the Lagrangian multiplier approach:

$$min \sum_{i=1}^{N} D_i^{l_i} + \lambda \cdot \sum_{i=1}^{N} R_i^{l_i}, \quad l_i \in \mathcal{I} \tag{6.3}$$

where λ is Lagrange multiplier.

The problem of Lagrange multiplier is to find a value of λ_o, that yields the least distortion under target rate constraints. The bisection search can be applied over all possible values of λ to find the suitable value. For a constant λ the minimization can also be done for each cluster independently.

Geometrical Interpretation

Considering the set of RD points $\mathcal{P}_i$ for the cluster $\mathbf{G}_i$. The lagrange optimization can be interpreted with a series of parallel lines of constant slope λ. The optimal solution $(R_i^{l_i^*}, D_i^{l_i^*})$ is obtained when the line is tangent to the lower convex hull of the set RD points $\mathcal{P}_i$ as illustrated in figure 6.4.

Furthermore, alternatively, the optimal solution can be found by direct computation of the lower convex hulls of all set $\mathcal{P}_i$, $(i = 1, ..., N)$ 6.4.

6.5.3 Incremental Computation of the Rate-Distortion Optimization

In our compression algorithm, we introduce an R-D optimization which is based on an incremental computation of the convex hull [121].

For simplicity, and since the number of bits increases with the size of the basis vectors, we define the rate R as the number of basis vectors rather than the number of bits. Briefly, we define the optimization algorithm in the following:

1. For each cluster $\mathcal{C}_i$ we compute:
   - The number of components l_i that corresponds to the smallest rate;
   - The number of components k_i that corresponds to the next rate-distortion point on the lower convex hull;
   - The slope λ_i between the points $(R_i^{l_i}, D_i^{l_i})$ and $(R_i^{k_i}, D_i^{k_i})$.
2. We compute the total rate $R_t = \sum_{i=1}^{N} R_i^{l_i}$
3. As long as ($\sum_{i=1}^{N} R_i^{l_i} \leq R_{target}$) is verified, we:
   - Select the cluster S_n whose λ_n is minimal;
   - Update R_t
   - Modify the number of components l_i of the cluster $\mathcal{C}_i$ by k_i;
   - Determine the number of components k_i that corresponds to the next rate-distortion point on the lower convex hull;
   - Compute λ_n.

6.6 Compression Parameters

The compression parameters define the desired amount of compression. In our approach, there are three parameters that govern the compression ratio:

- *The number of basis vectors/rate L*: If this number is fixed for all clusters, then the user defines it (depending on the desired accuracy). The larger this number is, the better reconstruction will be (at the expense of less compression). If the RDO is used, then we will need only to specify the amount of compression (rate) or the maximum number of basis vectors that are to be used to approximate each cluster as we do in our coding. The number of vectors in each cluster is then optimally selected such that the total rate is below the given user-specified rate (or the total number of vectors is below the given user-specified maximum number of vectors).

- *The number of clusters* N: If this number is very small, then the cluster may contain vertices of different behavior and their local coordinates will have a large variation over time. However, it is difficult to find a linear space that efficiently represents these coordinates using PCA.

- *The reconstruction error*: This error presents the deviation of the reconstructed positions from the original one. It is measured using L2-norm or the metric which we call da [102, 62]. Moreover, the metric L2-norm controls the compression during the RDO. This number should increase with a decrease in the number of clusters or the number of eigenvectors.

6.7 Conclusion

We introduced a new compression technique for the animated meshes which is based on LPCA. The mesh vertices are clustered using the motion in the *LCF*. Then, the world coordinates of each cluster are transformed into local coordinates. This step enables the algorithm to compress an animated mesh efficiently. It exploits the "local" behavior of the local coordinates. Finally, an LPCA is performed in each cluster with the rate distortion optimization. The experimental results are detailed in chapter 6, section 8.6.

This approach is simple, outperforms the SplitCoder. It also achieves similar or better results than other current existing compression techniques. We obtain a better rate distortion performance than the standard PCA [3], LPC (Linear predictive Coding) and TG [117]. This result is obvious since the animation coding based on static techniques (TG) only exploit the spatial coherence and the linear prediction coding only uses the temporal coherence. Furthermore, the standard PCA only approximates the global linearity and is less effective for nonlinear animation. For the CPCA (based on local PCA) [102] and AWC (based on wavelet) [39] algorithms, we achieve better or similar results. More results are given in section 8.6.

This method is computationally inexpensive compared to a global PCA for the full mesh and its decompression process is very fast. It is applicable to meshes and point-based models. It performs well for animations with a large number of vertices and well-suited for progressive transmission. For very long sequences, we suspect that the motion of a local coordinates also becomes complex and non-linear. Therefore, one can split the sequences into small clips and perform RLPCA.

As seen before, the algorithm considers the entire sequence of meshes and exploits the coherence globally in term of linear space. The drawback of this approach is that the compression is computationally expensive.

In order, to exploit the coherence locally and achieve very low bitrate (unlike the Split-Coder), we develop compression approach based on Predictive and Discrete Cosine Transform coders (PDCT) and encode the mesh sequence frame by frame (see next chapter). We benefit from the clustering and encode each cluster with few non-zero coefficients.

CHAPTER 7

Predictive-DCT based Compression

In chapter 5 we presented connectivity-guided predictive coding belonging to the vertex based coding approach. This chapter introduces a new approach based on predictive and spectral techniques. The algorithm is independent of the connectivity information and belongs to the cluster based prediction approach. We follow the same strategy presented in previous chapters, the animated mesh is segmented into almost rigid clusters, then the predictive and DCT coding is performed in each cluster, frame per frame, instead of PCA.

7.1 Introduction

Two algorithms have been previously proposed for animated meshes. The first algorithm exploits the coherence locally frame by frame. It is vertex based predictive coding where one vertex is processed at a time in almost lossless way. These approaches are simple, not expensive, near-lossless and well-suited for real-time applications. The drawback of these methods is that they do not support progressive transmission. The second algorithm is cluster based coding, where a set of vertices are processed at a time and encoded in a lossy way using PCA. The algorithm considers the entire sequence of sub-meshes and exploits the coherence globally. The global linear behavior of the vertices through all frames is approximated in terms of linear space. The animation sequence can be reduced to a few principal components and coefficients. The efficiency of this technique increases when the datasets are segmented or clustered, so that each group is individually encoded by PCA. This type of method supports progressive transmission. The drawback of this approach is that it is computationally expensive

This chapter presents an alternative compression algorithm based on predictive and DCT transform in the local coordinate systems. In contrast to vertex based predictive coding, the algorithm is cluster based predictive coding.

The method is inspired from video coding. We first split the animated mesh into several clusters (similar to macroblocks in video coding) using the simple and efficient clustering process developed previously. Then, we perform a prediction in the local coordinate systems. Finally, we transform the resulting delta vectors (between the predicted and the original vertex locations) of each cluster in each frame into the frequency domain using Discrete Cosine Transform.

7.2 Overview

In such sequence of meshes, neighboring vertices have a strong tendency to behave similarly and the degree of dependencies between their locations in two successive frames is very large which can be efficiently exploited using a combination of *Predictive* and *DCT* coders (PDCT).

The local coordinate system has an important property that exhibits a large clustering over time and the locations of the vertex tend to form a cluster around one position (over all frames). Regardless what kind of deformation the vertices undergo, i.e. rotation, or translation or scaling or combination of all three relations, the vertices will generally keep their positions, at least between two successive frames. This property was used in different ways and in different contexts in previous chapters: in segmentation and in PCA based compression.

The proposed technique in this chapter also uses this property to perform a predictive coder followed by DCT algorithm which transforms the resulting residuals from the spatial domain to the frequency domain.

The vertices of each cluster have small variation over a time relative to the LCF. Therefore, the location of each new vertex is well-predicted from its location in the previous frame relative to the LCF of its cluster. The difference between the original and the predicted local coordinates are then transformed into frequency domain using DCT. The resulting DCT coefficients are quantized and compressed with entropy coding. The original sequence of meshes can be reconstructed from only a few non-zero DCT coefficients without significant loss in visual quality.

Basically, the algorithm consists of four steps:

1. **Clustering process:** As described in the previous chapter, the vertices are clustered

into a given number of clusters depending on their motion in the LCFs. Indeed, the vertex should belong to the cluster where its deviation in the LCF through all frames is very small compared to the other LCFs. Thereby, the vertices displacements between two successive frames will be close to zero and the efficiency of the prediction through time increases. Moreover, the clustering will preserve the global shape when DCT coding is performed (spatially) in each cluster. Note that the clustering is used in this chapter in the concept of prediction.

In PCA based algorithm the goal of segmentation was to cluster the vertices into near rigid bodies. Relative to LCF these regions are near invariant. Thus, performing PCA in the LCF yields large number of PCA coefficients that are close to zero. Thereby, low bitrate is archived.

2. **Lossless coding of LCFs:** The locations of the vertices that contribute to the construction of the LCF of each cluster should be losslessly encoded. In order to ensure that the decoder could use the same LCF, we decode and reconstruct the LCF to be used during the compression of the remaining vertices.

3. **Predictive coding:** This step allows the reduction of space-time redundancy. It is performed on the local coordinates rather than the world coordinates, which makes the coding more efficient. It produces very small prediction errors. The powerfulness of the predictor strongly relies on the clustering process. If the vertex is associated with a LCF whose motion is not similar to its motion then the local coordinates of the vertex will have a large variation over all frames and the prediction will produce large delta vectors. In chapter 5, we presented an alternative connectivity-guided predictive algorithm, this algorithm processes one vertex at a time. For each vertex one LCF is defined upon a previously traversed triangle and its coordinates are split into two components: tangential and normal components. In contrast, in this chapter we proposed predictive coding that requires one LCF per cluster.

4. **Transform-based coding or DCT:** For further compression, the coordinates of delta vectors are represented as 1D signals then transformed into frequency domain using DCT, producing uncorrelated coefficients. These coefficients are more compressible with the entropy coding than delta vectors. Moreover, many coefficients of low values can be zeroed without significant loss in visual quality.

To avoid error accumulation that may occur, we simulate the decoding process during encoding to make sure that during the encoding, we use exactly the same information

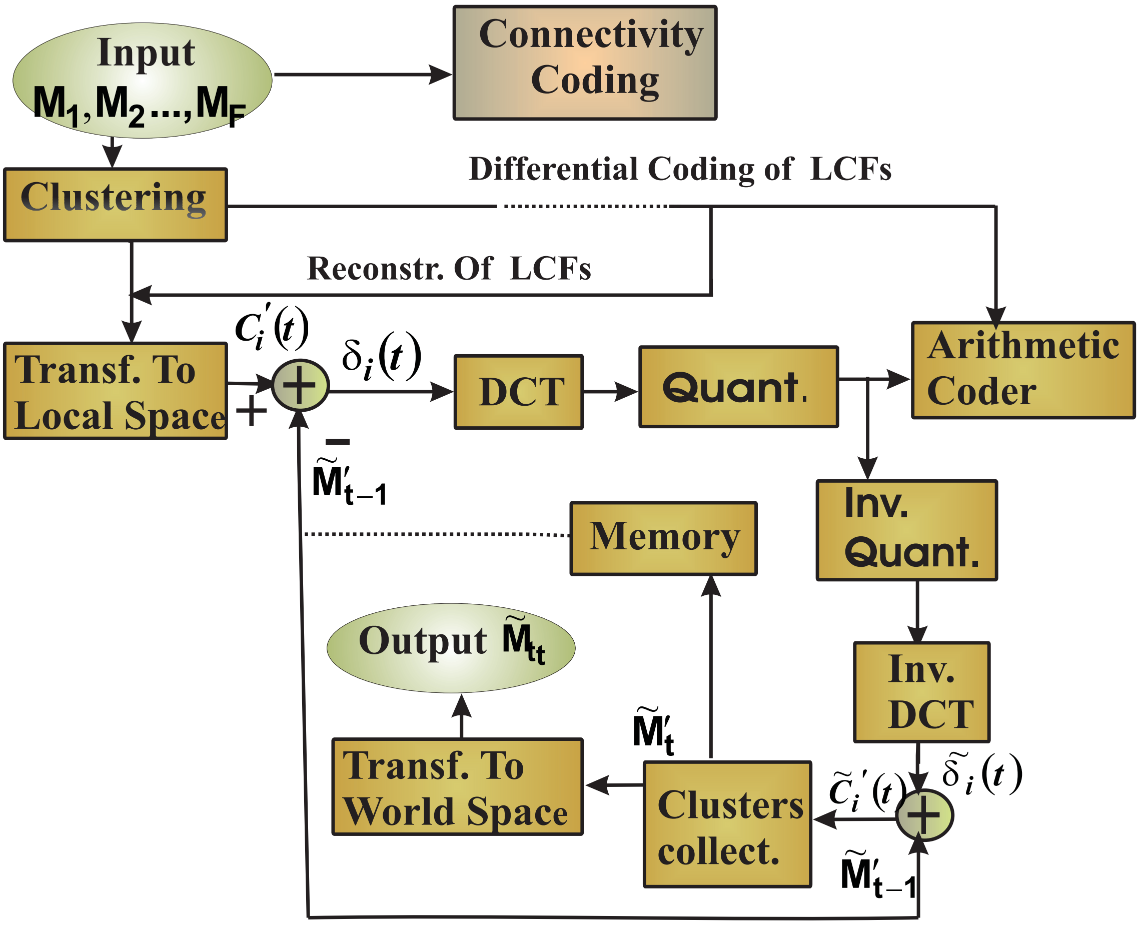


Figure 7.1 Overview of the compression pipeline

available to the decoding algorithm. After the compression of each frame, we should substitute the original vertex locations by the decoded locations.

7.3 Compression Pipeline

Given a sequence of triangle meshes $M_f, f = 1, ..., F$ with V vertices and F frames, we encode the first frame separately from the rest of the frames in the sequence using static mesh compression presented in chapter 3.

An overview of the whole compression pipeline is illustrated in Figure 7.1.

7.3.1 Local Coordinate Frames

We follow the same strategy described in the chapter 4 section 4.5.2 to select seed triangles and to construct the local coordinate frames.

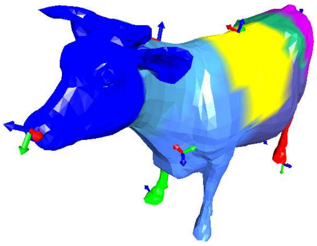


Figure 7.2 Illustration of the local coordinate frames assigned to the clusters

7.3.1.1 Seed Triangles Selection

The first step in our algorithm is to find N seed triangles upon which we construct the LCFs as described in section 4.4.2 in chapter 4, using the far distance approach [126]. Then, we associate with each seed one of its incident triangles and call this triangle the seed triangle. We denote the three vertices of seed triangle of k-th cluster in the f-th frame as $(\mathbf{p}^f_{k,1}, \mathbf{p}^f_{k,2}, \mathbf{p}^f_{k,3})$

7.3.1.2 Local Coordinate Frame Construction

Each cluster is initialized with the three vertices of the seed triangle. Each cluster $\mathcal{C}_k$ has its own LCF defined on the seed triangle $(\mathbf{p_1}, \mathbf{p_2}, \mathbf{p_3})$ as illustrated in Figure 7.2 (and as described in the previous chapters).

The transformation of a point $\mathbf{p}$ to its local coordinate system $\mathbf{q}$ can be accomplished by an affine transformation with a translation $\mathbf{o}$ and a linear transformation $\mathbf{T}$ ($\mathbf{T}$ is an orthonormal matrix):

$$\mathbf{q} \quad = \quad \mathbf{T}(\mathbf{p} - \mathbf{o})$$

For each frame f $(1 \leq f \leq F)$ and for each cluster $\mathcal{C}^f_k$ $(1 \leq k \leq N)$, we have $\{\mathbf{T}^f_k, \mathbf{o}^f_k\}$ computed from the points of the seed triangle $(\mathbf{p}^f_{k,1}, \mathbf{p}^f_{k,2}, \mathbf{p}^f_{k,3})$.

7.3.2 Motion in LCF based Clustering

The clustering process starts with several seed triangles upon which the LCFs are constructed. Then, the clustering is obtained by assigning the vertices to the seed triangle where they have minimal local coordinate deviation across the F frames according to

$$\theta_{k,i} \quad = \quad \sum_{f=1}^{F} \|\mathbf{q}^f_{k,i} - \mathbf{q}^{f-1}_{k,i}\|^2$$

$q_{k,i}$ are the local coordinates of the vertex i with cluster k and $\theta_{k,i}$ represents its total motion in the LCF associated with the cluster k.

The vertex should then belong to the cluster k for which the deviation is very small, note k_{min}:

$$k_{min} \quad := \quad argmin_{1\leq k\leq N}\{\theta_{k,i}\}$$

7.3.3 Differential Coding of LCFs

Generally, our approach first transforms the world coordinates of each vertex into local coordinate frame of its cluster. Then, it performs the compression. At reconstruction, the local coordinates are decoded then transformed back to world coordinates. A lossy compression of the vertices of the seed triangle may damage the coordinate frames at the decoding step and as a result, the transformed local coordinates will be damaged. Therefore, the LCF of each cluster should be encoded *losslessly*.

We assume that the LCFs of the first frame is already encoded. For each frame and for each new LCF, we encode the locations of their three vertices with the differential encoding. We subtract their coordinates in previously encoded frame from its current coordinates. We quantize the prediction differences, we apply the arithmetic coder to the resulting integers and we update the current locations with the decoded locations.

7.3.4 Spatial-Temporal Predictive Coding

Once the segmentation process is finished, and all LCFs are decoded (during the coding), the prediction assumes that the current point does not change relative to the LCF of its cluster. So, for each new point $\mathbf{p}_{k,i}^{f}$ in the cluster $\mathcal{C}_k^f$ of the frame f, one transforms its world coordinate into local coordinates $\mathbf{q}_{k,i}^{f}$. Then, one predicts its location from the decoded local coordinates of its location in previous frame $f-1$ by:

$$\mathbf{pred} \quad = \quad \tilde{\mathbf{q}}_{k,i}^{f-1}$$

The delta vectors are computed:

$$\delta_{k,i}^{f} \quad = \quad \mathbf{q}_{k,i}^{f} - \mathbf{pred}$$

Unlike the current predictive animated mesh compression techniques [56, 45, 63] where the delta vectors are encoded in world coordinate frame, here they are computed in the local coordinates.

7.3.5 DCT Coding

After prediction, we represent the x,y,z coordinates of the delta vectors of each cluster C_k^f as 1D separate signals of length $V_k - 3$ ($V_k - 3$ is the number of vertices in the cluster $\mathcal{C}_k$, minus the three vertices of seed triangle) and encode them with DCT coding.

For each cluster we have three signals:

$$\begin{aligned}\mathbf{X}_k^f &= \{x_{k,4}^f, x_{k,5}^f, ..., x_{k,V_k}^f\}\\ \mathbf{Y}_k^f &= \{y_{k,4}^f, y_{k,5}^f, ..., y_{k,V_k}^f\}\\ \mathbf{Z}_k^f &= \{z_{k,4}^f, z_{k,5}^f, ..., z_{k,V_k}^f\}\end{aligned}$$

where $k \in 1, ..., N$ and $f \in 1, ..., F$.

For the whole sequences, the number of signals we obtain is $N \times 3 \times F$. We transform each signal vector into the frequency domain using 1D DCT to obtain a more compact representation. Simple 1D DCT is defined as:

$$\mathcal{X}_{k,l}^f = \alpha(l) \sum_{i=4}^{V_k} x_{k,i}^f cos(\frac{\pi(l-4)(2(i-4)+1)}{2(V_k-3)})$$

for $l = 4, ..., V_k$, and $\alpha(l)$ is defined as:

$$\alpha(l) = \begin{cases} \frac{1}{\sqrt{V_k - 3}} & \text{for } l = 4 \\ \sqrt{\frac{2}{V_k - 3}} & \text{for } l \neq 4 \end{cases}$$

The inverse DCT is similarly defined as:

$$x_{k,i}^f = \sum_{l=4}^{V_k} \alpha(l) \mathcal{X}_{k,l}^f cos(\frac{\pi(l-4)(2(i-4)+1)}{2(V_k-3)})$$

where $i = 4, ..., V_k$.

After DCT transform, the majority of signal energy concentrates on the low frequencies and little on the high frequencies. Hence the high frequencies (insignificant coefficients) can be zeroed yielding a significant reduction in the overall entropy and the signal can then be represented by few high value coefficients without significant distortion. Note that the high frequencies close to zero can also be set to zero automatically using quantization module only.

In our algorithm, we arrange the DCT coefficients from high to low values to easily set the coefficients to zero from bottom to a certain number of coefficients depending on the compression rate and the desired quality.

7.3.6 Quantization and Arithmetic Coder

The low frequency coefficient (high values) correspond to the coarse details of the cluster while the high frequency coefficients (low values) correspond to the fine details. On the other hand the human eye can perceive the coarse details much more accurately than the fine details. This means that if we use a coarse quantization or set the low value coefficients to zero, the cluster will still retain an acceptable visual quality and we will obtain better compression ratios.

In this version of the algorithm, we uniformly quantize the coefficients to a user specified number of bits per coefficient. Typically, we use a number between 8 and 12 bits, depending on how many DCT coefficients are zeroed. The more coefficients that are zeroed, the more coarser the quantization is, and that better the compression will be at the expense of visual appearance. The finer details can be preserved when only a finer quantization is used and few coefficients are thrown away. For example, if 50% of coefficients have zero values then we use 10 bits quantization. If 90% we use 8 bits only.

One might possibly improve on the present quantization approach by introducing different levels of quantization in each cluster. The high frequencies can be coarsely quantized while the low frequencies can be finely quantized.

Note that, the delta vectors of the first frame are encoded using 12 bits quantization while the delta vectors of the LCFs in the whole sequence are quantized to 16 bits.

For further compression the resulting integer values are well encoded with an arithmetic coder [123].

7.3.7 Reconstruction

To reconstruct the original data cluster, we simply de-quantize the coefficients and perform the inverse DCT to find out the delta vectors and add these latter to the predicted location from the perviously decoded frame to recover the original local coordinates. Then, we transform them to world coordinates.

7.4 Conclusion

This chapter introduced a simple and efficient compression technique for dynamic 3D meshes based on predictive and DCT coding. First, the algorithm clusters the vertices into a given number of clusters depending on their motion in their LCF. Second, the location of each new vertex in the current frame, is predicted from its location in the previous frame. The effectiveness of prediction coding depends strongly on the clustering process. Indeed, if the vertices are well-clustered then the motion relative to the LCF between two successive frames tends to be zero. Third, the delta vectors are further encoded with DCT transform to reduce the code length since the entropy in frequency domain is smaller than the entropy coding of delta vectors. The resulting DCT coefficients are quantized and encoded with an arithmetic coder.

The experimental results are detailed in chapter 7, section 8.7. This algorithm is lossy techniques, computationally expensive compared with SplitCoder, requiring more computation steps for DCT and inverse DCT but it allows low bit rate. The distortion introduced by PDCT is almost less than the distortion produced by SplitCoder. At similar bitrate (see section 8.8), PDCT yields better reconstruction due to DCT based coding that may reduces the blocky artifact at low bitrate. Indeed, after DCT transform, the majority of signal energy concentrates on the low frequencies and little on the high frequencies. Hence, the high frequencies can be zeroed yielding a significant reduction in the overall entropy. The signal can then be represented by few high value coefficients without significant distortion. The gain of PDCT method over SplitCoder is up to 73%. Of course, the gain varies with compression parameters and animation sequences. Experimental results also show that our algorithm is competitive when compared to the state-of-the-art techniques. It performs better than the standard PCA [3], LPC, KG [62] and TG [117] and comes close to CPCA [102] and wavelet AWC [39] algorithms. PDCT is applicable to meshes and point-based models regardless of how the animation is generated. The drawback of the proposed approach is that it does not support progressive transmission.

CHAPTER 8

Evaluation and Comparison

In previous chapters, we have presented our new approaches, and we have detailed the theory and the design of each algorithm. This chapter presents the experimental results. We discuss and evaluate different parameters that affect compression performance, we compare the proposed techniques with other methods and we also compare between our approaches.

Note that the experimental results of segmentation methods are given in chapter 4. This chapter is restricted to the results of the compression approaches only.

8.1 Implementations

We have implemented our algorithms in c++ programming language and Matlab. We also used an object oriented framework called DaViS (Data Visualization System), a very powerful tool for evaluating new algorithms and visualization techniques. The implementation runs on a standard PC with Pentium 4 with 2.53 GHz, Nvidia Geforce 4 MX 420, 750 MB of RAM (it is used for timing results of compression algorithms) and AMD Athlon(TM) XP 3000+ 2.10 GHz, 1.00GB of RAM (it used for timing results of segmentation algorithms).

Models	vertices	Triangles	Frames
chicken	3030	5664	400
dance	7061	14118	201
dolphin	6179	12278	101
cowheavy	2904	5804	204
snake	9179	18354	134

Table 8.1 Characteristic of animation sequences used for analysis.

8.2 Input Datasets

To evaluate our approaches, we used different static and dynamic triangular meshes, used by the majority of the current existing approaches. The proposed approaches are independent of how the input datasets (particulary the animation) are generated. Indeed, we consider different animations generated in different ways.

Figure 8.1 and Figure 8.2 show the static and dynamic models, respectively, used for the evaluation. Their characteristics are reported in tables 8.1 and 8.4 respectively.

All tested 3D models have different shape and smoothness, as well as movements of different degrees of complexity in the animated case. The chicken character is distributed solely for the purpose of comparison of geometry compression techniques. The Dance animation has been created by skinning a skeleton with motion capture data. Its movement is quite smooth and its mesh is quite regular. The Cow animation consists of extreme deformations and the mesh is very irregular. The Dolphin animation is generated with sinusoidal movements and the snake animation of large body deformation.

8.3 Measurement

To show the efficiency of our schemes, we measured the number of bits per vertex for static meshes and number of bits per vertex per frame (bpvf) for animated meshes. To measure the distortion in the reconstructed animation with regard to the original animation, we used a metric *da* similar to [102, 62]. The error is defined as follows:

$$da = 100\frac{||\mathbf{M} - \hat{\mathbf{M}}||}{||\mathbf{M} - E(\mathbf{M})||}$$

where $\mathbf{M}$ is the geometric matrix $(3V \times F)$ containing the original geometry sequence. $\hat{\mathbf{M}}$ is the reconstructed geometry sequence. $E(\mathbf{M})$ is an average matrix whose columns contain the average vertex positions of all meshes over time.

We also computed the distortion per frame using the L^2norm of all reconstructed vertex positions relative to the original positions of each frame.

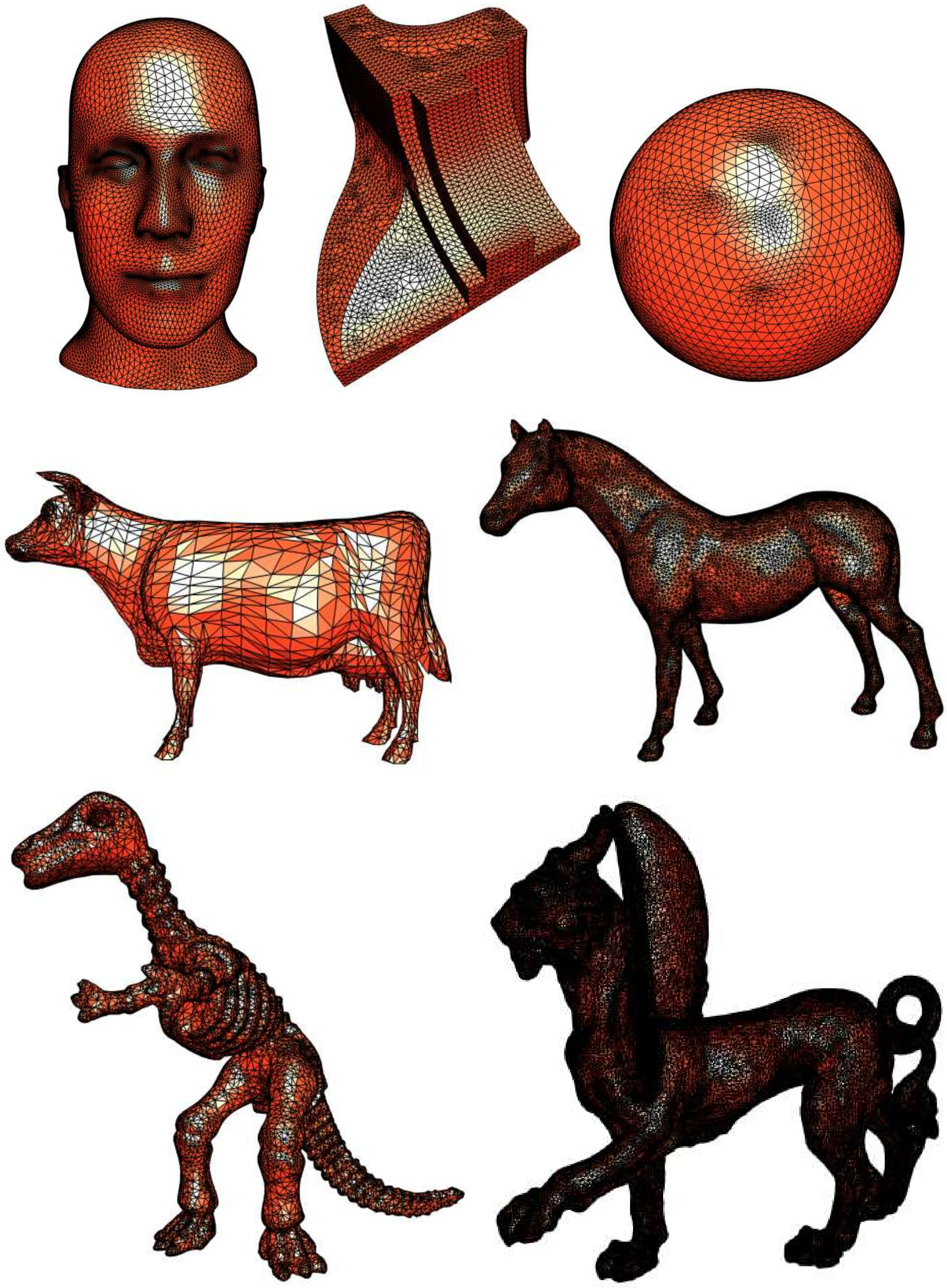


Figure 8.1 The static models used for analysis: head, fandisk, random, cow, horse, dino and feline. These models are provided us by Pierre Alliez and Lee Haeyoung.

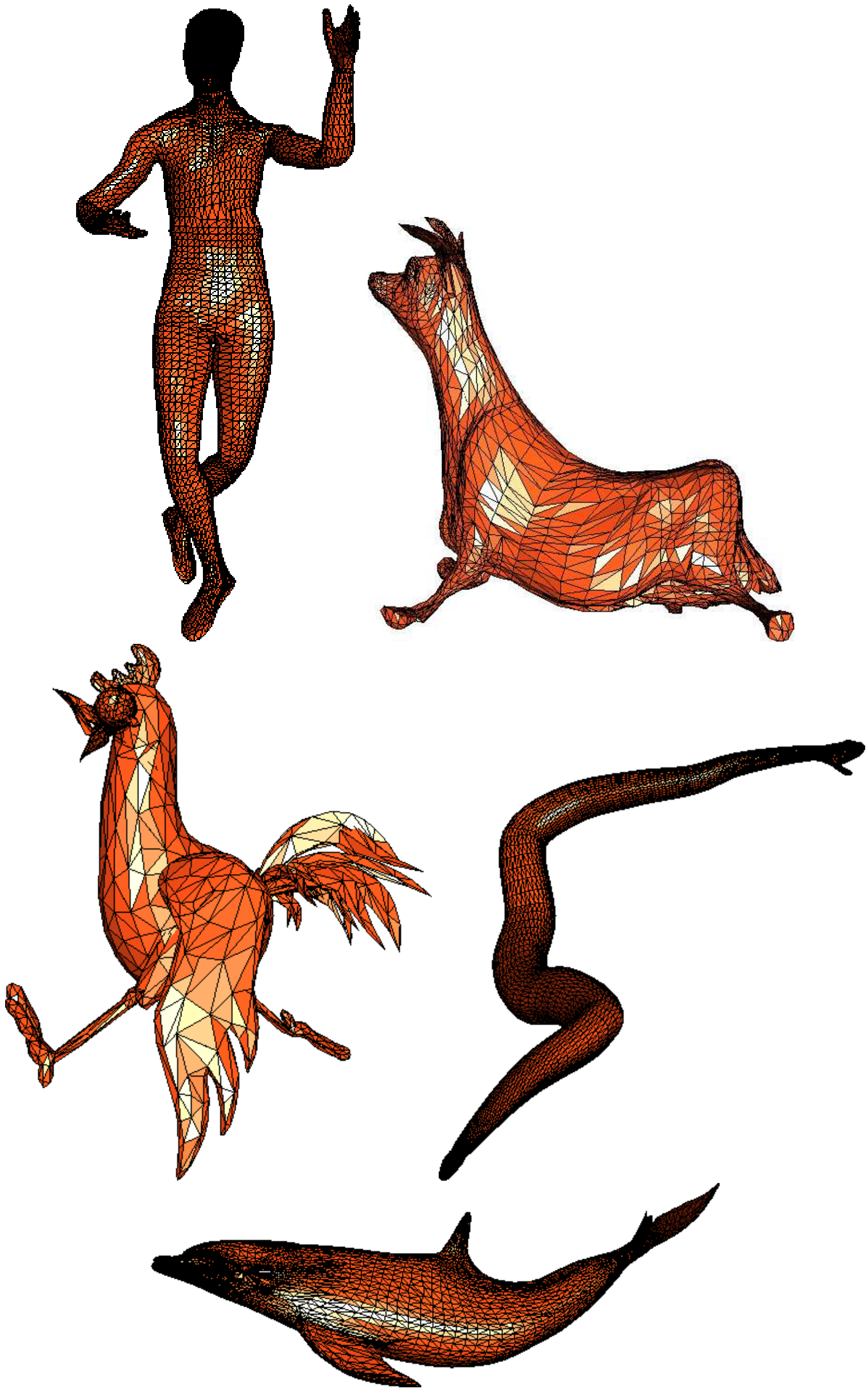


Figure 8.2 The animations used for analysis. The dance animation was created by the MIT CSAIL Graphics Lab. The chicken character is property of Microsoft Inc. and was created by Andrew Glassner, Tom McClure, Scott Benza and Mark Van Langeveld. The dolphin animation is given us by Zachi Karni. The cow and snake animations can be found in [1].

8.4 Higher Order Prediction for Static Geometry Coding

In this approach, we have developed a higher order prediction scheme for geometry compression. Instead of encoding the coordinates of the vertex, we encode its tangential components and its normal component as bending angle. The tangential components are encoded with parallelogram prediction. For the normal encoding, we first fit a polynomial surface to the previously encoded vertices in the vicinity of the current gate edge. Then, we intersect the tangential circle given by the tangential components, which are encoded in advance, with the polynomial surface yielding the prediction for the bending angle. We also fit a sphere to a small number of vertices as a fast compromise between polynomial surface fitting and simple angle prediction.

In order to see the gain of the higher order prediction scheme based on polynomial graph function fitting (explicit function), we measured the consumed bits per vertex separately for the tangential and normal components. Table 8.2 shows the results, when the parallelogram prediction is used for tangential prediction, and table 8.3 the results for the multi-way prediction.

In both tables the second columns give the number of bits consumed for the tangential components. The third columns contain the cost for the bending angle, when predicted as proposed by Touma and Gotsman [117]. In the fourth columns the result of the higher order prediction scheme can be seen and in the last columns we entered the gain in the angular component in percent. Both tables show, that there is an average gain of 20% in the coding of the bending angle independent of the tangential prediction strategy.

We summarize the experimental results in table 8.4, where we compare the total coding cost for the used models with the official implementation of the approach of Touma and Gotsman. The third column shows the cost for the connectivity in bits per vertex consumed by Touma and Gotsman's method. The fourth and fifth columns compare the total geometry cost of Touma and Gotsman's approach to ours. In the last column we finally tabulate the gain of the total cost in percent – geometry and connectivity.

On the first sight, the result for the cow in table 8.4 is surprising as there is a gain in tables 8.2 and 8.3 for our implementation of the Touma-Gotsman angle predictor. It turned out that no angle prediction yielded an over all compression result of 20.2 bits per vertex. Thus we suspect that also the official implementation of the Touma-Gotsman coder turns off angle prediction, when there is no gain.

Figure 8.5 also shows the total coding cost when we fit polynomial graph (HOPE) and implicit (HOPI) function and sphere (HOPS) as well as the total geometry cost of Touma and Gotsman's approach. In the last three columns we tabulate the gain of the total cost in percent. On can see that the polynomial graph performed better on a subset of the used

Models	x, y	α_{TG}	α_{HO}	gain(α)
horse	11.41	4.16	3.61	13.22
random	12.32	3.12	1.85	40.70
sphere	4.25	1.97	0.85	56.85
head	9.92	4.10	3.19	22.20
dino	12.53	5.65	5.00	11.50
fandisk	9.72	4.24	2.68	36.80
feline	10.68	4.04	3.71	8.17
cow	14.90	7.52	7.08	5.85
Average	10.70	4.35	3.50	24.41

Table 8.2 Geometry coding results for the parallelogram prediction: cost for tangential components x and y in bits per vertex, cost for normal component with bending angle prediction according to Touma and Gotsman (subscript TG) and according to higher order prediction (subscript HO) and the gain of our method in percent.

Models	x, y	α_{TG}	α_{HO}	gain(α)
horse	10.64	4.16	3.61	13.07
random	11.01	3.13	1.84	41.07
sphere	3.39	1.77	0.83	52.64
head	8.91	4.08	3.19	21.59
dino	12.01	5.65	4.99	11.53
fandisk	10.14	4.23	2.71	35.71
feline	9.98	4.07	3.76	7.59
cow	14.47	7.54	7.05	6.44
Average	10.07	4.32	3.50	23.71

Table 8.3 Geometry coding results for the multi-way prediction with the same columns as table 8.2.

Models	vrts	Conn.	Geom.	ours	gain
horse	19851	2.34	15.16	14.26	5.94
random	4338	0.41	15.64	12.85	17.80
sphere	10242	0.02	6.95	4.23	39.13
head	11703	0.45	12.64	12.11	4.21
dino	14070	2.39	17.40	17.01	2.24
fandisk	6475	1.08	13.82	12.86	6.93
feline	49864	2.38	14.17	13,74	3.03
cow	2904	1.88	20.38	21.52	-5.65
Average		1.37	14.52	13.57	9.20

Table 8.4 Models and final coding results : number of vertices, connectivity and geometry coding cost in bits per vertex for Touma and Gotsman's method, geometry cost for our method, total gain in percent, using 12 bits quantization, a weighting exponent of 1.3 and a maximum number of 18 gathered fit vertex.

models than the other two prediction schemes. And non of these two methods proofed to be superior on most of the cases.

In order to find out the best maximum number of vertices to be gathered, the weighting exponent and the threshold angle cosine, we plotted the number of bits consumed by the normal component for different values of these parameters for cow model. The resulting plots are illustrated in figures 8.4, 8.5and 8.3. On can see that with higher order prediction based explicit function fitting the gain becomes important with the growing of the number of vertices to be gathered to fit a surface while with implicit function, the gain is nearly constant and both methods are superior than TG's method. HOPE and HOPI show a minimal coding cost for more than 22 and 8 gathered vertices, a weighting exponent of approximately 1.37 and 2.68, and angle cosine 0.67 and 0.9, respectively.

We also plotted the number of bits per vertex consumed by the normal component for different weighting exponents over the number of gathered fit vertices where the bits per vertex were averaged over the collection of sample models illustrated in 8.1. Figure 8.6 shows a minimal coding cost for eighteen gathered vertices and a weighting exponent of approximately 1.3. Note that these values were used for the measurements in 8.3 8.4 8.2

It is quite obvious that the compression and decompression speeds with high-order prediction are significantly slower than with the simple prediction rule of Touma and Gotsman. Most of the time is spent for the gathering of the fit vertices and the computation of matrix $\mathbf{F}$ via equation 3.2. If a maximum of eighteen vertices are gathered, 72% if the coding time is consumed for gathering and 22% for the eigenvalue decomposition of $\mathbf{F}$. For the simple bending angle prediction only 5% of the higher order coding time are consumed. If we only gather 10 vertices as the second minimum in figure 8.6 suggests gathering is 58%, eigenvalue decomposition is 33% and simple coding is 9% of higher order coding time, resulting in an eleven times slower coding algorithm. In sphere fitting we fit sphere to a small number of vertices, i.e. four points. Thus the speed is much lower than the surface fitting.

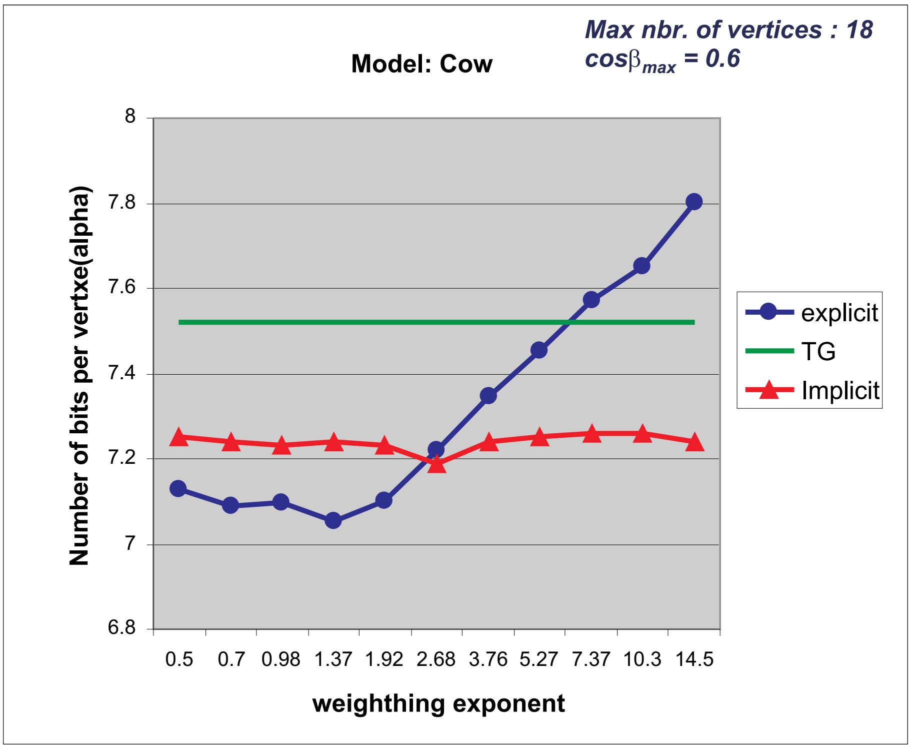


Figure 8.3 Plot of bits per vertex consumed for the normal component with different weighting exponents for a given angle cosine and maximum number of gathered.

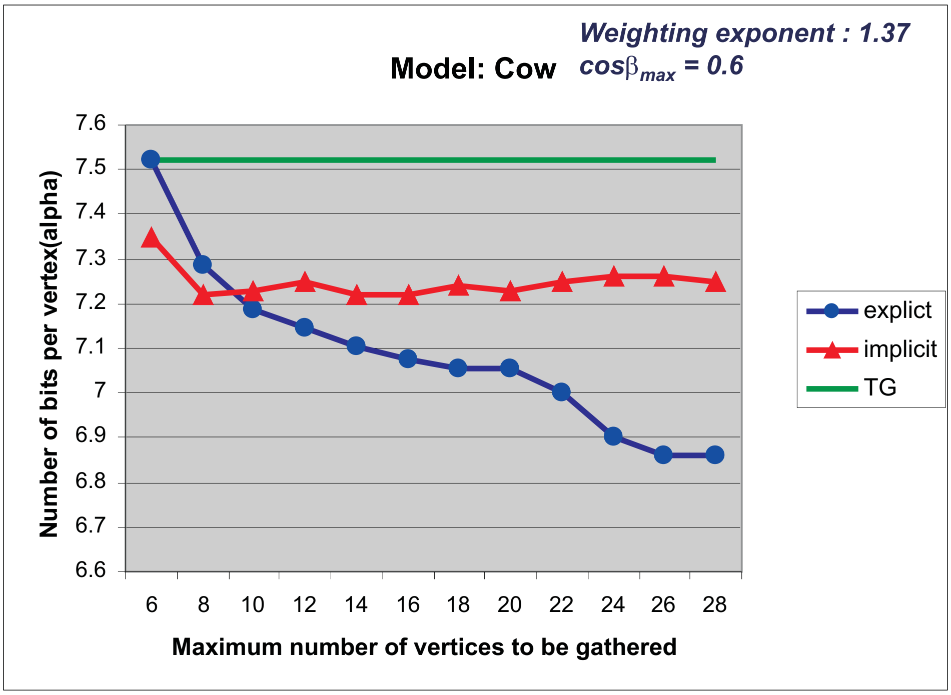


Figure 8.4 Plot of bits per vertex consumed for the normal component with different number of gathered fit vertices for a given weighting exponent and threshold angle cosine. The straight red line illustrates the performance of the angle prediction via Touma and Gotsman.

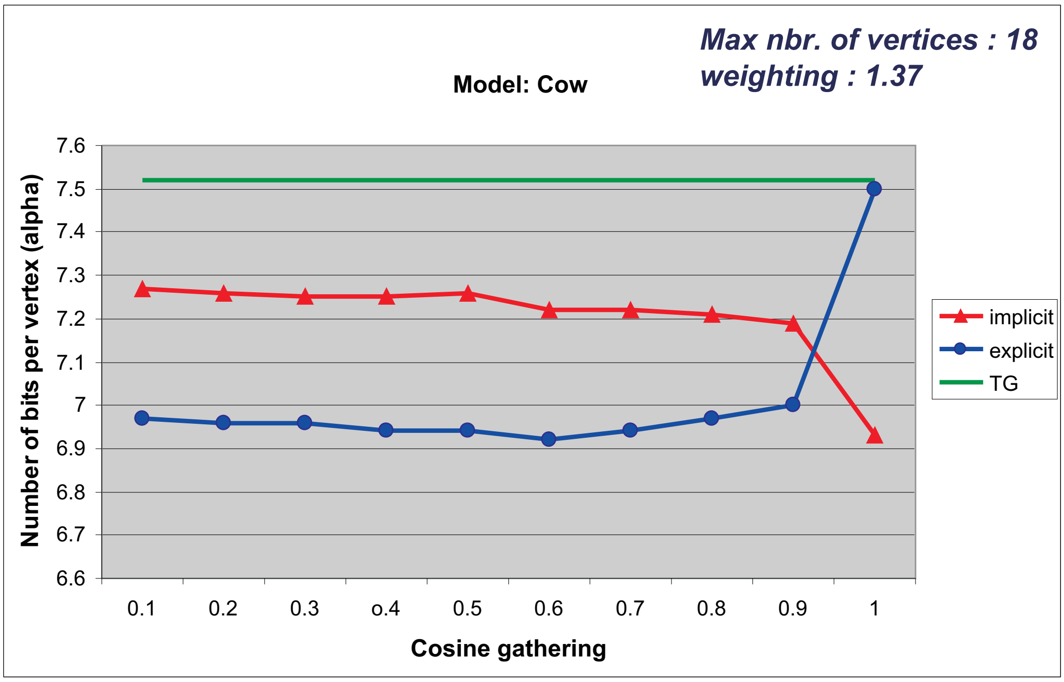


Figure 8.5 Plot of bits per vertex consumed for the normal component with different angle cosine values for a given weighting exponent and maximum number of gathered.

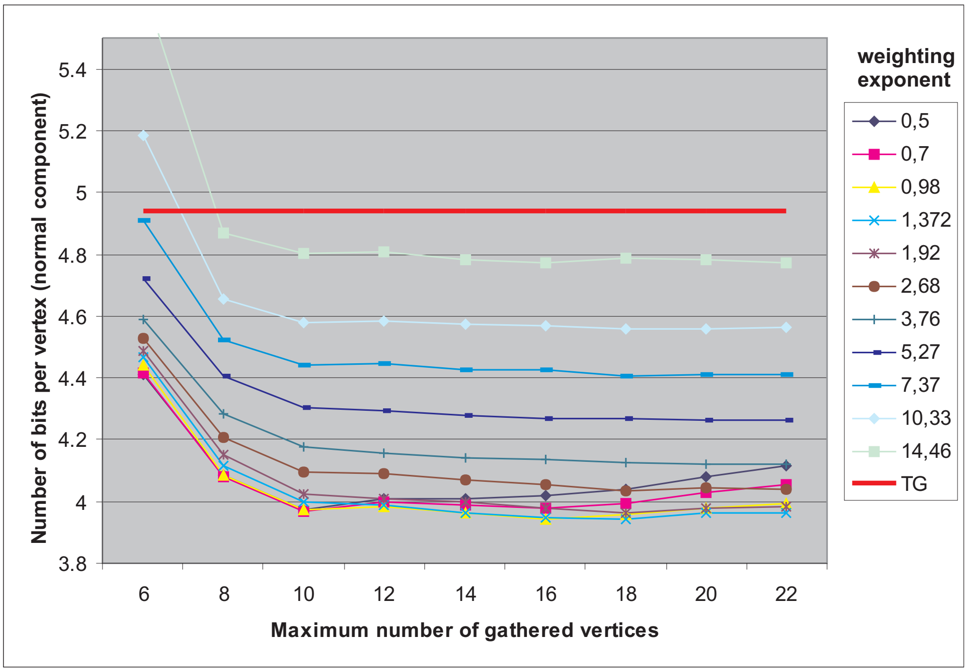


Figure 8.6 Plot of bits per vertex consumed for the normal component with different number of gathered fit vertices and different weighting exponents. The straight red line illustrates the performance of the angle prediction via Touma and Gotsman.

Models	Geom_TG	HOPE	HOPI	HOPS	Gain_E	Gain_I	Gain_S
horse	15.16	14.26	15.00	15.00	5.93	1.00	1.00
random	15.64	12.85	16.00	15.58	17.83	-2.3	0.38
sphere	6.95	4.23	5.60	5.59	39.13	19.42	19.56
head	12.64	12.11	13.64	13.57	4.19	-7.91	-7.35
dino	17.4	17.01	17.70	17.70	2.24	-1.72	-1.72
fandisk	13.82	12.86	12.5	12.91	6.94	9.55	6.58
cow	20.38	21.5	22.1	21.93	-5.49	-8.43	-7.60
Cow	20.38	21.52	22.10	21.93	-5.49	-8.43	-7.60
Average	14.57	13.54	14.64	14.61	10.11	1.38	1.56

Table 8.5 Coding results. Geometry coding cost in bits per vertex for Touma and Gotsman' method, and our methods based on graph $f(x, y)$, polynomial implicit function and sphere fitting, their total gain in percent using 12 bits quantization.

8.5 Near-lossless Predictive Coding for Dynamic Meshes

This approach, which we called SplitCoder, introduces single rate near lossless compression for animated meshes of fixed connectivity using a predictive technique. The algorithm traverses the triangular mesh of each frame in a region growing order. Every time a new vertex is encountered during the traversal, we split its position into its tangential and normal components. Then, we encode each component separately using prediction and quantization coding.

To find out the best predictive coding for the tangential and normal component, we implemented different predictors for both components.

8.5.1 Evaluation of Different Predictors

In order to demonstrate the efficiency of each predictor in terms of compression performance for each component, we measured the consumed bits per vertex separately for the tangential and normal components.

8.5.1.1 Tangential components

Space-only predictor

We implemented a space-only predictor to encode the parametric information of each frame separately. The coherence is exploited over the direct neighbors only, neglecting the temporal coherence. Two mode of prediction are evaluated:

- In parallelogram prediction mode (space(PP)), we use the formula of Touma and

Gotsman but we use the decoded vertex locations in order to avoid error accumulation.

$$pred_{paral} = p^f_{v-1} + p^f_{v-2} - p^f_{v-3}$$

- In multi-way prediction mode (space(Multi)), when a new vertex is encountered there can be more than one possible reference triangle as illustrated in Figure 2. The idea is to use all possible reference triangles for parallelogram predictions and average the resulting predicted locations in world coordinates.

$$pred_{multi} = \sum_i pred_{paral,i}$$

Then we transform the predicted location to the local coordinate system of the actually selected gate edge, to then carry out the predicted tangential components $\mathbf{p'_{tan}}$.

$$[p'_{tan,v}, \alpha] = transfToLocal(pred_{paral/multi})$$

$$predict_{tan}(v, f) = p'_{tan}$$

Eventually, in multi-way mode, we use all possible reference triangles for parallelogram predictions, then we average the resulting predicted locations in the world coordinates and transform its coordinates into local coordinate frame to find out the tangential components.

Time only Duplicata Predictor

The Duplicata predictor is a time-only predictor in the model space. Since we use the local coordinate frame constructed upon three direct neighbors in the current frame (spatial domain) and the locations in the previous frame (temporal domain), it is therefore a space-time predictor in the world space.

As described before, the tangential components x and y are predicted from their recovered tangential components in the previously decoded frame.

$$predict_{tan}(v, f) = \tilde{p}^{f-1}_{tan,v}$$

This predictor predict perfectly the parametric information and the prediction errors are close to zero.

Space-time predictor

The space-time predictor is an average of the time predictor and space predictor defined by the parallelogram (space(PP)_time) or the multiway (Multi_aver) predictor.

We computed the average of the time predictor (time_only) and the space predictor defined by the parallelogram (space(PP)_time) or the multiway (space(Multi)_time) predictor:

$$predict_{tan}(v,f) = \frac{1}{2}\left(\tilde{p}^{f-1}_{tan,v} + predict_{tan,space}(v,f)\right)$$

where $predict_{tan,space}(v,f)$ is the predicted tangential components from the neighbors in the current frame (space domain) and $\tilde{p}^{f-1}_{tan}$ is the temporal predicted components (time domain), the reconstructed tangential components in the previous frame.

Evaluation

The coding of tangential components using the described predictors is evaluated independently of the coding of the normal components.

Figure 8.7 (a) shows the rate distortion curves of the tangential components, for the cow animation, using temporal prediction (time_only), the average space-time predictions using parallelogram (space(PP)_time) and multiway (space(Multi)_time) prediction. Time_only shows a significant improvement in the compression ratio independent of the angle prediction, it saves about 26% to 41% bits over the space(PP)_time and space(Multi) _time, and about $\%31$ to 52% over spatial predictors (space(PP), space(Multi)), at similar distortion levels (*da*) and using 8 to 15 bits quantization. Recall that here the temporal prediction is relative to a local space and it is a space-time predictor in the world space.

Similar, for the dance animation 8.8 (a), time_only saves up to 46% bits over the space(PP)_time and space(Multi) _time and up to 59% over spatial predictors (space(PP), space(Multi)) at similar reconstruction error.

8.5.1.2 Normal component

The normal component is predicted in a cylindrical coordinate system around the gate as the bending angle between the reference triangle and the newly encoded triangle. Similar to the tangential component, we tested different predictors for the bending angle:

Space-only Predictor

The space-only predictor approximates the bending angle from the other available bending angles of the reference triangle:

$$predict_{\alpha}(v, f) = 180 - ({\alpha_{\mathbf{l}}}^{f} + {\alpha_{\mathbf{r}}}^{f})/2$$

Time Angle Duplicata Predictor

The bending angle is predicted from the recovered bending angle in the previous frame:

$$predict_{\alpha}(v, f) = \tilde{\alpha}_{v}^{f-1}$$

The curvature or the angle is almost preserved between two successive as seen chapter 5. Therefore, the predictor replicates exactly the curvature at the current edge gate from the curvature in the previous frame.

Spatial-temporal prediction

The predicted bending angle is obtained by averaging the predicted angles computed above:

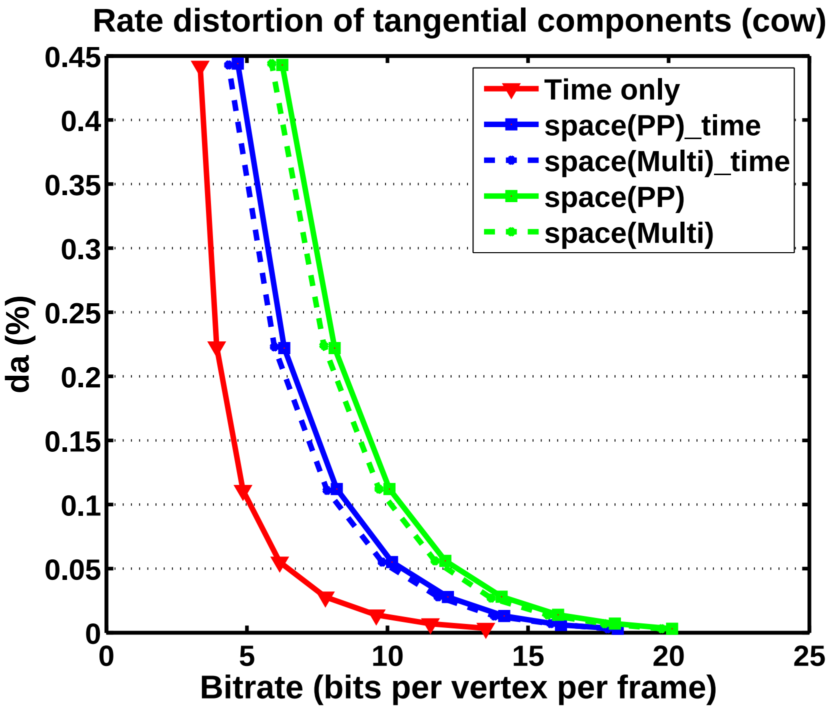
$$predict_{\alpha}(v, f) = (\alpha'(f-1) + \alpha'(f))/2$$

where $\alpha'(f-1)$ is the predicted angle using time prediction (time domain) and $\alpha'(f)$ is the predicted angle using the TG method (space domain).

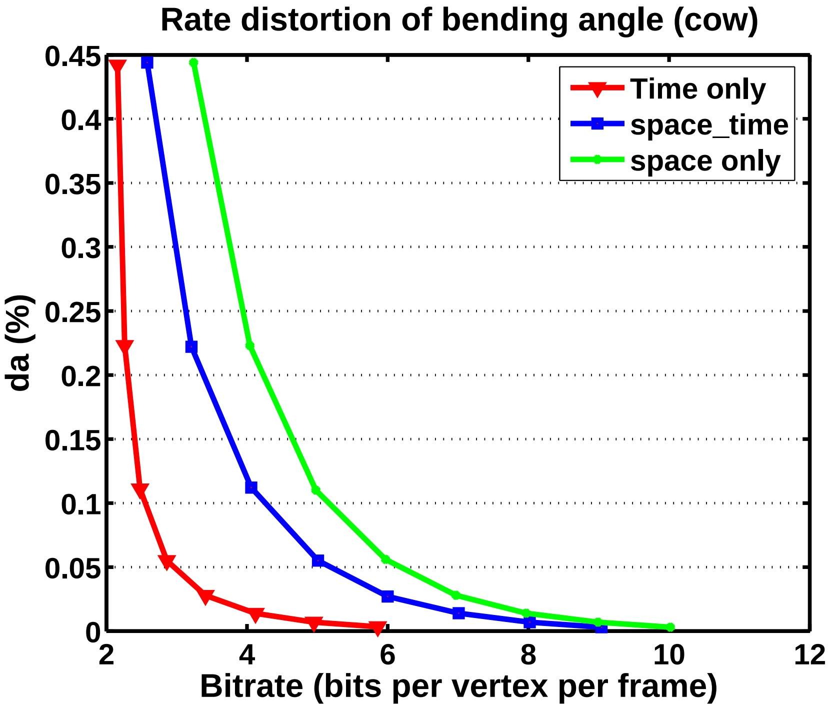
Evaluation

Figure 8.7 (b) shows the rate distortion of cow and dance animations respectively, using different predictors. Temporal prediction (in LCF) yielded a better compression ratio, independent of the tangential prediction strategy. The compression gain over space-time prediction (in LCF) based coding is up to 52% at the same level of distortions and using 8 to 15 bits quantization. Similar for the dance animation 8.8 (b), the temporal predictor saves up to 65% at similar distortion da.

The significant improvement means that the approximation of local coordinates in the temporal domain is significant and yields delta vectors and angles close to zero, (see chapter 5). Thereby, a significant reduction in the overall entropy can be achieved.

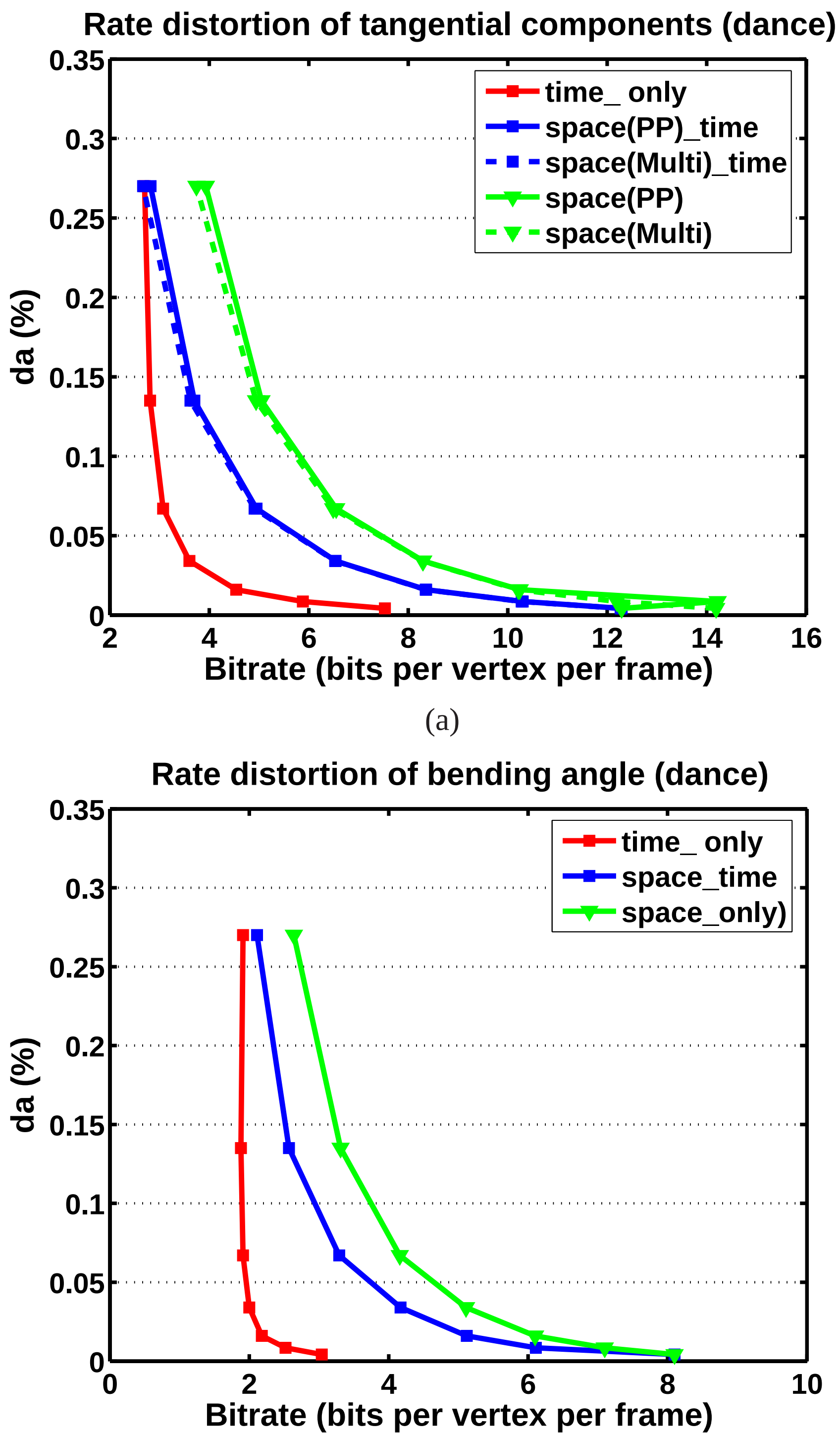


(a)



(b)

Figure 8.7 Rate-distortion curves of tangential components (a) and for bending angle (b) using different quantization bits (cow animation). Note that here we call temporal predictor relative to the LCF but it is space-time domain in world coordinate frame.



(a)

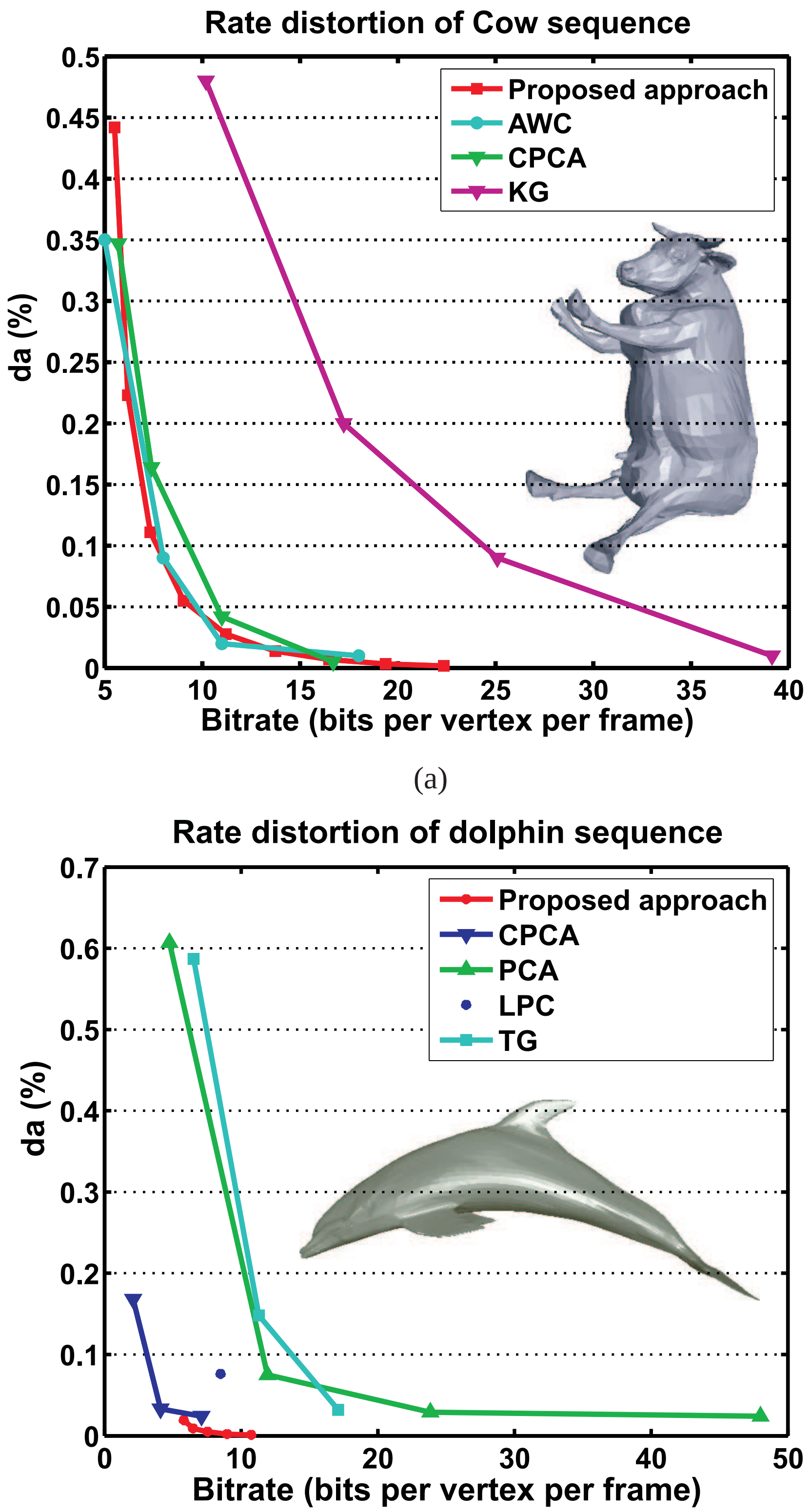
(b)

Figure 8.8 Rate-distortion curves of tangential components (a) and for bending angle (b) using different quantization bits (dance animation).

8.5.2 SplitCoder vs. State-of Art

Figure 8.20 illustrates the results of running our coder on the cow animation, which contains extreme deformations, and the dolphin animation, compared with different methods. To obtain different rate distortion points, we used different levels of resolution ($q = 8, ..., 14$). At first glance, we can see that our approach achieves a better rate distortion performance than the standard TG [117], linear predictive coding (LPC), KG [62] and PCA [3]. This result is obvious since animation coding based on static techniques only exploit spatial coherence and the linear prediction coding uses the temporal coherence only. Furthermore, the standard PCA only approximates the global linearity and is less effective for nonlinear animation. The LPCA overcomes this problem but still exploits the coherence of local regions over all frames. For the AWC [39] and CPCA [102] algorithms, we achieve similar results.

In Figure 8.10, we compare our approach SplitCoder to the Dynapack algorithm [45] that uses the Lorenzo predictor, the angle preserving predictor and CPCA. At similar numbers of bits (Figure 8.10 (a)), our approach achieves better quality (up to 80%) over Dynapack and (up to 22%) over the angle preserving predictor (maverg+angle) [113]. At similar quality (Figures 8.10 (b)), our coder archives gains up to 3%, 8% and 18%, over (maverg+angle), CPCA and Dynapack, respectively.



(a)

(b)

Figure 8.9 Comparison of our method with different compression algorithms.

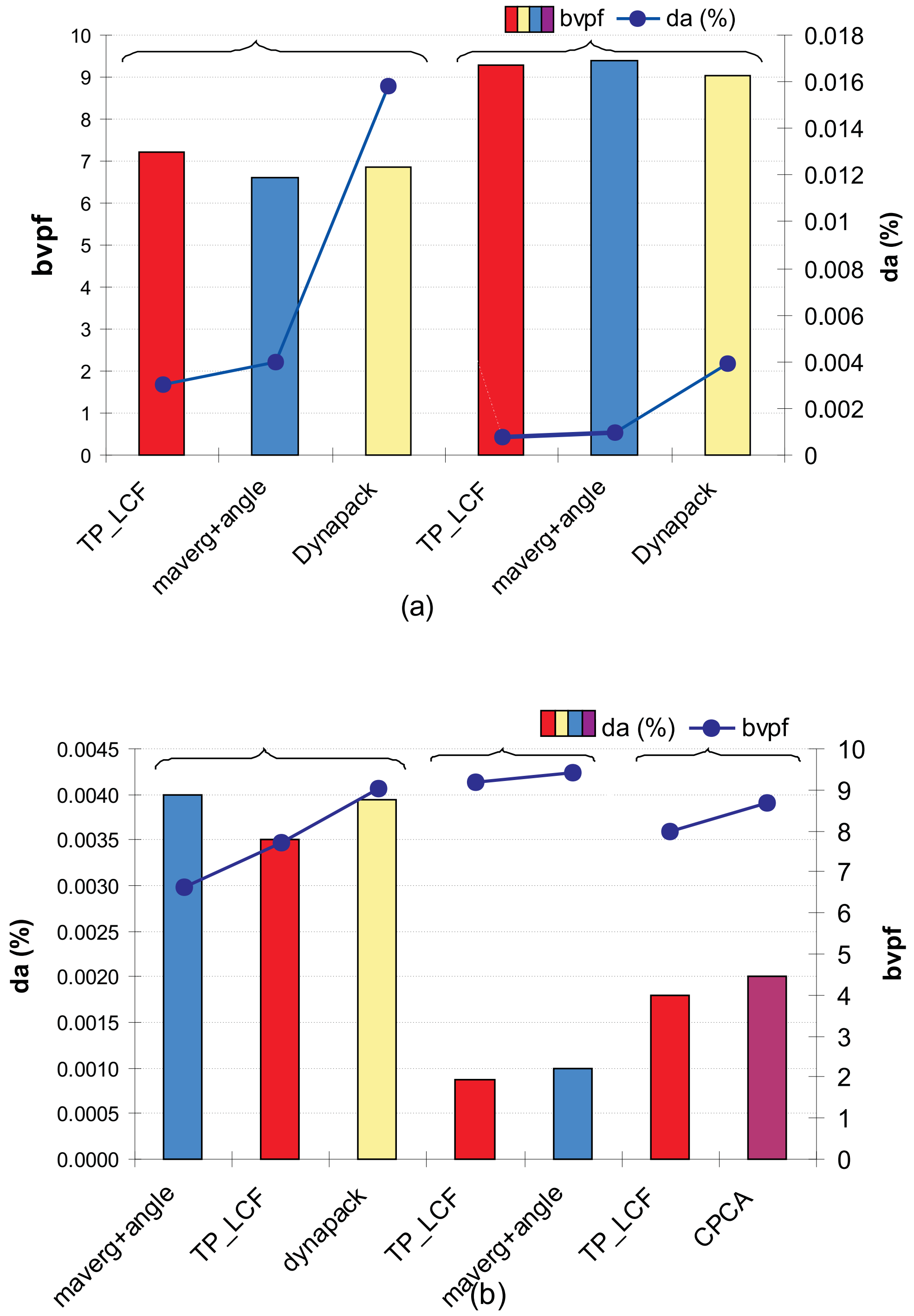


Figure 8.10 Comparison of our method SplitCoder (TP_LCF) with different compression algorithms at similar bitrates (a) and at similar reconstruction error (b)(chicken sequence). TP_LCF refers to the time only predictor in LCF.

8.6 RLPCA based Compression for Dynamic Meshes

This approach segments the mesh vertices into several groups of similar motion. Then, the local vertex coordinates of each set are transformed into another basis. To find the appropriate number of PCA components to represent the original data in each set, we introduce a rate distortion optimization for PCA based coding, that trades off between rate and the quality of the reconstructed animations. The compression is achieved by encoding the PCA components and coefficients.

8.6.1 Compression Parameters

In this approach, we consider three parameters that can govern the compression ratio as discussed before in chapter 6. We recall these parameters:

- *The number of basis vectors/rate L*: If this number is fixed for all clusters, then the user defines it (depending on the desired accuracy). The larger this number is, the better reconstruction will be (at the expense of less compression). If the RDO is used, then we will need only to specify the amount of compression (rate) or the maximum number of basis vectors that are to be used to approximate each cluster as we do in our coding. The number of vectors in each cluster is then optimally selected such that the total rate is below the given user-specified rate (or the total number of vectors is below the given user-specified maximum number of vectors).
- *The number of clusters* N: If this number is very small, then the cluster may contain vertices of different behavior and their local coordinates will have a large variation over time. However, it is difficult to find a linear space that efficiently represents these coordinates using PCA.
- *The reconstruction error*: This error presents the deviation of the reconstructed positions from the original one. It is measured using L2-norm or the metric *da*. Moreover, the metric L2-norm controls the compression during the RDO. This number should increase with decreases in the number of clusters or the number of eigenvectors.

8.6.2 RLPCA vs. LPCA

We want to find the influence of the clustering and the local coordinates on the bitrate and on the reconstruction of animation. We performed LPCA in the world coordinate system as well as in the local coordinate systems for a given numbers of clusters, components

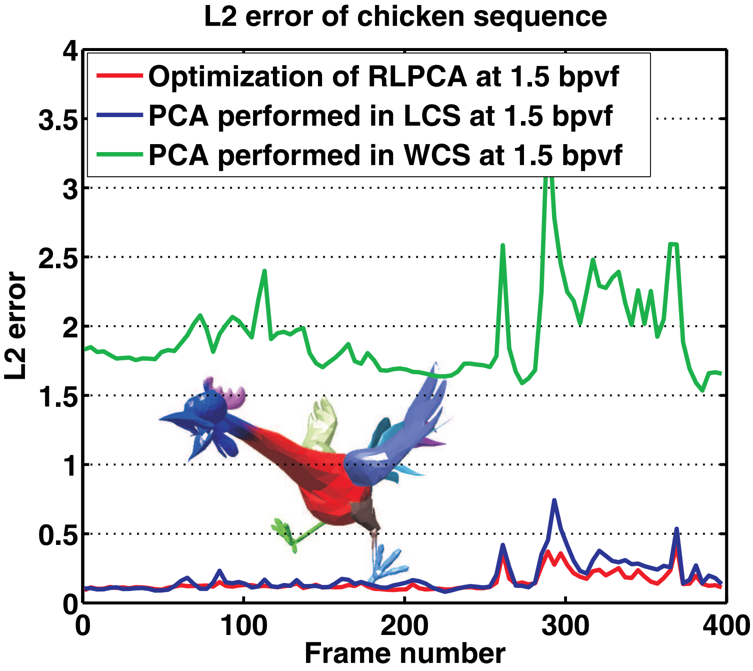


Figure 8.11 The error plot for the chicken sequence using LPCA in the world (standard LPCA) and the local coordinates (RLPCA) and using the RD-optimization (ORLPCA) (10 clusters,10 components). The error is measured by the L2-norm between the original and reconstructed frame.

and bits of quantization N, L and q_c respectively. Furthermore, we compared LPCA with the standard PCA.

Figure 8.11 shows the reconstruction results relative the original frame using $q_c = q_u = 12$ and $L = 10$ when the LPCA is performed in the world coordinates (green) and in the local coordinates (blue) and when the R-D optimization is introduced (red) at the same number of bit per vertex per frame. We can see that the local coordinates are more compressible than the original coordinates.

The improvement in the second curve (blue) (Figure 8.11) is due to the transformation of the original data into local coordinates which forces the coordinates of a vertex to cluster around one point (see chapter 6). This improvement increases (red) when the optimization were introduced.

Figure 8.12 and Figure 8.14 (a) shows the effect of the number of clusters and the component on the frame reconstruction for the chicken animation using (N, L) =$\{(20, 10)$; $(20, 20)$; $(10, 10)\}$ and on the rate-distortion curves for the dance animation using 10, 20 and 30 clusters.

Figures 8.13 shows the reconstructed two frames in the chicken sequence when the world and the local coordinates are used and when the optimization is introduced using 10 components and 10 clusters.

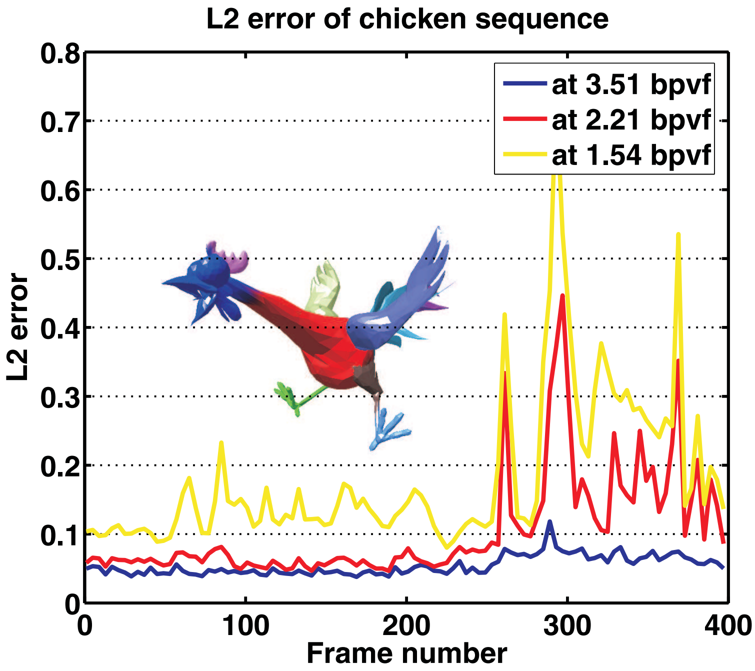


Figure 8.12 The error plot for the chicken sequence using different numbers of clusters N and components L. (N, L) =$\{(20, 20)$ (blue); $(20, 10)$ (red); $(10, 10)(yellow)\}$. The error is measured by the L2-norm between the original and reconstructed frame.

8.6.3 RLPCA and ORLPCA vs. State of Art

Figures 8.14 (b) and 8.15 also illustrate the comparison to other methods as rate-distortion curves for the cow (a), dolphin (d), and chicken (c) animations. At first glance, we can see that our approach achieves a better rate distortion performance than the standard PCA, LPC and TG for the three models. This result is obvious since the animation coding based on static techniques only exploit the spatial coherence and the linear prediction coding only uses the temporal coherence. Furthermore, the standard PCA only approximates the global linearity and is less effective for nonlinear animation.

For the CPCA and AWC algorithms, we achieve better or similar results. Figure 8.15 (a) shows that for the cow animation our method is significantly better than the method of KG (PCA+ LPCA) and than the CPCA. And it comes close to AWC. For the dolphin and the chicken sequences our method performs better than all the above methods. This improvement is due to the segmentation of the model into meaningful parts (whose vertices move quit similarly) as well as to the use of local coordinates rather than world coordinates. On the other hand, the RLPCA performs well for the models of large number of vertices in contrast to KG. Therefore, by combining RLPCA with LPC, we might achieve

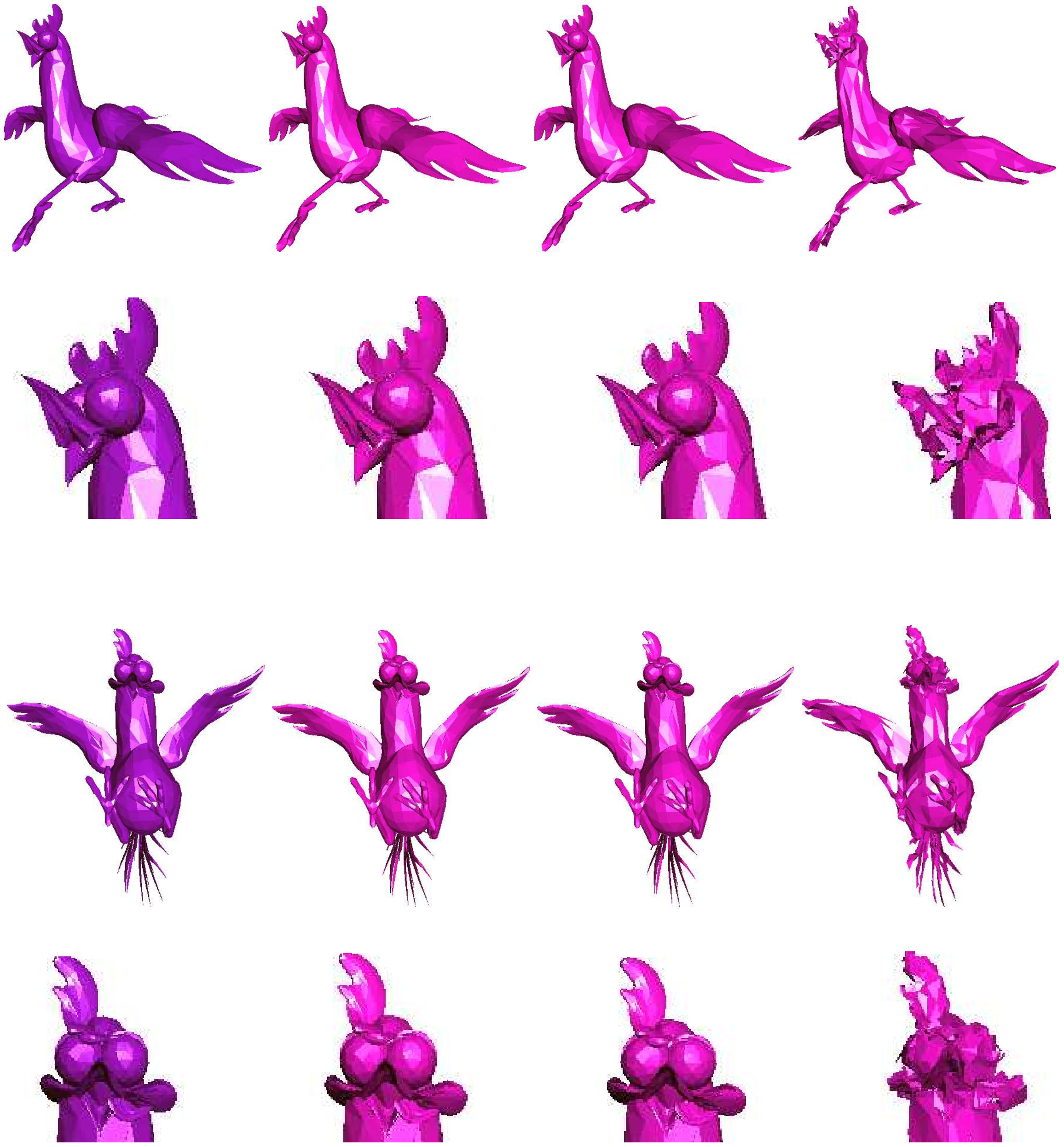


Figure 8.13 Reconstructed chicken: Frame 314 and its zoomed view (the two top raws) and frame 400 and its zoomed view (the two bottom raws) . **From left to right**: Original, optimized RLPCA, RLPCA, and LPCA performed in world coordinates (10 clusters; 10 components).

a better compression ratio. Figures 8.14 (b) and 8.15 also demonstrate that the rate distortion optimization we introduce in our algorithm (ORLPCA) is important for achieving better compression performances especially when the number of vertices is large and the animation is complex.

From the computational viewpoint, PCA is computational expensive but in combination with LPC [62], it gives a better compression performance, particularly for a long sequence of just a few number of vertices. CPCA [102] outperforms both methods since they explore a robust segmentation which is based on a data analysis technique but remains expensive. In contrast, our RLPCA uses a simple segmentation and transformations and achieves a better compression ratio.

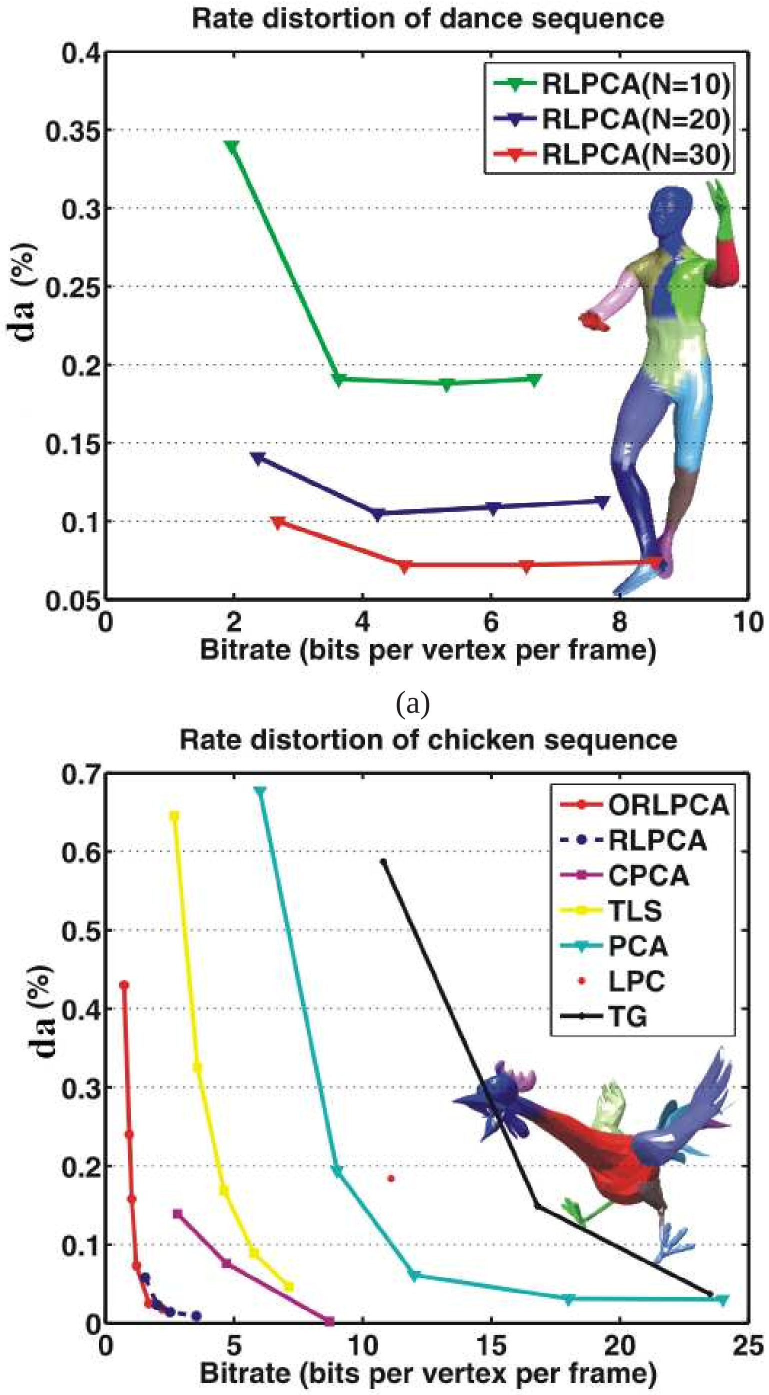
Our Algorithm achieves an increased compression performance, is computationally inexpensive compared to a PCA for the full mesh and it is well-suited for progressive transmission

8.6.4 Timings

Table 8.6 shows the timings in seconds of the coding (t^{enc}) and decoding (t^{dec}) processes (without optimization) for the four animations with a comparison to CPCA $(t^{FPS}$ for display while decoding). We observe that for the chicken and cow animations, our coder is much faster and performs better than CPCA. Our timing results are measured on Pentium 4 with 2.53 GHz and CPCA on AMD Athlon64 XP 3200+.

Table 8.6 Comparison compression and decompression (RLPCA-Clustering) timings with CPCA.

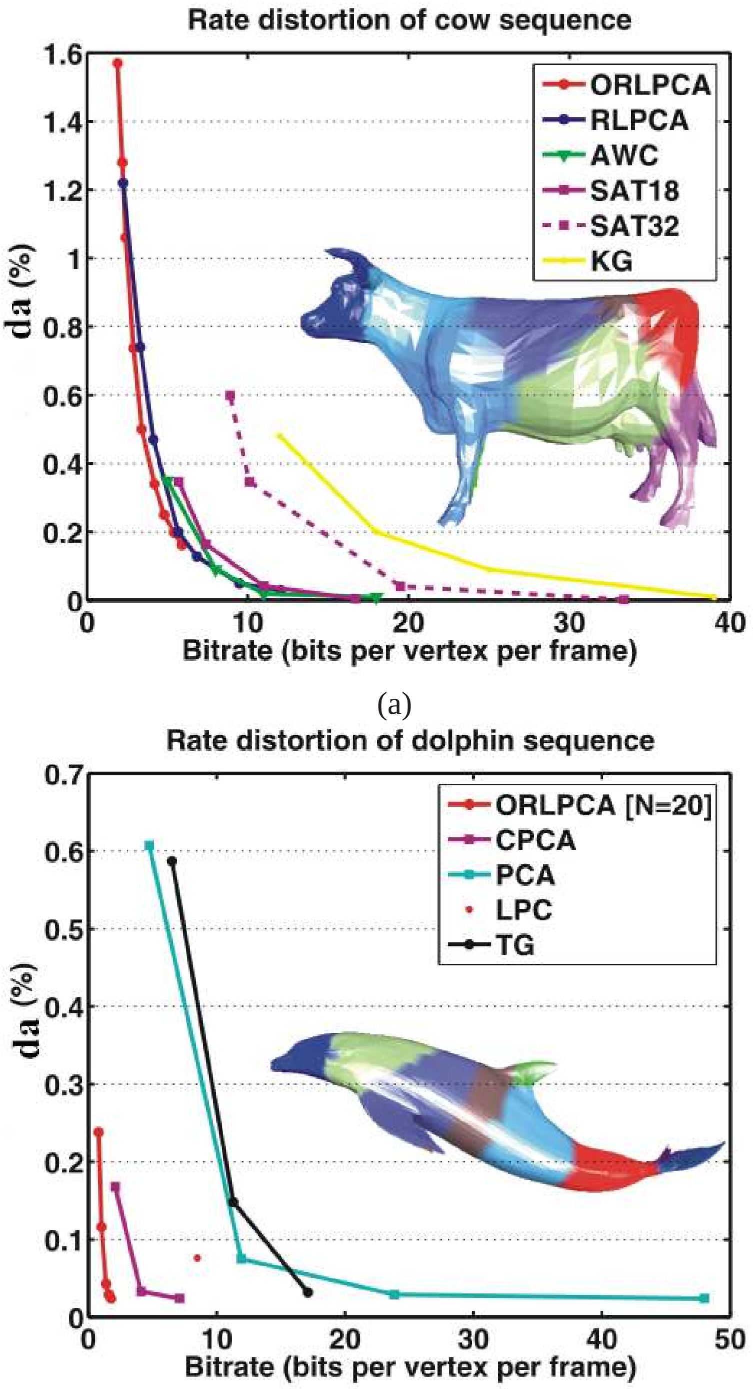
	CPCA				RLPCA					
Models	bpvf	da	$t^{enc}_{(sec)}$	$t^{FPS}_{(sec)}$	bpvf	da	N	L	$t^{enc}_{(sec)}$	$t^{dec}_{(sec)}$
chicken	4.7	0.076	206	214	3.5	0.008	20	20	120	69
	2.8	0.139	395	215	2.2	0.043	20	10	115	69
	2.8	0.139	395	215	1.5	0.057	10	10	110	47
cow	7.4	0.16	75	145	6.8	0.128	30	20	82	46
	3.8	0.5	59	218	4.1	0.470	30	20	40	50
	2.0	1.47	55	284	2.2	1.220	10	10	70	23
dolphin	7.1	0.024	-	-	3.9	0.016	20	10	74	40
	4.1	0.033	-	-	2.1	0.018	20	5	78	32
	2.1	0.168	-	-	1.9	0.066	10	5	39	25



(a)

(b)

Figure 8.14 Rate distortion curves for dance and chicken sequences.



(a)

(b)

Figure 8.15 Rate distortion curves for cow and dolphin sequences using the metric da.

8.7 Predictive-DCT based Compression

This section shows the results of the algorithm *PDCT* that is based on cluster based-prediction and DCT and the effect of different compression parameters on compression performance.

8.7.1 Compression Parameters

Influence of Cluster Numbers

The number of clusters N is an important compression parameter that affects the compression performance as seen in previous method RLPCA. The bigger this number is, the smoother the shape reconstruction will be and the lower the bit rate that is obtained. If this number is too small, the vertices of the same cluster may behave differently relative to their LCF. Thereby, the prediction in the LCF becomes poor yielding poor compression. In contrast, If N is big, the variation of the vertex relative to the LCF of its cluster becomes smaller and the prediction is more effective.

Figures 8.17 and 8.16 illustrates the curves DCT coefficients/bitrate and coefficients/*da* for different numbers of clusters.

Figure 8.22 also shows the rate-distortion curves for different animations at different numbers of clusters: dolphin using 10 and 40 clusters, chicken using 10, 25 and 40 and dance using 10, 20 and 40 clusters. We observe that 40 clusters provide better error quality and bit rate than using 10 or 20 clusters.

Influence of DCT Coefficients

To find the influence of the number of DCT coefficients on the rate and on the reconstruction of animation, we have run our coding on different resolution. Figure 8.17 and 8.16 show the results of the number of these coefficients percent for chicken animation. When more coefficients are discarded, better compression (Figure 8.17) is achieved at the expense of the reconstruction quality (Figure 8.16).

The effect of the cluster and coefficient numbers can also be seen in Figures 8.18 and 8.23.

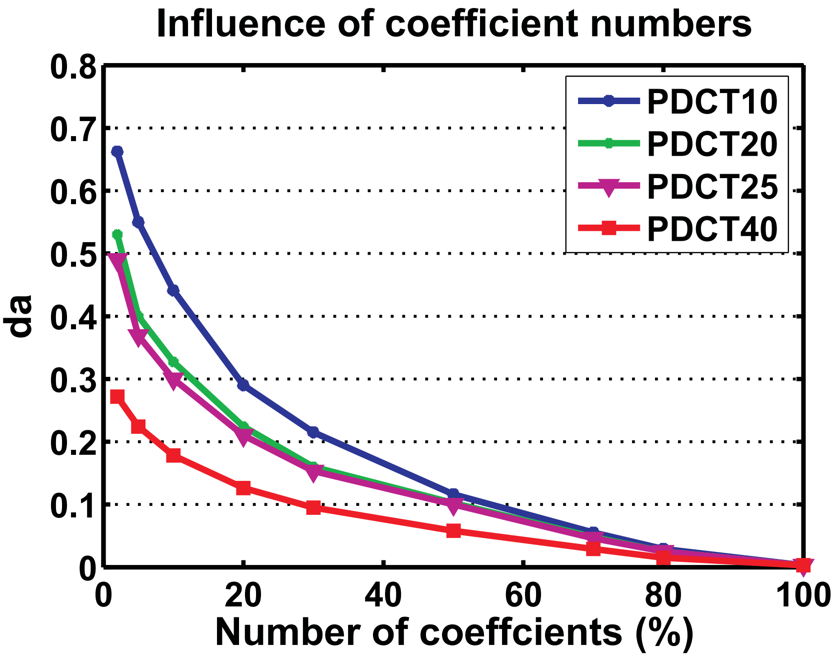


Figure 8.16 Influence of different numbers of zeroed DCT coefficients (%) on the reconstruction quality *da* using different number of clusters.

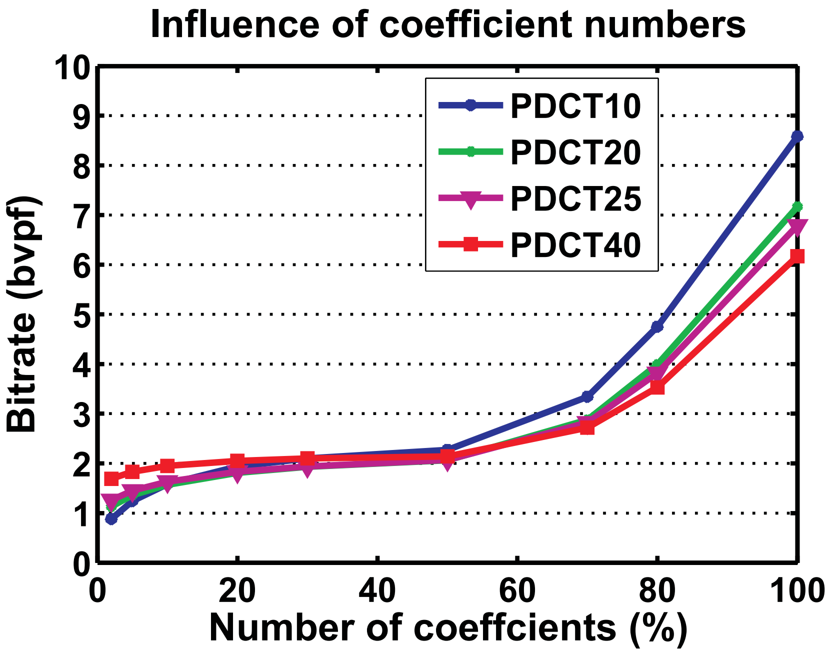


Figure 8.17 Influence of different numbers of zeroed DCT coefficients (%) on the bitrate using different number of clusters.

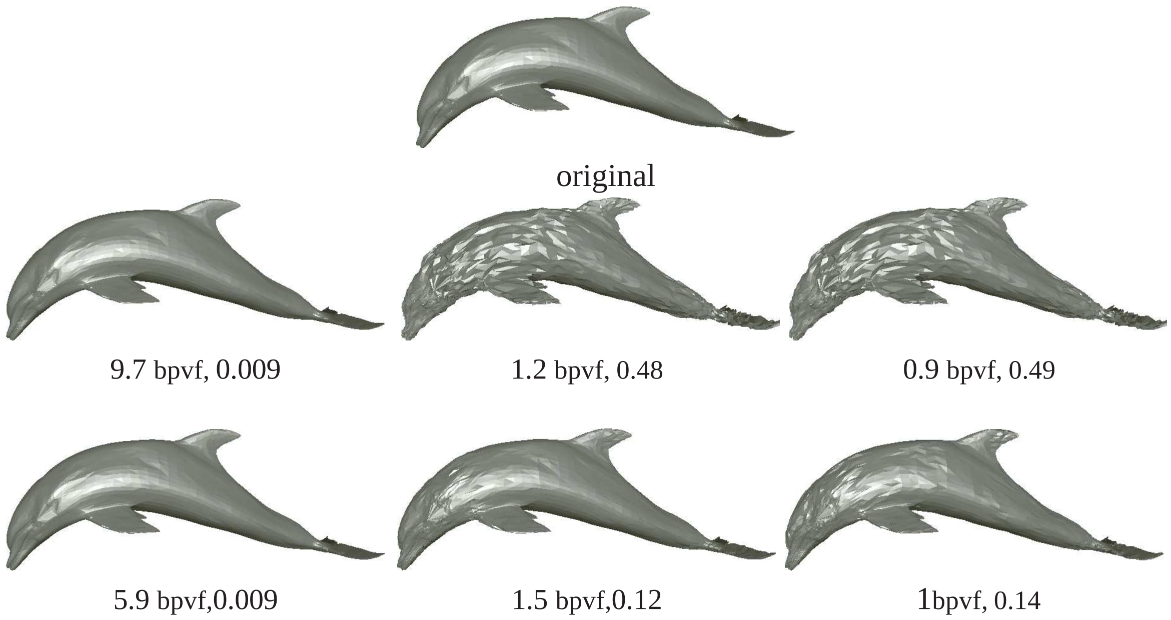


Figure 8.18 Reconstruction frame 60 of dolphin sequence, original mesh (top arrow), using 10 clusters (middle arrow) and 40 cluster(bottom arrow). From left to right: using different numbers of non-zero coefficients ($\%$) and quantization levels: (100%,12 bits), (2%,12 bits) and (2%,8 bits), at various bit rates in bit per vertex per frame and decoding error (*da*).

Influence of Quantization Level

Figure 8.19 illustrates the reconstruction samples of cow animation for different quantization levels 6, 8, 12 bits. If a coarse quantization is used then the low value DCT coefficients will be zeros. Consequently, the fine details are lost and only the coarse details are detected. Moreover, the clustering process will retain an acceptable visual quality.

The more coefficients that are zeroed, the more coarser the quantization is, and that better the compression will be at the expense of visual appearance. The finer details can be preserved when only a finer quantization is used and few coefficients are thrown away. For example, if 50% of coefficients have zero values then we use 10 bits quantization. If 90% we use 8 bits only.

Recall that the delta vectors of the first frame are encoded using 12 bits quantization while the delta vectors of the LCFs in the whole sequence are quantized to 16 bits.

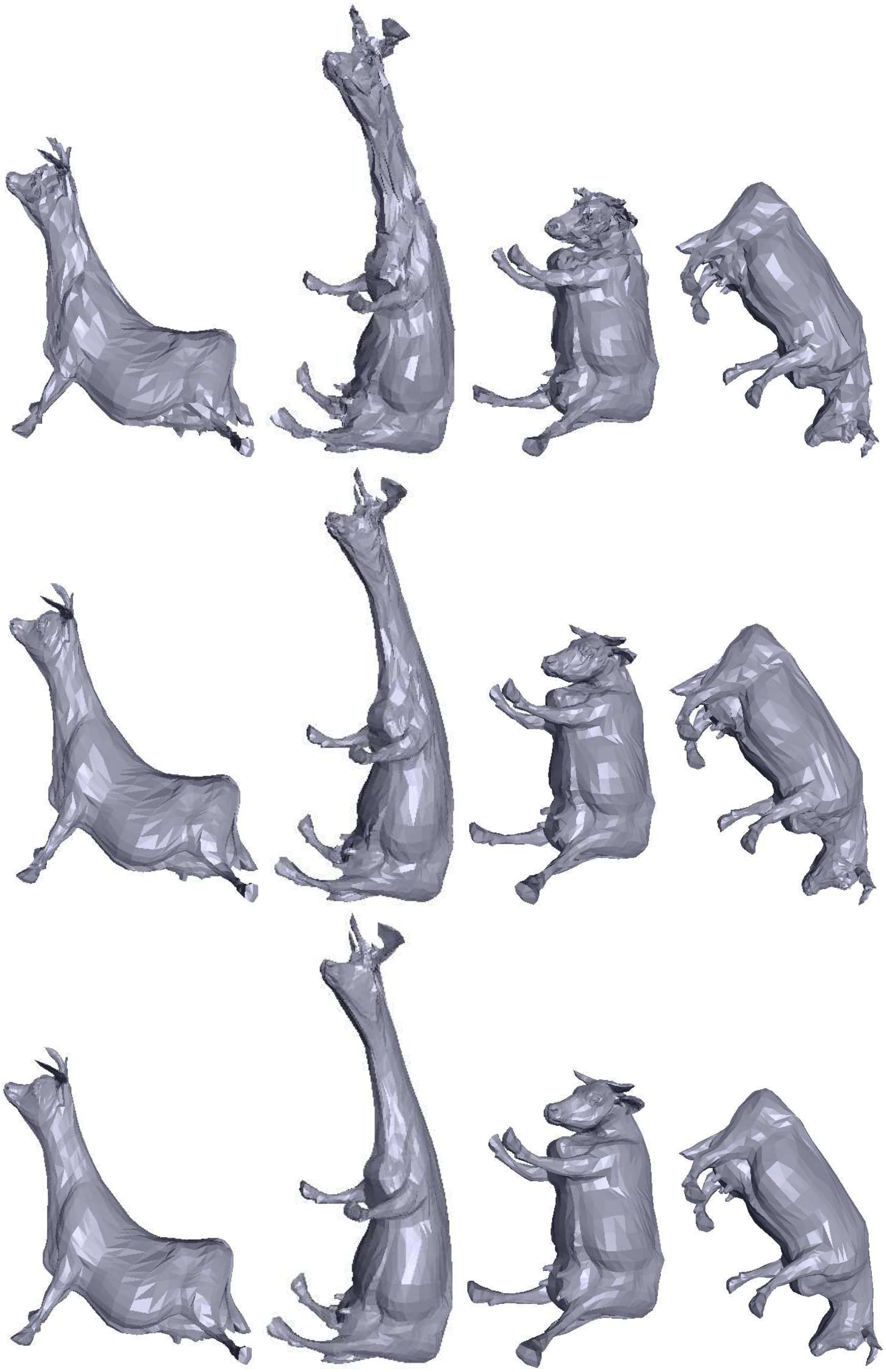
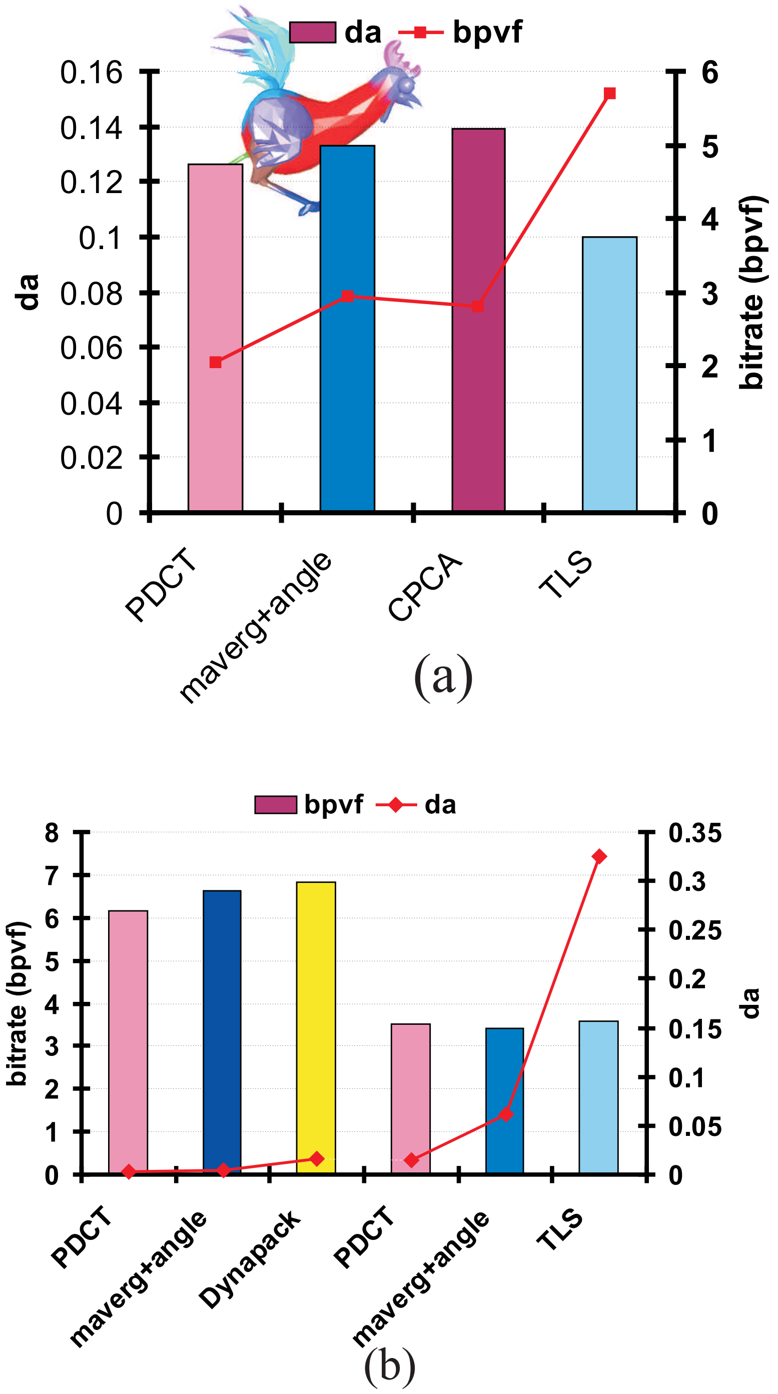


Figure 8.19 Reconstruction sample frames of cow animation using different quantization levels. From top to bottom: 6, 8, 12 bits.



(a)

(b)

Figure 8.20 Comparison of our method with different compression algorithms at almost similar bitrates (a) and at similar reconstruction error (b) (chicken sequence).

8.7.2 PDCT vs. State of Art

Figure 8.22 illustrate the results of running of our coder on three animations compared with different methods.

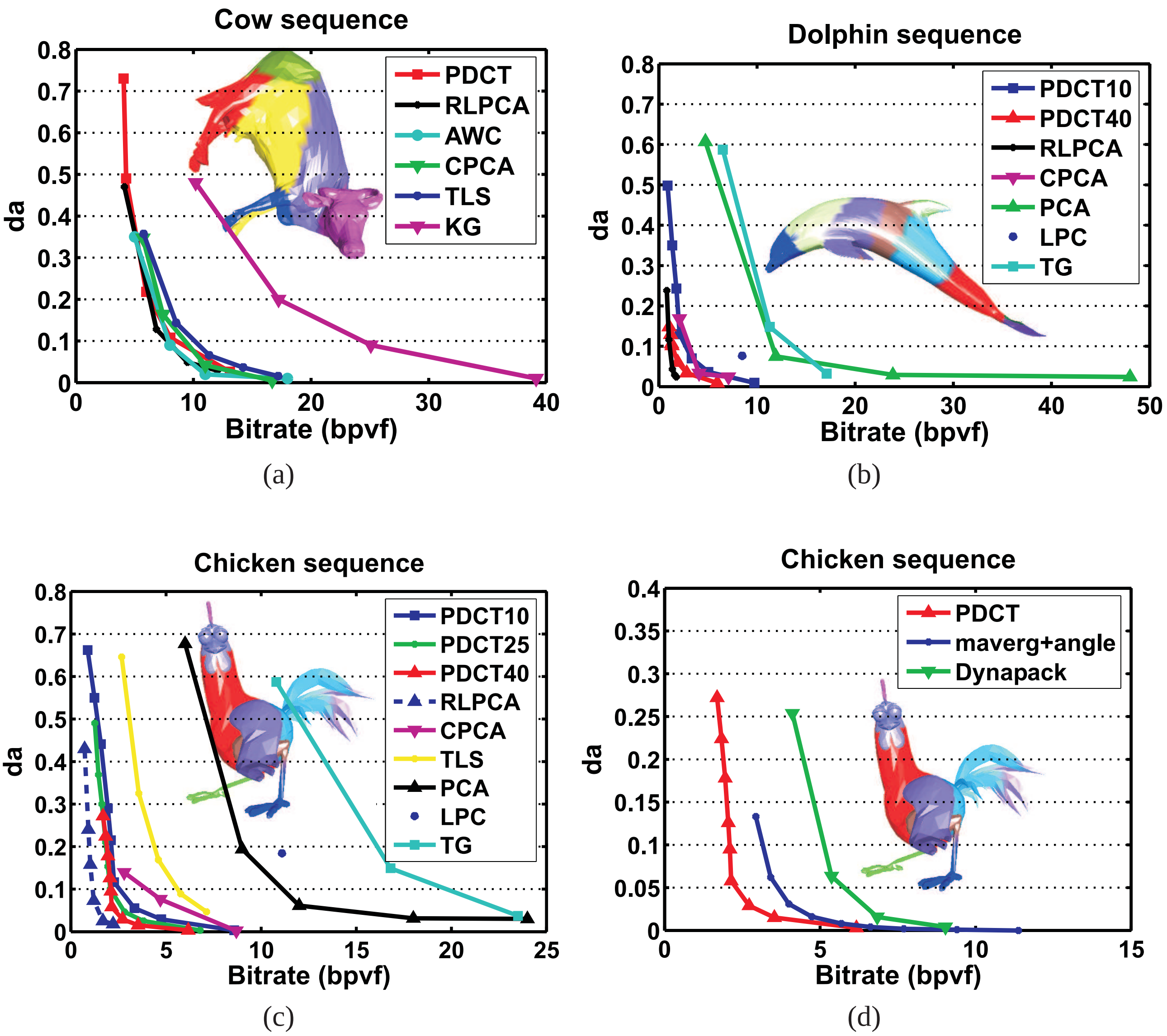
For the three models PDCT preforms better than the standard PCA, LPC, KG and TG. This is because, as motioned before, the animation coding based on static mesh compression technique only (TG) exploits the spatial coherence only, the linear prediction coding (LPC) uses the temporal coherence only and PCA approximates the global linearity and is less effective for nonlinear animation. while PDCT exploit the coherence in both space a time domain.

For the CPCA and AWC algorithms, we achieve better or similar results. Figure 8.22 (a) shows that for the cow animation which contains extreme deformations, our method is significantly better than the KG method and comes close to the CPCA and to wavelet based methods (TLS [92] and AWC).

For the chicken and the dolphin sequences, our method performs better than all the above methods, including the predictive techniques (Dynapack and maverg+angle). This improvement is due in one hand to the clustering of the model into rigid parts, making the prediction more efficient in the local rather than the world, coordinates, and on the other hand to the further DCT coding, which leads to a significant reduction in the overall entropy.

The RLPCA method overcomes all other methods, including ours, for chicken animation while it comes close to our for the other models. Our method PDCT uses as similar clustering process as RLPCA scheme. However, the difference arises in the way of encoding the local coordinates. The RLPCA considers the entire cluster sequence and exploit the global coherence using PCA. While PDCT have to encode frame by frame using predictive and spatial DCT coding, the method is well suited for real-time compression.

In Figure 8.20, we compare our approach against several approaches. At (almost) similar quality (Figures 8.20 (a)), our coder archives gains up to 30% and 27%, over the angle preserving predictor (maverg+angle) and CPCA respectively. At (almost) similar numbers of bits (Figures 8.20 (b)), our approach obtains better animation quality from 28% up to 76% over maverg+angle, 81% over Dynapack (using Extended Lorenzo Predictor) and 95% over TLS.



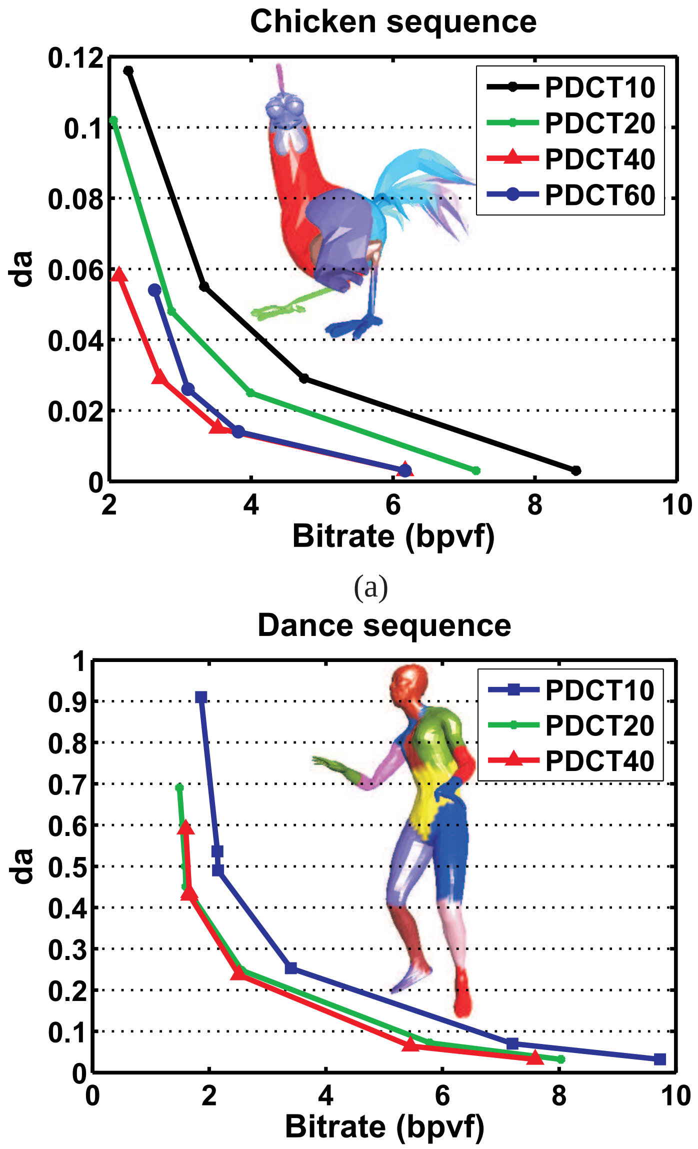
(a)

(b)

(c)

(d)

Figure 8.21 Rate distortion curves for the cow (a), dolphin (b), and chicken (c), (d) sequences.



(a)

(b)

Figure 8.22 Rate distortion curves for the chicken (a) and dance (f) sequences using different number of clusters.

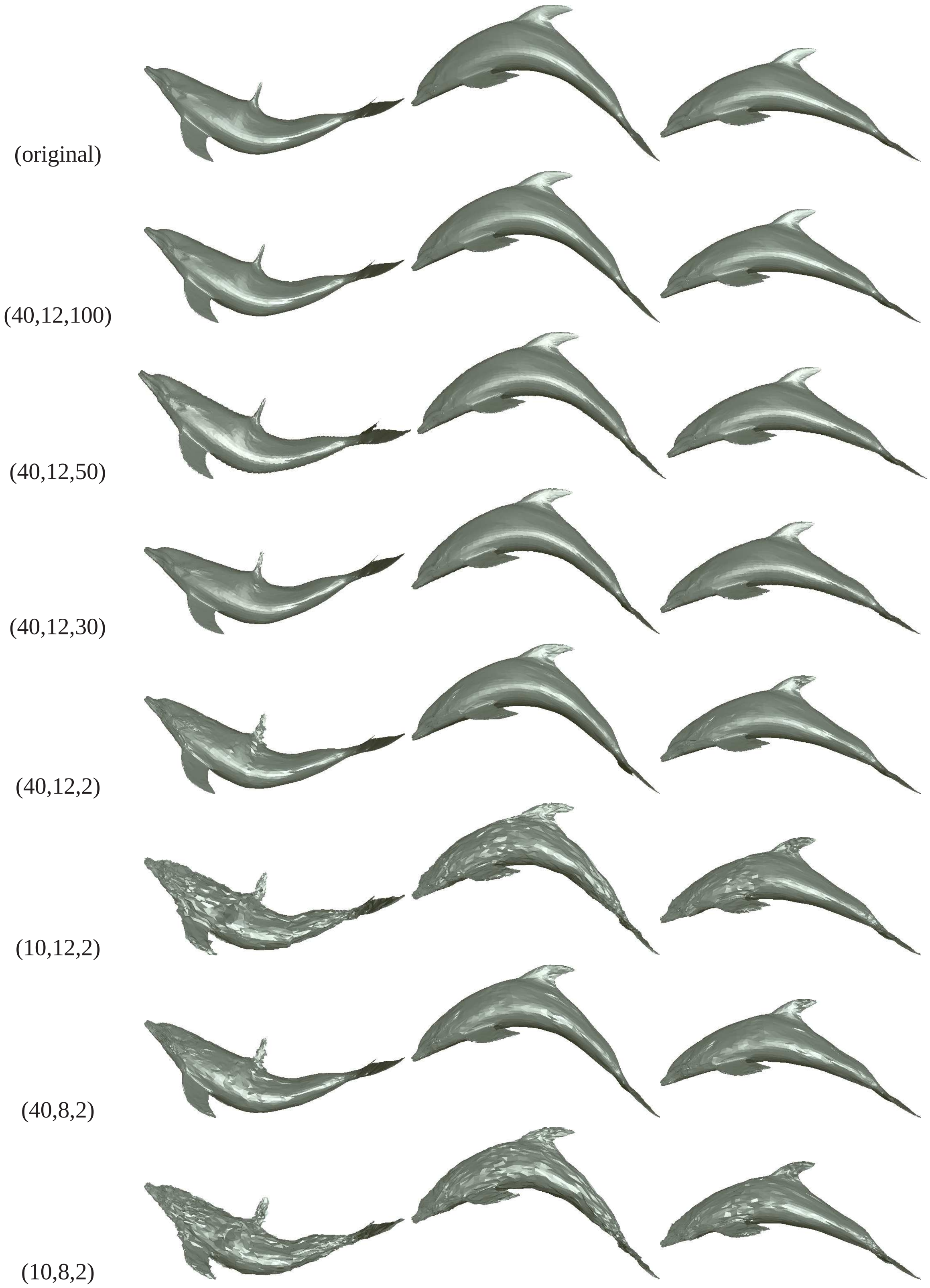


Figure 8.23 Reconstruction sample frames of dolphin sequence. The numbers in the first column are the number of clusters, quantization level and coefficient number (%).

8.8 Comparison of the Proposed Methods

We already compared our algorithms to some methods from state-of-art based on their results reported in their papers. Indeed, we did not obtain the reconstructed animation sequences of these methods. We also did not use their source code of the distortion measurement that computes the reconstruction distortion to ensure the comparison is properly performed. Of course, the compression parameters affect the sequence reconstruction and the comparison to the other methods.

In this section, we restricted the final experiments to the comparison between our implementation codes only.

It is obvious that the method that exploits both spatial and temporal coherence achieves better compression ratio than the method that exploits only one. Our proposed methods for 3D dynamic mesh compression combine different schemes to exploit the coherence in both space and time:

1. SplitCoder:
   - LCF: spatial coherence
   - vertex based-predictor (relative to LCF assigned for each vertex): spatial and temporal coherence.
2. PDCT:
   - clustering: spatial and temporal coherence.
   - LCF: spatial coherence.
   - differential coding of LCF: temporal coherence.
   - cluster based-predictor: spatial and temporal coherence
   - DCT: spatial coherence.
3. RLPCA:
   - clustering: spatial and temporal coherence.
   - LCF: spatial coherence.
   - differential coding of LCF: temporal coherence.
   - PCA: temporal coherence.

Table 8.7 summarizes the characteristic of the proposed compression methods for animated meshes.

Table 8.7 Characteristic of the proposed approaches SplitCoder, PDCT, and RLPCA

	splitCoder	PDCT	RLPCA
lossless/lossy	not lossy	lossy	lossy
category	vertex-based	cluster-based	cluster-based
coherence	locally (frame by frame)	locally (frame by frame)	globally
connectivity	dependant	$\sim$ independent	$\sim$ independent
complexity	*	**	**
encoding time	* few sec.	*** few sec. to min.	** few sec.
decoding time	* fast	**	* very fast
remarks	well-suited for realtime applications	suited for realtime applications	allows progressive transmission more practical

SplitCoder is vertex based-predictor, for each vertex on have to construct LCF and perform predictive coding. The advantage of SplitCoder is its lossless nature. The distortion is due only to the quantization. Typically, between 10 and 12 bits, we obtain an animation sequence that is indistinguishable from the original one. Moreover, it is simple, fast (in the order of few seconds), well-suited for real time application, and outperforms several existing compression techniques based on PCA, or on predictive coding. The only disadvantage of this method is that it does not allow very low bit rates. This is because at low quantization level (typically 5 and 6 bits), some triangles may become degenerated and consequently, the reconstruction of the LCF is impossible.

PDCT is lossy technique, more complex than splitCoder but allows low bit rate and we need only one LCF per cluster. DCT is introduced for further data reduction, yielding uncorrelated coefficients, which are well compressible than the delta vectors. This methods is more expensive than SplitCoder because for each frame and for each cluster, one have to perform DCT coding and computer the inverse DCT to reconstruct the current frame for further encoding. Recall that always the current frame to be encoded, is predicted from the previously decoded one, to avoid error accumulation.

RLPCA is lossy compression, less complexity than PDCT but more complex than splitCoder, while decompression algorithm is the most simple and the fastest one, compared to the above methods. When a bit allocation is introduced (ORLPCA), better compression ratio is obtained at the expense of processing time of encoding.

Figures 8.25, 8.26 and 8.24 illustrate the comparison between the three methods as rate-distortion curves for the dolphin, dance, cow, chicken and snake animations. One can see that RLPCA and ORLPCA outperforms PDCT and SplitCoder. At very low bitrate, this later is not efficient. In case of high bitrate, lower distortion is obtained and all methods come close to each others.

The following figures 8.27, 8.28, 8.29, 8.30, 8.31, 8.32, 8.33, 8.34, 8.35 and 8.36 show the original and the reconstructions of sample frames from the dance, dolphin, cow, snake and chicken animations, using the proposed approaches at similar bitrate.

At high bitrate it is difficult to see the difference of visual appearance between the original and the reconstructions. In case of more distortion, high compression ratios is achieved.

One may also notice that the reconstructed frames using RLPCA are nearly identical to the original frames. The distortion incurred is always very small compared with the error incurred by PDCT and SplitCoder methods and the good compression ratios make the method RLPCA the most attractive one.

The distortion introduced by PDCT is also almost less than the distortion produced by SplitCoder. This is obvious, at low bitrate, SplitCoder is not efficient. Indeed, the mesh reconstructions exhibit blocky artifact due to the low quantization level.

At similar bitrate, PDCT yields better reconstruction due to DCT based coding that may reduces the the blocky artifact at low bitrate. Indeed, after DCT transform, the majority of signal energy concentrates on the low frequencies and little on the high frequencies. Hence, the high frequencies can be zeroed yielding a significant reduction in the overall entropy. The signal can then be represented by few high value coefficients without significant distortion.

Note that the blocking effect also appears in PDCT if the number of coefficients to be zeroed is very large or if the quantization level is very low.

We also computed the gain of each method at similar level of distortion. We found that the gain of OLPCA over RLPCA is up to 42% due to the bit allocation introduced for PCA. The gain of RLPCA over DCT and SplitCoder are up to 32% and 69%, respectively. The gain of PDCT method over SplitCoder is up to 73%. Note that the gain varies with compression parameters and animation sequences. For example, for the snake animation, RLPCA come close to PDCT while their gain over SplitCoder is up to 73%. For chicken animation, the gain of RLPCA over DCT is about 36% and over SplitCoder is about 48%.

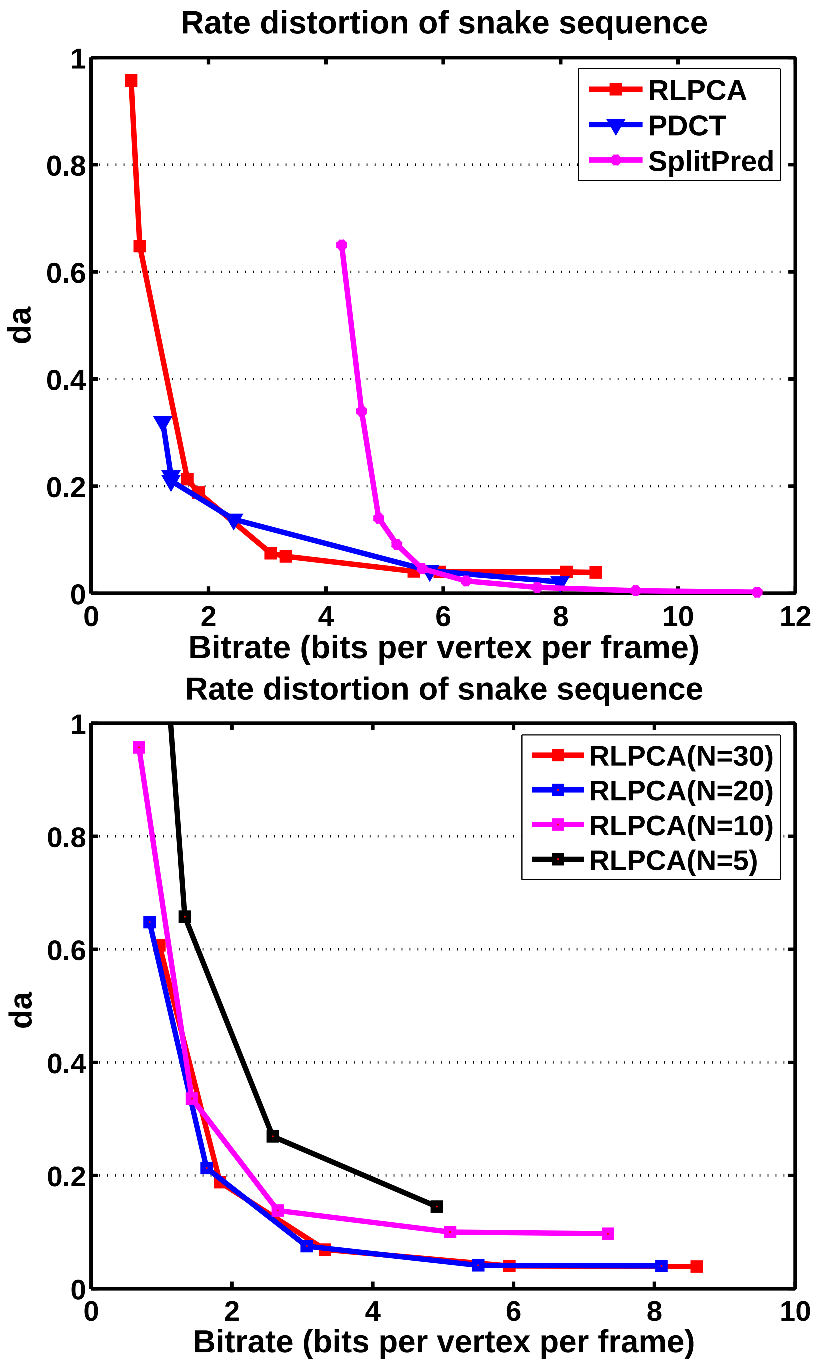


Figure 8.24 Rate distortion curves for snake sequence using different number of clusters

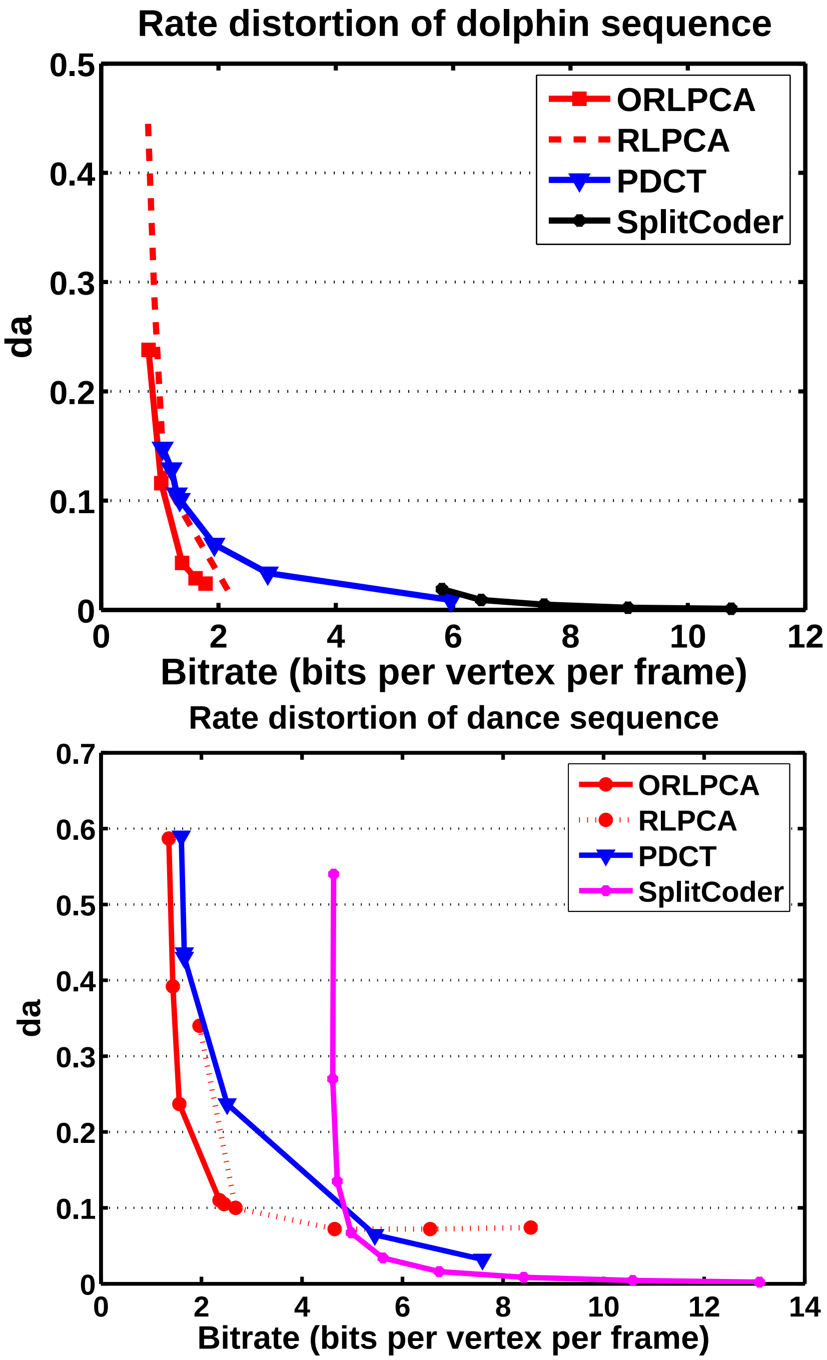


Figure 8.25 Comparison between the proposed approaches for dolphin and dance animations

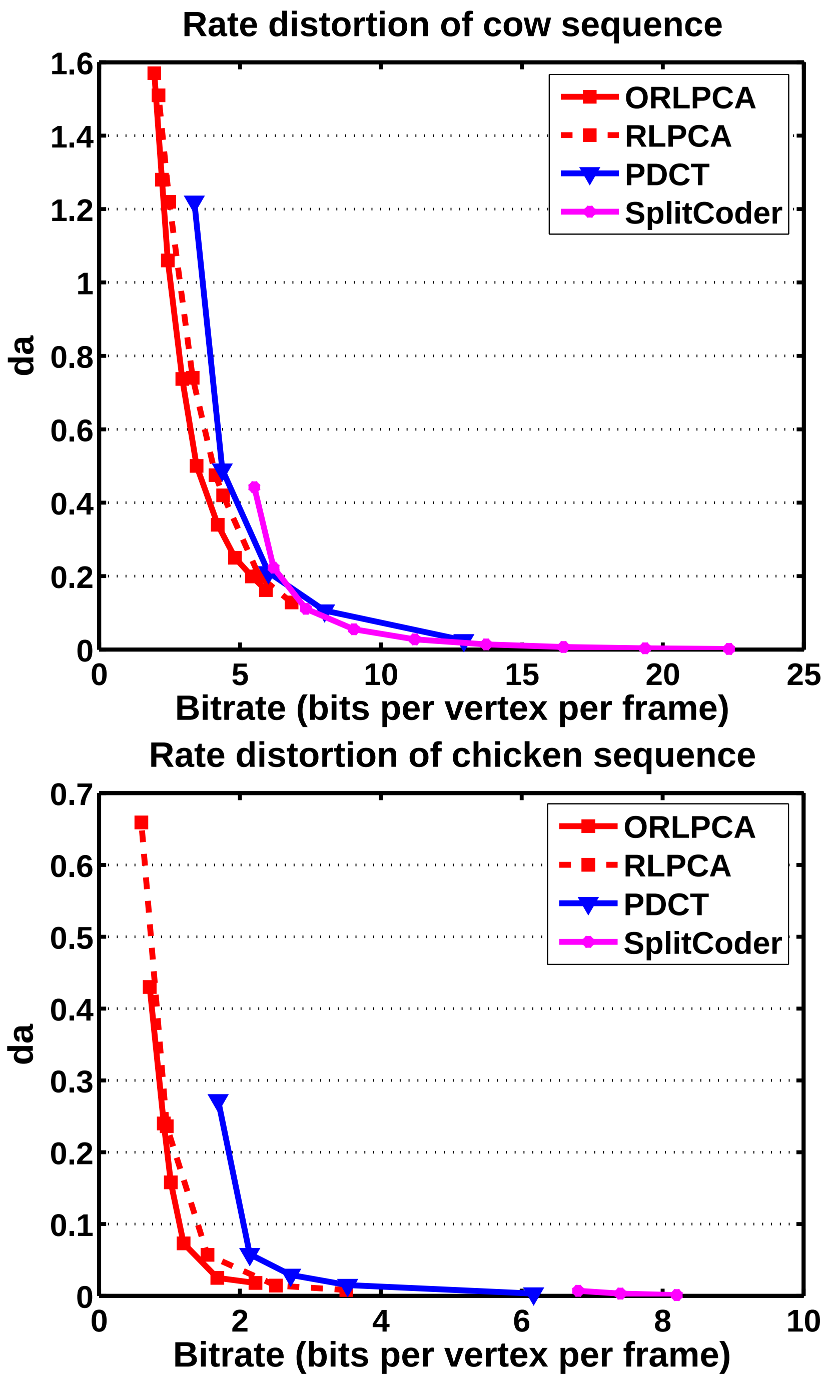


Figure 8.26 Comparison between the proposed approaches for cow and chicken animations

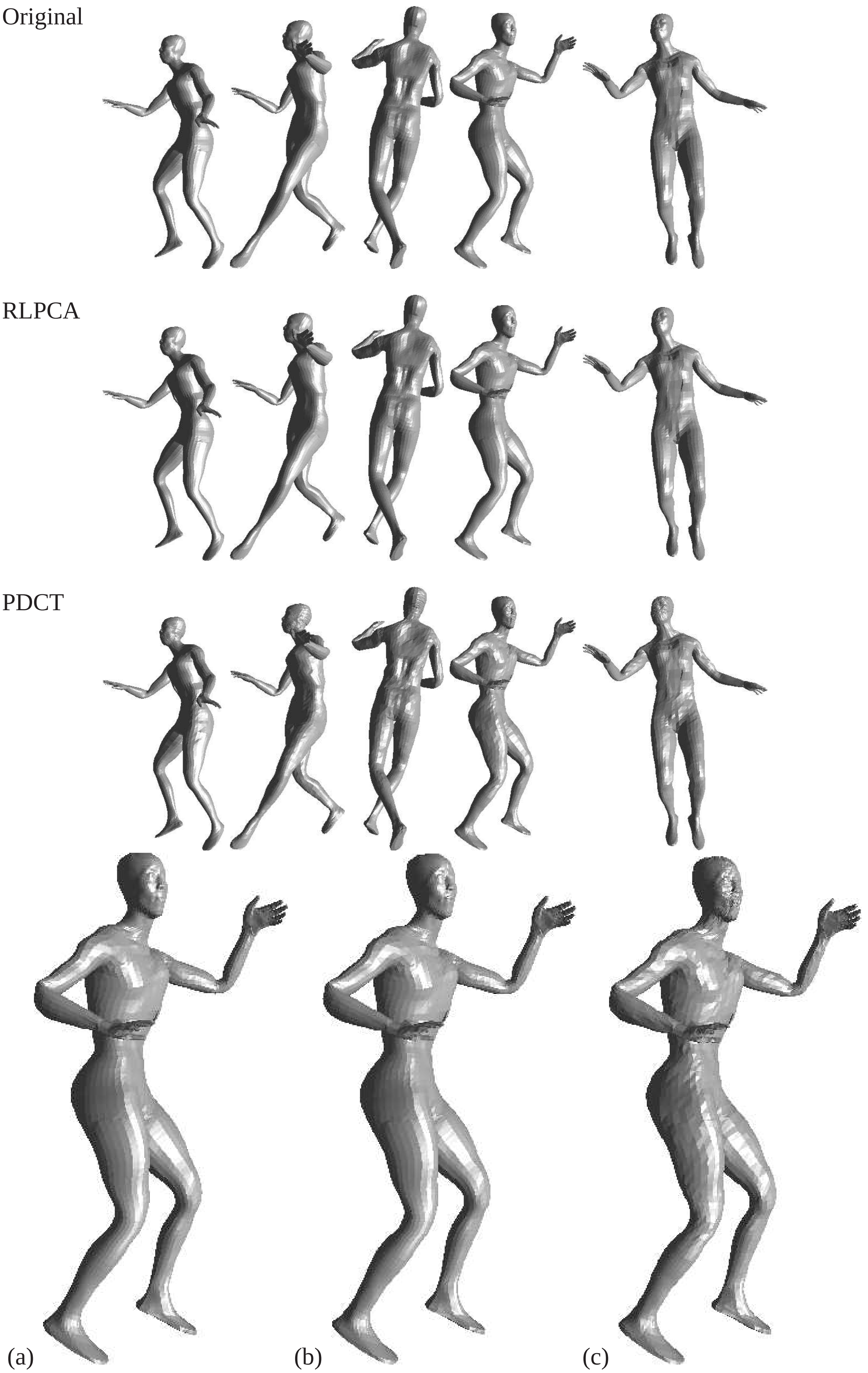


Figure 8.27 Comparison of RLPCA and PDCT at a similar bitrate. Frame 50: (a) original and (b) RLPCA and (c) PDCT.

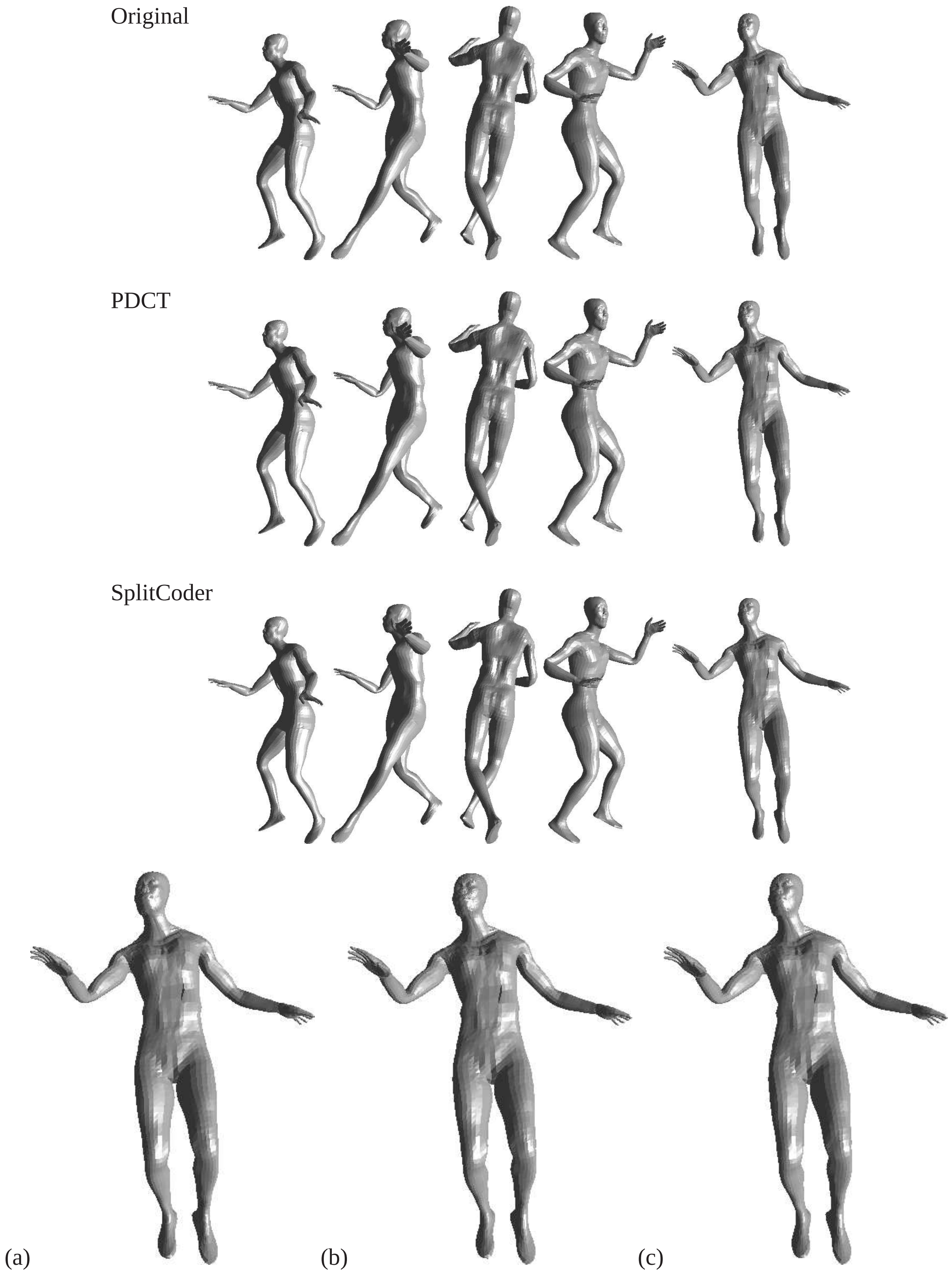


Figure 8.28 Comparison of PDCT and SplitCoder at a similar bitrate. Frame 80: (a) original and (b) PDCT and (c) SplitCoder.

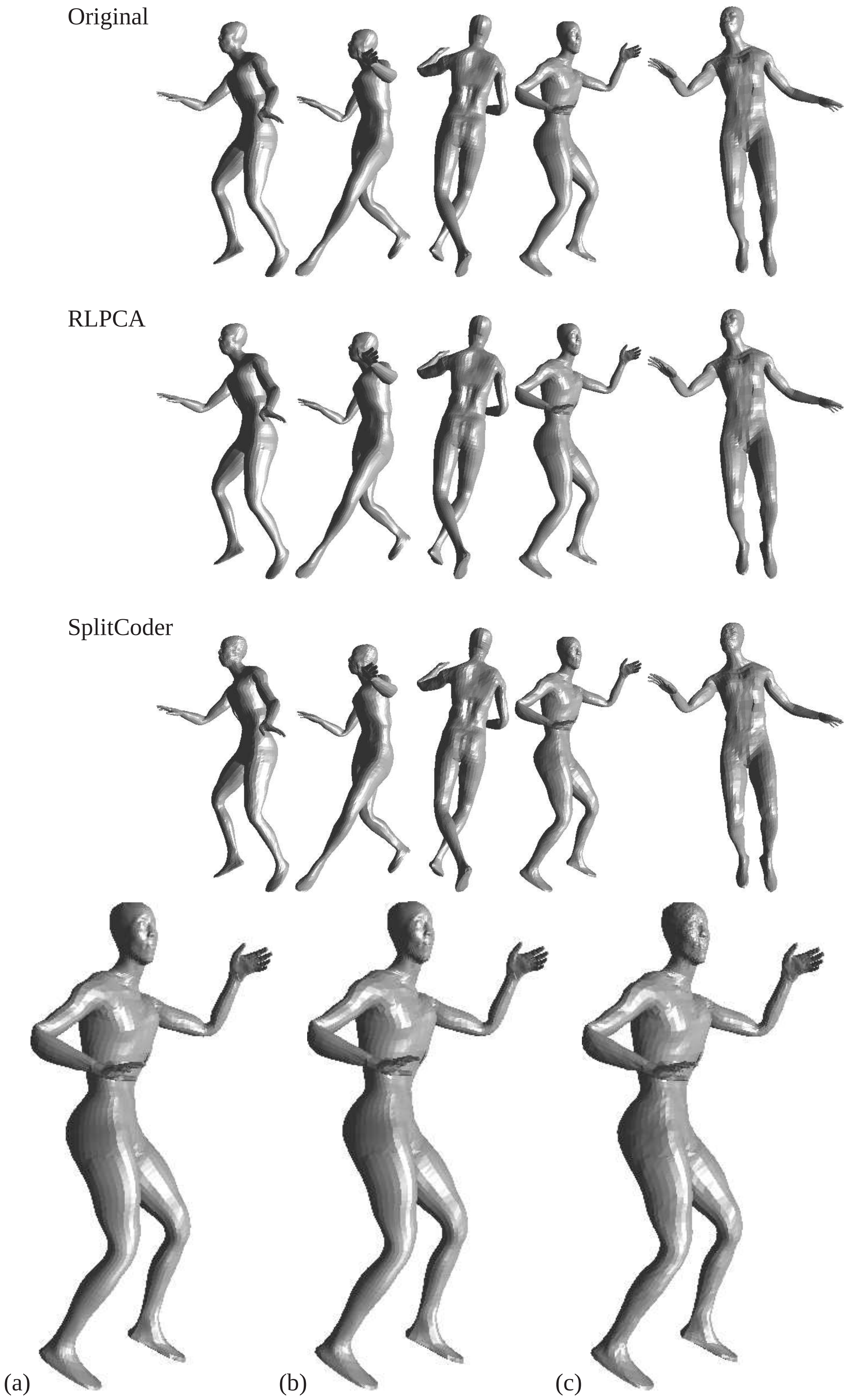
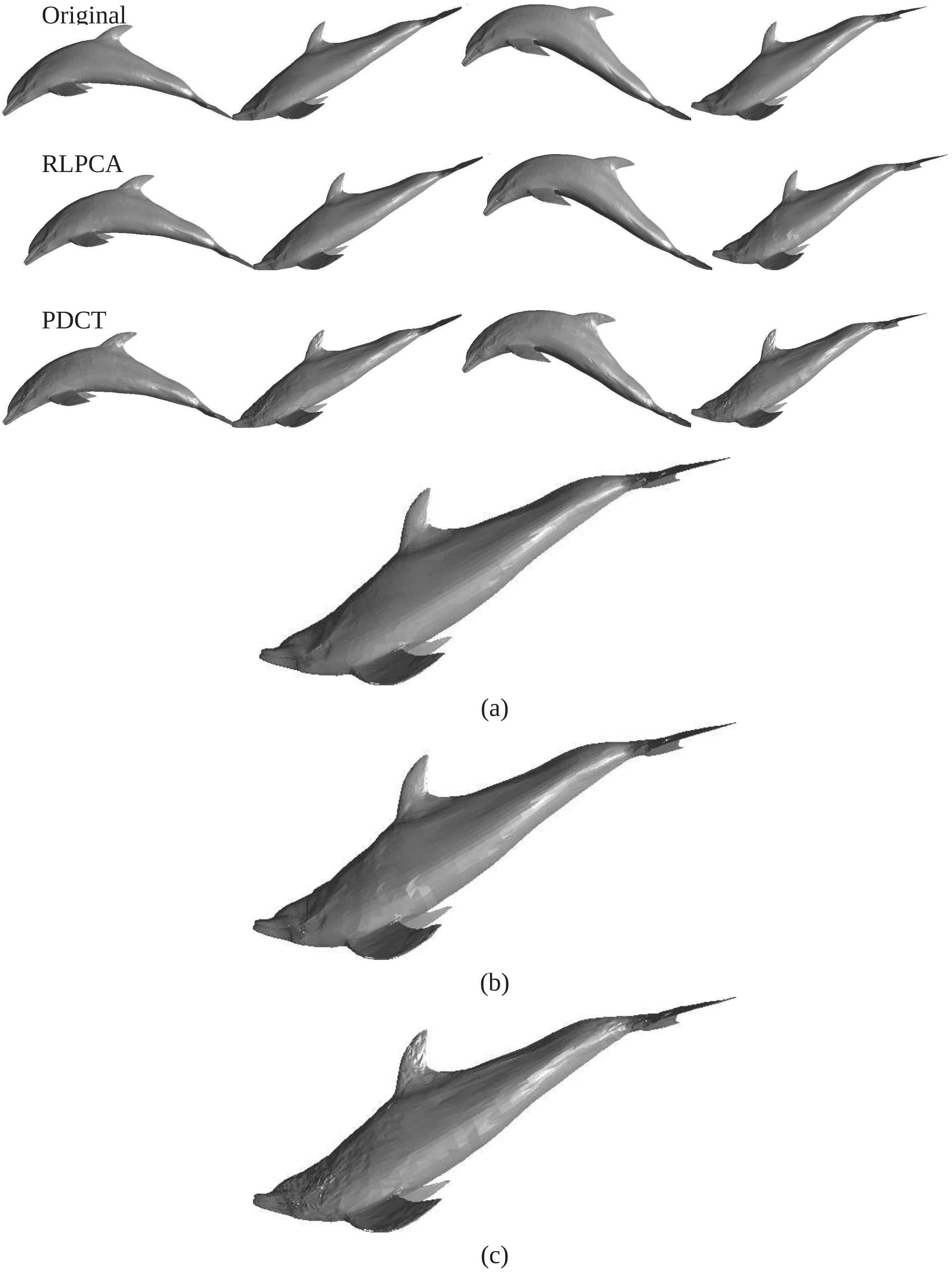


Figure 8.29 Comparison of RLPCA and SplitCoder at a similar bitrate. Frame 50: (a) original and (b) PDCT and (c) SplitCoder.

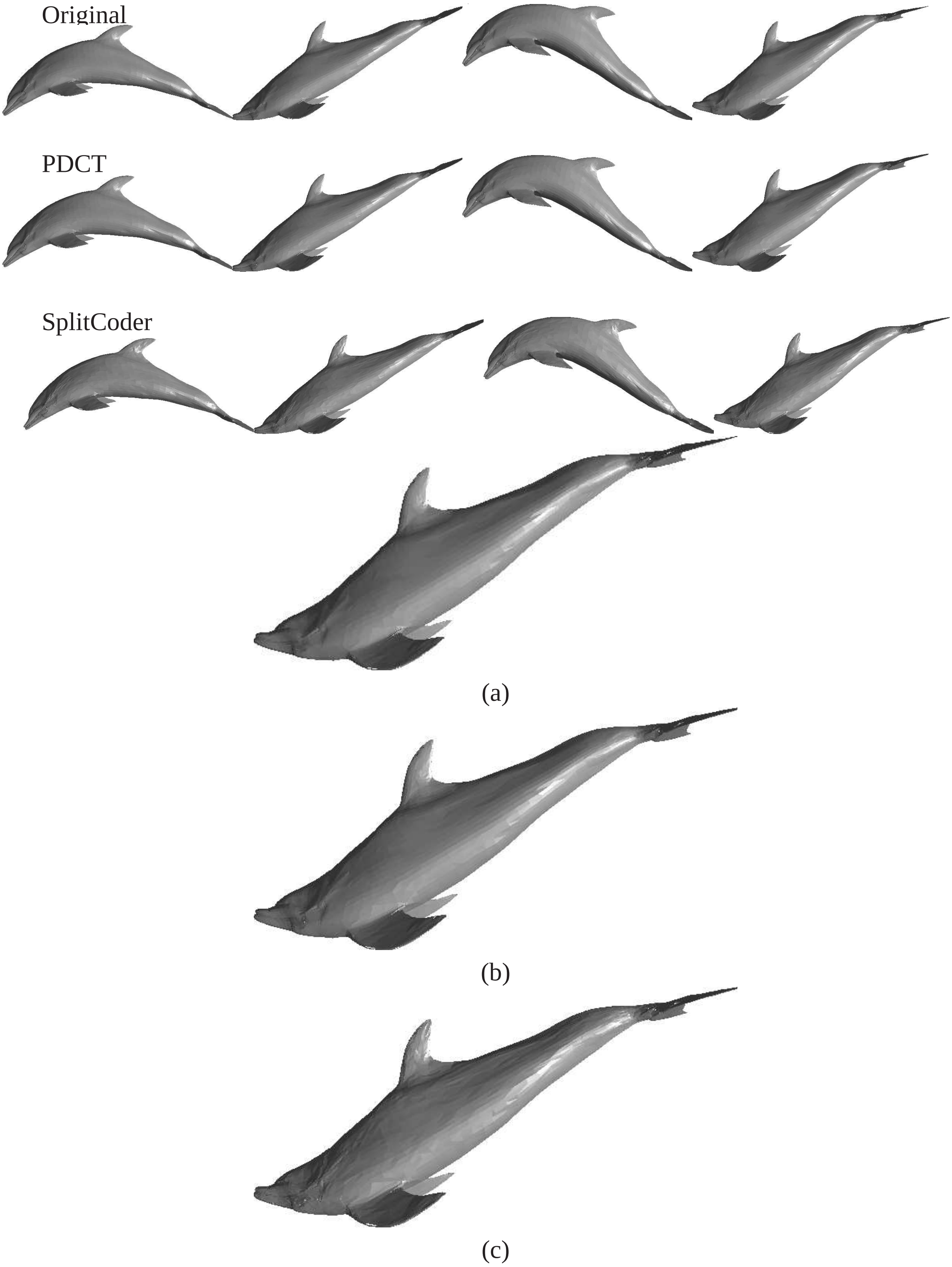


(a)

(b)

(c)

Figure 8.30 Comparison of RLPCA and PDCT at a similar bitrate. Frame 80: (a) original and (b) RLPCA and (c) PDCT.



(a)

(b)

(c)

Figure 8.31 Comparison of PDCT and SplitCoder at a similar bitrate. Frame 80: (a) original and (b) PDCT and (c) SplitCoder.

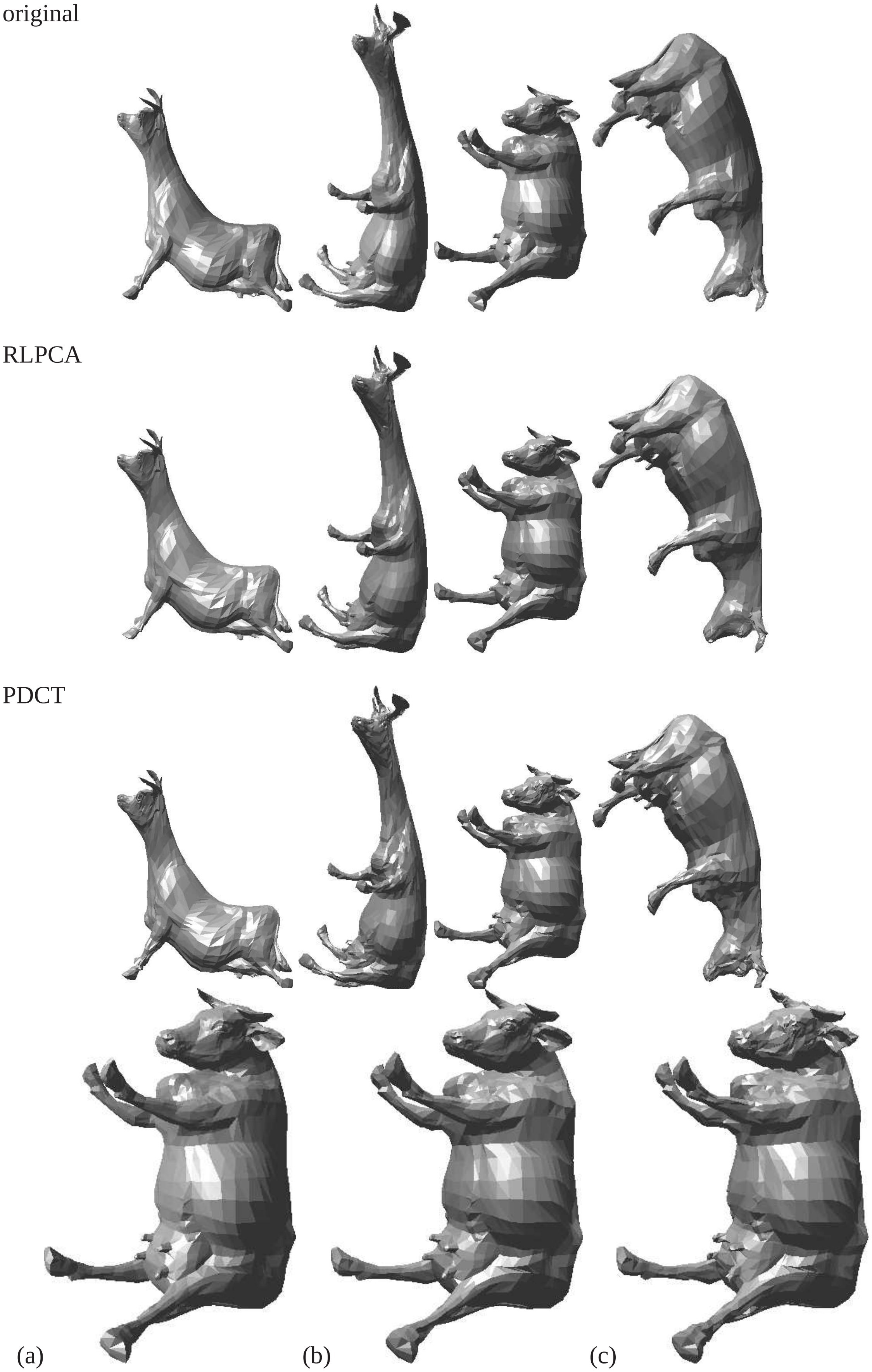


Figure 8.32 Comparison of RLPCA and PDCT at a similar bitrate. Frame 59: (a) original and (b) RLPCA and (c) PDCT.

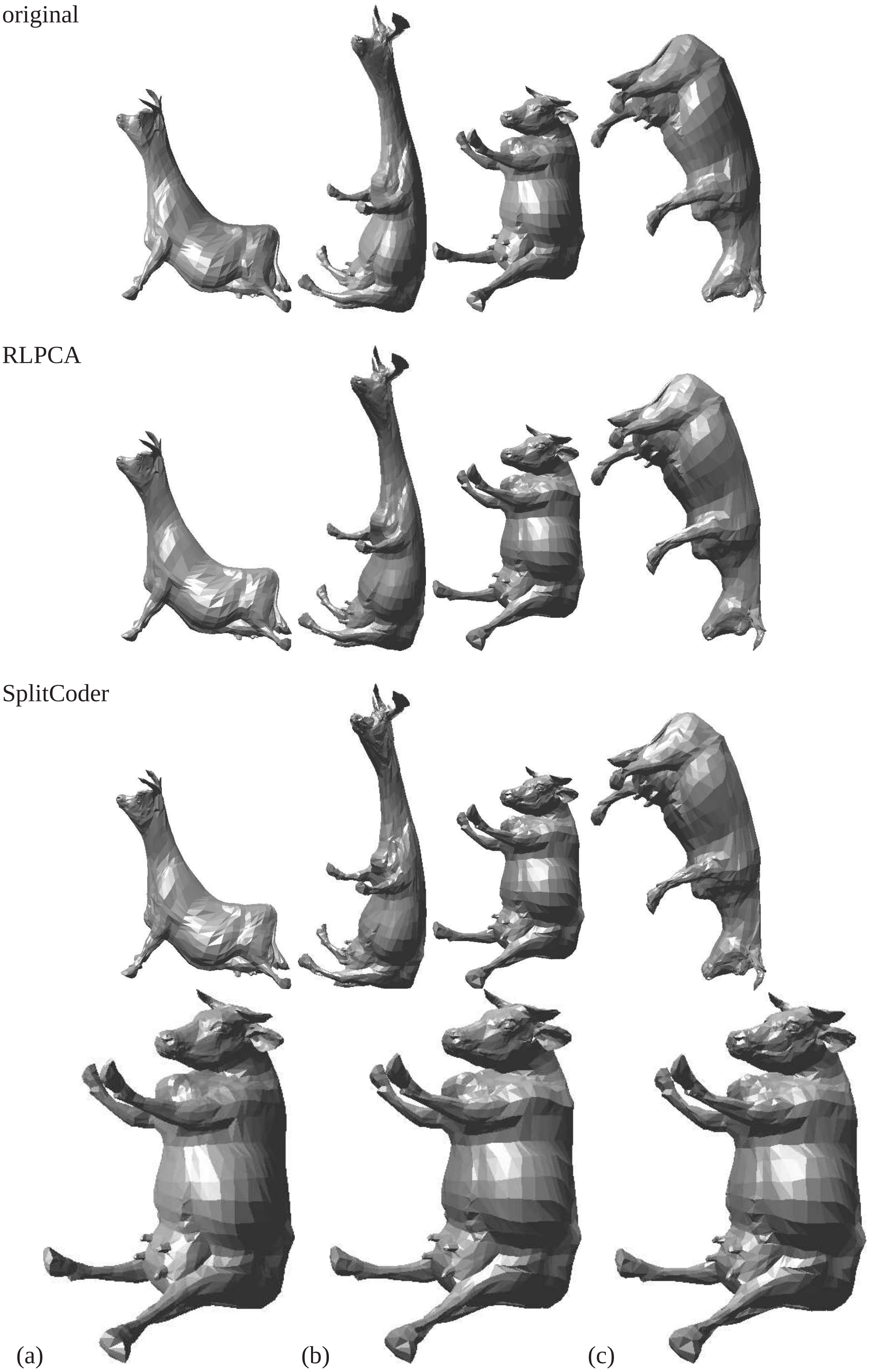


Figure 8.33 Comparison of RLPCA and SplitCoder at a similar bitrate. Frame 59: (a) original and (b) RLPCA and (c) SplitCoder.

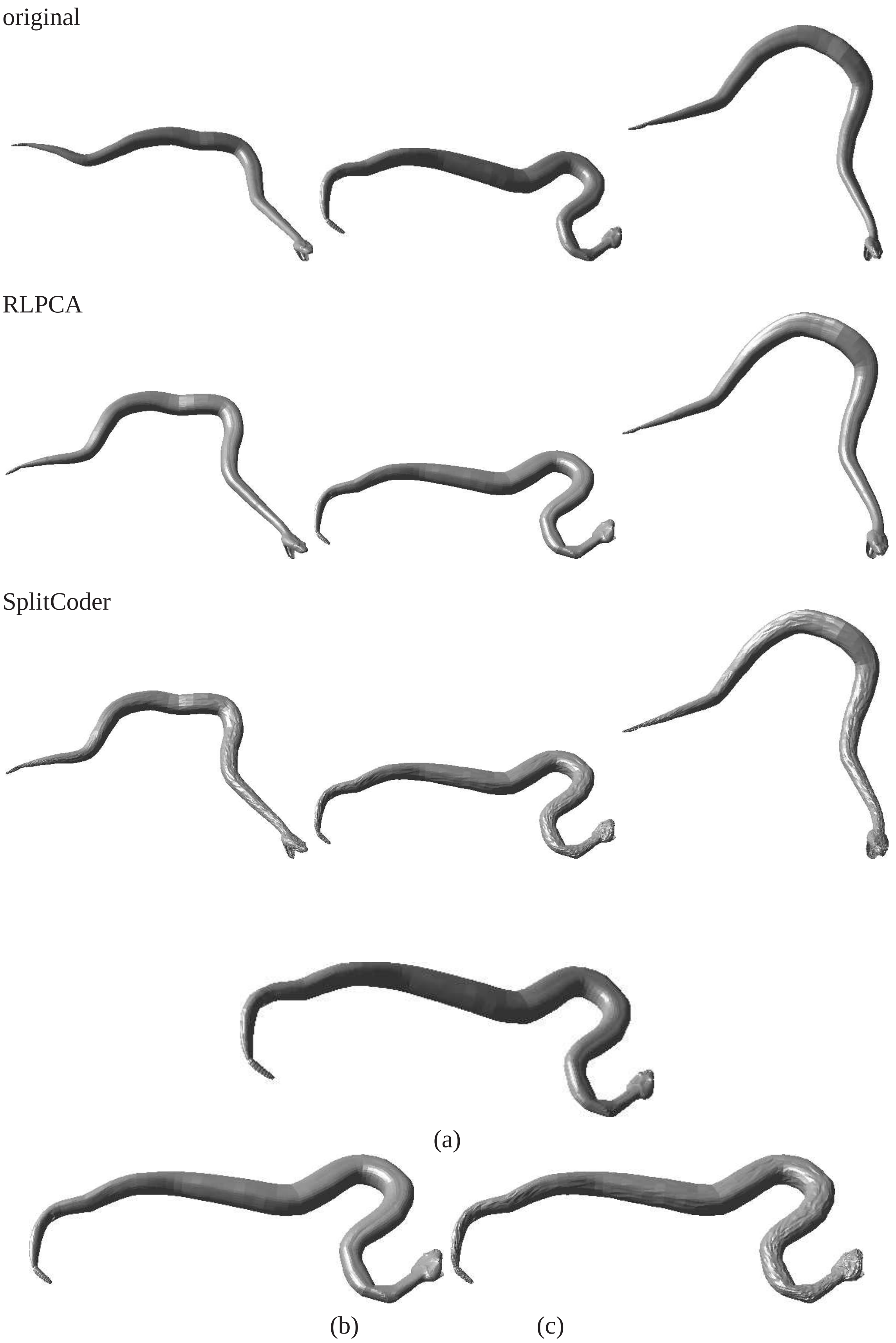


Figure 8.34 Comparison of RLPCA and SplitCoder at a similar bitrate. Frame 40: (a) original and (b) RLPCA and (c) SplitCoder.

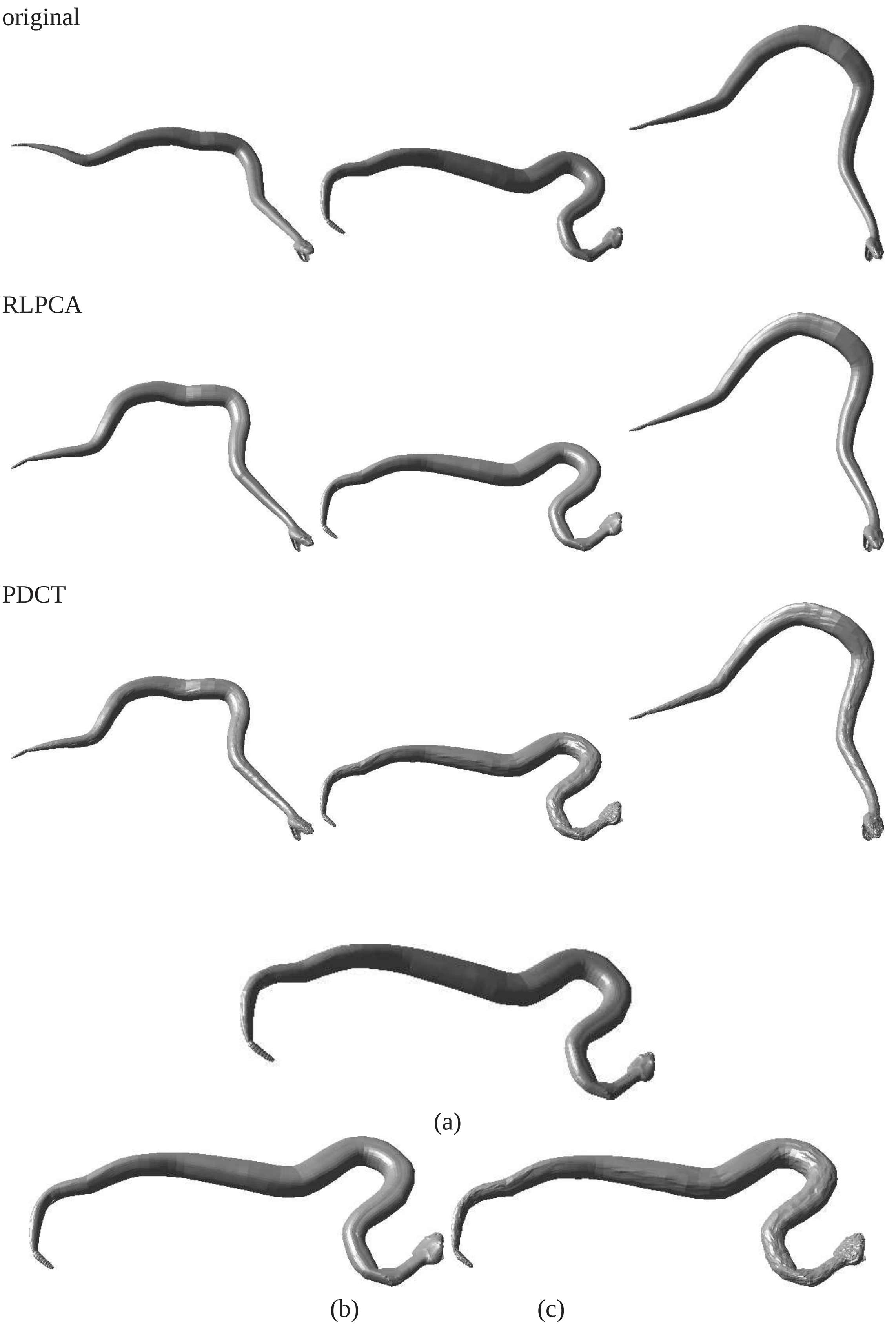


Figure 8.35 Comparison of RLPCA and PDCT at a similar bitrate. Frame 40: (a) original and (b) RLPCA and (c) PDCT.

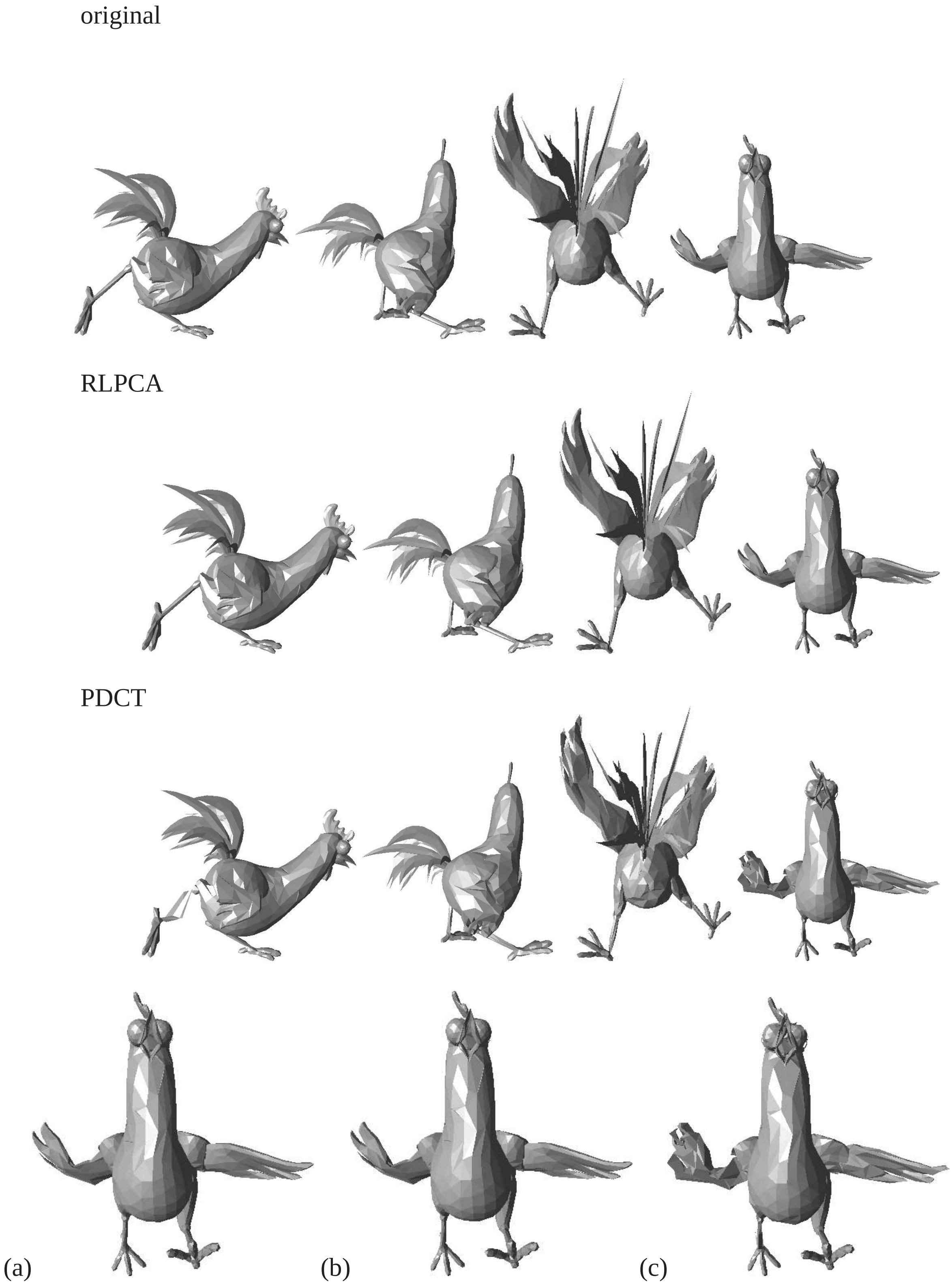


Figure 8.36 Comparison of RLPCA and PDCT at a similar bitrate. Frame 337: (a) original and (b) RLPCA and (c) PDCT.

CHAPTER 9

Conclusion and Future Work

This thesis addressed the problem of storage space and transmission of static and dynamic three-dimensional meshes.

We began with a review of static and animated mesh compression techniques, as well as some general compression methods which are used as a post-processing step in the current mesh compression algorithms. Then, we presented new compression techniques for static and animated meshes, and segmentation approaches for animated meshes.

For static mesh compression, we proposed a higher order prediction technique to encode the mesh geometry. We split the coding into tangential and normal components and we encoded them separately. For an improved encoding of the normal component, we developed higher order predictors based on surface fitting and sphere fitting, taking advantage of the large correlation between neighboring vertices.

In the context of animated mesh compression, we developed segmentation approaches to group the vertices of similar motion within all types of animation. Segmentation has the advantage that it decreases computational costs, it preserves global shape when lossy compression is performed and that it increases prediction efficiency, thereby achieving the best bit-rate compression. We have proposed three methods based on region growing, clustering and adaptive clustering, taking advantage of the spatial and temporal coherence. Visually as well as metrically, clustering approaches exhibit better partitioning into almost rigid groups than the region growing based algorithm. These approaches can be useful for many applications.

For animated mesh compression, we have developed three approaches:

The first approach, SplitCoder, is a connectivity-guided, predictive compression technique. The algorithm splits the vertex coordinates into parametric and curvature information. As result, the coordinate space exhibits a large clustering behavior, allowing more compact representation. Then, each component is encoded separately using predictive coding and quantization. The proposed predictor produces offsets whose distribution is close to zero and allows a low-entropy. This approach is simple, efficient and has a low computational cost which makes it well-suited for real-time compression and decompression. The drawback is that it does not support a progressive compression and is not efficient at very low bit rate. Introducing vector quantization might be very efficient, as it enables a higher quantization error in the tangential direction, resulting in a change of parametrization only and not in a larger geometric error. This primarily reduces the code length of the tangential components. Moreover, it will allow compression at very low bit rates.

The second approach, RLPCA, is based on the local PCA, which captures the linear correlations present in the datasets and represent mesh vertices using very few components and coefficients. Taking advantage of mesh clustering, which produces near-rigid clusters, we have transformed the original vertex coordinates into the local coordinate frame of their cluster, making each one slightly invariant to any deformation over time. Therefore, performing a PCA in these invariant groups of vertices leads to a more compact representation than the original data. In order to achieve an optimal tradeoff between the bitrate and the quality, we have introduced the rate distortion optimization for PCA, based on an incremental computation of the convex hull.

The third method, PDCT, is inspired from video coding. In contrast to the vertex based, predictive coding SplitCoder, the algorithm uses a cluster based predictive coding. During pre-processing, the animated mesh vertices are partitioned into clusters, similar to macroblocks in video coding. Next, we made predictions in the local coordinate frame. Then, we transformed the resulting delta vectors of each cluster in each frame into the frequency domain using a discrete cosine transform to reduce the code length, since the entropy in frequency domain is smaller than the entropy coding of delta vectors.

The effectiveness of prediction coding depends strongly on the clustering process. Indeed, if the vertices are well clustered then the motion relative to the LCF between two successive frames tends to be zero. The drawback of the proposed approach is that it does not support progressive transmission. Similar to the RLPCA, the clustering in PDCT produces clusters of different sizes and one has to choose a fixed number of coefficients to be discarded, for all clusters. Consequently, one may have over-fitting and under-fitting.

Typically, when very few coefficients are used, not all frames can be reconstructed at the same desired level of quality. Therefore, to overcome this drawback, one can introduce a rate distortion optimization that trades off between rate and the total distortion. Another alternative DCT based approach, is to combine the temporal-DCT with predictive coding in the local coordinates. This approach is more suitable for progressive transmission.

Note that both approach SplitCoder and PDCT take advantage of frame-to-frame coherence. SplitCoder processes vertex after vertex and PDCT cluster by cluster. In contrast, RLPCA considers the entire mesh sequence and analyzes the global coherence of each animated sub-meshes.

For a large sequence of meshes, the animation may become more complex and the clustering can produce poor prediction for some successive frames. Therefore, we propose to cut the sequence into short clips and update the clustering for each clip separately. In the PDCT approach, the first frame of each clip should be encoded spatially as an I-frame while in the PCA based approach, the RLPCA can be directly performed in each clip.

According to the experimental results, RLPCA yields a good rate/distortion ratio, and significantly outperforms PDCT and SplitCoder. Furthermore, the compression parameters (the number of clusters, the number of PCA components and DCT coefficients and the level of quantization) strongly affect compression performance.

On the other hand, the three approaches are competitive when compared to the state-of-the-art techniques. They even provide a significant improvement in compression ratio over some existing coders.

Our methods are significantly better than the KG method and comes close or even better than the CPCA and wavelet based methods (TLS and AWC). Also, we achieve rate distortion performance better than the standard TG which exploits the spatial coherence only, the LPC which exploits the temporal cohence only, KG (PCA+LPC) and the standard PCA which approximates the global linearity and it is less effective for nonlinear animation.

The proposed approaches also perform better than the predictive techniques (Dynapack andmaverg+angle). For PDCT, the improvement is due to the clustering of the model into rigid parts making the prediction more efficient in the local rather than the world coordinates and to the further DCT coding which leads to a significant reduction in the overall entropy. SplitCoder's efficiency is due the prediction and the quantization which are performed in the local space. This shows high clustering behavior of the vertex coordinates as well as the coordinates of delta vectors. Thereby, a significant reduction in the overall entropy is achieved.

In the end, it is important to note that our coders PDCT and RLPCA as well as the clustering process can be applicable to point-based models also, regardless of how the animation is generated.

Another challenging problem is to develop a new error measurement. During our experiments, we observed that sometimes one may have a larger error but have better visual quality. It is possible that, for a similar error metric *da*, the visual quality of both the original and the reconstructed frames, are different. Recently, some work has been done in this direction, but there is till much to be done. We plan to develop a metric that takes in to consideration both the spatial and the temporal information, locally and globally over all sequences.

To close this dissertation, our experience is the following:

For the compression of static meshes, the higher order prediction scheme based on polynomial graph function fitting shows to be efficient for encoding of mesh geometry.

For the segmentation process, if the time is critical in your application then the clustering approach would be efficient to break down the animated object into rigid parts, otherwise the adaptive clustering approach is used. Very often, it is desirable for many application to have a segmentation that produces automatically the number of segments. Typically, in the context of compression, in static clustering, we recommend to use a number between 18 and 25 clusters. This number should increase with the complexity of animation to obtain rigid parts.

For compression of mesh sequences, if all frames are not known then predictive based compression is the most suited algorithm. If all frames are known, and a low bit rate is desired, then RLPCA is the preferred approach. This algorithm is cheapest, most efficient to use and its decompression process is very fast. The quantization level is fixed at 16 bit for the points that construct the local coordinate frames and 12 for PCA coefficients.

Bibliography

[1] http://www710.univ-lyon1.fr/ hbriceno//research/geometryvideos/. 130

[2] Mpeg-4 standard. http://www.mpeg-4.com. In *http://www.mpeg-4.com*. 32

[3] Marc Alexa and Wolfgang Müller. Representing animations by principal components. *Computer graphics Forum*, 19(3):411–426, 2000. ISSN 0167-7055. 43, 114, 125, 142

[4] P. Alliez and M. Desbrun. Valence-driven connectivity encoding for 3D meshes. In *Eurographics ’01 Conference Proceedings*, pages 480–489, 2001. 28

[5] Pierre Alliez and Mathieu Desbrun. Progressive compression for lossless transmission of triangle meshes. In *SIGGRAPH ’01: Proceedings of the 28th annual conference on Computer graphics and interactive techniques*, pages 195–202. ACM, 2001. 37

[6] Pierre Alliez and Craig Gotsman. Recent advances in compression of 3d meshes. In *Advances in Multiresolution for Geometric Modelling serie*, pages 3–26. Springer, Berlin, 2005. 21, 22, 34, 37

[7] R. Amjoun, R. Sondershaus, and W. Straßer. Compression of complex animated meshes. *Computer Graphics International 2006 Conference, LNCS by Springer*, 4035:606–613, June 2006. 4, 5, 6, 7, 8, 47

[8] Rachida Amjoun. Compression of 3d dynamic mesh sequences. Technical report, Wilhelm Schickard Institute for Computer Science, Graphical-Interactive Systems (WSI/GRIS), University of Tübingen, 2003. 3, 6, 8

[9] Rachida Amjoun. Exploiting temporal and spatial coherence in coordinate systems. Technical Report WSI-2006-14, Wilhelm Schickard Institute for Computer Science, Graphical-Interactive Systems (WSI/GRIS), University of Tübingen, 2006. 8

[10] Rachida Amjoun and Wolfgang Straßer. Efficient compression of 3d dynamic mesh sequences. In *Journal of the WSCG*, 2007. 4, 5, 6, 7, 8, 45, 47

[11] Rachida Amjoun and Wolfgang Straßer. Encoding Animated Meshes in Local Coordinates . In *International Conference on Cyberworlds (CW'07), NSAGEM*, pages 437–446, October 2007. 4, 6, 8

[12] Rachida Amjoun and Wolfgang Straßer. Predictive-spectral compression of dynamic 3d meshes. In *2nd International Conference on Computer Graphics Theory (Grapp)*, Mars 2007. 4, 7, 8

[13] Rachida Amjoun and Wolfgang Straßer. Predictive-dct coding for 3d mesh sequences compression. *Journal of Virtual Reality and Broadcasting*, 5(6), July 2008. 4, 7, 8

[14] Rachida Amjoun and Wolfgang Straßer. Segmentation and compression of animated meshes. Technical report, Wilhelm Schickard Institute for Computer Science, Graphical-Interactive Systems (WSI/GRIS), University of Tübingen, 2009. 6, 8

[15] Rachida Amjoun and Wolfgang Straßer. Single-rate near lossless compression of animated geometry. *Journal of Computer-Aided Design*, 2009. 4, 6, 8

[16] Marco Attene, Bianca Falcidieno, and Michela Spagnuolo. Hierarchical mesh segmentation based on fitting primitives. *Vis. Comput.*, 22(3):181–193, 2006. 45

[17] Chandrajit L Bajaj, Valerio Pascucci, and Guozhong Zhuang. Single resolution compression of arbitrary triangular meshes with properties. In *DCC '99: Proceedings of the Conference on Data Compression*, page 247, Washington, DC, USA, 1999. IEEE Computer Society. 28

[18] Ulug Bayazit, Ozgur Orcay, Umut Konur, and Fikret S. Gurgen. Predictive vector quantization of 3-d mesh geometry by representation of vertices in local coordinate systems. *J. Vis. Comun. Image Represent.*, 18(4):341–353, 2007. 30

[19] 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, 2003. 44

[20] Bernard Chazelle, David P. Dobkin, Nadia Shouraboura, and Ayellet Tal. Strategies for polyhedral surface decomposition: an experimental study. In *SCG '95: Proceedings of the eleventh annual symposium on Computational geometry*, pages 297–305, New York, NY, USA, 1995. ACM. 45

[21] Yizong Cheng. Mean shift, mode seeking, and clustering. *IEEE Trans. Pattern Anal. Mach. Intell.*, 17(8):790–799, 1995. 46

[22] Peter H. Chou and Teresa H. Meng. Vertex data compression through vector quantization. *IEEE Transactions on Visualization and Computer Graphics*, 8(4):373–382, 2002. 30, 33

[23] Mike M. Chow. Optimized geometry compression for real-time rendering. In *VIS '97: Proceedings of the 8th conference on Visualization '97*, pages 347–ff., Los Alamitos, CA, USA, 1997. IEEE Computer Society Press. 24, 30

[24] Paolo Cignoni, Claudio Rocchini, and Roberto Scopigno. Metro: Measuring error on simplified surfaces. *Comput. Graph. Forum*, 17(2):167–174, 1998. 52

[25] D. Cohen-Or, R. Cohen, and R. Irony. Multi-way geometry encoding. 2002. Technical Report. 3, 57, 67

[26] G Collins and A Hilton. A rigid transform basis for animation compression and level of detail. In *Proceedings of the IMA Conference on Vision, Video and Graphics*, pages 21–28, 2005. 39

[27] Dorin Comaniciu and Peter Meer. Mean shift: A robust approach toward feature space analysis. volume 24, pages 603–619. IEEE Computer Society, 2002. 46

[28] Michael Deering. Geometry compression. In *SIGGRAPH '95: Proceedings of the 22nd annual conference on Computer graphics and interactive techniques*, pages 13–20, New York, NY, USA, 1995. ACM. 24, 30

[29] N. Faller. An adaptive system for data compression. *Record of the 7th Asilomar Conference on Circuits, Systems, and Computers*, pages 593–597, 1973. 15

[30] R.G. Gallagher. Variations on a theme by huffman. *IEEE Transactions on Information Theory*, IT-24(6):668–674, 1978. 13, 15

[31] Pierre-Marie Gandoin and Olivier Devillers. Progressive lossless compression of arbitrary simplicial complexes. In *SIGGRAPH '02: Proceedings of the 29th annual conference on Computer graphics and interactive techniques*, pages 372–379, New York, NY, USA, 2002. ACM. 37

[32] Michael Garland, Andrew Willmott, and Paul S. Heckbert. Hierarchical face clustering on polygonal surfaces. In *I3D '01: Proceedings of the 2001 symposium on Interactive 3D graphics*, pages 49–58, New York, NY, USA, 2001. ACM. 45

[33] Arthur D. Gregory, Andrei State, Ming C. Lin, Dinesh Manocha, and Mark A. Livingston. Interactive surface decomposition for polyhedral morphing. *The Visual Computer*, 15(9):453–470, 1999. 45

[34] Lavoue Guillaume, Dupont Florent, and Baskurt Atilla. Curvature tensor based triangle mesh segmentation with boundary rectification. In *CGI '04: Proceedings of the Computer Graphics International*, pages 10–17, Washington, DC, USA, 2004. IEEE Computer Society. 45

[35] S. Gumhold and W. Straßer. Real time compression of triangle mesh connectivity. In *SIGGRAPH '98 Conference Proceedings*, pages 133–140, 1998. 25

[36] Stefan Gumhold and Rachida Amjoun. Higher order prediction for geometry compression. In *International Conference On Shape Modelling And Applications*, pages 59–68, 2003. 3, 6, 8

[37] Johannes Günther, Heiko Friedrich, Ingo Wald, Hans-Peter Seidel, and Philipp Slusallek. Ray tracing animated scenes using motion decomposition. *Computer Graphics Forum*, 25(3):517–525, September 2006. (Proceedings of Eurographics). 4, 46

[38] Sumit Gupta, Kuntal Sengupta, and Ashraf A. Kassim. Compression of dynamic 3d geometry data using iterative closest point algorithm. *Comput. Vis. Image Underst.*, 87(1-3):116–130, 2002. 4, 40, 46, 47, 70

[39] Igor Guskov and Andrei Khodakovsky. Wavelet compression of parametrically coherent mesh sequences. In *ACM SIG./Eurog. symp. on Comput. anim.*, pages 183–192, 2004. ISBN. 44, 114, 125, 142

[40] Igor Guskov, Kiril Vidimce, Wim Sweldens, and Peter Schröder. Normal meshes. In Kurt Akeley, editor, *Siggraph 2000, Computer Graphics Proceedings*, pages 95–102. ACM Press / ACM SIGGRAPH / Addison Wesley Longman, 2000. 37

[41] Bruce Hendrickson and Robert Leland. A multilevel algorithm for partitioning graphs. In *Supercomputing '95: Proceedings of the 1995 ACM/IEEE conference on Supercomputing (CDROM)*, page 28, New York, NY, USA, 1995. ACM. 40, 47, 70

[42] Hugues Hoppe. Progressive meshes. In *Proceedings of the 23rd annual conference on Computer graphics and interactive techniques*, pages 99–108. ACM Press, 1996. ISBN 0-89791-746-4. 23, 35, 36

[43] Hugues Hoppe. Efficient implementation of progressive meshes. *Computers and Graphics*, 22(1):27–36, 1998. 37

[44] David A. Huffman. A method for the construction of minimum-redundancy codes. *Proceedings of the Institute of Radio Engineers*, 40(9):1098–1101, 9 1952. 12

[45] Lawrence Ibarria and Jarek Rossignac. Dynapack: space-time compression of the 3d animations of triangle meshes with fixed connectivity. In *SIG./Eurog. Symp. on Computer Animation*, pages 126–135. Eurographics Association, 2003. 100, 122, 142

[46] Keisuke Inoue, Takayuki Itoh, Atsushi Yamada, Tomotake Furuhata, and Kenji Shimada. Face clustering of a large-scale cad model for surface mesh generation. *Computer-Aided Design*, 33(3):251–261, 2001. 45

[47] M. Isenburg and P. Alliez. Compressing polygon mesh geometry with parallelogram prediction. In *IEEE Visualization '02 Conference Proceedings*, pages 141–146, 2002. 34

[48] Martin Isenburg and Stefan Gumhold. Out-of-core compression for gigantic polygon meshes. *ACM Trans. Graph.*, 22(3):935–942, 2003. ISSN 0730-0301. 37

[49] Martin Isenburg, Peter Lindstrom, and Jack Snoeyink. Lossless compression of predicted floating-point geometry. *Computer-Aided Design*, 37(8):869–877, 2005. 34

[50] Martin Isenburg and Jack Snoeyink. Mesh collapse compression. In *SCG '99: Proceedings of the fifteenth annual symposium on Computational geometry*, pages 419–420. ACM, 1999. 28

[51] Martin Isenburg and Jack Snoeyink. Face fixer: compressing polygon meshes with properties. In *SIGGRAPH '00: Proceedings of the 27th annual conference on Computer graphics and interactive techniques*, pages 263–270. ACM Press/Addison-Wesley Publishing Co., 2000. 26

[52] Martin Isenburg and Jack Snoeyink. Spirale reversi: reverse decoding of the edgebreaker encoding. *Computational Geometry*, 20(1-2):39–52, 2001. 26

[53] Martin Isenburg and Jack Snoeyink. Early-split coding of triangle mesh connectivity. In *GI '06: Proceedings of Graphics Interface 2006*, pages 89–97, Toronto, Ont., Canada, Canada, 2006. Canadian Information Processing Society. 28

[54] Doug L. James and Christopher D. Twigg. Skinning mesh animations. *ACM Transactions on Graphics (SIGGRAPH 2005)*, 24(3), August 2005. 4, 45, 46, 47, 70

[55] Euee S. Jang, James D. K. Kim, Seok Yoon Jung, Mahnjin Han, and Sang Oak Woo. Interpolator data compression for mpeg-4 animation. *IEEE Trans. Circuits Syst. Video Techn.*, 14(7):989–1008, 2004. 42

[56] Yang J.H., Kim C.S., and Lee S.U. Compression of 3-d triangle mesh sequences based on vertex-wise motion vector prediction. *Cir. Sys Video*, 12(12):1178–1184, December 2002. ISSN 1051-8215. 100, 122

[57] Zhongping Ji, Ligang Liu, Zhonggui Chen, and Guojin Wang. Easy mesh cutting. *Computer Graphis Forum*, 25(3):283–291, 2006. 45

[58] Dan Julius, Vladislav Kraevoy, and Alla Sheffer. D-charts: Quasi-developable mesh segmentation. In *Computer Graphics Forum, Proceedings of Eurographics 2005*, volume 24, pages 581–590, Dublin, Ireland, 2005. Eurographics, Blackwell. 45

[59] Felix Kälberer, Konrad Polthier, Ulrich Reitebuch, and Max Wardetzky. Freelence - coding with free valences. *Computer Graphics Forum*, 24(3):469–478, 2005. 28, 34

[60] Tapas Kanungo, David M. Mount, Nathan S. Netanyahu, Christine D. Piatko, Ruth Silverman, and Angela Y. Wu. An efficient k-means clustering algorithm: Analysis and implementation. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 24(7):881–892, 2002. 47, 70

[61] Zachi Karni and Craig Gotsman. Spectral compression of mesh geometry. In *SIGGRAPH '00: Proceedings of the 27th annual conference on Computer graphics and interactive techniques*, pages 279–286, 2000. 34, 45

[62] Zachi Karni and Craig Gotsman. Compression of soft-body animation sequences. *Computer and Graphics*, 28(1):25–34, 2004. ISSN 0097-8493. 43, 114, 125, 128, 142, 149

[63] Matthias Kautzner Peter Eisert Thomas Wiegand Karsten Muller, Aljoscha Smolic. Predictive compression of dynamic 3d meshes. In *International Conference on Image Processing*, pages 621–624, 2005. 40, 100, 122

[64] Sagi Katz, George Leifman, and Ayellet Tal. Mesh segmentation using feature point and core extraction. *The Visual Computer*, 21(8-10):649–658, 2005. 45, 46

[65] Sagi Katz and Ayellet Tal. Hierarchical mesh decomposition using fuzzy clustering and cuts. In *SIGGRAPH '03: ACM SIGGRAPH 2003 Papers*, pages 954–961, New York, NY, USA, 2003. ACM. 45

[66] Andrei Khodakovsky and Igor Guskov. Compression of normal meshes. *Geometric Modeling for Scientific Visualization. Springer-Verlag, 2003.*, 2003. 37

[67] Andrei Khodakovsky, Peter Schröder, and Wim Sweldens. Progressive geometry compression. In Kurt Akeley, editor, *Siggraph 2000, Computer Graphics Proceedings*, pages 271–278. ACM Press / ACM SIGGRAPH / Addison Wesley Longman, 2000. 37

[68] Donald E. Knuth. Dynamic huffman coding. *J. Algorithms*, 6(2):163–180, 1985. 15

[69] Vladislav Kraevoy and Alla Sheffer. Cross-parameterization and compatible remeshing of 3d models. *ACM Trans. Graph.*, 23(3):861–869, 2004. 45

[70] B. Kronrod and C. Gotsman. Optimized compression of triangle mesh geometry using prediction trees. In *International Symposium on 3D Data Processing Visualization and Transmission*, 2002. 28, 33

[71] Yu-Kun Lai, Qian-Yi Zhou, Shi-Min Hu, and Ralph R. Martin. Feature sensitive mesh segmentation. In *SPM '06: Proceedings of the 2006 ACM symposium on Solid and physical modeling*, pages 17–25, New York, NY, USA, 2006. ACM. 45

[72] Guillaume Lavoué, Florent Dupont, and Atilla Baskurt. A new cad mesh segmentation method, based on curvature tensor analysis. *Computer-Aided Design*, 37(10):975–987, September 2005. 45

[73] Eung-Seok Lee and Hyeong-Seok Ko. Vertex data compression for triangular meshes. In *PG 0́0: Proceedings of the 8th Pacific Conference on Computer Graphics and Applications*, page 225, Washington, DC, USA, 2000. IEEE Computer Society. 30, 32, 33, 52, 55

[74] H. Lee, P. Alliez, and M. Desbrun. Angle-analyzer: A triangle-quad mesh codec. In *Eurographics '02 Conference Proceedings*, pages 383–392, 2002. 34, 67

[75] Tong-Yee Lee, Ping-Hsien Lin, Shaur-Uei Yan, and Chun-Hao Lin. Mesh decomposition using motion information from animation sequences: Animating geometrical models. *Comput. Animat. Virtual Worlds*, 16(3-4):519–529, 2005. 46

[76] Tong-Yee Lee, Yu-Shuen Wang, and Tai-Guang Chen. Segmenting a deforming mesh into near-rigid components. *Vis. Comput.*, 22(9):729–739, 2006. 46, 47, 70

[77] Jerome Edward Lengyel. Compression of time-dependent geometry. In *Proceedings of ACM symposium on Interactive 3D graphics*, pages 89–95. ACM Press, 1999. ISBN 1-58113-082-1. 4, 39, 46, 47, 70

[78] Jiankun Li, Jin Li, and C. C. Jay Kuo. Progressive compression of 3d graphic models. In *International Conference on Multimedia Computing and Systems*, pages 135–142, 1997. 37

[79] Y. Linde, A. Buzo, and R. M. Gray. An algorithm for vector quantizer design. *IEEE Transactions on communications*, 28(1):84–95, January 1980. 30

[80] Rong Liu and Hao Zhang. Segmentation of 3d meshes through spectral clustering. In *PG '04: Proceedings of the Computer Graphics and Applications, 12th Pacific Conference*, pages 298–305, Washington, DC, USA, 2004. IEEE Computer Society. 45

[81] Rong Liu and Hao Zhang. Mesh segmentation via spectral embedding and contour analysis. *Computer Graphics Forum (Special Issue of Eurographics 2007)*, 26(3):385–394, 2007. 45

[82] Stuart P. Lloyd. Least squares quantization in PCM. *IEEE Transactions on Information Theory*, IT-28(2):129–137, March 1982. 43, 45

[83] Khaled Mamou, Titus Zaharia, and Françoise J. Prêteux. A preliminary evaluation of 3d mesh animation coding techniques. In *Mathematical Methods in Pattern and Image Analysis*, volume 5916, pages 44–55, August 2005. 22

[84] Khaled Mamou, Titus Zaharia, and Françoise Prêteux. A skinning approach for dynamic 3d mesh compression. *Computer Animation Virtual Worlds*, 17, 2006. ISSN 1546-4261. 4, 45, 46

[85] Khaled Mamou, Titus B. Zaharia, and Françoise J. Prêteux. Multi-chart geometry video: A compact representation for 3d animations. In *3DPVT*, pages 711–718, 2006. 47, 70

[86] Alan P. Mangan and Ross T. Whitaker. Partitioning 3d surface meshes using watershed segmentation. *IEEE Transactions on Visualization and Computer Graphics*, 5(4):308–321, 1999. 45

[87] K. Muller, A. Smolic, M. Kautzner, P. Eisert, and T. Wiegand. Rate-distortion-optimized predictive compression of dynamic 3d mesh sequences. *Signal Process: Image Communication*, 21(9):812–828, October 2006. 40

[88] D. L. Page, A.F. Koschan, and M. A. Abidi. Perception-based 3d triangle mesh segmentation using fast marching watersheds. *Computer Vision and Pattern Recognition, IEEE Computer Society Conference on*, 2:27, 2003. 45

[89] Renato Pajarola and Jarek Rossignac. Compressed progressive meshes. *IEEE Transactions on Visualization and Computer Graphics*, 6(1):79–93, 2000. 36

[90] Xiang Pan, Xiuzi Ye, and Sanyuan Zhang. 3d mesh segmentation using a two-stage merging strategy. In *CIT*, pages 730–733, 2004. 45

[91] Marais Patrick, James Gain, and Dave Shreiner. Distance-ranked connectivity compression of triangle meshes. In *Computer Graphics Forum*, volume 26, pages 871–876, 2007. 34

[92] Frederic Payan and Marc Antonini. Wavelet-based compression of 3d mesh sequences. In *Proceedings of IEEE ACIDCA-ICMI'2005*, Tozeur, Tunisia, november 2005. 44, 157

[93] Jingliang Peng and C.-C. Jay Kuo. Geometry-guided progressive lossless 3d mesh coding with octree (ot) decomposition. *ACM Trans. Graph.*, 24(3):609–616, 2005. 37

[94] J.L. Peng, C.S. Kim, and C.C.J. Kuo. Technologies for 3d mesh compression: A survey. *JVCIR*, 16(6):688–733, December 2005. 22, 34

[95] Jovan Popović and Hugues Hoppe. Progressive simplicial complexes. In *SIGGRAPH '97: Proceedings of the 24th annual conference on Computer graphics and interactive techniques*, pages 217–224, New York, NY, USA, 1997. ACM Press/Addison-Wesley Publishing Co. 37

[96] J. Rossignac. Edgebreaker: Connectivity compression for triangle meshes. *IEEE Transactions on Visualization and Computer Graphics*, 5(1):47–61, 1999. 25

[97] J. Rossignac. *Surface simplification and 3D geometry compression*. Chapter 54 in Handbook of Discrete and Computational Geometry 2004. 22

[98] Jarek Rossignac and Andrzej Szymczak. Wrap&zip decompression of the connectivity of triangle meshes compressed with edgebreaker. *Comput. Geom. Theory Appl.*, 14(1-3):119–135, 1999. 26

[99] Amir Said. Introduction to arithmetic coding: theory and practice. Technical report, 2004. 17

[100] Pedro V. Sander, John Snyder, Steven J. Gortler, and Hugues Hoppe. Texture mapping progressive meshes. In *SIGGRAPH '01: Proceedings of the 28th annual conference on Computer graphics and interactive techniques*, pages 409–416, New York, NY, USA, 2001. ACM. 45

[101] Pedro V. Sander, Zoë J. Wood, Steven J. Gortler, John Snyder, and Hugues Hoppe. Multi-chart geometry images. In *Symposium on Geometry Processing*, pages 146–155, 2003. 45

[102] Mirko Sattler, Ralf Sarlette, and Reinhard Klein. Simple and efficient compression of animation sequences. In *ACM SIG./Eurog. symp. on Computer animation*, pages 209–217. ACM Press, 2005. ISBN 1-7695-2270-X. 4, 43, 45, 46, 47, 70, 114, 125, 128, 142, 149

[103] Khalid Sayood. *Introduction to data compression*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1996. 12

[104] Ariel Shamir. A formulation of boundary mesh segmentation. In *3DPVT '04: Proceedings of the 3D Data Processing, Visualization, and Transmission, 2nd International Symposium*, pages 82–89, Washington, DC, USA, 2004. IEEE Computer Society. 45

[105] Ariel Shamir. A survey on mesh segmentation techniques. *Computer Graphics Forum*, 27(6):1539–1556, September 2008. 45

[106] Ariel Shamir and Valerio Pascucci. Temporal and spatial level of details for dynamic meshes. In *VRST '01: Proceedings of the ACM symposium on Virtual reality software and technology*, pages 77–84, New York, NY, USA, 2001. ACM. 44

[107] Jerome M. Shapiro. Embedded image coding using zerotrees of wavelet coefficients. pages 124–141, San Francisco, CA, USA, 2001. Morgan Kaufmann Publishers Inc. 37

[108] Alla Sheffer. Model simplification for meshing using face clustering. *Computer-Aided Design*, 33(13):925–934, 2001. 45

[109] Dinesh Shikhare, Sushil Bhakar, and Sudhir P. Mudur. Compression of large 3d engineering models using automatic discovery of repeating geometric features. In *VMV '01: Proceedings of the Vision Modeling and Visualization Conference 2001*, pages 233–240. Aka GmbH, 2001. 34

[110] Shymon Shlafman, Ayellet Tal, and Sagi Katz. Metamorphosis of polyhedral surfaces using decomposition. In *Computer Graphics Forum*, pages 219–228, 2002. 45

[111] A. Smolic, R. Sondershaus, N. Stefanoski, L. Vasa, K. Mueller, J. Ostermann, and T. Wiegand. *A survey on coding of static and dynamic 3D meshes*, volume 0. Springer Verlag, dez 2007. 37

[112] Olga Sorkine, Daniel Cohen-Or, and Sivan Toledo. High-pass quantization for mesh encoding. In *Proceedings of the Eurographics/ACM SIGGRAPH Symposium on Geometry Processing*, pages 42–51. Eurographics Association, 2003. 34

[113] Nikolce Stefanoski and Joern Ostermann. Connectivity-guided predictive compression of dynamic 3d meshes. In *International Conference on Image Processing*, pages 2973–2976, oct 2006. 42, 100, 142

[114] G. Taubin and J. Rossignac. Geometric compression through topological surgery. *ACM Transactions on Graphics*, 17(2):84–115, 1998. 31

[115] Gabriel Taubin, André Guéziec, William Horn, and Francis Lazarus. Progressive forest split compression. In *SIGGRAPH '98: Proceedings of the 25th annual conference on Computer graphics and interactive techniques*, pages 123–132, New York, NY, USA, 1998. ACM. 36

[116] Gabriel Taubin and Jarek Rossignac. Geometric compression through topological surgery. *ACM Trans. Graph.*, 17(2):84–115, 1998. 24

[117] Costa Touma and Craig Gotsman. Triangle mesh compression. In *Graphics Interface*, pages 26–34, 1998. 3, 27, 31, 32, 33, 34, 35, 67, 114, 125, 131, 142

[118] G. Turan. Succinct representations of graphs. *Discrete Applied Math*, 8:289–294, 1984. 24

[119] Miguel Vieira and Kenji Shimada. Surface mesh segmentation and smooth surface extraction through region growing. *Comput. Aided Geom. Des.*, 22(8):771–792, 2005. 45

[120] Jeffrey Scott Vitter. Design and analysis of dynamic huffman codes. *J. ACM*, 34(4):825–845, 1987. 15

[121] M. Wagner and D. Saupe. Rd-optimization of hierarchical structured adaptive vector quantization for video coding. In *Proceedings of IEEE on Data Compression*, page 576, 2000. 104, 113

[122] T. A. Welch. A technique for high-performance data compression. volume 17, pages 8–19, Los Alamitos, CA, USA, 1984. IEEE Computer Society Press. 10

[123] Ian H. Witten, Radford M. Neal, and John G. Cleary. Arithmetic coding for data compression. *Communications of the ACM*, 30(6):520–540, 1987. ISSN 0001-0782. 26, 57, 101, 109, 124

[124] Jianhua Wu and Leif Kobbelt. Structure recovery via hybrid variational surface approximation. *Comput. Graph. Forum*, 24(3):277–284, 2005. 45

[125] Hitoshi Yamauchi, Seungyong Lee, Yunjin Lee, Yutaka Ohtake, Alexander Belyaev, and Hans-Peter Seidel. Feature sensitive mesh segmentation with mean shift. In *SMI '05*, pages 238—245, Washington, DC, USA, 2005. IEEE Computer Society. 45

[126] Zhidong Yan, Sunil Kumar, and C.-C. Jay Kuo. Error-resilient coding of 3-d graphic models via adaptive mesh segmentation. *IEEE Trans. Circ. Syst. Video Tech.*, 11(7):860–873, 2001. ISSN 1051-8215. 73, 121

[127] Eugene Zhang, Konstantin Mischaikow, and Greg Turk. Feature-based surface parameterization and texture mapping. *ACM Trans. Graph.*, 24(1):1–27, 2005. 45

[128] Hao Zhang and Rong Liu. Mesh segmentation via recursive and visually salient spectral cuts. In *Proc. of Vision, Modeling, and Visualization*, pages 429–436, 2005. 45

[129] Kun Zhou, John Snyder, Baining Guo, and Heung-Yeung Shum. Iso-charts: Stretch-driven mesh parameterization using spectral analysis. In *Symposium on Geometry Processing*, pages 47–56, 2004. 45

[130] Jacob Ziv and Abraham Lempel. A universal algorithm for sequential data compression. *IEEE Transactions on Information Theory*, 23:337–343, 1977. 10

[131] Malte Zöckler, Detlev Stalling, and Hans-Christian Hege. Fast and intuitive generation of geometric shape transitions. *The Visual Computer*, 16(5):241–253, 2000. 45

[132] E. Zuckerberger. Polyhedral surface decomposition with applications. *Computers and Graphics*, 26(5):733–743, October 2002. 45